

# Portfolios in Stochastic Local Search: Efficiently Computing Most Probable Explanations in Bayesian Networks

Ole J. Mengshoel  
RIACS  
NASA Ames Research Center  
Mail Stop 269-3  
Moffett Field, CA 94035  
omengshoel@riacs.edu

Dan Roth  
Department of Computer Science  
University of Illinois at Urbana-Champaign  
1304 West Springfield Avenue  
Urbana, IL 61801  
danr@cs.uiuc.edu

David C. Wilkins  
Center for the Study of Language and Information  
Stanford University  
Stanford, CA 94305  
dwilkins@stanford.edu

## Abstract

In this article we investigate the use of portfolios (or collections) of heuristics when solving computationally hard problems using stochastic local search. We consider uncertainty reasoning, specifically the computation of most probable explanations in Bayesian networks (BNs). Our contribution is two-fold. First, we introduce a portfolio-based stochastic local search approach that utilizes an initialization portfolio and a search portfolio. Second, and leveraging our portfolio approach, we investigate determinism in BNs. We introduce a novel additive measure of gain (or gradient), which is tailored to partly deterministic Bayesian networks, and show how it generalizes the MAXSAT measure of gain from stochastic local search for the satisfiability (SAT) problem. This measure of gain provides, along with an explanation’s gain in probability and their respective noisy variants, different heuristics that locally improve an explanation.

Our approach is implemented in a stochastic local search system, Stochastic Greedy Search. Stochastic Greedy Search is here compared to the state-of-the-art inference system HUGIN, which performs BN inference by compilation to and propagation in clique trees. We report on experiments using partly deterministic Bayesian networks from applications as well as synthetically generated networks. On synthetic networks, Stochastic Greedy Search speeds up computation approximately two orders of magnitude compared to HUGIN. In application networks, our approach is highly competitive in Bayesian networks with a high degree of determinism. In addition to showing that stochastic local search can be competitive with clique tree clustering, our empirical results provide an improved understanding of the circumstances under which portfolio-based stochastic local search outperforms clique tree clustering and vice versa. While our focus here is not on adaptation or learning during search, we believe that our framework and results can in turn enable innovations in those areas, thereby further enhancing stochastic local search algorithms.

## 1 Introduction

In this article we study the problem of computing a most probable explanation (MPE) in Bayesian networks [45]. Informally, an MPE is an instantiation of all non-evidence nodes in a BN such that no other instantiation has greater probability. (A formal definition of MPE is provided in Section 2 of this article.) MPE computation is a problem that is common to probabilistic formulations of diagnosis, image processing, and error correction decoding [10, 11, 31, 55, 56]. Ideally, one would prefer to use exact algorithms for MPE computation — for example structure-based algorithms such as clique tree clustering [2, 28, 53], conditioning, [23, 44, 45], or variable elimination [29, 58]. However, the MPE problem is computationally hard [1, 54], and this hardness manifests itself in slow execution times even in relatively simple networks that are used in current applications. The complexity of structure-based algorithms depends on the treewidth of a BN’s

underlying graph or the optimal maximal clique size of a BN’s induced clique tree [3,6,7]. Due to their very large treewidths or optimal maximal clique sizes, exact algorithms have proven to be infeasible or impractical in many application BNs.

Because of the limitations of exact algorithms, along with the importance of the MPE problem in applications, the development of improved algorithms for MPE computation is crucially important. Stochastic local search (SLS) algorithms have proven to be competitive in solving computationally hard problems including satisfiability (SAT) [14,21,48,50,51], the most probable explanation [25,26,30,34,37], and the maximum a posteriori (MAP) hypothesis in BNs [41,42]. Unfortunately, the theoretical understanding of SLS algorithms has been lagging [19], and despite recent significant progress [18,36,38] it is clear that further advances are needed. This work is part of a larger research effort where the ultimate goal is the development of highly adaptive but well-understood SLS algorithms, including SLS algorithms for computation in BNs. Progress on such adaptive SLS algorithms has already been made, for example in the areas of adaptive noise [8,17,33] and learning predictive models [22,27,47,57]. In these adaptive SLS algorithms, there is a need to search in the space of SLS search parameters in addition to the fundamental SLS search process (say, for an MPE). In other words, there is a two-level or two-step process: object-level SLS search, and meta-level search for approximately optimal SLS parameters that control the object-level SLS search process. To make the vision of this two-level or two-step process a reality, we believe that an improved understanding of the object-level search space is essential, hence we set in this paper out to provide such improved understanding.

This article provides an improved understanding of the role of portfolios of heuristics in stochastic local search. More specifically, our contribution is two-fold. First, we investigate the use of portfolios of heuristics in stochastic local search. These methods support the combination of a wide range of exact and stochastic algorithms, and seem to be a promising research direction [12,13,24,25,37,57]. Algorithmically, we introduce a portfolio-based stochastic local search approach which utilizes an initialization portfolio and a search portfolio. Our approach is implemented in the Stochastic Greedy Search (SGS) system, which is presented in this article. There are in fact two SGS algorithms both of which are discussed here; we call them SIMPLESGS and OPERATORSGS respectively. The latter is a portfolio-based [12,24,37,57] approach to MPE computation using SLS. It provides a flexible and general framework for stochastic explanation initialization and search. Specifically, OPERATORSGS allows us to combine different initialization operators (or initialization heuristics) and different search operators (greedy and noisy heuristics). Given our portfolio approach, one does not need to take a winner-takes-all approach to these different heuristics or operators. Instead, one can combine (and eventually adaptively tune) them according to the problem instance and application-specific requirements at hand. Using and analyzing this portfolio approach, we make progress related to the use of a variety of different initialization and search algorithms operators.

As a second contribution, and leveraging our portfolio approach, we consider novel heuristics for handling determinism in BNs. This approach, which we here carefully relate to MAXSAT, adds rather than multiplies probabilities when computing the utility of an explanation, and we therefore call it additive utility. Gain (i.e., change in utility) functions merely measure the progress made when one BN node’s state is flipped in an explanation. The traditional gain function, multiplicative gain, computes change in an explanation’s probability resulting from a flip. Additive utility (or gain) is a generalization of MAXSAT utility (or gain) from the GSAT and WALKSAT algorithms and their descendants [21,48,50,51] to the probabilistic setting. We carefully formalize and experiment with multiplicative and additive gain. Both gain functions are utilized in greedy and noisy search heuristics, which make up the search portfolio, where the noisy heuristics are used to effectively escape local but non-global optima.

Empirically, the performance of SGS is compared to that of the state-of-the-art inference system HUGIN, which implements a clique (or join) tree algorithm [2,5,28]. The clique tree algorithm is well-known and among the best performing inference methods for Bayesian networks. With minor adjustments, the clique tree algorithm computes either marginals [28] or MPEs [5]. We experiment with HUGIN and SGS on both synthetic and application BNs. Comparisons to HUGIN show that Stochastic Greedy Search performs significantly better for certain randomly generated Bayesian networks as well as for partly deterministic Bayesian networks from applications.

Randomly generated problem instances may be quite easy, and therefore experimenting with them may not constitute a proper evaluation of the algorithms considered [51]. To address this problem, we have developed an experimental paradigm for generating hard and easy synthetic BN instances [35,39]. Let  $V$  be the number of root nodes in a BN, and let  $C$  be the number of leaf nodes, and consider synthetic bipartite BNs generated by systematically varying  $C$  and  $V$  [34,39]. Our results on synthetic BNs, where root nodes have uniform distributions and leaf nodes are **or**-nodes, are as follows. First, as the  $C/V$ -ratio increases, the measured run times of both HUGIN and SGS increase at an approximately exponential rate as a function

of increasing  $C/V$ -ratio. However, given a suitable measure of gain, and specifically additive gain, SGS is approximately two orders of magnitude faster than HUGIN.

We have also found that our algorithm is quite effective on application networks with substantial determinism, and in many cases it performs comparably to or better than HUGIN. Here, one significant component in the success of SGS is the stochastic initialization portfolio, in particular its dynamic programming and forward simulation algorithms [37, 38]. The heuristics in the initialization portfolio suggest good starting points to the stochastic local search component of SGS (see also [26, 41, 42]). Two other significant reasons for SGS’s success in application BNs are the ability to exploit different search operators in OPERATORSGS, and the additive measure of gain also turns out to be surprisingly powerful in these networks.

The rest of this article is organized as follows. Section 2 introduces the problem of computing a most probable explanation (MPE) as well as related results, definitions, and notation. Section 3 discusses measures of utility and gain which form the basis of all local search algorithms, including SGS, and pays special attention to the additive approach. In Section 4 we describe the overall structure of our stochastic local search approach, SGS. We present two SGS algorithms, namely SIMPLESGS and OPERATORSGS, and also discuss related research. Section 5 discusses analytical results. In Section 6 we turn to the experimental results of the article: Section 6.1 presents results for synthetically generated Bayesian networks, while in Section 6.2 we discuss experimental results for application BNs. Section 7 concludes and discusses future work. This article extends and revises our earlier reports on SGS [34, 37].

## 2 Preliminaries

This section briefly reviews some of the key definitions and results for Bayesian networks and MPE computation as they relate to our research. A Bayesian network (BN), formally introduced in Definition 1 below, represents a multi-variate probability distribution as a directed acyclic graph (DAG), where the nodes represent the random variables.

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

$$\Pr(\mathbf{x}) = \Pr(x_1, \dots, x_n) = \Pr(X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^n \Pr(x_i | \pi_{X_i}). \quad (1)$$

A conditional probability distribution  $\Pr(X_i | \Pi_{X_i})$  is also known as a conditional probability table (CPT). While BN nodes can be continuous, this article is restricted to the discrete case and we will take “BN node” to mean “discrete BN node”. Suppose that a BN node  $X$  has states  $\{x_1, \dots, x_m\}$ . We then use the notation  $\Omega_X = \Omega(X) = \{x_1, \dots, x_m\}$ . For simplicity, but without loss of generality, we often use binary nodes in the following, in which case a BN node  $X$  has  $|\Omega_X| = |X| = |\{0, 1\}| = 2$  states.

A BN may be given *evidence* by clamping some of its nodes to their observed states. An instantiation of the remaining nodes is an explanation, formally defined as follows.

**Definition 2 (Explanation)** *Consider a BN  $\beta = (\mathbf{X}, \mathbf{E}, \mathbf{P})$  with  $\mathbf{X} = \{X_1, \dots, X_n\}$  and evidence  $\mathbf{e} = \{X_1 = x_1, \dots, X_m = x_m\}$  where  $m < n$ . An explanation  $\mathbf{x}$  is defined as  $\mathbf{x} = \{x_{m+1}, \dots, x_n\} = \{X_{m+1} = x_{m+1}, \dots, X_n = x_n\}$ .*

One is often interested in computing  $\Pr(\mathbf{x} | \mathbf{e})$ . In order to simplify the exposition, we typically consider  $\Pr(\mathbf{x}, \mathbf{e})$  instead of  $\Pr(\mathbf{x} | \mathbf{e})$ ; this does not fundamentally change the computation since  $\Pr(\mathbf{x}, \mathbf{e}) = \Pr(\mathbf{x} | \mathbf{e})\Pr(\mathbf{e})$ .

Given evidence, one can perform various forms of BN inference. This article focuses on computing the most probable explanation, which is defined as follows.

**Definition 3 (Most probable explanation (MPE))** *Computing a most probable explanation (MPE) in a BN is the problem of finding an explanation  $\mathbf{x}^* \in \Omega(X_1) \times \dots \times \Omega(X_n)$  such that  $\Pr(\mathbf{x}^*) \geq \Pr(\mathbf{y})$ , where  $\mathbf{y} \in \Omega(X_1) \times \dots \times \Omega(X_n)$  is any other explanation in the BN. The set of the  $k$  most probable explanations is defined as  $\mathbf{X}^* = \{\mathbf{x}_1^*, \dots, \mathbf{x}_k^*\}$  where  $\Pr(\mathbf{x}^*) = \Pr(\mathbf{x}_1^*) = \dots = \Pr(\mathbf{x}_k^*)$ .*

In other words, no other explanation has higher probability than  $\mathbf{x}^* \in \mathbf{X}^*$ . When discussing an explanation  $\mathbf{x}$ , the BN  $\beta$  is typically left implicit. Note that there might be many explanations with the same probability, and for this reason we say “an” MPE rather than “the” MPE.

In this article we emphasize the connection between the SAT and MPE problems and now introduce a few relevant definitions.

**Definition 4 (SAT, SAT problem)** *A satisfiability (SAT) formula  $\phi = (\mathbf{x}, \mathbf{q}, \mathbf{c})$  is defined by  $V$  variables  $\mathbf{x} = \{x_1, \dots, x_V\}$ , a set of  $L$  literals  $\mathbf{q} = \{q_1, \dots, q_L\}$ , where  $q_i = x$  or  $q_i = \bar{x}$  for  $x \in \mathbf{x}$ , and  $C$  distinct clauses  $\mathbf{c} = \{c_1, \dots, c_C\}$ , where each clause consists of literals combined using the **or** (“ $\vee$ ”) connective. The satisfiability (SAT) problem is to determine whether there is a truth assignment  $\boldsymbol{\tau}: \mathbf{x} \rightarrow \{0, 1\}^V$  that makes  $c_j = 1$  for all  $1 \leq j \leq C$ . Such an assignment is called a satisfying (truth) assignment (or model).*

SAT is a celebrated NP-complete decision problem, and the SAT problem and algorithms for solving it are of central importance both in theoretical computer science and in artificial intelligence [14, 40, 49, 51]. In a SAT problem instance, we have  $c_i = 1$  if the  $i$ -th clause is satisfied;  $c_i = 0$  if it is not satisfied. The SAT utility measure is  $\prod_{i=1}^C c_i$ . Consequently,  $\prod_{i=1}^C c_i = 1$  for some truth assignment to variables if the formula is satisfiable, else  $\prod_{i=1}^C c_i = 0$  for all variable assignments.

The SAT utility measure is not very useful for hill-climbing purposes, since it does not distinguish between few and many satisfied clauses. The MAXSAT utility measure is therefore often used, and one considers the MAXSAT optimization problem.

**Definition 5 (MAXSAT, MAXSAT utility)** *Let  $\phi = (\mathbf{x}, \mathbf{q}, \mathbf{c})$  be a SAT formula where  $C = |\mathbf{c}|$ . The MAXSAT utility measure  $U_S$  of  $\phi$ , given a truth assignment  $\boldsymbol{\tau}$ , is the number of true clauses in  $\phi$ :  $U_S(\boldsymbol{\tau}) = |\{c_i \in \mathbf{c} \mid c_i = 1 \text{ and } 1 \leq i \leq C\}|$ . The maximum satisfiability problem (MAXSAT) is to find a truth assignment  $\boldsymbol{\tau}^*$  such that the number of true clauses  $U_S(\boldsymbol{\tau}^*)$  in  $\phi$  is maximized.*

MAXSAT defines an optimization problem where one optimizes the number of satisfied clauses,  $\sum_{i=1}^C c_i$ . Note that  $\sum_{i=1}^C c_i = C$  for some truth assignment if the formula is satisfiable, else  $\sum_{i=1}^C c_i < C$  for all variable assignments. If  $\sum_{i=1}^C c_i = C$  is reached in MAXSAT optimization, one has also solved the underlying SAT problem. Since SAT is NP-complete, clearly MAXSAT is NP-complete as well, however MAXSAT is much more useful for hill-climbing.

A key contribution in this article, see Section 3, is how the MAXSAT utility measure can be generalized to an additive BN measure which is then utilized in MPE search. In order to set the stage we introduce these definitions.

**Definition 6 (SAT-equivalent, SAT-like)** *A BN  $\beta$  is SAT-equivalent to a SAT formula  $\phi$  (and vice versa) if there is a one-to-one mapping between  $\beta$  and  $\phi$ . A BN  $\beta$  is SAT-like if there exists a formula  $\phi$  that is SAT-equivalent to  $\beta$ .*

The constructions underlying Definition 6 are well-known [4, 46, 54] and we do not discuss details in this article. Along similar lines, one can define an explanation  $\mathbf{x}$  in a BN  $\beta$  as SAT-equivalent to a truth assignment  $\boldsymbol{\tau}$  in a formula  $\phi$ , where  $\beta$  is SAT-equivalent to  $\phi$  and there is a one-to-one mapping between the individual assignment of states to nodes in  $\mathbf{x}$  and the assignments of truth values to individual variables in  $\boldsymbol{\tau}$ . Here, a special case of great interest occurs when an MPE  $\mathbf{x} = \mathbf{x}^*$  is SAT-equivalent to a satisfying assignment  $\boldsymbol{\tau} = \boldsymbol{\tau}^*$ .

We now introduce a few additional SAT-related concepts. Below,  $\tau$  maps one variable  $x$  to  $\{0, 1\}$  while  $\bar{\tau}$  maps all  $V$  variables  $\mathbf{x}$  in  $\phi$  to  $\{0, 1\}^V$ . The notation  $\overline{\tau(x)}$  means that variable  $x$ ’s truth assignment  $\tau(x)$  is inverted;  $\overline{\tau(x)} = 1$  iff  $\tau(x) = 0$  and  $\overline{\tau(x)} = 0$  iff  $\tau(x) = 1$ .

**Definition 7** *In a SAT problem instance  $\phi = (\mathbf{x}, \mathbf{q}, \mathbf{c})$  with truth assignment  $\boldsymbol{\tau}$ , let  $x \in \mathbf{x}$ . The notation  $\bar{\boldsymbol{\tau}}(\phi, x, \boldsymbol{\tau})$  (often abbreviated  $\bar{\boldsymbol{\tau}}$ ) is a new truth assignment with  $\bar{\boldsymbol{\tau}}(x) = \overline{\tau(x)}$  and where for all  $y \in \mathbf{x} - \{x\}$ ,  $\bar{\boldsymbol{\tau}}(y) = \tau(y)$ . We say that  $\bar{\boldsymbol{\tau}}(\phi, x, \boldsymbol{\tau})$  is flipped compared to  $\boldsymbol{\tau}$ .*

SAT is relevant to the study of BNs for several reasons. First, BNs often have many deterministic nodes, of which the **or**- and **and**-nodes found in SAT are special cases. For examples of application BNs with many deterministic nodes, we refer to Figure 3. Second, significant progress on stochastic local search to solve the satisfiability problem has been made in recent years [14, 21, 48, 50, 51]. Of particular relevance to our work is (i) the use of the GSAT measure of gain, which is based on the MAXSAT utility measure, and (ii) the use

of noise as a mechanism of escape from local optima. In the research reported here, we have extended the MAXSAT utility measure and the GSAT measure of gain to the probabilistic setting, giving the additive utility measure and the additive measure of gain respectively.

It can be shown by reduction from SAT that MPE computation is NP-hard [54]. Approximating an MPE to within a constant ratio-bound has also been proven to be NP-hard [1]. In fact, the problem is much harder, since the evaluation problem is  $\#P$  complete [46]. Since inference in BNs is computationally hard and the MPE problem is important in applications, we believe that it is important to study SLS algorithms for MPE computation, where estimates of  $\mathbf{x}^*$  are computed.

**Definition 8 (MPE (lower-bound) estimate)** *Let  $\mathbf{x}^*$  be an MPE. A best-effort estimate of  $\mathbf{x}^*$  is denoted  $\hat{\mathbf{x}}^*$ ; if  $\Pr(\hat{\mathbf{x}}^*) \leq \Pr(\mathbf{x}^*)$  then  $\hat{\mathbf{x}}^*$  is a lower-bound estimate.*

In the remainder of this article, our main focus is on lower-bound MPE estimates  $\hat{\mathbf{x}}^*$  computed using stochastic local search combined with initialization algorithms. Particular attention is paid to SLS algorithms as Las Vegas algorithms [21]. In other words, we typically assume that  $\Pr(\mathbf{x}^*)$  is a known SLS input parameter, therefore SLS terminates once an  $\mathbf{x}^* \in \mathbf{X}^*$  has been found. Las Vegas algorithms are used in theoretical computer science as well as in SLS research [12, 21], and enables our scientific study of SLS algorithms as further discussed in Section 5.1.

SLS algorithms can be analyzed using discrete time Markov chains with discrete state spaces, and we consider time-homogenous Markov chains with finite state spaces in this article. A Markov chain is a stochastic process  $(A_t, t \geq 0) = (A_0, A_1, \dots)$  induced by a 3-tuple  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$ , where  $\mathcal{S}$  is  $k = |\mathcal{S}|$  discrete states, a  $k$ -dimensional vector  $\mathcal{V} = (\pi_1, \dots, \pi_k)$  represents the initial probability distribution, and a  $k \times k$  matrix  $\mathcal{P}$  represents the transition probabilities. As we will elaborate in Section 5,  $\mathcal{M}$  is given by the objective function, the SLS algorithm, and the SLS algorithm's parameter settings.

In  $\mathcal{M}$ , some states  $\mathcal{O} \subset \mathcal{S}$  represent optimal states, and we introduce the following definition.

**Definition 9 (SLS model)** *Let  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$  define a Markov chain. Further, assume an objective function  $f : \mathcal{S} \rightarrow \mathbb{R}$  and an optimal objective function value  $f^* \in \mathbb{R}$  that defines optimal states  $\mathcal{O} = \{s \mid s \in \mathcal{S} \text{ and } f(s) = f^*\}$ . The 2-tuple  $(\mathcal{M}, \mathcal{O})$  defines an SLS model.*

The objective function  $f$  and the optimal states  $\mathcal{O}$  are independent of the SLS algorithm and its parameters; finding an  $s^* \in \mathcal{O}$  is the purpose of SLS search including MPE computation. We emphasize maximization in the form of MPE computation in this article, therefore  $f$  in Definition 9 is given by  $\Pr(\mathbf{x})$  in Definition 1. We often consider BNs with binary nodes. In this case, explanations can be represented as bitstrings of length  $n$ ,  $\mathbf{b} \in \{0, 1\}^n$ , and  $s^* = \mathbf{b}^* \in \{0, 1\}^n$ .

Consider an SLS model  $(\mathcal{M}, \mathcal{O})$ . A hitting time analysis of  $\mathcal{M}$ , known from Markov chain theory, gives the expected number of search steps needed to reach  $s^* \in \mathcal{O}$ . Hitting times are based on first passage times.

**Definition 10 (First passage time)** *Consider an SLS model  $(\mathcal{M}, \mathcal{O})$  and let  $s_i \in \mathcal{S}$  where  $\mathcal{S}$  is  $\mathcal{M}$ 's states. The first passage time  $T$ , a random variable, into  $s^* \in \mathcal{O}$  is given by  $T = \min(j \geq 0 : A_j \in \mathcal{O})$ . The expected value of  $T$ , given initial state  $A_0 = s_i$ , is defined as*

$$m_{i,\mathcal{O}} := E(T \mid A_0 = s_i).$$

Definition 10 covers  $|\mathcal{O}| \geq 1$  and thus includes first passage time into multiple optimal states. For simplicity we emphasize the one-state case  $|\mathcal{O}| = 1$  here, with  $\mathcal{O} = \{s^*\} = \{s_k\}$ . First passage time now simplifies to  $T = \min(j \geq 0 : A_j = s_k)$ , and  $m_{i,\mathcal{O}}$  simplifies to  $m_{i,k}$ .

Using conditional expectations, one obtains from Definition 10 the following well-known result.

**Theorem 11 (Expected hitting time)** *Let  $\mathcal{M}$  be a Markov chain with state space  $\mathcal{S} = \{s_1, \dots, s_k\}$  and first passage time  $T$  (into  $s^* = s_k$ ). The expected hitting time  $h$  is then*

$$h := \sum_{i=1}^k E(T \mid A_0 = i) \Pr(A_0 = i) = \sum_{i=1}^k m_i \pi_i. \quad (2)$$

Expected hitting time can be used to analyze the expected time to reach an optimal state  $s^* \in \mathcal{O}$  in an SLS model  $(\mathcal{M}, \mathcal{O})$ , and is often closely related to the observed mean run time for an SLS algorithm. For SLS, the hitting time  $h$  is with respect to some state in  $\mathcal{O}$  and depends on the algorithm's input parameters including the problem instance. Previously, we have studied hitting time  $h(p_N)$ , where  $p_N$  is the

noise probability [36]. By varying  $p_N$ , we have constructed expected hitting time curves that are analytical counterparts to experimental noise response curves [36]. These expected hitting time curves provide an analytical foundation for finding optimal noise level  $p_N^*$ . This noise level is optimal in the sense that it minimizes expected hitting time, or  $p_N^* = \arg \min_{0 \leq p \leq 1} h(p)$ .

### 3 Measures of Utility and Gain

Stochastic local search algorithms are based on evaluating the absolute goodness (utility) of an explanation and relative goodness (gain) of an explanation compared to explanations that are neighbors in the search space. Gain is, informally speaking, change in utility from one explanation  $\mathbf{x}$  to another explanation  $\mathbf{x}'$ . The explanation  $\mathbf{x}'$  may be derived from  $\mathbf{x}$ , for example by changing (flipping) one node’s state.

Measures of utility and gain play essential roles in any local search algorithm, including SGS, and this section focuses on these matters. We first present measures of utility in Section 3.1, then measures of gain in Section 3.2. We highlight two different approaches, the multiplicative approach and the additive approach. While the multiplicative approach is natural, the additive approach is related to MAXSAT and turns out to give excellent performance in partly deterministic BNs. The basis for how the measures of gain are utilized in stochastic local search is described in Section 3.3.

Readers interested in our portfolio approach but not in our approach to handling zeros in BNs may want to skip this section.

#### 3.1 Measures of Utility

A utility measure (or function) is also known as a cost function, an energy function, a fitness function, or an objective function. We introduce the notion of a utility measure  $U(\mathbf{x})$ , where  $\mathbf{x}$  is an explanation in a BN.

##### 3.1.1 Multiplicative Utility

As one might expect, the definition of a joint probability from Equation 1 is used as a utility measure in SLS. For purposes of notational convenience and uniform treatment in our SLS algorithms we also consider it a multiplicative utility  $U_M$ .

**Definition 12 (Multiplicative utility)** *Let  $\mathbf{x}$  be an explanation in a BN with  $n$  nodes. The multiplicative utility is defined as*

$$U_M(\mathbf{x}) = \Pr(\mathbf{x}) = \prod_{i=1}^n \Pr(x_i \mid \pi(X_i)). \quad (3)$$

While multiplicative utility is a natural utility measure, it does have limitations, in particular in BNs with many zeros in their CPTs and consequently many explanations  $\mathbf{x}$  with  $U_M(\mathbf{x}) = \Pr(\mathbf{x}) = 0$ .

To obtain further insight into partly deterministic BNs, suppose that zeros are assigned CPT entries in an existing BN according to the following randomized algorithm. The algorithm is controlled by a probability parameter  $\rho$ , where  $0 \leq \rho \leq 1$  (see [9] for details). With the exception of the last entry in a CPT column,  $\rho$  is the probability that a CPT entry is set to zero. Specifically,  $\rho$  controls the setting of zeros in the CPT entries of a BN node in the following manner. We assume a random number generator  $\text{RANDOM}(a, b)$  that generates a real number  $r$ , where  $a \leq r \leq b$ , uniformly at random. Let  $V$  be an arbitrary non-root node in the BN, and let  $\mathbf{y}$  be an arbitrary instantiation of parent nodes  $\mathbf{Y} = \Pi(V)$ . The algorithm sequentially assigns probabilities to  $V$ ’s states, given  $\mathbf{y}$ , and distinguishes these two cases. Case 1 is for a state  $v_i \in \Omega_V$  that is not the last state of  $V$ : If  $\text{RANDOM}(0, 1) < \rho$  then let  $\Pr(V = v_i \mid \mathbf{Y} = \mathbf{y}) = 0$ , else let  $\Pr(V = v_i \mid \mathbf{Y} = \mathbf{y}) > 0$ . Case 2 is for the state  $v_{|V|} \in \Omega_V$  that is the last state: Let  $\Pr(V = v_{|V|} \mid \mathbf{Y} = \mathbf{y}) = 1 - \sum_{i=1}^{|V|-1} \Pr(V = v_i \mid \mathbf{Y} = \mathbf{y})$ . The situation where  $V$  is a root node in the BN is handled in a similar manner.

For the above randomized algorithm, the following theorem holds.

**Theorem 13** *Consider a BN  $\beta$  with  $n$  nodes, with  $w = \Omega(V_1) = \dots = \Omega(V_n)$ , and where CTPs are set according to the above randomized algorithm. Let  $\mathbf{x}$  be an explanation picked uniformly at random in  $\beta$ . Then*

$$\Pr(U_M(\mathbf{x}) = 0) = 1 - \left(1 - \frac{w-1}{w} \times \rho\right)^n. \quad (4)$$

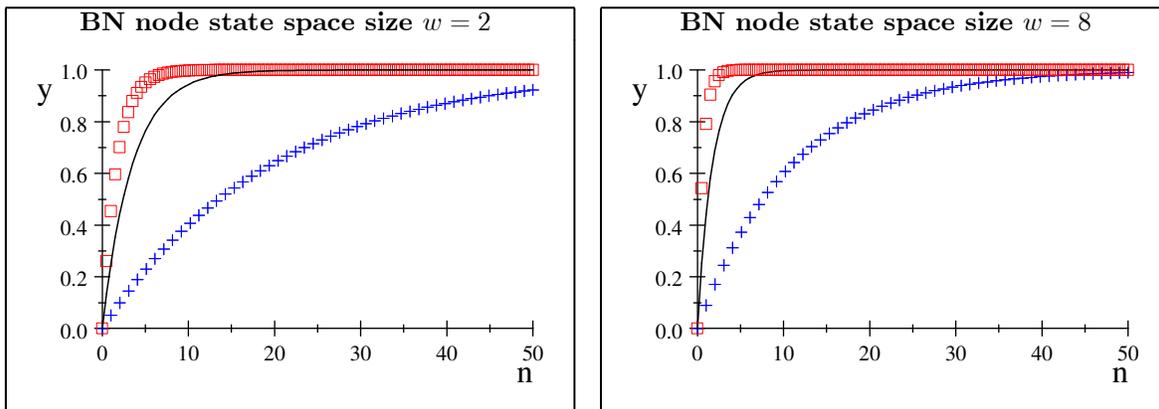


Figure 1: The probability of a randomly picked BN explanation  $\mathbf{x}$  being zero, or  $y = \Pr(U_M(\mathbf{x}) = 0)$ , as a function of varying BN size  $n$  (along the  $x$ -axis), CPT entry zero probability  $\rho$ , and BN node state space size  $w$ . The curves are for varying  $\rho$ , with  $\rho = 0.1$  (curve indicated by blue crosses),  $\rho = 0.5$  (curve indicated by black line), and  $\rho = 0.9$  (curve indicated by red squares). The BN node state space size is different in these two panels. *Left panel*: State space size  $w = 2$ ; *Right panel*: State space size  $w = 8$ .

This theorem is a slight variation on a result in [9]. Figure 1 illustrates (4), and clearly shows the rapid speed at which the probability  $\Pr(U_M(\mathbf{x}) = 0)$  grows, for an explanation  $\mathbf{x}$  chosen uniformly at random, with growing BN size  $n$  and probability parameter  $\rho$ . The slowest growth for  $\Pr(U_M(\mathbf{x}) = 0)$  in Figure 1 is for  $w = 2$  and  $\rho = 0.1$ , where for  $n = 10$  we have  $\Pr(U_M(\mathbf{x}) = 0) \approx 0.4$ . The fastest growth, on the other hand, takes place for  $w = 8$  and  $\rho = 0.9$ , where already for  $n = 10$  we have  $\Pr(U_M(\mathbf{x}) = 0) \approx 1.0$ . In other words, almost all explanations picked uniformly at random have zero probability or  $U_M(\mathbf{x}) = 0$ . Since most application BNs of interest contain substantially more than  $n = 10$  nodes, the effect of zeros in CPTs is often much worse than indicated by these two examples.

As presented in Table 3, a large percentage of zeros can often be found in CPTs of application BNs. Theorem 13 suggests that in such BNs, if an explanation  $\mathbf{x}$  is picked uniformly at random,  $U_M(\mathbf{x}) = \Pr(\mathbf{x}) = 0$  with high probability. Further, by the SAT reduction it is NP-hard to find an explanation  $\mathbf{x}$  such that  $\Pr(\mathbf{x}) > 0$  [46, 54]. The case  $U_M(\mathbf{x}) = \Pr(\mathbf{x}) = 0$  therefore presents a serious complication for the SLS approach to computing MPEs. With a large part of the search space of explanations equal to zero, there is often no gradient to hill-climb for an SLS algorithm. This limitation of the multiplicative measure leads us to the additive approach, which we discuss next.

### 3.1.2 Additive Utility

Due to the limitations of multiplicative utility, we now consider a complementary utility measure, namely additive utility  $U_A$  [34, 37]. This utility measure is based on the MAXSAT utility measure, which was used in the seminal GSAT family of algorithms [50, 51].

**Definition 14 (Additive utility)** *Let  $\mathbf{x}$  be an explanation in a BN with  $n$  nodes. The additive utility is defined as*

$$U_A(\mathbf{x}) = \sum_{i=1}^n \Pr(x_i \mid \pi(X_i)). \quad (5)$$

Additive utility clearly avoids multiplicative utility’s problem with zeros as discussed above in Section 3.1.1. The detailed investigation and use of the additive utility measure in BN inference is, to our knowledge, a novel aspect of our research as reported in this article.

We provide formal and experimental arguments in support of additive utility and gain in the following. First, as shown in Theorem 15 below, additive utility is a generalization of MAXSAT utility in the following sense. MAXSAT utility adds the number of satisfied CNF clauses [49, 50], given a truth assignment to the logic variables. In a SAT-like BN, additive utility adds the probabilities of root and leaf nodes, given an instantiation of root nodes (which correspond to SAT variables) and clamping of the leaf nodes (which correspond to SAT clauses) to 1. Second, additive utility is an approximation to multiplicative utility.

(Formal results concerning the relationship between additive gain and multiplicative gain are provided later in Section 3.) Third, it turns out that additive utility and gain give excellent empirical results in certain BNs as reported in Section 6.2 and Section 6.1.

**Theorem 15 (Generalization of MAXSAT)** *Let  $\phi = (\mathbf{y}, \mathbf{q}, \mathbf{c})$  be a SAT problem instance with  $|\mathbf{y}| = V$ ,  $|\mathbf{c}| = C$ , and  $n = C + V$ . Let  $\beta = (\mathbf{X}, \mathbf{E}, \mathbf{P})$  be a Bayesian network that is SAT-equivalent to  $\phi$  and with leaf nodes clamped, i.e.  $\mathbf{e} = \{X_1 = 1, \dots, X_C = 1\}$ . Let  $\boldsymbol{\tau}$  be a truth assignment to the logic variables  $\mathbf{y} = \{y_1, \dots, y_V\}$  in  $\phi$  and let  $\mathbf{x} = \{X_{C+1} = x_{C+1}, \dots, X_n = x_n\}$  be an equivalent explanation over the  $V$  root nodes in  $\beta$ . Then  $U_S(\boldsymbol{\tau}) = k$  if and only if<sup>1</sup>*

$$U_A(\mathbf{x}) = k + \frac{V}{2}. \quad (6)$$

**Proof.** Case ( $\Leftarrow$ ): Consider a SAT-equivalent BN  $\beta$  with an explanation  $\mathbf{x}$ , and suppose that  $U_A(\mathbf{x}) = k + \frac{V}{2}$ . Since  $\beta$  has  $n$  nodes  $\{X_1, \dots, X_n\}$ , we can form

$$U_A(\mathbf{x}) = \sum_{i=1}^n \Pr(X_i = x_i \mid \pi(X_i)).$$

Because  $\beta$  is SAT-equivalent to some formula  $\phi$ ,  $\beta$  is bipartite with  $V$  root nodes (corresponding to variables in  $\phi$ ) and  $C = n - V$  leaf nodes (corresponding to clauses in  $\phi$ ). We decompose the additive utility  $U_A(\mathbf{x})$  accordingly to

$$U_A(\mathbf{x}) = \sum_{i=1}^C \Pr(X_i = x_i \mid \pi(X_i)) + \sum_{i=C+1}^n \Pr(X_i = x_i), \quad (7)$$

where  $\{X_{C+1}, \dots, X_n\}$  are root nodes and have no parents. By construction, the  $V = n - C$  root nodes are binary and each root node state has probability mass  $\frac{1}{2}$  [4, 46, 54]. Since there are  $V = n - C$  of them we obtain for the second term in (7)

$$\sum_{i=C+1}^n \Pr(X_i = x_i) = \sum_{j=1}^V \Pr(X_{j+C} = x_{j+C}) = \frac{V}{2}. \quad (8)$$

Clearly, (7) can be written as

$$\sum_{i=1}^C \Pr(X_i = x_i \mid \pi(X_i)) = U_A(\mathbf{x}) - \sum_{i=C+1}^n \Pr(X_i = x_i) = k, \quad (9)$$

where the last equality follows from  $U_A(\mathbf{x}) = k + \frac{V}{2}$  and (8). Now, in  $\beta$ ,  $C$  leaf nodes are being clamped as follows:  $\mathbf{e} = \{X_1 = 1, \dots, X_C = 1\}$ . Without loss of generality we assume a suitable ordering, and from (9) we know that for explanation  $\mathbf{x}$  the leaf nodes in  $\beta$  look like this:

$$\Pr(X_1 = 1 \mid \pi_{X_1}) = 1, \dots, \Pr(X_k = 1 \mid \pi_{X_k}) = 1, \Pr(X_{k+1} = 1 \mid \pi_{X_{k+1}}) = 0, \dots, \Pr(X_C = 1 \mid \pi_{X_C}) = 0;$$

Due to the equivalence between explanations in  $\beta$  and truth assignments in  $\phi$ , there is an assignment  $\boldsymbol{\tau}$  for  $\phi$  that is SAT-equivalent to explanation  $\mathbf{x}$ . And from the above it follows that exactly  $k \leq C$  clauses are satisfied in  $\phi$  under  $\boldsymbol{\tau}$ , or  $U_S(\boldsymbol{\tau}) = k$ . Case ( $\Rightarrow$ ) is similar, giving the desired result. ■

While the MAXSAT measure  $U_S$  applies to the variables in a SAT formula, the additive measure  $U_A$  applies to the full explanation in a BN, with BN nodes in a SAT-like BN corresponding to both variables and clauses in the SAT-equivalent formula. More importantly, additive utility is not restricted to the logical setting as MAXSAT is. The additive measure  $U_A$  can therefore be applied in all BNs, not only in SAT-like BNs.

Even though the  $U_A$  measure clearly does not compute an explanation's probability, it is very useful for the special but important case of deterministic BN nodes. As already noted, surprisingly many BNs from applications have substantial deterministic fragments. Examples of such partly deterministic BNs are SAT-like BNs as discussed above (with **or**-nodes), error correction decoding BNs (with **xor**-nodes) from information theory [10, 31], and other BNs from applications — see Table 3.

<sup>1</sup>Recall that  $U_s$  is MAXSAT utility; see Definition 5 for the formal definition.

## 3.2 Measures of Gain

For an explanation  $\mathbf{x}'$ , a measure of gain, say  $\Delta U$ , simply measures change in utility  $U$  compared to another explanation  $\mathbf{x}$ . When a node  $X$  is flipped in an explanation  $\mathbf{x}$ , it has a localized impact in the BN at hand. The following definition introduces a vocabulary for such localized changes. To simplify the exposition, we often assume  $|\Omega_X| = 2$  in the remainder of this section; this can easily be generalized to  $|\Omega_X| \geq 2$ .

**Definition 16 (Flipped node, flipped explanation)** *Consider an instantiated BN node  $X_i = x_i$ . If one puts  $X_i = \bar{x}_i$ , where  $\bar{x}_i = 0$  if  $x_i = 1$  and  $\bar{x}_i = 1$  if  $x_i = 0$ , then the node is flipped (from  $X_i = x_i$ ). An explanation  $\mathbf{x}'$  is flipped from another explanation  $\mathbf{x}$  if at least one node  $X_i$  has  $X_i = x_i$  in  $\mathbf{x}$  while  $X_i = \bar{x}_i$  in  $\mathbf{x}'$ .*

We note that the terminology “flipping” does *not* imply that CPTs are being changed — we are not performing machine learning but rather search.

In the following definition, the notation  $\mathbf{x}[X_i = x_i]$  means that the current state of the BN node  $X_i$  is  $x_i$  in the explanation  $\mathbf{x}$ ; no changes are made. The notation  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ , on the other hand, means that  $X_i$ , which was assigned state  $x_i$  in  $\mathbf{x}$ , is now (after the assignment  $X_i \leftarrow \bar{x}_i$ ) assigned state  $\bar{x}_i$  in  $\mathbf{x}'$ . This is similar to assignment in programming languages.

**Definition 17 (One-flipped)** *An explanation  $\mathbf{x}' = \{X_1 = x_1, \dots, X_i = \bar{x}_i, \dots, X_n = x_n\}$  is one-flipped from an explanation  $\mathbf{x} = \{X_1 = x_1, \dots, X_i = x_i, \dots, X_n = x_n\}$  if exactly one node, here the  $i$ -th node, is flipped. For one-flipped explanations the notation  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ , where  $\mathbf{x}'$  is the new explanation, is also used. A one-flipped parent instantiation, denoted  $\bar{\pi}_X$ , means that some node  $Y \in \Pi_X$  has its state flipped from  $y$  to  $\bar{y}$  in  $X$ 's parent instantiation  $\pi_X$ .*

At the core of SLS algorithms is an evaluation of whether the state of a node  $X_i$  should be flipped from  $x_i$  to  $\bar{x}_i$ , leading to a corresponding one-flipped change in an explanation from  $\mathbf{x}[X_i = x_i]$  to  $\mathbf{x}[X_i \leftarrow \bar{x}_i]$ . These evaluations are based on the concept of gain. We now discuss multiplicative gain, which is derived from multiplicative utility.

### 3.2.1 Multiplicative Gain

The multiplicative gain is obtained as follows. Let  $\mathbf{x}$  be the explanation before flipping,  $\mathbf{x}'$  the explanation after flipping some number of nodes. A measure of the goodness of those flips is introduced in the following definition.

**Definition 18 (Multiplicative gain, one-flip multiplicative gain)** *Let  $\mathbf{x}$  and  $\mathbf{x}'$  be explanations, assume that  $U_M(\mathbf{x}) > 0$ , and suppose that we flip from  $\mathbf{x}$  to  $\mathbf{x}'$ . The multiplicative gain  $\Delta U_M(\mathbf{x}, \mathbf{x}')$  is defined as*

$$\Delta U_M(\mathbf{x}, \mathbf{x}') = \frac{U_M(\mathbf{x}')}{U_M(\mathbf{x})}. \quad (10)$$

*Suppose that  $x_{i,k} \in \Omega_{X_i}$ ,  $x_{i,j} \in \Omega_{X_i}$ , and  $\mathbf{x}[X_i = x_{i,k}]$ . The general case of one-flipping  $\mathbf{x}$ 's  $X_i$  from  $x_{i,k}$  to  $x_{i,j}$ , where  $j \neq k$ , is defined as*

$$\Delta U_M(\mathbf{x}, x_{i,j}) = \frac{U_M(\mathbf{x}[X_i \leftarrow x_{i,j}])}{U_M(\mathbf{x}[X_i = x_{i,k}])}. \quad (11)$$

**Definition 19 (Binary one-flip multiplicative gain)** *Assume that  $X_i$  is binary,  $\mathbf{x}[X_i = x_i]$ , and let  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ . Binary one-flip multiplicative gain is then defined as*

$$\Delta U_M(\mathbf{x}, \bar{x}_i) = \frac{U_M(\mathbf{x}')}{U_M(\mathbf{x})} = \frac{U_M(\mathbf{x}[X_i \leftarrow \bar{x}_i])}{U_M(\mathbf{x}[X_i = x_i])}. \quad (12)$$

We now show how multiplicative gain computation can be simplified compared to the above definitions.

**Theorem 20 (One-flip multiplicative gain)** *Let  $\mathbf{x}$  and  $\mathbf{x}'$  be explanations such that  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ . Further, let  $\mathbf{C} = \Psi_{X_i}$ , let  $C_j \in \mathbf{C}$ , and let  $\pi(C_j) \subseteq \mathbf{x}$  (respectively  $\bar{\pi}(C_j) \subseteq \mathbf{x}'$ ) be the instantiation of all parent nodes for children of  $X_i$  before (respectively after) a one-flip of  $X_i$ . Suppose that  $U_M(\mathbf{x}) > 0$ . The one-flip multiplicative gain  $\Delta U_M(\mathbf{x}, \bar{x}_i)$  is*

$$\Delta U_M(\mathbf{x}, \bar{x}_i) = \frac{\Pr(\bar{x}_i \mid \pi(X_i)) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j \mid \bar{\pi}(C_j))}{\Pr(x_i \mid \pi(X_i)) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j \mid \pi(C_j))}, \quad (13)$$

*with  $\Pr(x_i \mid \pi(X_i)) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j \mid \pi(C_j)) > 0$ .*

**Proof.** We start with the definitions of multiplicative gain and utility, obtaining

$$\Delta U_M(\mathbf{x}, \bar{x}_i) = \frac{U_M(\mathbf{x}[X_i \leftarrow \bar{x}_i])}{U_M(\mathbf{x})} = \frac{\Pr(\mathbf{x}[X_i \leftarrow \bar{x}_i])}{\Pr(\mathbf{x})}. \quad (14)$$

We now consider  $X_i$ 's children  $\mathbf{C} = \Psi_{X_i}$ . Let  $\mathbf{Y}$  be the BN nodes other than  $X_i$  and  $\mathbf{C}$ :  $\mathbf{Y} = \mathbf{V} - (\{X_i\} \cup \mathbf{C})$ , and rewrite Equation 14 as

$$\Delta U_M(\mathbf{x}, \bar{x}_i) = \frac{\Pr(X_i = \bar{x}_i | \pi(X_i)) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \bar{\pi}(C_j)) \times \prod_{Y \in \mathbf{Y}} \Pr(Y = y | \pi(Y))}{\Pr(X_i = x_i | \pi(X_i)) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j)) \times \prod_{Y \in \mathbf{Y}} \Pr(Y = y | \pi(Y))}.$$

Since only one node  $X_i$  is flipped (from  $x_i$  to  $\bar{x}_i$ ), we can exploit the Markov blanket locality of the BN and only need to consider the CPT in  $X_i$  itself as well as CPTs for children  $\mathbf{C}$ , and obtain

$$\Delta U_M(\mathbf{x}, \bar{x}_i) = \frac{\Pr(X_i = \bar{x}_i | \pi_{X_i}) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \bar{\pi}(C_j))}{\Pr(X_i = x_i | \pi_{X_i}) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j))}.$$

This is exactly the multiplicative gain  $\Delta U_M(\mathbf{x}, \bar{x}_i)$  in Equation 13. Obviously, since by assumption  $U_M(\mathbf{x}) > 0$  it is clear that  $\Pr(x_i | \pi(X_i)) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j)) > 0$  as well. ■

Note that the computation of  $\Pr(\mathbf{x}')$  of a flipped explanation can be expressed as taking the probability of the old explanation,  $\Pr(\mathbf{x})$ , and multiplying it with the gain from Equation 10, giving

$$\Pr(\mathbf{x}') = \Delta U_M(\mathbf{x}, \mathbf{x}') \Pr(\mathbf{x}). \quad (15)$$

We typically restrict ourselves to the one-flipped case in this article, and consequently have  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$  and thus obtain

$$\begin{aligned} \Pr(\mathbf{x}') &= \Pr(\mathbf{x}[X_i \leftarrow \bar{x}_i]) \\ &= \Delta U_M(\mathbf{x}, \bar{x}_i) \Pr(\mathbf{x}) \\ &= \frac{\Pr(X_i = \bar{x}_i | \pi_{X_i}) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \bar{\pi}(C_j))}{\Pr(X_i = x_i | \pi_{X_i}) \times \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j))} \Pr(\mathbf{x}). \end{aligned} \quad (16)$$

Especially when the number of nodes  $n$  is large, using (16) iteratively rather than evaluating all of  $\mathbf{x}'$  to compute  $\Pr(\mathbf{x}')$  provides substantial computational savings.

In the use of multiplicative gain in SGS and other SLS algorithms there is a slight but crucial complication in that multiplicative gain is not defined, according to Definition 18, if the denominator  $U_M(\mathbf{x}) = 0$ . In practice,  $U_M(\mathbf{x}) = 0$  is very common in partly deterministic BNs. We discuss how this can be handled in an ad-hoc manner in SGS in Section 4.1. However, this limitation of  $U_M(\mathbf{x})$  also motivates our investigation of additive gain which we turn to now.

### 3.2.2 Additive Gain

The assumption  $U_M(\mathbf{x}) = \Pr(\mathbf{x}) > 0$  made for multiplicative gain in Equation 10 is sometimes not realistic and leads us to introduce additive gain based upon additive utility.

**Definition 21 (Additive gain, one-flip additive gain)** *Let  $\mathbf{x}$  and  $\mathbf{x}'$  be explanations. The additive gain  $\Delta U_A(\mathbf{x}, \mathbf{x}')$  is defined as*

$$\Delta U_A(\mathbf{x}, \mathbf{x}') = U_A(\mathbf{x}') - U_A(\mathbf{x}). \quad (17)$$

*Suppose that  $x_{i,k} \in \Omega_{X_i}$ ,  $x_{i,j} \in \Omega_{X_i}$ , and  $\mathbf{x}[X_i = x_{i,k}]$ . The general case of one-flipping  $\mathbf{x}$ 's  $X_i$  from  $x_{i,k}$  to  $x_{i,j}$ , where  $j \neq k$ , is defined as*

$$\Delta U_A(\mathbf{x}, x_{i,j}) = U_A(\mathbf{x}[X_i \leftarrow x_{i,j}]) - U_A(\mathbf{x}[X_i = x_{i,k}]). \quad (18)$$

**Definition 22 (Binary one-flip additive gain)** *Assume that  $X_i$  is binary,  $\mathbf{x}[X_i = x_i]$ , and let  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ . Binary one-flip additive gain is then defined as*

$$\Delta U_A(\mathbf{x}, \bar{x}_i) = U_A(\mathbf{x}') - U_A(\mathbf{x}) = U_A(\mathbf{x}[X_i \leftarrow \bar{x}_i]) - U_A(\mathbf{x}[X_i = x_i]).$$

The additive gain is well-defined in cases where the multiplicative gain is not. As a consequence, we can in the SGS implementation of additive gain computation (see Figure 4) handle the case  $\Pr(\mathbf{x}) = 0$  without complicating the stochastic local search algorithm as is needed for multiplicative gain (see Figure 3). As we will see in experiments, additive gain proves useful as a complement to multiplicative gain.

**Theorem 23 (One-flip additive gain)** *Let  $\mathbf{x}$  and  $\mathbf{x}'$  be explanations such that  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ . Further, let  $\mathbf{C} = \Psi_{X_i}$ , let  $C_j \in \mathbf{C}$ , and let  $\pi(C_j) \subseteq \mathbf{x}$  (respectively  $\bar{\pi}(C_j) \subseteq \mathbf{x}'$ ) be the instantiation of all parent nodes for children of  $X_i$  before (respectively after) a one-flip of  $X_i$ . The one-flip additive gain  $\Delta U_A(\mathbf{x}, \bar{x}_i)$  is then*

$$\Delta U_A(\mathbf{x}, \bar{x}_i) = \Pr(\bar{x}_i | \pi_{X_i}) + \sum_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \bar{\pi}(C_j)) - \Pr(x_i | \pi_{X_i}) - \sum_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j)). \quad (19)$$

**Proof.** The one-flip additive gain can be derived in a manner similar to the proof of one-flip multiplicative gain  $\Delta U_M(\mathbf{x}, \bar{x})$  in Theorem 20. ■

### 3.2.3 Additive Gain and Multiplicative Gain

We have several arguments for why additive gain is “reasonable”. One of our results, which is stated formally in Theorem 27 below, relates additive gain and multiplicative gain. In order to improve readability, we introduce in Definition 24 and Definition 25 alternative ways of expressing additive and multiplicative gains when flipping a node  $X_i$  with children  $\mathbf{C}$ .

**Definition 24 (Simple additive gain notation)** *Let  $\alpha = |\mathbf{C}| + 1$  and put in Theorem 23:  $p_1 = \Pr(\bar{x}_k | \pi_{X_k})$ ,  $\sum_{i=2}^{\alpha} p_i = \sum_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \bar{\pi}(C_j))$ ,  $q_1 = \Pr(x_k | \pi_{X_k})$ , and  $\sum_{i=2}^{\alpha} q_i = \sum_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j))$ . This gives the following simplified notation for additive gain:*

$$\Delta U_A(\mathbf{x}, \bar{x}_k) = \sum_{i=1}^{\alpha} p_i - \sum_{i=1}^{\alpha} q_i.$$

**Definition 25 (Simple multiplicative gain notation)** *Let  $|\mathbf{C}| = \alpha - 1$  and put in Theorem 20:  $p_1 = \Pr(\bar{x}_k | \pi_{X_k})$ ,  $\prod_{i=2}^{\alpha} p_i = \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \bar{\pi}(C_j))$ ,  $q_1 = \Pr(x_k | \pi_{X_k})$ , and  $\prod_{i=2}^{\alpha} q_i = \prod_{C_j \in \mathbf{C}} \Pr(C_j = c_j | \pi(C_j))$ . This gives the following simplified notation for multiplicative gain:*

$$\Delta U_M(\mathbf{x}, \bar{x}_k) = \prod_{i=1}^{\alpha} \frac{p_i}{q_i}, \quad (20)$$

again under the assumption  $\prod_{i=1}^{\alpha} q_i > 0$ .

The following fact follows easily from the definitions of  $\Delta U_A$  and  $\Delta U_M$ .

**Proposition 26** *Let  $\mathbf{x}$  and  $\mathbf{x}'$  be explanations and suppose that  $\Delta U_M(\mathbf{x}) > 0$ .  $\Delta U_A(\mathbf{x}, \mathbf{x}') = 0$  if and only if  $\Delta U_M(\mathbf{x}, \mathbf{x}') = 1$ .*

We now turn to the more general case where  $\Delta U_A(\mathbf{x}, \mathbf{x}') \neq 0$  and  $\Delta U_M(\mathbf{x}, \mathbf{x}') \neq 1$ .

**Theorem 27** *Consider the definitions of additive gain  $\Delta U_A(\mathbf{x}, \bar{x}_k) = \sum_{i=1}^{\alpha} p_i - \sum_{i=1}^{\alpha} q_i$  and multiplicative gain  $\Delta U_M(\mathbf{x}, \bar{x}_k) = \prod_{i=1}^{\alpha} \frac{p_i}{q_i}$  from Definition 24 and Definition 25 respectively. Suppose that there exists a permutation  $\sigma$  of  $\{q_1, \dots, q_{\alpha}\}$ , denoted  $\{q_{\sigma(1)}, \dots, q_{\sigma(\alpha)}\}$ , and a permutation  $\rho$  of  $\{p_1, \dots, p_{\alpha}\}$ , denoted  $\{p_{\rho(1)}, \dots, p_{\rho(\alpha)}\}$ , such that*

$$p_{\rho(1)} > q_{\sigma(1)}, \dots, p_{\rho(\kappa)} > q_{\sigma(\kappa)}, p_{\rho(\kappa+1)} = q_{\sigma(\kappa+1)}, \dots, p_{\rho(\alpha)} = q_{\sigma(\alpha)} \quad (21)$$

for some  $1 \leq \kappa \leq \alpha$ . If (21) is true and  $\Delta U_A(\mathbf{x}, \bar{x}_k) > 0$  then  $\Delta U_M(\mathbf{x}, \bar{x}_k) > 1$ .

**Proof.** Since, by assumption,  $\Delta U_A(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\alpha} p_i - \sum_{i=1}^{\alpha} q_i > 0$ , we must have  $\kappa \geq 1$  in (21). For any  $1 \leq j \leq \kappa$ , it is the case that  $p_{\rho(j)} > q_{\sigma(j)}$  or in other words  $\frac{p_{\rho(j)}}{q_{\sigma(j)}} > 1$ , giving also

$$\prod_{j=1}^{\kappa} \frac{p_{\rho(j)}}{q_{\sigma(j)}} > 1. \quad (22)$$

For the remaining elements in (21) we have  $p_{\rho(\kappa+1)} = q_{\sigma(\kappa+1)}, \dots, p_{\rho(\alpha)} = q_{\sigma(\alpha)}$  and thus

$$\prod_{j=\kappa+1}^{\alpha} \frac{p_{\rho(j)}}{q_{\sigma(j)}} = 1. \quad (23)$$

Going back to the definition (20) as well as (22) and (23) gives

$$\Delta U_M(\mathbf{x}, \bar{x}_k) = \prod_{i=1}^{\alpha} \frac{p_i}{q_i} = \prod_{j=1}^{\kappa} \frac{p_{\rho(j)}}{q_{\sigma(j)}} \prod_{j=\kappa+1}^{\alpha} \frac{p_{\rho(j)}}{q_{\sigma(j)}} > 1$$

as desired. ■

Theorem 27 justifies the use of  $\Delta U_A$  in MPE local search algorithms as follows: When flipping from explanation  $\mathbf{x}$  to explanation  $\mathbf{x}'$  — and if  $\Delta U_A(\mathbf{x}, \mathbf{x}') > 0$  as well as condition (21) holds during local search — one is also making progress according to  $\Delta U_M$ .

Despite the result above, we emphasize that a positive additive gain  $\Delta U_A$  does not always imply an increase in an explanation’s probability or multiplicative gain  $\Delta U_M$ . This is stated formally in the following theorem.

**Proposition 28** *Let  $\mathbf{x}$  be an explanation, and  $\mathbf{x}'$  be one-flipped from  $\mathbf{x}$ . It is not always the case that if  $\Delta U_A(\mathbf{x}, \mathbf{x}') > 0$  then  $\Delta U_M(\mathbf{x}, \mathbf{x}') > 1$ .*

**Proof.** A counter-example in a BN with two nodes  $X_1$  and  $X_2$  is provided. The nodes both have state space  $\{0, 1\}$ . Node  $X_1$  is  $X_2$ ’s parent. Let the CPTs be  $\{\Pr(X_1 = 0) = 0.2, \Pr(X_1 = 1) = 0.8, \Pr(X_2 = 0 \mid X_1 = 0) = 0.4, \Pr(X_2 = 1 \mid X_1 = 0) = 0.6, \Pr(X_2 = 0 \mid X_1 = 1) = 0.05, \Pr(X_2 = 1 \mid X_1 = 1) = 0.95\}$ . Now consider  $\mathbf{x} = \{X_1 = 0, X_2 = 0\}$  and  $\mathbf{x}' = \{X_1 = 1, X_2 = 0\}$ . In this case we have  $\Delta U_A(\mathbf{x}, \mathbf{x}') = U_A(\mathbf{x}') - U_A(\mathbf{x}) = (0.8 + 0.05) - (0.2 + 0.4) = 0.25 > 0$  while  $\Delta U_M(\mathbf{x}, \mathbf{x}') = U_M(\mathbf{x}')/U_M(\mathbf{x}) = (0.8 \times 0.05)/(0.2 \times 0.4) = 0.5 \not> 1$ . ■

### 3.2.4 Additive Gain and GSAT Gain

The reasoning behind the GSAT gain [50,51] and the additive gain shown in Equation 19 is similar. Reflecting this, we introduce the following two definitions based on previous research [50,51].

**Definition 29 (GSAT gain)** *Let  $\tau$  and  $\tau'$  be truth assignments. Then GSAT gain  $\Delta U_S(\tau, \tau')$  is defined as*

$$\Delta U_S(\tau, \tau') = \Delta U_S(\tau') - \Delta U_S(\tau).$$

The notation  $\Delta U_S(\tau, \tau')$  is used for GSAT gain over truth assignments  $\tau$  and  $\tau'$  for SAT formulas, similar to additive gain  $\Delta U_A(\mathbf{x}, \mathbf{x}')$  for explanations  $\mathbf{x}$  and  $\mathbf{x}'$  in BNs as introduced in Definition 21.

**Definition 30 (One-flip GSAT gain)** *Let  $\tau$  be a truth assignment to the variables  $\mathbf{y}$  in a SAT instance  $\phi = (\mathbf{y}, \mathbf{q}, \mathbf{c})$ . Suppose that  $\tau$  is changed to  $\bar{\tau}$  by flipping one variable  $y \in \mathbf{y}$  from  $\tau(y)$  to  $\bar{\tau}(y)$ . Let  $\text{MAKECOUNT}(\tau, \bar{y})$  be the number of clauses satisfied (or made 1) by the flip of  $y$ . Let  $\text{BREAKCOUNT}(\tau, \bar{y})$  be the number of clauses unsatisfied (or made 0) by the flip of  $y$ . One-flip GSAT gain is defined as*

$$\Delta U_S(\tau, \bar{y}) = \text{MAKECOUNT}(\tau, \bar{y}) - \text{BREAKCOUNT}(\tau, \bar{y}).$$

In a manner similar to GSAT gain’s success in SAT, our additive gain turns out to be powerful on deterministic BN nodes, which occur in application BNs, as well as in SAT-like BNs which encode satisfiability instances [4,46]. In fact, GSAT gain is a special case of additive gain as expressed in the following result.

**Theorem 31** *Let BN  $\beta = (\mathbf{Y}, \mathbf{E}, \mathbf{P})$  and suppose that SAT instance  $\phi = (\mathbf{y}, \mathbf{q}, \mathbf{c})$  is SAT-equivalent to  $\beta$ . Further, let two explanations  $\mathbf{x}_1$  and  $\mathbf{x}_2$  in  $\beta$  be SAT-equivalent to two truth assignments  $\tau_1$  and  $\tau_2$ , respectively, for  $\phi$ . Then  $\Delta U_A(\mathbf{x}_1, \mathbf{x}_2) = \Delta U_S(\tau_1, \tau_2)$ .*

**Proof.** Applying Theorem 15 to both  $\mathbf{x}_1$  and  $\mathbf{x}_2$  gives

$$U_A(\mathbf{x}_1) = U_S(\tau_1) + \frac{V}{2} \quad (24)$$

$$U_A(\mathbf{x}_2) = U_S(\tau_2) + \frac{V}{2}, \quad (25)$$

and when (24) is subtracted from (25) we obtain

$$U_A(\mathbf{x}_2) - U_A(\mathbf{x}_1) = U_S(\boldsymbol{\tau}_2) - U_S(\boldsymbol{\tau}_1). \quad (26)$$

Simply substituting the definitions of  $\Delta U_A(\mathbf{x}_1, \mathbf{x}_2)$  (Definition 21) and  $\Delta U_S(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2)$  (Definition 29) into (26) gives the desired result. ■

The following result follows easily from Theorem 31.

**Corollary 32** *Let  $\Delta U_A(\mathbf{x}_1, \bar{x}_i)$  be the additive gain obtained by flipping a root node  $X_i \in \mathbf{X}$  in a SAT-like BN  $\beta = (\mathbf{Y}, \mathbf{E}, \mathbf{P})$ . Assume that the SAT instance  $\phi = (\mathbf{y}, \mathbf{q}, \mathbf{c})$  is SAT-equivalent to  $\beta$  and that  $\boldsymbol{\tau}_1$  is equivalent to  $\mathbf{x}_1$ . Further, let  $\Delta U_S(\boldsymbol{\tau}_1, \bar{y}_i)$  be the GSAT gain obtained, under  $\boldsymbol{\tau}_1$ , by flipping the variable  $y_i \in \mathbf{y}$  corresponding to the root node  $X_i \in \mathbf{X}$ . It is then the case that  $\Delta U_A(\mathbf{x}_1, \bar{x}_i) = \Delta U_S(\boldsymbol{\tau}_1, \bar{y}_i)$ .*

**Proof.** Put  $\mathbf{x}_2 \leftarrow \mathbf{x}_1[X_i \leftarrow \bar{x}_i]$  and let  $\boldsymbol{\tau}_2 \leftarrow \bar{\boldsymbol{\tau}}(\phi, y_i, \boldsymbol{\tau}_1)$  (copying  $\boldsymbol{\tau}_1$  into  $\boldsymbol{\tau}_2$  but exchanging  $\boldsymbol{\tau}(y_i)$  with  $\bar{\boldsymbol{\tau}}(y_i)$ ). Observe that node  $X_i$  corresponds to the variable  $y_i$ . Clearly, the constructed truth assignment  $\boldsymbol{\tau}_2$  is equivalent to the constructed explanation  $\mathbf{x}_2$ , thus Theorem 31 applies and the result follows. ■

### 3.3 Criteria of Choice

In addition to measures of utility and gain, SLS algorithms including SGS need *criteria of choice*. These criteria determine how an SLS algorithm decides, based on its gain computations, which BN node(s) to flip next. Gains in a neighborhood around the current explanation  $\mathbf{x}$  are computed and placed in a candidate array (or set), defined as follows.

**Definition 33 (Candidate array)** *Let  $\mathbf{x}$  be an explanation and let  $g_{i,j} = \Delta U(\mathbf{x}, x_{i,j})$  be the one-flip gain obtained by flipping explanation  $\mathbf{x}$ 's  $i$ -th node  $X_i$  from its  $k$ -th state  $k \in \{1, \dots, |\Omega_{X_i}|\}$  to its  $j$ -th state, where  $j \in \{1, \dots, |\Omega_{X_i}|\} - \{k\}$ . The set of 3-tuples*

$$\mathbf{A} = \{(X_i, x_{i,j}, g_{i,j}) \mid X_i \text{ is the } i\text{-th BN node, } x_{i,j} \text{ is } X_i\text{'s } j\text{-th state, and } g_{i,j} \text{ is the one-flip gain}\}$$

is denoted the candidate array.

At present,  $g_{i,j} = \Delta U(\mathbf{x}, x_{i,j})$  is either multiplicative gain  $\Delta U_M(\mathbf{x}, x_{i,j})$  (11) or additive gain  $\Delta U_A(\mathbf{x}, x_{i,j})$  (18). The candidate array describes explanation  $\mathbf{x}$ 's local neighborhood and is used to determine — using a criterion of choice — which node to actually flip in  $\mathbf{x}$ . We have investigated two types of criteria of choice, the greedy criterion  $C_G$  and the noisy criterion  $C_N$ . Both criteria rely on the candidate array  $\mathbf{A}$  and a tuple of gains in  $\mathbf{A}$ , formally  $\mathbf{G} = (g_1, \dots, g_m)$ .<sup>2</sup>

**Definition 34 (Greedy criterion  $C_G$ )** *The greedy (or maximizing) criterion  $C_G(\mathbf{A})$  is to pick the  $i$ -th tuple with gain  $g_i$ , for  $1 \leq i \leq k \leq m$ , such that  $g_i \geq g_j$  for all  $1 \leq j \leq m$ . Among these  $k$  maximal tuples,  $C_G(\mathbf{A})$  picks the  $i$ -th with probability  $p_i = 1/k$ .*

**Definition 35 (Noisy criterion  $C_N$ )** *The noisy (or stochastic) criterion  $C_N(\mathbf{A})$  is to pick the  $i$ -th tuple with gain  $g_i$ , for  $1 \leq i \leq m$ , according to the probability distribution*

$$p_i = g_i \Big/ \sum_{j=1}^m g_j .$$

In SGS, the maximizing criterion in Definition 34 chooses greedily from the candidate array  $\mathbf{A}$  and yields classical steepest-ascent hill-climbing. The stochastic criterion yields probabilistic hill-climbing, since it probabilistically decides which node and state to flip to, using gain information in the candidate array  $\mathbf{A}$ . The stochastic criterion Definition 35 is closely related to the approach used in the stochastic simulation algorithm [45] as well as to noise strategies such as “random walk” used in stochastic local search for SAT [50]. We also note that the  $p_i$ 's in Definition 34 and Definition 35 in fact define rows in the transition matrix of an exact Markov chain model; see Section 5.2 and in particular Definition 51 for details.

A measure of gain, then, defines how to measure progress for candidate local search steps. A criterion of choice, on the other hand, determines how to choose between the different candidate local search steps, based on the candidate gains, using a maximizing criterion or a stochastic criterion.

<sup>2</sup>For the binary case and when there is no evidence,  $m = n$  since an explanation in a BN with  $n$  nodes has  $n$  possible one-flip gains.

## 4 Stochastic Local Search Algorithms

Stochastic greedy search (SGS) is a local search approach augmented with stochastic initialization and noise steps. This section presents two variants of SGS, in order of increasing complexity and power. Section 4.1 presents simple SGS (or SIMPLESGS), while Section 4.2 presents operator-based SGS (or OPERATORSGS). Among these, SIMPLESGS is most similar to other SLS algorithms, and acts as a stepping stone to reach OPERATORSGS, our portfolio-based main contribution. Section 4.3 discusses related work and compares our two SGS variants.

### 4.1 Simple Stochastic Greedy Search

SIMPLESGS, which is presented in Figure 2, starts from an explanation  $\mathbf{x}$  as constructed by an initialization algorithm INITIALIZE and performs one-flip local changes to  $\mathbf{x}$  in order to improve the utility  $U(\mathbf{x})$ . The input parameter  $U$  controls which measure of utility and gain is used, and is currently either  $U_M$  or  $U_A$ . The algorithm is related to the GSAT and WALKSAT algorithms [50, 51] as well as other SLS algorithms as discussed in Section 4.3. Specifically, the structure of SIMPLESGS is similar to that of GSAT and WALKSAT: There is an outer loop for tries (or restarts), and an inner loop for flips (or stochastic local search steps). The parameter MAX-FLIPS limits how many flips are done before a restart, while the number of tries is upper bounded by MAX-TRIES. The parameter  $p_N$  controls the noise level. Applying noise amounts to taking a random step with probability  $p_N$ , and taking a greedy step using the chosen measure of gain with probability  $1 - p_N$ . This is similar to WALKSAT with random noise [50]. SIMPLESGS terminates when an explanation  $\hat{\mathbf{x}}^*$  is found such that  $U(\hat{\mathbf{x}}^*) \geq U_{\min}$ . Other termination criteria can easily be incorporated into SIMPLESGS. In particular, we have implemented the option of terminating after executing SIMPLESGS for a certain time period, even though this is not shown in Figure 2.

The SIMPLESGS algorithm returns a two-valued tuple  $(\mathbf{b}, \hat{\mathbf{x}}^*)$ . The first value  $\mathbf{b}$  is a Boolean value signifying whether search was successful or not; the second value  $\hat{\mathbf{x}}^*$  is the explanation with the highest utility score. The input value  $p_N = 0$  means that no noise should be applied; in this case SIMPLESGS returns when reaching a local maximum. With  $U_{\min} > 0$ , SIMPLESGS is successful if utility  $U_{\min}$  or better is attained.

**Theorem 36** *Let  $\text{MAX-TRIES} = \infty$ ,  $U = U_M$ , and  $U_{\min} = \text{Pr}(\mathbf{x}^*)$  in SIMPLESGS. Then SIMPLESGS is a Las Vegas algorithm and returns  $(\mathbf{true}, \hat{\mathbf{x}}^*)$  such that  $\hat{\mathbf{x}}^* \in \mathbf{X}^*$ .*

**Proof.** Since  $\text{MAX-TRIES} = \infty$ , there are two statements from which SIMPLESGS can return, and they both read “**if**  $(U(\hat{\mathbf{x}}^*) \geq U_{\min})$  **return**  $(\mathbf{true}, \hat{\mathbf{x}}^*)$ .” By substituting in the assumptions, we obtain

$$U(\hat{\mathbf{x}}^*) = U_M(\hat{\mathbf{x}}^*) = \text{Pr}(\hat{\mathbf{x}}^*) \geq U_{\min} = \text{Pr}(\mathbf{x}^*),$$

which clearly only holds when  $\text{Pr}(\hat{\mathbf{x}}^*) = \text{Pr}(\mathbf{x}^*)$ , giving the desired result. ■

SIMPLESGS uses gain computation algorithms COMPUTEMULTIPLICATIVEGAIN and COMPUTEADDITIVEGAIN, see Figure 3 and Figure 4 respectively, to evaluate candidate tuples and construct the candidate array  $\mathbf{A}$ . Which gain computation is used depends on the value of the input parameter  $U$ . Both gain computation algorithms iterate over all non-evidence nodes and over all states of each node  $X_i$ , excluding a node’s current state  $x_{i,k}$ . Within the inner-most loops there are calls to NODECHILDRENPROBABILITY, which are used to compute the gain  $g$  associated with flipping node  $X_i$ ’s state in explanation  $\mathbf{x}$ .

We now discuss how the COMPUTEMULTIPLICATIVEGAIN and COMPUTEADDITIVEGAIN algorithms implement multiplicative gain  $\Delta U_M$  and additive gain  $\Delta U_A$  respectively. In the implementation of the multiplicative measure as shown in COMPUTEMULTIPLICATIVEGAIN in Figure 3, the assignment to Old is the denominator in multiplicative gain, while the assignment to New is multiplicative gain’s numerator. Recall from Definition 18 and Theorem 20 that multiplicative gain is not defined if the denominator is zero. Consequently, there is a test for zero in COMPUTEMULTIPLICATIVEGAIN, and the following action is taken: If Old (denominator in Equation 12) and New (numerator in Equation 12) are both zero, we define multiplicative gain to be one. If Old is zero but New is non-zero, we greedily pick the new state  $s$ , which is  $\bar{x}$  in the binary case, and return. So in this case where Old (denominator) is zero but New (numerator) is non-zero, the algorithm is more greedy than in the other cases, since the first state in the first node where a flip leads from a zero to a non-zero probability is picked as a candidate for flipping.

Both COMPUTEMULTIPLICATIVEGAIN and COMPUTEADDITIVEGAIN return a filled-in candidate array  $\mathbf{A}$  to SIMPLESGS. After computation of  $\mathbf{A}$ , SIMPLESGS calls the CHOOSESTATE function. CHOOSESTATE

SIMPLESGS(MAX-TRIES, MAX-FLIPS,  $p_N$ ,  $U$ ,  $C$ ,  $U_{\min}$ )

**Input:** MAX-TRIES maximum number of restarts  
MAX-FLIPS maximum number of flips  
 $p_N$  noise probability  
 $U$  measure of utility (and gain):  $U_M$  (and  $\Delta U_M$ ) or  $U_A$  (and  $\Delta U_A$ )  
 $C$  criterion of choice:  $C_G$  or  $C_N$   
 $U_{\min}$  utility lower bound for termination

**Output:** ( $\mathbf{b}$ ,  $\hat{\mathbf{x}}^*$ )  $\mathbf{b} \in \{\mathbf{true}, \mathbf{false}\}$ ;  $\hat{\mathbf{x}}^*$  is a best effort estimate of MPE  $\mathbf{x}^*$

**begin**

$\hat{\mathbf{x}}^* \leftarrow \text{INITIALIZE}()$  {initialization uniformly at random (using NU operator)}

**for**  $i \leftarrow 1$  to MAX-TRIES {loop of tries}

$\mathbf{x} \leftarrow \text{INITIALIZE}()$  {initialization uniformly at random (using NU operator)}

**if** ( $U(\mathbf{x}) > U(\hat{\mathbf{x}}^*)$ )  $\hat{\mathbf{x}}^* \leftarrow \mathbf{x}$

**if** ( $U(\hat{\mathbf{x}}^*) \geq U_{\min}$ ) **return** (**true**,  $\hat{\mathbf{x}}^*$ )

**for**  $j \leftarrow 1$  to MAX-FLIPS {loop of flips}

Noise  $\leftarrow$  (RANDOMIZE()  $< p_N$ ) {decide between hill-climbing and noise, set Noise flag}

**if** (**not** Noise) {Noise = **false**, perform greedy hill-climbing step}

**if** ( $U = U_M$ )  $\mathbf{A} \leftarrow \text{COMPUTEMULTIPLICATIVEGAIN}(\mathbf{x})$  {put  $U_M$  gains in array  $\mathbf{A}$ }

**else**  $\mathbf{A} \leftarrow \text{COMPUTEADDITIVEGAIN}(\mathbf{x})$  {put  $U_A$  gains in array  $\mathbf{A}$ }

**if** ( $\mathbf{A} = \emptyset$ )

**if** ( $p_N = 0$ ) **break**

**else** Noise  $\leftarrow$  **true** {no candidates, apply noise}

**else**

$(X_i, x_{i,j}) \leftarrow \text{CHOOSESTATE}(\mathbf{A}, C)$  {pick node  $X_i$  and state  $x_{i,j}$  from array  $\mathbf{A}$ }

**endif**

**endif**

**if** (Noise) {Noise = **true**, perform noise step}

$X_i \leftarrow$  pick node  $X_i$  from BN at random

$x_{i,j} \leftarrow$  pick state  $x_{i,j} \in \Omega_{X_i} - \{x_{i,k}\}$  at random {avoid the current state  $x_{i,k}$ }

**endif**

$\mathbf{x} \leftarrow \mathbf{x}[X_i \leftarrow x_{i,j}]$  {set  $X_i = x_{i,j}$  in explanation  $\mathbf{x}$ }

**if** ( $U(\mathbf{x}) > U(\hat{\mathbf{x}}^*)$ )  $\hat{\mathbf{x}}^* \leftarrow \mathbf{x}$

**if** ( $U(\hat{\mathbf{x}}^*) \geq U_{\min}$ ) **return** (**true**,  $\hat{\mathbf{x}}^*$ )

**endfor** {loop of flips}

**endfor** {loop of tries}

**if** ( $U_{\min} = 0$ ) **return** (**true**,  $\hat{\mathbf{x}}^*$ ) **else return** (**false**,  $\hat{\mathbf{x}}^*$ )

**end**

Figure 2: The simple stochastic greedy search algorithm SIMPLESGS. The algorithm repeatedly interleaves noise and greedy hill-climbing steps; hill-climbing is done by calling either COMPUTEMULTIPLICATIVEGAIN or COMPUTEADDITIVEGAIN. These functions return a candidate array  $\mathbf{A}$  which contains candidates for the next hill-climbing step. A candidate is a tuple  $(X_i, x_{i,j}, g_{i,j})$ , where  $X_i$  is the BN node potentially being flipped,  $x_{i,j}$  is a state of that node, and  $g_{i,j}$  is the gain obtained by flipping  $X_i$  to  $x_{i,j}$ . The CHOOSESTATE function applies the criterion of choice  $C$  to pick a node  $X_i$  and state  $x_{i,j}$ , which is then subsequently flipped to.

```

COMPUTEMULTIPLICATIVEGAIN( $\mathbf{x}$ )
  Input:  $\mathbf{x}$  explanation
  Output:  $\mathbf{A}$  candidate array of gains computed using multiplicative gain
begin
  INIT( $\mathbf{A}, 1$ ) {for multiplicative gain initialize candidate array  $\mathbf{A}$  with 1s}
  for each non-evidence node  $X_i \in \mathbf{X}$ 
     $o \leftarrow x_i$  {remember original state  $x_i$ }
    Old  $\leftarrow$  NODECHILDRENPROBABILITY( $X_i, \mathbf{x}, U_M$ )
    for each state  $s \in \Omega_{X_i} - \{x_i\}$  {try all states except state  $x_i$ }
       $\mathbf{x} \leftarrow \mathbf{x}[X_i \leftarrow s]$  {try state  $s$  in  $i$ -th node  $X_i$ }
      New  $\leftarrow$  NODECHILDRENPROBABILITY( $X_i, \mathbf{x}, U_M$ )
      if (Old = 0)
        if (New = 0)  $g \leftarrow 1$ 
        else
          FORCE( $\mathbf{A}, X_i, s$ ) {"force" state  $s$  into candidate array  $\mathbf{A}$ }
          return  $\mathbf{A}$ 
        endif
      endif
       $g \leftarrow$  New / Old
      ADD( $\mathbf{A}, X_i, s, g$ ) {add tuple ( $X_i, s, g$ ) to candidate array  $\mathbf{A}$ }
    endfor
     $\mathbf{x} \leftarrow \mathbf{x}[X_i \leftarrow o]$  {reset state of  $X_i$  to original value  $o$ }
  endfor
  return  $\mathbf{A}$ 
end

```

Figure 3: This algorithm, which implements the multiplicative measure of gain  $\Delta U_M$ , is called by SGS and is used to fill in the candidate array  $\mathbf{A}$  with gains for neighbors of the current explanation  $\mathbf{x}$ .

```

COMPUTEADDITIVEGAIN( $\mathbf{x}$ )
  Input:  $\mathbf{x}$  explanation
  Output:  $\mathbf{A}$  candidate array of gains computed using additive gain
begin
  INIT( $\mathbf{A}, 0$ ) {for additive gain initialize candidate array with 0s}
  for each non-evidence node  $X_i \in \mathbf{X}$ 
     $o \leftarrow x_i$  {remember original state  $x_i$  of node  $X_i$ }
    Old  $\leftarrow$  NODECHILDRENPROBABILITY( $X_i, \mathbf{x}, U_A$ )
    for each state  $s \in \Omega_{X_i} - \{x_i\}$  {try all states except state  $x_i$ }
       $\mathbf{x} \leftarrow \mathbf{x}[X_i \leftarrow s]$  {try state  $s$  in  $i$ -th node  $X_i$ }
      New  $\leftarrow$  NODECHILDRENPROBABILITY( $X_i, \mathbf{x}, U_A$ )
       $g \leftarrow$  New - Old
      ADD( $\mathbf{A}, X_i, s, g$ ) {add tuple ( $X_i, s, g$ ) to candidate array  $\mathbf{A}$ }
    endfor
     $\mathbf{x} \leftarrow \mathbf{x}[X_i \leftarrow o]$  {reset state of node  $X_i$  to original value  $o$ }
  endfor
  return  $\mathbf{A}$ 
end

```

Figure 4: This algorithm, which implements the additive measure of gain  $\Delta U_A$ , is used by SGS to fill in the candidate array  $\mathbf{A}$  with gains for neighbors of the current explanation  $\mathbf{x}$ .

applies the criterion of choice and picks which state  $x_{i,j}$  to flip to. Then, after a possible noise step, the utility measure  $U(\mathbf{x})$  is applied. Using either multiplicative utility  $U_M$  or additive utility  $U_A$ , the better explanation of  $\mathbf{x}$  and  $\hat{\mathbf{x}}^*$  is kept. The algorithm then terminates or iterates.

SIMPLESGS has been very successful in certain BNs, as is demonstrated in Section 6.1. It turned out that the algorithm also has some limitations, including the following. First, SIMPLESGS does not accommodate an arbitrary number of different initialization and search operators in the same invocation of the algorithm. Second, the measures of utility and gain are coupled through the input parameter  $U$ , and it is not possible to apply different measures of gain in the same invocation of the algorithm. Addressing these limitations is crucial to obtaining strong performance on application BNs. We now discuss how they are addressed in the operator-based variant of SGS.

## 4.2 Operator-Based Stochastic Greedy Search

The operator-based variant of SGS, denoted OPERATORSGS, is presented in Figure 5. OPERATORSGS retains the structure of SIMPLESGS, with two nested loops for tries and flips. What is novel in OPERATORSGS is how the algorithm chooses from two sets of operators or heuristics when an initialization or search step is performed. This operator-based approach gives better flexibility and extensibility, resulting in substantially better performance on a range of different BNs compared to SIMPLESGS. We will return a more detailed comparison after describing OPERATORSGS as well as formally introducing its initialization and search operators.

**Definition 37 (Initialization operator, initialization operators)** *An initialization operator is a heuristic that computes, given a BN  $\beta$  and evidence  $\mathbf{e}$ , an explanation  $\mathbf{x}$ . A set of initialization operators is denoted  $\Phi_I$ .*

The initialization operators that have so far been explored using OPERATORSGS are: uniform initialization (UN), forward simulation (FS), backward dynamic programming (BDP), and forward dynamic programming (FDP); we thus have  $\Phi_I = \{\text{UN,FS,BDP,FDP}\}$  [38]. Other algorithms, for example the mini-bucket algorithm [26], can easily be incorporated as initialization operators in OPERATORSGS.

**Definition 38 (Search operator, search operators)** *A (local) search operator is a heuristic that computes, for a BN  $\beta$  with evidence  $\mathbf{e}$  and an explanation  $\mathbf{x}$ , a new explanation  $\mathbf{x}'$ . A one-flip (local) search operator is a search operator where  $\mathbf{x}' = \mathbf{x}[X_i \leftarrow \bar{x}_i]$ . A set of search operators is denoted  $\Phi_S$ .*

Our search operators are closely related to flip selection strategies such as GSAT [51], WSAT-G [33], WSAT-B [33], WSAT [33,50], Novelty [33], Novelty+, and DLM [48] in the SAT setting, and similar strategies in the BN setting [26,30,37,41,42]. Flip selection strategies, though, typically hard-code a greedy part and a noisy part similar to what is done in SIMPLESGS. In OPERATORSGS, on the other hand, these two parts are typically represented by means of two distinct operators in a search portfolio. We will return to the issue of flip selection strategies and search operators after introducing the following concept.

**Definition 39 (Compound operator)** *A compound operator is a 2-tuple  $(\Delta U(\mathbf{x}, x_{i,j}), C(\mathbf{A}))$ , where  $\Delta U(\mathbf{x}, x_{i,j})$  is a gain function and  $C(\mathbf{A})$  is a criterion of choice operating on a candidate array  $\mathbf{A}$ .*

Example gain functions referred to in the above definition are additive gain  $\Delta U_A(\mathbf{x}, x_{i,j})$  and multiplicative gain  $\Delta U_M(\mathbf{x}, x_{i,j})$ . Compound operators are used by COMPOUNDSEARCH, see Figure 6.

The operators  $\{\text{MG,AG,MN,AN}\}$  are compound operators that we have used in OPERATORSGS so far; these operators are defined as follows.

**Definition 40 (Compound operators MG, AG, MN, and AN)** *Using  $C(\mathbf{A}) = C_G(\mathbf{A})$  (the greedy criterion) we define these greedy search operators.<sup>3</sup>*

$$\begin{aligned} \text{MG} &:= (\Delta U_M(\mathbf{x}, x_{i,j}), C_G(\mathbf{A})) \text{ \{multiplicative gain with greedy criterion\}} \\ \text{AG} &:= (\Delta U_A(\mathbf{x}, x_{i,j}), C_G(\mathbf{A})) \text{ \{additive gain with greedy criterion\}}. \end{aligned}$$

Using  $C(\mathbf{A}) = C_N(\mathbf{A})$  (the noisy criterion) we define these noisy search operators:

$$\begin{aligned} \text{MN} &:= (\Delta U_M(\mathbf{x}, x_{i,j}), C_N(\mathbf{A})) \text{ \{multiplicative gain with noisy criterion\}} \\ \text{AN} &:= (\Delta U_A(\mathbf{x}, x_{i,j}), C_N(\mathbf{A})) \text{ \{additive gain with noisy criterion\}}. \end{aligned}$$

<sup>3</sup>In previous work [37,38] MG was called BM, AG was called GM, MN was called BS, AN was called GS.

The AG operator is formally justified by Theorem 27 and Theorem 32. Informally, there is the following relationship between maximizing GSAT gain and maximizing additive gain: Using GSAT gain, we flip the variable which gives the highest increase in the number of satisfied clauses [51]. Using the additive operator AG, we flip the BN node which gives the greatest additive increase in values of the affected CPTs.

In addition to the compound operators introduced in Definition 40, there are operators that are not formed by combining a gain function and a criterion of choice. For example, we have investigated the uniform noise (NU) and stochastic simulation (SS) operators. Uniform noise means that a non-evidence node, picked uniformly at random, is flipped with probability  $p_N$ . Stochastic simulation is Gibbs sampling in BNs, where nodes are sequentially sampled [45]. Altogether, this gives search operators  $\Phi_S = \{\text{NU}, \text{SS}, \text{MG}, \text{AG}, \text{MN}, \text{AN}\}$ .

An operator portfolio  $\Lambda$ , formally introduced in Definition 41 below, is generally speaking a set of operators along with an approach to operator selection. We focus here on selection according to a probability distribution defined by associating a probability with each operator as follows.<sup>4</sup>

**Definition 41 (Stochastic portfolio)** *Let  $q \geq 1$  and let  $\Phi = \{\phi_1, \dots, \phi_q\}$  be a set of operators. A stochastic portfolio over  $\Phi$  is a set of  $q$  operator tuples  $\Lambda = \{\nu_1, \dots, \nu_q\} = \{(\phi_1, p_1), \dots, (\phi_q, p_q)\}$  where  $0 \leq p_i \leq 1$ ,  $\sum_{i=1}^q p_i = 1$ , and  $(\phi_i, p_i)$  means that the  $i$ -th operator  $\phi_i$ , where  $1 \leq i \leq q$ , is selected (and then executed) with probability  $p_i$  whenever some operator is selected from  $\Lambda$ .*

Initialization algorithms have turned out to be important components in SGS as well as in other similar algorithms [26, 30, 38, 42]. In OPERATORSGS, the input parameters therefore include a stochastic portfolio of initialization operators  $\Lambda_I$ , defined as follows.

**Definition 42 (Initialization portfolio)** *Let  $\Phi_I$  be a set of initialization operators. The OPERATORSGS initialization portfolio  $\Lambda_I$  is a stochastic portfolio over  $\Phi_I$  according to Definition 41.*

The initialization algorithms construct explanations which make up starting points that the local search steps incrementally improve, leading to increasingly better MPE estimates. We have investigated the following initialization operators for stochastic generation of initial explanations: uniform initialization (UN), forward simulation (FS), forward dynamic programming (FDP), and backward dynamic programming (BDP) [38]. These stochastic initialization operators all have a time complexity of  $O(n)$  in the number of nodes. A wide range of initialization heuristics can easily be incorporated as initialization operators in OPERATORSGS.

The stochastic portfolio  $\Lambda_S$  for the local search heuristics is formally introduced as follows.

**Definition 43 (Search portfolio)** *Let  $\Phi_S$  be a set of search operators. The OPERATORSGS search portfolio  $\Lambda_S$  is a stochastic portfolio over  $\Phi_S$  according to Definition 41.*

The input parameters of OPERATORSGS include a portfolio of search operators  $\Lambda_S$  and a portfolio of initialization operators  $\Lambda_I$ . OPERATORSGS chooses only among the operators included in the portfolios  $\Lambda_I$  and  $\Lambda_S$ . This operator-based approach allows us to use a mixture of different operators, thus allowing us to emulate a wide range of SLS algorithms investigated previously. Here are a few examples from the literature.

**Example 44** *For computing SAT, Selman et al. investigated, in the mixed random walk strategy, greedy search (GSAT) combined with a random walk (RW) [50]:  $\Lambda_S = \{(\text{GSAT}, p_{\text{GSAT}}), (\text{RW}, p_{\text{RW}})\}$ .*

**Example 45** *For computing MPE, Kask and Dechter investigated greedy search (GR) with stochastic simulation (SS) using mini-bucket (MB) initialization [26]:  $\Lambda_S = \{(\text{GR}, p_{\text{GR}}), (\text{SS}, p_{\text{SS}})\}$  and  $\Lambda_I = \{(\text{MB}, 1)\}$ .*

**Example 46** *For computing MAP, Park and Darwiche investigated (among a total of 11 SLS algorithms) hill-climbing (HILL) with random noise (NU), using MPE-based initialization (MPE) [42]:  $\Lambda_S = \{(\text{HILL}, p_{\text{HILL}}), (\text{NU}, p_{\text{NU}})\}$  and  $\Lambda_I = \{(\text{MPE}, 1)\}$ .*

For further examples, Schuurmans and Southey investigated eight flip selection strategies [48]; these could also be formulated as OPERATORSGS search portfolios.<sup>5</sup>

<sup>4</sup>An alternative to the stochastic portfolio is a round-robin portfolio or schedule. A round-robin portfolio is a sequence of  $q$  operators  $\Phi = (\phi_0, \dots, \phi_{q-1})$  such that the  $j$ -th time an operator is selected for execution from  $\Phi$ , operator  $\phi_i$  where  $i = j \bmod r$  is selected. We do not further investigate round-robin portfolios in this article.

<sup>5</sup>Schuurmans and Southey introduced a novel flip selection strategy called DLM; in addition they considered the existing strategies GSAT, HSAT, WSAT-G, WSAT-B, WSAT, Novelty, and Novelty+.

|   | <b>Greedy:</b> $p_G = p_{AG} + p_{MG}$   | <b>Noisy:</b> $p_N = p_{AN} + p_{MN}$   |
|---|--|---|
| <b>Additive:</b><br>$p_A = p_{AG} + p_{AN}$       | Additive & greedy: (AG, $p_{AG}$ )       | Additive & noisy: (AN, $p_{AN}$ )       |
| <b>Multiplicative:</b><br>$p_M = p_{MG} + p_{MN}$ | Multiplicative & greedy: (MG, $p_{MG}$ ) | Multiplicative & noisy: (MN, $p_{MN}$ ) |

Table 1: Joint probability table, containing OPERATORSGS search operator and selection probability pairs, and decomposed according to the two orthogonal dimensions of (i) greedy versus noisy search and (ii) additive versus multiplicative gain computation. Taken together, these two dimensions define four pairs of search operators and selection probabilities as illustrated.

OPERATORSGS( $\beta, U, \text{MAX-FLIPS}, \text{MAX-TRIES}, U_{\min}, \Lambda_I, \Lambda_S$ )

**Input:**  $\beta$  Bayesian network (BN)  
 $U$  measure of utility:  $U_A$  or  $U_M$   
MAX-FLIPS upper bound on the number of flips  
MAX-TRIES upper bound on the number of tries  
 $U_{\min}$  lower bound for termination  
 $\Lambda_I$  initialization portfolio — initialization operators  
 $\Lambda_S$  search portfolio — greedy and noisy search operators

**Output:** ( $\mathbf{b}, \hat{\mathbf{x}}^*$ )  $\mathbf{b} \in \{\text{true}, \text{false}\}$ ;  $\hat{\mathbf{x}}^*$  is estimate of MPE

**begin**

```

 $\hat{\mathbf{x}}^* \leftarrow \text{INITIALIZE}(\beta, \Lambda_I)$  {use initialization operator to create explanation  $\hat{\mathbf{x}}^*$ }
for  $i \leftarrow 1$  to MAX-TRIES {outer loop of tries}
   $\mathbf{x} \leftarrow \text{INITIALIZE}(\beta, \Lambda_I)$  {use initialization operator to create explanation  $\mathbf{x}$ }
  for  $j \leftarrow 1$  to MAX-FLIPS {inner loop of flips}
     $\mathbf{x} \leftarrow \text{SEARCH}(\beta, \mathbf{x}, \Lambda_S)$  {apply search operator to update  $\mathbf{x}$ }
    if ( $U(\mathbf{x}) > U(\hat{\mathbf{x}}^*)$ ) then  $\hat{\mathbf{x}}^* \leftarrow \mathbf{x}$ 
    if ( $U(\hat{\mathbf{x}}^*) \geq U_{\min}$ ) then return (true,  $\hat{\mathbf{x}}^*$ )
  endfor {inner loop of flips}
endfor {outer loop of tries}
return (false,  $\hat{\mathbf{x}}^*$ )

```

**end**

Figure 5: The operator-based stochastic greedy search algorithm (OPERATORSGS). OPERATORSGS computes an MPE estimate  $\hat{\mathbf{x}}^*$  given the input parameters, and operates in two main phases: an initialization phase and a local search phase. The initialization algorithm INITIALIZE applies initialization operators from  $\Lambda_I$ , while the search algorithm SEARCH applies search operators from  $\Lambda_S$ . OPERATORSGS terminates if an explanation  $\hat{\mathbf{x}}^*$  of utility  $U_{\min}$  or greater is found or if the computation exceeds MAX-TRIES tries.

Turning to our additive approach, we now introduce a particular search portfolio which contains additive operators; see also Table 1.

**Definition 47** Consider the compound operators from Definition 40 and suppose that  $p_{AN} + p_{AG} + p_{MN} + p_{MG} = 1$ . We define  $\Lambda_S(p_{AN}, p_{AG}, p_{MN}) = \{(\text{AN}, p_{AN}), (\text{AG}, p_{AG}), (\text{MN}, p_{MN}), (\text{MG}, p_{MG})\}$ , where  $p_{MG} = 1 - p_{AN} - p_{AG} - p_{MN}$ , as a search portfolio.

In Section 6.2 we will see that this search portfolio is very effective on application BNs. Also, the search portfolio in Definition 47 enables us to discuss an analogue between the probabilistic use of our additive measure and the use of noise in SLS. Varying the noise probability  $p_N$ , or the probability of applying a non-greedy heuristic, has been found to have a dramatic impact on SLS run time [8, 17, 20, 33, 36, 49, 50]. Utilizing our additive utility measure, we introduce the concept of additive probability  $p_A$ , or the probability of applying an additive heuristic, which is similar but orthogonal to noise probability  $p_N$ . In Definition 47, only the operators AN and AG are additive, hence  $p_A = p_{AN} + p_{AG}$ . Varying the additive probability  $p_A$  turns out to have a major impact on SLS computation time when searching for MPEs in partly deterministic BNs, as presented in Section 6.

```

COMPOUNDSEARCH( $\Delta U, C, \mathbf{x}$ )
  Input:  $\Delta U$    measure of gain:  $\Delta U_A$  or  $\Delta U_M$ 
            $C$      criterion of choice:  $C_G$  or  $C_N$ 
            $\mathbf{x}$    explanation
  Output:  $\mathbf{x}$    explanation
begin
  if ( $\Delta U = \Delta U_M$ )
     $\mathbf{A} \leftarrow \text{COMPUTEMULTIPLICATIVEGAIN}(\mathbf{x}, C)$ 
  else {the case  $\Delta U = \Delta U_A$ }
     $\mathbf{A} \leftarrow \text{COMPUTEADDITIVEGAIN}(\mathbf{x}, C)$ 
  ( $X_i, x_i$ )  $\leftarrow \text{CHOOSESTATE}(\mathbf{A}, C)$  {pick node  $V$  and state  $v$  from candidate array  $\mathbf{A}$ }
   $\mathbf{x} \leftarrow \mathbf{x}[X_i \leftarrow x_i]$ 
  return  $\mathbf{x}$ 
end

```

Figure 6: An algorithm used by the SEARCH algorithm in OPERATORSGS. It performs stochastic local search on the input explanation  $\mathbf{x}$ , using a compound operator defined by the input parameters  $\Delta U$  and  $C$ . The result is an updated explanation  $\mathbf{x}$ , which is output.

In the operator-based version of SGS presented in Figure 5, the number of restarts (or tries) is controlled by the input parameter MAX-TRIES.<sup>6</sup> The proof of the following theorem, where MAX-TRIES =  $\infty$ , is similar to that of Theorem 36.

**Theorem 48** *Let MAX-TRIES =  $\infty$ ,  $U = U_M$ , and  $U_{\min} = \Pr(\mathbf{x}^*)$  in OPERATORSGS. Then OPERATORSGS is a Las Vegas algorithm and returns (**true**,  $\hat{\mathbf{x}}^*$ ) such that  $\hat{\mathbf{x}}^* \in \mathbf{X}^*$ .*

The OPERATORSGS subroutines INITIALIZE and SEARCH employ the multiplicative and additive measures of gain according to their input portfolios, and operate as follows. INITIALIZE( $\beta$ ,  $\Lambda_I$ ) returns a new explanation  $\mathbf{x}$  for  $\beta$ , created using the initialization operators in the initialization portfolio  $\Lambda_I$  (see Definition 42). SEARCH( $\beta$ ,  $\mathbf{x}$ ,  $\Lambda_S$ ) performs a search step on the explanation  $\mathbf{x}$ , using the measure of gain associated with the search operator chosen from the search portfolio  $\Lambda_S$ . A search operator is picked from the search portfolio  $\Lambda_S$  as described in Definition 43. Compound operators are invoked by SEARCH using COMPOUNDSEARCH, see Figure 6.

### 4.3 Discussion and Comparison to Related Work

The problem of computing a most probable explanation (MPE) in Bayesian networks has been addressed using exact algorithms [2, 23, 28, 29, 44, 53, 58] as well as inexact algorithms [25, 26, 30, 34, 37]. Since a comprehensive literature review is well beyond the scope of this article, and our main focus is on a subclass of the inexact algorithms, namely stochastic local search algorithms, we now discuss previous research most closely related to our work.

We first consider related research on stochastic local search for solving the satisfiability problem (SAT) [14, 21, 33, 48, 50, 51]. An early contribution was the GSAT algorithm, which searches for a satisfying truth assignment in a CNF formula [51]. GSAT is controlled by the number of local search steps before a restart, MAX-FLIPS or MAX-FLIPS, and the number of restarts, MAX-TRIES or MAX-TRIES [51]. A major contribution of GSAT was its “sideways moves”; the algorithm continues to flip variables even when the number of satisfied clauses stays the same. Experiments showed that GSAT outperformed the Davis-Putnam (DP) algorithm on synthetic instances [40], and strong performance on other problems including graph coloring, the N-queens problem, and Boolean induction were also reported [51].

The GSAT algorithm has been extended in a number of directions (see [21] for a summary). For instance, the three heuristic techniques of clause weighing, averaging in of previous near solution, and (mixed) random walk have all been found to improve the performance of basic GSAT [49]. Mixed random walk, which is also known as WALKSAT [33, 50], combines random walk and greedy local search and is more focused than random noise in that it applies noise only in variables that occur in unsatisfied clauses. The main conclusion of an empirical evaluation was that mixed random walk is superior to simulated annealing and

<sup>6</sup> An alternative, not further pursued in this article, would be to use an upper bound on computation time in wall-clock time. Further discussion of termination criteria can be found in Section 5.1.

random noise in a wide range of cases [50]. Further flip selection strategy improvements — such as WSAT-G [33], WSAT-B [33], Novelty [33], Novelty+, and DLM [48] — have followed, along with progress in noise adaptation [8, 17, 33].

We now discuss related research using stochastic search techniques to compute MPE or MAP in Bayesian networks [26, 30, 34, 37, 41, 42, 45]. Pearl argued early on that stochastic simulation, also known as Gibbs sampling, can be used for computing the MPE although the algorithm was primarily intended for belief updating [45, p. 216, p. 262]. In stochastic simulation, evidence nodes are clamped, and then random samples are created by randomly picking among a node’s states based on their probabilities as given the node’s Markov blanket. Kask and Dechter empirically investigated MPE computation using stochastic simulation [26], and concluded that it does not perform as well as a greedy approach or as an approach where stochastic simulation and greedy search is combined. In addition, they found that augmenting the approach with the mini-bucket algorithm for initialization was important, which is in line with our results. We note that both stochastic simulation and greedy search are operators in OPERATORSGS, such that combining greedy search and stochastic simulation, as well as additional algorithms, within the SGS framework is easy.

Another early investigation of MPE computation by means of stochastic search techniques considers iterative local search (ILS), simulated annealing (SA), and genetic search (GS) [30]. ILS is a probabilistic hill-climbing that combines next-ascent and random-mutation hill climbing. ILS stops on (local) maxima, however since it iterates, it typically finds many maxima, one of which might be an MPE. For the empirical study of these algorithms a bipartite BN version of the QMR medical knowledge base, QMR-DT, was used. Experiments with ILS, SA, and GS showed that they converged, in many cases, to an MPE. In terms of computational speed, ILS was found to be significantly faster than SA which was significantly faster than GS. However, SA was more accurate than ILS and GS. It is speculated that the weaker performance of ILS is due to the local search getting trapped in the plethora of local maxima [30], a hypothesis which has later been confirmed.

Our work is also related to research on hybrid or portfolio algorithms [12, 13, 24, 57]. Huberman, Lukose, and Hogg investigate hard computational problems and how heuristic algorithms, often randomized, have been developed to solve such problems [24]. Their main emphasis is on portfolios of independent Las Vegas algorithms, where run times can be described using probability distributions, and consequently one can form expectation and variance (or standard deviation). By considering the 2-dimensional space spanned by expectation and standard deviation, an efficient frontier can be formed, similar to what is done in economics [52], based on varying the fractions of CPU-time allocated to different Las Vegas algorithms in a portfolio. Empirically, the NP-hard problem of graph coloring was studied using the Brelaz heuristic. By taking a portfolio approach, it was found that expected performance could be increased by 30% while also reducing risk (or standard deviation) [24]. While the main emphasis of Huberman, Lukose, and Hogg is on completely independent algorithms, they also discuss cooperating algorithms which is our main focus in SGS.

Gomes and Selman note that performance profiles vary dramatically among different algorithms over different problem instances [12]. In response, they investigate portfolios of algorithms, and find that it can be beneficial to combine algorithms with high run time variance in portfolios. Gomes and Selman consider one or more Las Vegas algorithms running independently and in parallel on multiple processors. They investigate constraint satisfaction and mixed integer programming problems empirically, and conclude that optimal portfolio design strongly depends on details of the run time distribution.

Lagoudakis and Littman formalize algorithm selection as a Markov decision process, and investigate a reinforcement learning approach to learning the value function [27]. Algorithms are selected from the portfolio in a cooperative, not independent, manner. Experimentally, they obtain promising results on algorithms for sorting and order statistics selection. Xu et al. integrate a learning approach with SAT algorithm portfolios [57]. All algorithms in the portfolio execute all problem instances in the training set in order to learn empirical hardness models. Learning is performed using features that characterize the problem instances, and learned ridge regression models are used to predict run times for individual problem instances. During test, a problem instance is solved by the SAT algorithm predicted to be fastest, hence there is no cooperation between algorithms. Xu et al. report strong results in the 2007 SAT competition.

We now discuss how our SGS approach is similar to and different from related research, and also how OPERATORSGS and SIMPLESGS compare. SIMPLESGS was primarily inspired by previous seminal research on stochastic local search for SAT, in particular the GSAT and WALKSAT algorithms [50, 51]. SGS is related to ILS and SLS, and extends ILS by applying to a wider range of BN topologies, using the additive measure, and using more advanced initialization operators before local search commences. We also note that the random-mutation hill climbing of ILS gives an effect similar to that of SGS noise operators. SLS [26], which was developed independently of SGS, shares several characteristics with SGS, such as stochastic

steps and initialization that goes beyond initialization uniformly at random. However, there are several important differences between SGS and SLS, including the initialization algorithms, the fact that SGS can use different measures of gain (including additive gain), as well as the fact that SGS has an operator-based variant OPERATORSGS.

OPERATORSGS makes a stronger distinction between utility and gain than SIMPLESGS and other SLS algorithms. A key point is that *each compound search operator has an associated gain, which is unrelated to how an explanation’s overall utility is computed using  $U$* . So, for example, some search operators in  $\Lambda_S$  can use additive gain  $\Delta U_A$ , while the overall utility of an explanation can be computed using multiplicative utility  $U_M$ . This provides greater flexibility than what is present in SIMPLESGS and other SLS algorithms. Operators that use the additive measure are optimized for the special but often occurring case of deterministic nodes, while operators using the multiplicative measure are optimized for the general case of probabilistic nodes. SIMPLESGS as well as other SLS algorithms can not use both measures in the same invocation of the algorithm, while OPERATORSGS can. Since many application BNs contain both probabilistic nodes and deterministic nodes, it turns out that using  $\Delta U_A$  and  $\Delta U_M$  in combination gives, in many cases, faster computation of MPEs using SGS. See Section 6.2 for supporting experiments with application BNs.

OPERATORSGS can simulate SIMPLESGS in the following manner: One can regard the noise and hill-climbing steps of SIMPLESGS as two operators chosen from a stochastic search portfolio  $\Lambda_S$  of size two, where one search operator is a greedy operator such as AG, the other is a stochastic operator such as NU:  $\Lambda_S = \{(\text{NU}, p_N), (\text{AG}, 1 - p_N)\}$ . And we can regard SIMPLESGS initialization as an initialization operator chosen from an OPERATORSGS stochastic initialization portfolio  $\Lambda_I$  of size one:  $\Lambda_I = \{(\text{UN}, 1)\}$ . In the following, we often do not distinguish between these two variants of SGS: SIMPLESGS on the one hand, and OPERATORSGS with  $\Lambda_S = \{(\text{NU}, p_N), (\text{AG}, 1 - p_N)\}$  and  $\Lambda_I = \{(\text{UN}, 1)\}$  on the other hand. The advantage of OPERATORSGS compared to standard SGS is generality and flexibility, which enables, as shown in experiments in Section 6.2, strong performance on application BNs. Further, as detailed in Section 6.1, the speed of OPERATORSGS and SIMPLESGS is essentially the same on SAT-like BNs.

Previous portfolio-based approaches have emphasized running multiple heuristics independently, either on the same computer or independently on multiple computers [12, 24]. Alternatively, they have picked, for a particular problem instance, the best algorithm among a portfolio of algorithms [57]. In comparison, there is in OPERATORSGS a cooperation between heuristics (similar to [27]), in the sense that initialization and search operators work on the same explanation  $\mathbf{x}$  to compute an MPE estimate  $\hat{\mathbf{x}}^*$ . There is currently no learning in OPERATORSGS, unlike in some related work [22, 27, 47, 57]. Finally, we note that our Las Vegas analysis and experimental approach is different from most previous efforts in several ways. First, we emphasize finding an MPE, and not just getting close to it, and therefore we compare with an exact method, the HUGIN clique tree clustering algorithm. Second, our synthetic networks are constructed in a systematic way, generating instances of varying hardness. This allows us to experimentally show that SGS can outperform HUGIN on synthetic BN of increasing hardness.

## 5 Analysis of Portfolios in Stochastic Local Search

The goal of this section is to analyze SLS portfolio effects and the initialization and search portfolios of OPERATORSGS in particular. Our analysis therefore complements previous SLS analysis efforts [18, 36, 38], and given our observation that this approach can simulate many other SLS algorithms, including SIMPLESGS, the analysis is hopefully of broad interest.

Section 5.1 discusses the incompleteness of SGS, and argues for the Las Vegas analysis and experimentation approach we have used in this article. We discuss our Markov chain-based analysis approach in Section 5.2, present results for the initialization portfolio in Section 5.3, and for the search portfolio in Section 5.4.

### 5.1 Incompleteness, Termination, and Las Vegas Algorithms

Stochastic local search algorithms, including SGS, are incomplete algorithms. A key limitation of incomplete algorithms is that one cannot say whether an estimate  $\hat{\mathbf{x}}^* \in \mathbf{X}^*$  or  $\hat{\mathbf{x}}^* \notin \mathbf{X}^*$ , and this raises the question of when to terminate search. Incomplete algorithms are appropriate when complete methods are impractical due to excessive demands on time, space, or both. For example, with limited space a clique tree clustering algorithm such as HUGIN might not even be able to compile a given BN [34, 39, 55]. Alternatively, there might only be a fixed amount of time available that is too short for exact computation, and one might at the same time be willing to sometimes make a mistake. For example, there are situations where the cost of

making an occasional mistake by finding  $\hat{\mathbf{x}}^* \notin \mathbf{X}^*$ , is small compared to the cost — in terms of space or time — needed to use a complete BN inference algorithm such as clique tree clustering.

The answer to the question of when to terminate SLS is therefore, we believe, ultimately application-dependent. Certain applications, including real-time applications, have hard upper bounds on execution time and they might need to terminate MPE computation if that hard upper bound is exceeded. Other applications do not have hard real-time requirements and softer termination criteria would be appropriate. Giving a clear-cut recommendation beyond these general observations is certainly beyond the scope of this article.

Since SLS algorithms are randomized, their operation may be described using bivariate distributions with random variables for (i) the running time (in number of operations, say) and (ii) the current estimate of the utility of an optimal solution. Unfortunately, keeping track of both execution time and the quality of the MPE estimate complicates scientific analysis and experimentation. For simplicity, we generally take a Las Vegas approach [12] and use  $U_{\min} = \Pr(\mathbf{x}^*)$ , making SGS terminate only after an MPE is found. Clearly, in applications one will in general not know  $\Pr(\mathbf{x}^*)$ , and therefore cannot use  $U_{\min} = \Pr(\mathbf{x}^*)$  as a termination criterion in SGS. Using  $U_{\min} = \Pr(\mathbf{x}^*)$  might therefore by some readers be regarded as “cheating”. However, in a scientific study such as the present we argue that using  $U_{\min} = \Pr(\mathbf{x}^*)$  is entirely appropriate and indeed often superior. First, it is the hardest test of SLS algorithms in terms of computing an explanation, since they are not allowed to terminate prematurely. Second, one avoids additional concerns induced by complicated termination criterion — the SLS algorithm can terminate as soon as  $U_{\min}$  is reached. In this manner, the effect of the initialization and search algorithms (our main interest here) is isolated from the effect of the termination criterion, which in applications might be quite involved.

## 5.2 Markov Chains and Hitting Times

We now introduce definitions of a few random variables that will be used to characterize SLS performance.

**Definition 49 (Number of flips and operations)** *Let the flip length (number of SLS flips until termination) be a discrete random variable  $F$ . Let the run length (number of SLS operations, both initializations and flips, until termination) be a discrete random variable  $R$ .*

There are in fact families of random variables involved here. These families are characterized by the  $\eta$  input parameters of an SLS algorithm, a tuple  $(\Theta_1 = \theta_1, \dots, \Theta_\eta = \theta_\eta)$ . In OPERATORSGS,  $\eta = 7$  and of particular interest are the initialization portfolio parameter  $\Theta_{\eta-1} = \Theta_6 = \Lambda_I$  as well as the search portfolio parameter  $\Theta_\eta = \Theta_7 = \Lambda_S$ . In SIMPLESGS we have  $\eta = 6$ . Since  $F$  depends on the parameter values  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_\eta)$  used, we say  $F(\boldsymbol{\theta})$  when we want to make this explicit. For example, if we consider MAX-FLIPS and vary  $m$  in MAX-FLIPS =  $m$  while keeping the remaining  $\eta - 1$  input parameters constant, we may write  $F(m)$ ;  $R(m)$  is similar. In addition, we may condition on portfolio parameters  $\Lambda_I$  and  $\Lambda_S$  of OPERATORSGS; more about this below.

The search processes of SIMPLESGS and similar SLS algorithms can be analyzed using exact or approximate Markov Chain models as follows (see also [18, 36]). Let us for simplicity assume  $n$  binary BN nodes such that an exact Markov Chain model has  $2^n$  states. This Markov chain is structured as a hypercube, where each hypercube node  $\mathbf{b}$  represents a bitstring. A state  $\mathbf{b} \in \{0, 1\}^n$  in such a Markov chain has  $n$  neighbors, namely those states one flip away.

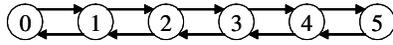
**Definition 50 (Neighborhood)** *Let  $\mathbf{b}$  be a bitstring of length  $n$ .  $\mathbf{b}$ 's neighborhood  $n(\mathbf{b})$ , strict neighborhood  $n'(\mathbf{b})$ , and non-neighborhood  $\bar{n}(\mathbf{b})$  are defined as follows*

$$\begin{aligned} n(\mathbf{b}) &= \left\{ \mathbf{c} \in \{0, 1\}^n \mid \sum_{i=1}^n |b_i - c_i| \leq 1 \right\} \\ n'(\mathbf{b}) &= n(\mathbf{b}) - \{\mathbf{b}\} \\ \bar{n}(\mathbf{b}) &= \{0, 1\}^n - n(\mathbf{b}). \end{aligned}$$

We now introduce the notion of exact Markov chains over bitstrings; they will be useful when analyzing how SLS algorithms process problem instances.

**Definition 51 (Exact Markov chain model)** *An exact Markov chain model  $\mathcal{M}$  has states  $\mathcal{S} = \{s_0, s_1, \dots\} = \{\mathbf{b} \mid \mathbf{b} \in \{0, 1\}^n\}$  and an initial probability vector  $\mathcal{V}$  with  $\Pr(A_0 = s_k) = \pi_k$  for  $1 \leq k \leq 2^n$ . The*

**Basic Random Walk:**



**Augmented Random Walk:**

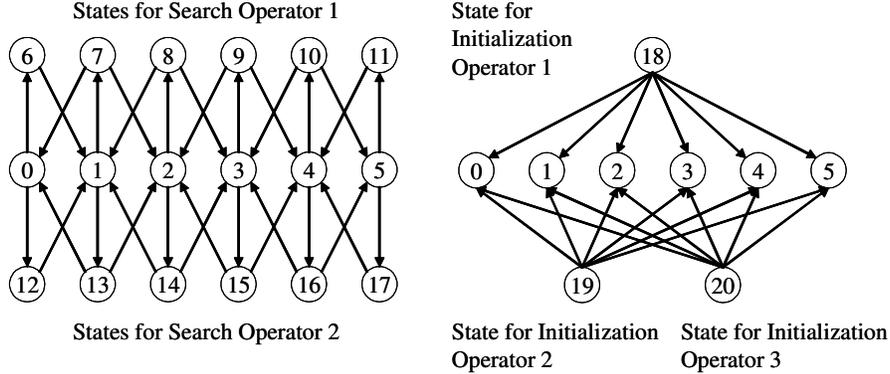


Figure 7: An example of a basic random walk (top) and a corresponding augmented random walk model (bottom) for OPERATORS<sub>SGS</sub> and other stochastic local search algorithms that use initialization and search portfolios. To reduce clutter, the states for the underlying search space are shown twice, namely in the context of the search portfolio with  $\chi = 2$  operators (bottom left) and in the context of the initialization portfolio with  $\xi = 3$  operators (bottom right). See also Example 53.

transition probability matrix  $\mathcal{P}$  is a stochastic matrix given by

$$\Pr(A_{t+1} = \mathbf{c} \mid A_t = \mathbf{b}) = 0 \text{ if } \mathbf{c} \in \bar{n}(\mathbf{b}) \quad (27)$$

$$\Pr(A_{t+1} = \mathbf{c} \mid A_t = \mathbf{b}) \geq 0 \text{ if } \mathbf{c} \in n(\mathbf{b}). \quad (28)$$

SIMPLESGS can be analyzed using exact Markov chain models up to MAX-FLIPS flips, similar to the SIMPLESLS algorithm [36]; we will discuss OPERATORS<sub>SGS</sub> shortly.

Due to the exponential growth of  $\mathcal{V}$  and  $\mathcal{P}$  as a function of  $n$  in Definition 51, we now formalize SLS behavior by means of an approximate Markov chain. Let the current state of SGS search be  $\mathbf{b}$ , and consider how search brings us closer to or farther away from an MPE  $\mathbf{b}^*$ . In the following definition we assume for simplicity, but without loss of generality [36], that  $\mathbf{b}^* = 1 \dots 1$ ;  $u(\mathbf{b})$  is a function that counts the number of 1s in  $\mathbf{b}$ . Clearly, when  $u(\mathbf{b}) = n$  we have  $\mathbf{b} = \mathbf{b}^*$ .

**Definition 52 (Positive and negative neighbors)** Let the bitstring  $\mathbf{b}$ , of length  $n$ , represent the current state of search and let  $\mathbf{b}^* = 1 \dots 1$  be the optimal bitstring. We then define positive neighbors  $n^+(\mathbf{b}) = \{\mathbf{c} \in n(\mathbf{b}) \mid u(\mathbf{c}) > u(\mathbf{b})\}$  and negative neighbors  $n^-(\mathbf{b}) = \{\mathbf{c} \in n(\mathbf{b}) \mid u(\mathbf{c}) < u(\mathbf{b})\}$ .

In an approximate Markov Chain model, we introduce states that count the number of correct bits  $u(\mathbf{b})$  in the current estimate  $\mathbf{b}$  versus  $\mathbf{b}^*$ . Clearly,  $0 \leq u(\mathbf{b}) \leq n$ . Therefore we have a Markov Chain model with  $n + 1$  states in which each state has two neighbors; a so-called random walk.

Can Markov chains such as random walks be applied also in the portfolio setting, and specifically to analyze OPERATORS<sub>SGS</sub>? As we will see in the following, the answer to this question is “yes”; we first illustrate our analysis by means of an example.

**Example 53** Suppose that we have a BN  $\beta$  with  $n = 5$  binary nodes. Let  $\Lambda_I = \{(\phi_1, p_1), (\phi_2, p_2), (\phi_3, p_3)\}$  and  $\Lambda_S = \{(\omega_1, q_1), (\omega_2, q_2)\}$ . Figure 7 shows an augmented Markov chain model  $\mathcal{M}$  (see also Definition 54) for OPERATORS<sub>SGS</sub> processing  $\beta$ , given these portfolios.

In addition to states representing the underlying search space, corresponding to the states in a basic random walk model, our novel augmented Markov chain model includes states that represent the search and initialization operators. The advantage of this model is that it allows us to analyze search and initialization operators within the same framework as the underlying search space.

**Basic Random Walk:**



**Augmented Random Walk:**

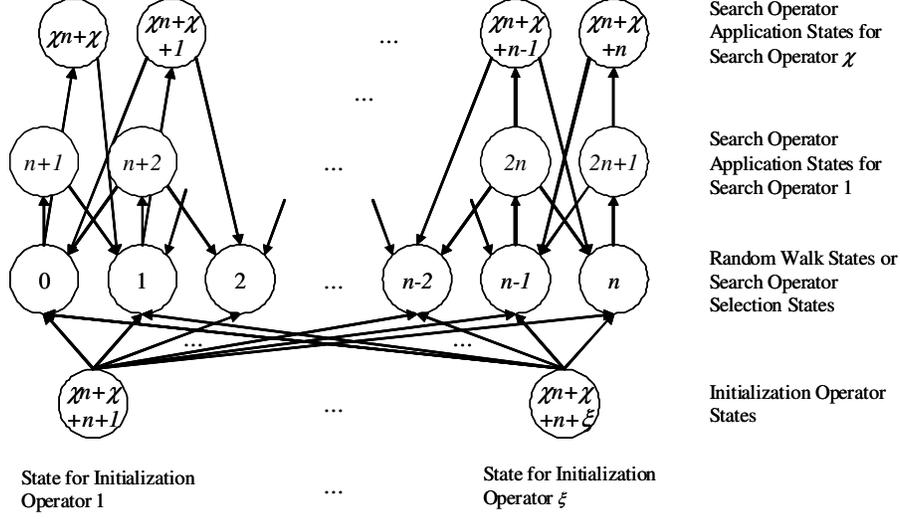


Figure 8: The general constructions for a basic random walk model (top) and a corresponding augmented random walk model (bottom) for stochastic local search algorithms that use initialization and search portfolios.

The details for Example 53 are as follows. We first consider  $\mathcal{S}$ . The number of states in  $\mathcal{S}$  is  $k = (n + 1)(\chi + 1) + \xi = 6 \times 3 + 3 = 21$ . Among these, states  $\{0, \dots, 5\}$  represent OPERATORSGS search right *before* search operator selection. States  $\{6, \dots, 11\}$  represent search right *after* selection of search operator  $\omega_1$ ; states  $\{12, \dots, 17\}$  represent search right after selection of  $\omega_2$ . States 18, 19, and 20 represent selection of initialization operators  $\phi_1, \phi_2$ , and  $\phi_3$  respectively. In  $\mathcal{V}$ ,  $\Pr(B_0 = i) = 0$  for  $0 \leq i \leq (n + 1)(\chi + 1) - 1 = 17$  and  $\Pr(B_0 = 17 + j) = p_j$  for  $1 \leq j \leq 3$ , where  $p_j$  are the initialization operator selection probabilities in Example 53. In  $\mathcal{P}$ , we have  $\Pr(B_{t+1} = i \mid B_t = j) \geq 0$  as indicated by edges in Figure 7. For all other entries in  $\mathcal{P}$ ,  $\Pr(B_{t+1} = i \mid B_t = j) = 0$ .

We now formally introduce our novel Markov chain construction that includes both initialization and search portfolios as illustrated in Example 53. We will in Theorem 55 formally show that it creates a Markov chain. To enable introduction of operators, we slightly extend the exact Markov chain notation in Definition 51 as follows. If there are  $1 \leq i \leq \xi$  different initialization operators, we have multiple initial probability vectors  $\mathcal{V}_i$  with  $\Pr(A_{i,0} = s_k)$  for  $1 \leq k \leq 2^n$ . If there are  $1 \leq j \leq \chi$  search operators, we have multiple transition probability matrixes  $\mathcal{P}_j$  where  $\Pr(A_{j,t+1} = \mathbf{c} \mid A_{j,t} = \mathbf{b}) \geq 0$  for  $\mathbf{c} \in n(\mathbf{b})$ , else for  $\mathbf{c} \in \bar{n}(\mathbf{b})$  we have  $\Pr(A_{j,t+1} = \mathbf{c} \mid A_{j,t} = \mathbf{b}) = 0$ .

**Definition 54 (Augmented random walk)** Consider a BN with  $n$  binary nodes. Suppose that we have an initialization portfolio  $\Lambda_I = \{(\phi_i, p_i) \mid 1 \leq i \leq \xi\}$  with initial probability vectors  $\mathcal{V}_i$  for  $1 \leq i \leq \xi$  and a search portfolio  $\Lambda_S = \{(\omega_j, q_j) \mid 1 \leq j \leq \chi\}$  with transition probability matrixes  $\mathcal{P}_j$  for  $1 \leq j \leq \chi$ . The augmented random walk model  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$  is defined as follows. Put  $m = (n + 1)$  and  $\ell = m(\chi + 1)$ .  $\mathcal{M}$  has  $k = \ell + \xi = |\mathcal{S}|$  discrete states named  $\{0, \dots, k - 1\}$ . The  $k$ -dimensional vector  $\mathcal{V}$  of initial probabilities is  $\Pr(B_0 = i) = 0$  for  $0 \leq i \leq \ell - 1$  and  $\Pr(B_0 = i) = p_{i - \ell + 1} \geq 0$  for  $\ell \leq i \leq k - 1$ . The potentially non-zero

entries in the  $k \times k$  matrix  $\mathcal{P}$  of conditional probabilities are:

$$\Pr(B_{t+1} = i \mid B_t = j + \ell - 1) = \sum_{\mathbf{u}(\mathbf{b})=i} \Pr(A_{j,0} = \mathbf{b}), \quad 0 \leq i \leq n \text{ and } 1 \leq j \leq \xi. \quad (29)$$

$$\Pr(B_{t+1} = jm + i \mid B_t = i) = q_j, \quad 0 \leq i \leq n \text{ and } 1 \leq j \leq \chi. \quad (30)$$

$$\Pr(B_{t+1} = i + 1 \mid B_t = jm + i) = \sum_{\mathbf{b}^+ \in n^+(\mathbf{b})} \Pr(A_{j,t+1} = \mathbf{b}^+ \mid A_{j,t} = \mathbf{b}), \quad 0 \leq i < n \text{ and } 1 \leq j \leq \chi. \quad (31)$$

$$\Pr(B_{t+1} = i - 1 \mid B_t = jm + i) = \sum_{\mathbf{b}^- \in n^-(\mathbf{b})} \Pr(A_{j,t+1} = \mathbf{b}^- \mid A_{j,t} = \mathbf{b}), \quad 0 < i \leq n \text{ and } 1 \leq j \leq \chi. \quad (32)$$

The remaining entries in  $\mathcal{P}$  are zero.

We now discuss Definition 54 in more detail; see also Figure 8. There is a one-to-one mapping between the  $\xi$  last states in the augmented random walk model and the tuples in the initialization portfolio  $\Lambda_I = \{(\phi_1, p_1), \dots, (\phi_\xi, p_\xi)\}$ . These  $\xi$  *initialization (operator) states* are shown at the bottom in Figure 8; the probabilities of the edges from these states are given by (29). There is also a relationship between states  $0 \leq i \leq (n+1)(\chi+1)$  and the tuples in the search portfolio  $\Lambda_S = \{(\omega_1, q_1), \dots, (\omega_\chi, q_\chi)\}$ : States  $0 \leq i \leq n$ , which we will call (*search operator*) *selection states*, correspond to the moment, in OPERATORSGS, right before search operator selection from  $\Lambda_S$  according to (30). States  $(n+1) \leq i \leq (n+1)(\chi+1)$  are called (*search operator*) *application states*. Each of these latter states correspond to a time when a search operator has been picked from  $\Lambda_S$ , and is about to be applied by OPERATORSGS. Once an operator has been selected and applied using (31) and (32), the Markov chain is again in a selection state  $0 \leq i \leq n$ , and the selection-application cycle continues until restart or termination.

There are two options for Markov chain analysis of our portfolio approach. First, we can perform analysis directly on the augmented random walk. Second, we can use the augmented random walk merely as an aid in constructing the basic random walk, and perform analysis on the basic random walk. In either case, since we have a Markov chain, Markov chain analysis techniques can be applied. In the rest of this work our main emphasis is on analysis directly on the augmented random walk, as reflected in the following result.

**Theorem 55** *An augmented random walk model  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$  induces a Markov chain  $(B_t, t \geq 0)$ .*

**Proof.** We consider  $\mathcal{S}$ ,  $\mathcal{V}$ , and  $\mathcal{P}$  in turn.  $\mathcal{S}$  has  $k = (n+1)(\chi+1) + \xi = \ell + \xi$  states and obviously defines a finite state space.  $\mathcal{V}$  is given by  $(\pi_0, \dots, \pi_{\ell+\xi-1})$ , where

$$\sum_{i=0}^{\ell+\xi-1} \pi_i = \sum_{i=0}^{\ell-1} \pi_i + \sum_{i=\ell}^{\ell+\xi-1} \pi_i = \sum_{j=1}^{\xi} p_j = 1$$

by the definition of  $\Lambda_I = \{(\phi_1, p_1), \dots, (\phi_\xi, p_\xi)\}$ . For  $\mathcal{P}$ 's  $\xi$  initialization states we have for any  $1 \leq j \leq \xi$ :

$$\sum_{i=0}^{\ell+\xi-1} \Pr(B_{t+1} = i \mid B_t = j + \ell - 1) = \sum_{i=0}^n \Pr(B_{t+1} = i \mid B_t = j + \ell - 1) = \sum_{\mathbf{u}(\mathbf{b})=i} \Pr(A_{j,0} = \mathbf{b}) = 1,$$

which follows from (29) and the fact that  $A_{j,0}$  is an initial probability vector in an exact Markov chain model. For  $\mathcal{P}$ 's  $n+1$  selection states we have for any  $0 \leq i \leq n$ :

$$\sum_{j=0}^{\ell+\xi-1} \Pr(B_{t+1} = j \mid B_t = i) = \sum_{j=1}^{\chi} \Pr(B_{t+1} = jm + i \mid B_t = i) = \sum_{j=1}^{\chi} q_j = 1,$$

which follows from (30) and the definition  $\Lambda_S = \{(\omega_1, q_1), \dots, (\omega_\chi, q_\chi)\}$ . Finally, for  $\mathcal{P}$ 's  $\chi(n+1)$  application states we have for  $1 \leq j \leq \chi$  and  $0 \leq i \leq n$ :

$$\begin{aligned} \sum_{k=0}^{\ell+\xi-1} \Pr(B_{t+1} = k \mid B_t = jm + i) &= \sum_{k=0}^n \Pr(B_{t+1} = k \mid B_t = jm + i) \\ &= \Pr(B_{t+1} = i + 1 \mid B_t = jm + i) + \Pr(B_{t+1} = i - 1 \mid B_t = jm + i), \end{aligned}$$

and by substituting (31) and (32) into the above we obtain:

$$\sum_{k=0}^{\ell+\xi-1} \Pr(B_{t+1} = k \mid B_t = jm + i) = \sum_{\mathbf{b}^+ \in n^+(\mathbf{b})} \Pr(A_{j,t+1} = \mathbf{b}^+ \mid A_{j,t} = \mathbf{b}) + \sum_{\mathbf{b}^- \in n^-(\mathbf{b})} \Pr(A_{j,t+1} = \mathbf{b}^- \mid A_{j,t} = \mathbf{b}) = 1,$$

where the last equality follows because  $\sum_{\mathbf{c} \in n'(\mathbf{b})} \Pr(A_{j,t+1} = \mathbf{c} \mid A_{j,t} = \mathbf{b}) = 1$  in an exact Markov chain model and  $n'(\mathbf{b})$  can be partitioned into  $n^+(\mathbf{b})$  and  $n^-(\mathbf{b})$ . Since we have now shown that (i)  $\mathcal{V}$  is a valid initial probability vector of size  $k = |\mathcal{S}|$  and (ii) all of  $\mathcal{P}$ 's  $\xi + n + 1 + \chi(n + 1) = k$  rows sum to one and hence  $\mathcal{P}$  is a valid  $k \times k$  conditional probability matrix, the desired result follows. ■

For SLS algorithms, one is typically interested in expected run time and its minimization. In this article, we study the expected hitting time  $h(\boldsymbol{\theta})$ . For our augmented random walk, the concept of expected hitting time is well-defined since our walk is a Markov chain. Given our results on additive utility and gain in Section 3 we are in particular interested in  $h(p_A)$ , where  $p_A$  is the probability of applying an additive operator in the search portfolio. We now turn to hitting time minimization and define  $\boldsymbol{\theta}^* = \arg \min h(\boldsymbol{\theta})$  and  $p_A^* = \arg \min h(p_A)$ . The general problems of efficiently computing  $\boldsymbol{\theta}^*$  and  $p_A^*$  are topics for ongoing research. In the rest of this section we provide a few analytical results; in addition there are substantial empirical results as reflected in the experiments of this article.

### 5.3 Analysis of Initialization

We now turn to an analysis of the OPERATORSGS initialization portfolio. We consider different types of initialization portfolios, and introduce the following terminology.

**Definition 56** *Let  $\Lambda$  be a stochastic portfolio. If  $|\Lambda| = 1$  then we call  $\Lambda$  a homogenous portfolio. If  $|\Lambda| \geq 2$  then we call  $\Lambda$  a heterogeneous portfolio.*

Instead of varying the size of a portfolio, as implied by Definition 56, one can set selection probabilities to zero. For example, a portfolio with exactly one operator is obviously equivalent to a portfolio in which exactly one operator has selection probability one while all other operators have selection probability zero. For simplicity we exclude portfolios in which operators have zero probability of selection in our analysis in this section.

We introduce a random variable  $\Phi$  to represent OPERATORSGS's random selection of an initialization operator from  $\Lambda_I$ . Suppose there are  $\xi$  operators in the initialization portfolio. Given  $\Phi$  and the random variable  $F$  representing the number of flips, we introduce conditional probabilities  $\Pr(F \mid \Phi = \phi_i)$  and conditional expectations  $E(F \mid \Phi = \phi_i)$  for  $0 \leq i \leq \xi$ . For the special case of MAX-FLIPS =  $\infty$ , we have

$$E(F \mid \Phi = \phi_i) = E(F \mid B_0 = j) = E(T \mid B_0 = j),$$

where  $j = i + (n + 1)(\chi + 1) - 1$  (see Definition 54). Here,  $F$  is the number of flips and  $T$  is the first passage time.

We may consider different initialization portfolios of the form  $\Lambda_I = \{(\phi_1, p_1), (\phi_2, p_2), \dots, (\phi_\xi, p_\xi)\}$ . Of particular interest is the optimal initialization portfolio  $\Lambda_I^*$ , formally defined as follows.

**Definition 57 (Optimal initialization)** *The optimal initialization portfolio  $\Lambda_I^* = \{(\phi_1, p_1^*), (\phi_2, p_2^*), \dots, (\phi_\xi, p_\xi^*)\}$  is defined by its optimal probabilities  $\mathbf{p}^* = (p_1^*, p_2^*, \dots, p_\xi^*)$*

$$(p_1^*, \dots, p_\xi^*) = \arg \min_{p_1, \dots, p_\xi} \left( \sum_{i=1}^{\xi} E(F \mid \Phi = \phi_i) p_i \right).$$

Clearly, the challenge here is to find  $\mathbf{p}^*$ . It turns out that homogenous initialization portfolios, defined as  $\Lambda_{I,1} = \{(\phi_1, 1)\}$ ,  $\Lambda_{I,2} = \{(\phi_2, 1)\}$ ,  $\dots$ ,  $\Lambda_{I,\xi} = \{(\phi_\xi, 1)\}$ , play a central role as reflected in the following result.

**Theorem 58** Let  $\xi$  be the number of operators in the initialization portfolio. Consider, for  $1 \leq i \leq \xi$ ,  $E(F | \Phi = \phi_i)$ .<sup>7</sup> Suppose that  $E(F | \Phi = \phi_i) \neq E(F | \Phi = \phi_j)$  for all  $1 \leq i, j \leq \xi$  where  $i \neq j$ . Then there is a unique optimal initialization portfolio  $\Lambda_I^*$ , namely some homogenous portfolio  $\Lambda_{I,i}$  for  $1 \leq i \leq \xi$ .

**Proof.** Since  $E(F | \Phi = \phi_i) \neq E(F | \Phi = \phi_j)$  for  $1 \leq i, j \leq \xi$  where  $i \neq j$ , we assume without loss of generality the strict ordering  $E(F | \Phi = \phi_1) < E(F | \Phi = \phi_2) < \dots < E(F | \Phi = \phi_\xi)$ . Consider now the expectation  $E(F)$  for an arbitrary heterogeneous initialization portfolio  $\{(\phi_1, p_1), (\phi_2, p_2), \dots, (\phi_\xi, p_\xi)\}$ . Using the law of conditional expectation gives

$$E(F) = \sum_{i=1}^{\xi} E(F | \Phi = \phi_i) \Pr(\Phi = \phi_i) = \sum_{i=1}^{\xi} E(F | \Phi = \phi_i) p_i. \quad (33)$$

By our strict ordering assumption of the homogenous portfolios, we can for  $2 \leq i \leq \xi$  write  $E(F | \Phi = \phi_i) = E(F | \Phi = \phi_1) + c_i$  for increasing constants  $c_i > 0$ . For convenience, we define  $c_1 = 0$ . Now, (33) can be written as

$$\begin{aligned} E(F) &= \sum_{i=1}^{\xi} (E(F | \Phi = \phi_1) + c_i) p_i \\ &= \sum_{i=1}^{\xi} E(F | \Phi = \phi_1) p_i + \sum_{i=1}^{\xi} c_i p_i \\ &= E(F | \Phi = \phi_1) + \sum_{i=1}^{\xi} c_i p_i. \end{aligned} \quad (34)$$

There are now two cases to consider. Case (i):  $p_1 = 1$ . From (34) it is easy to see that  $E(F) = E(F | \Phi = \phi_1)$  since  $p_i = 0$  for  $i \neq 1$ . Case (ii):  $p_1 < 1$ . From (34) it follows that  $E(F) > E(F | \Phi = \phi_1)$ , since clearly  $\sum_{i=1}^{\xi} c_i p_i > 0$ . Consequently,  $E(F | \Phi = \phi_1)$  is the minimal expectation, and the homogenous initialization portfolio containing exactly one operator is optimal, and specifically  $\Lambda_I^* = \Lambda_{I,1} = \{(\phi_1, 1)\}$ . ■

The implication of Theorem 58 is that once parameters  $\theta$  have been fixed, then best performance is ensured by using a homogenous initialization portfolio. Some readers might regard Theorem 58 as a negative result for initialization portfolios and might ask why we consider them in the first place. Here is why. First, Theorem 58 could not have been derived unless an initialization portfolio was assumed. Second, one is typically interested in performing inference over distributions of BNs, not one particular BN. And for different BNs, different initialization algorithms often turn out to be optimal, and the use of an initialization portfolio supports the desired flexibility in the use of initialization algorithms. Third, even though one initialization algorithm among many is optimal, it is often not a priori obvious which one it is. So there is a pre-processing phase, during which different initialization algorithms are tested, and the existence of an initialization portfolio enables this pre-processing phase.

## 5.4 Analysis of Search

We now turn our attention to SLS operations and assume that  $\text{MAX-TRIES} = \infty$  and that the initialization portfolio has been fixed. In OPERATORSGS an operation is executed when an operator is picked from either  $\Lambda_S$  or  $\Lambda_I$  and then run. In SIMPLESGS an operation is executed when INITIALIZE is called in the loop of tries and when SEARCH is called in the loop of flips.

The expected number of operations executed, including both initializations and local search steps, is characterized by the following result (see [38] for the derivation).

**Theorem 59 (Expected number of operations)** Let  $\text{MAX-FLIPS} = m$  and suppose that  $\Pr(R(m) \leq m) > 0$ . The expected number of SLS operations executed,  $E(R(m))$ , is given by

$$E(R(m)) = \frac{m(1 - \Pr(F(\infty) \leq m)) + \sum_{i=0}^m i \Pr(F(\infty) = i) + 1}{\Pr(F(\infty) \leq m)}. \quad (35)$$

<sup>7</sup>Note that  $E(T | \Phi = \phi_i)$  is a number while  $E(T | \Phi)$  is a random variable. In the former case the initialization portfolio  $\Phi$  is given as a specific portfolio  $\phi_i$ , while in the latter case  $\Phi$  is not specified.

Theorem 59 is a generalization of previous results [43, 48] in that it accounts for all operators including initialization operators in  $\Lambda_S$ . A limitation of Theorem 59, however, is that it does not include  $p_N$  or  $p_A$  as a parameter. We now address this limitation, utilizing the novel Markov chain model introduced in Definition 54.

Our analysis in the rest of this section assumes an SLS model  $(\mathcal{M}, \mathcal{O})$ , where  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$  is a Markov chain. Further,  $T$  is (as before) a random variable representing  $\mathcal{M}$ 's first passage time. The benefit of a Markov chain analysis is illustrated as follows. Suppose that we vary only the  $\Lambda_S$  part in  $\theta$ , and further suppose that  $\Lambda_S$  contains only two operators, namely a noisy operator (applied with probability  $p_N$ ) and greedy operator (applied with probability  $p_G = 1 - p_N$ ). In related work we have shown that the expected hitting time  $h$  as a function of  $p_N$ ,  $h(p_N)$ , are rational functions, or in other words ratios of polynomials  $P(p_N)$  and  $Q(p_N)$ :  $h(p_N) = P(p_N)/Q(p_N)$  [36]. The curves for  $h(p_N)$  are analytical counterparts to and explain theoretically the so-called noise response curves that have been extensively studied empirically [36]. The impact of noise varies with BN hardness, with optimal noise level  $p_N^*$  increasing with increasing problem hardness.

Analogously, we consider here hitting time as a function of additive probability  $p_A$ , with corresponding (empirical) additive response curves investigated in Section 6. Consider an SLS model  $(\mathcal{M}, \mathcal{O})$ , where the Markov Chain  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$  is defined over a bitstring of length  $n$  and with parameter  $p_A$ . Let  $\kappa = n + 1$ , assume optimum states  $\{s_\lambda, \dots, s_\kappa\} = \mathcal{O}$  where  $\lambda \leq \kappa$ , and form a system of equations for  $\mathcal{M}$ 's expected first passage times  $m_i$  for  $1 \leq i \leq \kappa$ . There exists an equivalent upper triangular system  $\mathbf{U}\mathbf{m} = \mathbf{b}$ , where  $\mathbf{m} = (m_1, \dots, m_{\lambda-1})^T$ , in which all coefficients in  $\mathbf{U}$  and  $\mathbf{b}$  are rational functions of  $p_A$ . Performing back substitution on  $\mathbf{U}\mathbf{m} = \mathbf{b}$ , we obtain the following expected hitting time result.

**Theorem 60 (Rationality of hitting time)** *Consider an SLS model  $(\mathcal{M}, \mathcal{O})$ , where  $\mathcal{M} = (\mathcal{S}, \mathcal{V}, \mathcal{P})$  is an augmented Markov Chain defined over a bitstring of length  $n$  and with parameter  $p_A$ . The expected hitting time for  $\mathcal{M}$  is a rational function of  $p_A$ ,  $h(p_A) = P(p_A)/Q(p_A)$ , where  $P(p_A)$  and  $Q(p_A)$  are polynomials.*

The proof of Theorem 60 is similar to that of our previous result for  $p_N$  [36], except that it (i) concerns  $p_A$  rather than  $p_N$ , and (ii) is based on our approximate (see Definition 54) rather than an exact Markov (see Definition 51) chain model. In the proof of the theorem, the key idea is to symbolically perform Gaussian elimination, such that the parameter  $p_A$  is preserved throughout the derivation.

The details for  $p_A$  closely parallel those for  $p_N$ , which already have been established [36], and consequently we only provide a few highlights here. Theorem 60, which is concerned with a single BN, can be extended in a natural way to multiple BNs by means of finite mixture distributions. Again, we consider additive probability to be the independent parameter. Because of closure properties [36], it follows from Theorem 60 that the SLS hitting time for a mixture of BNs is also a rational function  $h(p_A) = P(p_A)/Q(p_A)$ . To approximate the rational functions induced by single BNs as well as multiple BNs as one can use low-order polynomials [36]; this approach is motivated by Weierstrass' theorem and supported by empirical results.

## 6 Experiments

How useful is additive utility? How well does SGS perform compared to a state-of-the-art complete algorithm? In order to answer these and other questions, we now turn to experiments performed with the SGS system, which implements the SIMPLESGS and OPERATORSGS algorithms.

Our main emphasis in this section is on scientific experimentation rather than competitive experimentation [16]. In other words, the purpose of our experiments is not to show that our novel SLS techniques are uniformly faster than existing algorithms or heuristics for all problem instances (competitive experimentation [16]). Instead, we aim to complement our analytical results, and in particular improve the understanding of the portfolio approach as well as that of additive utility and gain (scientific experimentation [16]). Rather than investigating a large number of SLS systems at a superficial level, we study SGS in depth. Apart from SGS we know of no SLS system for MPE computation that implements our additive approach. In addition, there is clear evidence that a problem instance that is hard for one SLS algorithms is also hard for other SLS algorithms [20].

In addition, we investigate the question of how our SLS algorithms, as implemented by SGS, compare to exact algorithms, in particular clique tree clustering as implemented in the state-of-the-art tree clustering system HUGIN [5, 28]. HUGIN uses two phases to compute an MPE, a compilation phase and an execution phase. HUGIN was chosen since (i) we wanted to employ an exact algorithms to compute MPEs (as opposed to approximations) and (ii) HUGIN uses one of the best exact methods, the tree clustering algorithm. Due

to the fundamental differences between stochastic local search (including SGS) and clique tree clustering (including HUGIN), it is non-trivial to perform an “apples-to-apples” comparison of the two systems. To minimize differences, we used a Las Vegas methodology, motivated in Section 5.1, which has also been used extensively in experimental research on satisfiability [40]: HUGIN was always used before SGS, in order to find an MPE  $\mathbf{x}^*$  if one existed. The reason for focusing on instances which HUGIN could process is as follows. For these cases we know for sure that some non-zero MPE  $\mathbf{x}^*$  exists and can therefore input  $U_{\min} = \Pr(\mathbf{x}^*)$  to SGS to force convergence to an MPE. In addition, and following what is typically done in experiments, we assume that HUGIN’s compilation time as well as SGS’s parameter optimization time can be amortized over a large number of MPE queries to a BN.

In the rest of this section we provide experimental results for SGS and compare SGS to HUGIN. We use synthetic BNs in Section 6.1 and BNs from applications in Section 6.2. For the experiments discussed here, a Dell Dimension 4500 CPU, 2GHz Intel Pentium IV using 1GB of RAM and up to 2GB of swap space has been used. The computer was running Windows XP.

## 6.1 Experiments with Synthetic Bayesian Networks

We now compare SGS to HUGIN, using synthetically generated BNs. These results provide insights into the general patterns of performance of a stochastic local search algorithm such as SGS compared to a well-established baseline.

Section 6.1.1 outlines the methodology used. In Section 6.1.2 we focus on the effect, on HUGIN as well as on SIMPLESGS and OPERATORSGS, of varying the  $C/V$ -ratio in BNs. In Section 6.1.3 we investigate additive gain using OPERATORSGS.

### 6.1.1 Synthetic Networks: Methodology

In this section, we briefly discuss a paradigm for the generation of increasingly hard BNs for inference, namely the BPART algorithm [34, 39]. The BPART algorithm extends research on generating hard instances for the satisfiability problem [40], and we are exploiting the close relationship between computing an MPE and finding a satisfying truth assignment of a corresponding CNF formula [4, 46, 54]. Generating problem instances at random can result in very easy problems [40]. By carefully manipulating one or more BPART input parameters, one can construct BNs that existing tree clustering inference algorithms cannot handle due to an approximately exponential increase in clique tree size [35, 39]. The main quantity we vary here is the ratio  $C/V$ , where  $C$  and  $V$  are BPART input parameters representing the number of leaf and root nodes in a BN respectively. The  $C/V$ -ratio is perhaps easiest to motivate using bipartite BNs for medical diagnosis, where  $V$  is the number of diseases and  $C$  is the number of symptoms. Clearly, it is interesting to understand how the speed of MPE computation varies when the ratio of symptoms to diseases is varied, or in other words as the  $C/V$ -ratio is varied. For the experiments reported here we set BPART’s input parameters as follows, generating SAT-like BNs [39]: The CPT type of the root nodes was  $Q = \text{uniform}$ ; the CPT type of the non-root nodes was  $F = \text{or}$ ; the number of root nodes was  $V = 30$ ; the number of states per node was  $S = 2$ ; the number of parents per leaf node was  $P = 3$ ; and irregular BNs were created by setting  $R = \text{false}$ . Here, “irregular” refers to the fact that each root node has an irregular or randomly varying number of children. We varied the number of leaf nodes while keeping  $V = 30$  constant, giving  $C/V$ -ratios ranging from  $C/V = 2.0$  to  $C/V = 3.4$ .

Our main focus in the following is the comparison of the execution phase of HUGIN with that of SGS (using approximately optimal parameter settings) when computing an MPE. Our methodology is to generate a number of BN instances according to the BPART construction, run SGS and HUGIN on these instances, and record statistics for the time it takes to compute an MPE. We do not report results for BNs that HUGIN was not able to process, even in cases where SGS might have been able to find an MPE. For example, we did not experiment with as large SAT-like BNs as the SAT formulas used in earlier research [40], since HUGIN was not able to process these large networks for interesting  $C/V$ -ratios. Consequently, our results do not include the region where  $C/V \approx 4.25$ , which is SAT’s phase-transition region [40].

Even though these SAT-like BNs have a clear mapping to SAT instances, a few crucial differences make our experiments different from previous experiments in the SAT setting. Our BN data structures are different, since they are able to represent arbitrary CPTs, not just the logical true or false values that arise in pure logical inference. Along similar lines, our gain and utility computations are generalizations of their logical counterparts as discussed in Section 3 and Section 4. They need to handle the general, real-valued case as encountered in CPTs, not just the special, integer-valued case of logical inference.

| BNs<br>C/V | Total clique tree size (in 1000) |        |        |        |       | Size of maximal clique (in 1000) |         |         |        |         |
|------------|----------------------------------|--------|--------|--------|-------|----------------------------------|---------|---------|--------|---------|
|            | Mean                             | StDev  | Median | Max    | Min   | Mean                             | StDev   | Median  | Max    | Min     |
| 2.0        | 201.7                            | 95.17  | 172.1  | 513.3  | 53.34 | 70.94                            | 44.49   | 65.54   | 262.1  | 16.38   |
| 2.2        | 388.2                            | 211.1  | 337.9  | 1,212  | 53.60 | 150.6                            | 109.0   | 131.1   | 524.3  | 16.38   |
| 2.4        | 573.8                            | 288.3  | 504.5  | 1,628  | 131.8 | 238.6                            | 157.1   | 262.1   | 1,049  | 65.54   |
| 2.6        | 852.7                            | 449.1  | 718.8  | 2,302  | 190.9 | 329.6                            | 226.1   | 262.1   | 1,049  | 65.54   |
| 2.8        | 1,285                            | 789.0  | 1,158  | 5,463  | 278.5 | 503.3                            | 356.3   | 524.3   | 2,097  | 131.1   |
| 3.0        | 1,881                            | 991.7  | 1,651  | 6,506  | 580.9 | 720.9                            | 331.4   | 524.3   | 2,097  | 131.1   |
| 3.2        | 2,559                            | 1,241  | 2,337  | 6,468  | 607.6 | 1,077                            | 669.5   | 1,049   | 4,194  | 262.1   |
| 3.4        | 3,779                            | 1,807  | 3,504  | 11,204 | 689.1 | 1,565                            | 944.4   | 1,049   | 4,194  | 262.1   |
|            | Time, all propagations (seconds) |        |        |        |       | Time, one propagation (seconds)  |         |         |        |         |
|            | Mean                             | StDev  | Median | Max    | Min   | Mean                             | StDev   | Median  | Max    | Min     |
| 2.0        | 1.119                            | 0.5590 | 1.039  | 3.234  | 0.328 | 0.07038                          | 0.03405 | 0.06491 | 0.1902 | 0.01726 |
| 2.2        | 2.123                            | 1.220  | 1.860  | 7.047  | 0.375 | 0.1404                           | 0.08568 | 0.1236  | 0.5560 | 0.02344 |
| 2.4        | 3.102                            | 1.758  | 2.696  | 9.953  | 0.891 | 0.2291                           | 0.1345  | 0.1952  | 0.7864 | 0.04689 |
| 2.6        | 4.809                            | 3.234  | 3.898  | 24.55  | 0.718 | 0.3655                           | 0.2374  | 0.3015  | 1.444  | 0.1000  |
| 2.8        | 6.349                            | 4.249  | 5.258  | 25.64  | 1.234 | 0.5706                           | 0.4164  | 0.5137  | 3.205  | 0.1198  |
| 3.0        | 8.267                            | 5.039  | 7.008  | 36.31  | 1.782 | 0.8618                           | 0.4744  | 0.7428  | 3.026  | 0.2206  |
| 3.2        | 12.76                            | 8.332  | 10.79  | 41.91  | 2.875 | 1.293                            | 0.6811  | 1.120   | 3.224  | 0.2756  |
| 3.4        | 14.67                            | 9.867  | 9.781  | 66.5   | 2.797 | 1.864                            | 1.059   | 1.738   | 6.045  | 0.3108  |

Table 2: Clique tree statistics (top half) and propagation times (bottom half) for HUGIN on synthetic, bipartite Bayesian networks. Parameters characterizing these BNs are the number of root nodes  $V$  and the number of leaf nodes  $C$ .

### 6.1.2 Synthetic Networks: Hardness and the $C/V$ ratio

The purpose of the experiments reported here was to compare the performance of SGS, both SIMPLESGS and OPERATORSGS, with that of HUGIN as the  $C/V$ -ratio was varied. For each  $C/V$  level, using  $V = 30$  and for  $C/V = 2.0$  to  $C/V = 3.4$ , 100 BNs were generated using the BPART algorithm and processed using HUGIN and SGS.

Clique tree results for HUGIN are summarized in Table 2; each row shows statistics for BPART instances. There are dramatic increases in total clique tree size as well as in the size of the largest clique as  $C/V$  increases. The rapid growth of the total clique tree size with  $C/V$  causes MPE computation times to grow rapidly as well. In Table 2, sample means and standard deviations of MPE computation times for HUGIN are presented.

We also experimented with both variants of SGS, namely SIMPLESGS and OPERATORSGS, with parameters set as follows:  $\text{MAX-TRIES} = \infty$ ,  $U = U_A$ ,  $C = C_G$ , and  $U_{\min} = 75$  (for  $C = 60$ ) to  $U_{\min} = 117$  (for  $C = 102$ ).  $\text{MAX-FLIPS}$  and  $p_N$  were set to approximately optimal levels, varying with  $C/V$ . For both variants, an MPE was found for all problem instances.

In Figure 9 the results from these experiments are shown along with exponential regression lines of the form  $y = ae^{bx}$  where  $a, b \in \mathbb{R}$  and  $x = C/V$ . For SIMPLESGS and OPERATORSGS, each data point in the figure represent a total of 10,000 experiments. This is because there are 100 BNs per  $C/V$  level, and each SGS variant was executed 100 times for each BN. The exponential regression results in Figure 9 show that the computation times of HUGIN and SGS increase with  $x = C/V$  in a fashion that is well-approximated by exponential curves. For HUGIN, this confirms earlier results [35, 39]. For HUGIN, we note also that the execution phase consists of several propagations when there is more than one MPE [32]. The existence of multiple MPEs for this  $C/V$  range is reflected in the obvious difference between  $y = 0.03847e^{1.797x}$  (for all propagations) and  $y = 0.000870e^{2.289x}$  (for one propagation) in Figure 9.

The run time curves of SIMPLESGS and OPERATORSGS in Figure 9 are quite similar and in fact difficult to tell apart. A potential disadvantage of OPERATORSGS compared to SIMPLESGS, namely slower speed due to computational overhead associated with operator selection and application, is in other words minimized in our current implementation. Both SIMPLESGS and OPERATORSGS clearly outperforms HUGIN here. Specifically, for MPE computation we note that SIMPLESGS and OPERATORSGS on average are more than two orders of magnitude faster than HUGIN (the ‘‘all props’’ regression line in Figure 9) on these BPART BNs where  $2.0 \leq C/V \leq 3.4$ . Even if HUGIN had only needed one propagation (the ‘‘one prop’’ regression line in Figure 9) in order to compute an MPE, which clearly is the lower bound, both SGS variants are on average well over one order of magnitude faster than HUGIN for  $2.0 \leq C/V \leq 3.4$ .

### 6.1.3 Synthetic Networks: Investigation of Additive Gain for Varying $C/V$

The goal of this set of experiments was to investigate different OPERATORSGS variants. In particular, we wanted to investigate both multiplicative gain and additive gain in SAT-like BNs, generated using the BPART algorithm as discussed in Section 6.1.2. Since the additive measure is tailored to SAT-like BNs, one might expect SGS with  $\Lambda_S = \{(AG, p_{AG}), (NU, p_{NU})\}$  to outperform SGS with  $\Lambda_S = \{(MG, p_{MG}), (NU, p_{NU})\}$  in this experiment. However, SGS with  $\Lambda_S = \{(MG, p_{MG}), (NU, p_{NU})\}$  is augmented with a “force state” heuristic which handles deterministic nodes, as shown in Figure 3, and the question is how helpful this heuristic is compared to the additive measure. In this set of experiments, we again used  $V = 30$  root nodes, and considered  $C = 60$  and  $C = 102$ .

Results of the experiments are shown in Figure 12. In the two panels to the left in the figure, we show results for OPERATORSGS with  $\Lambda_S = \{(AG, p_{AG}), (NU, p_{NU})\}$ . In the two panels to the right, we show results for OPERATORSGS with  $\Lambda_S = \{(MG, p_{MG}), (NU, p_{NU})\}$ . Results for  $C/V = 2.0$  are displayed in the top two panels in Figure 12, while the bottom two panels show results for  $C/V = 3.4$ . In addition, we used three different values for  $p = p_{NU}$  (and consequently for  $p_{AG}$  and  $p_{MG}$  respectively) and varied MAX-FLIPS as shown on the  $x$ -axis in Figure 12.

Our main observations with respect to these experimental results are as follows. For both  $C/V = 2.0$  and  $C/V = 3.4$ , the additive approach (to the left in Figure 12) is the winner over the multiplicative approach in the sense of having the parameter settings that minimizes the number of flips. In addition, the additive approach is more robust in the sense that there is good performance for a wider range of parameter values. This experiment provide an experimental rationale for the “artificial” additive approach, which is used in OPERATORSGS with  $\Lambda_S = \{(AG, p_{AG}), (NU, p_{NU})\}$ , compared to the multiplicative approach used in OPERATORSGS with  $\Lambda_S = \{(MG, p_{MG}), (NU, p_{NU})\}$ .

## 6.2 Experiments with Application Bayesian Networks

We now investigate the SGS approach empirically using application instances. The OPERATORSGS algorithm was to a large extent motivated by our work with these BNs, which unlike the synthetic instances investigated in Section 6.1 have no restriction imposed on them. For instance, these BNs have no restrictions on topology, number of states, number of parents, or CPTs.

This section reports on the performance of OPERATORSGS when using different initialization portfolios, different search portfolios, and different measures of gain. In Section 6.2.1 we outline the methodology and briefly discuss the BNs used in experiments. Section 6.2.2 empirically compares OPERATORSGS and HUGIN. Section 6.2.3 covers the effect of using different search operators in OPERATORSGS, while Section 6.2.4 covers the effect of varying MAX-FLIPS in OPERATORSGS.

### 6.2.1 Application Networks: Methodology

The BNs investigated here, most of which are taken from Friedman’s Bayesian Network Repository, are denoted Munin1, Munin2, Mildew, and Water. (At the time of this writing, the location of the Bayesian Network Repository is <http://www.cs.huji.ac.il/labs/compbio/Repository/>.) Given our emphasis on additive utility and gain derived from MAXSAT, we focus on BNs with a substantial number of zeros; see Table 3.

The Munin BNs are medical BNs from the field of electromyography. The Munin1 BN has, among the BNs we consider here, the largest total clique tree size and the slowest compile and per-propagation execution time. The Water BN models the biological processes of water purification, while Mildew is for making dosage recommendations regarding the amount of fungicide needed to fight mildew in wheat. For each of these BNs, statistics for the CPT values along with the number of nodes and the average number of states per node are presented in Table 3.

For these application BNs, we first ran HUGIN. Following the methodology presented in Section 6.1.1, for each BN the resulting MPE probability  $\Pr(\mathbf{x}^*)$  was then used as utility limit  $U_{\min} = \Pr(\mathbf{x}^*)$  for SGS. This makes our comparison between algorithms more informative and reflects our discussion in Section 5.1.

### 6.2.2 Application Networks: SGS and Hugin

The goal of the experiments we report on now is to compare the performance of OPERATORSGS and HUGIN using BNs from applications. Results for HUGIN are shown in Table 4; the fill in weight triangulation heuristic was used. In Table 4, all HUGIN timing results are sample averages of 30 experiments.

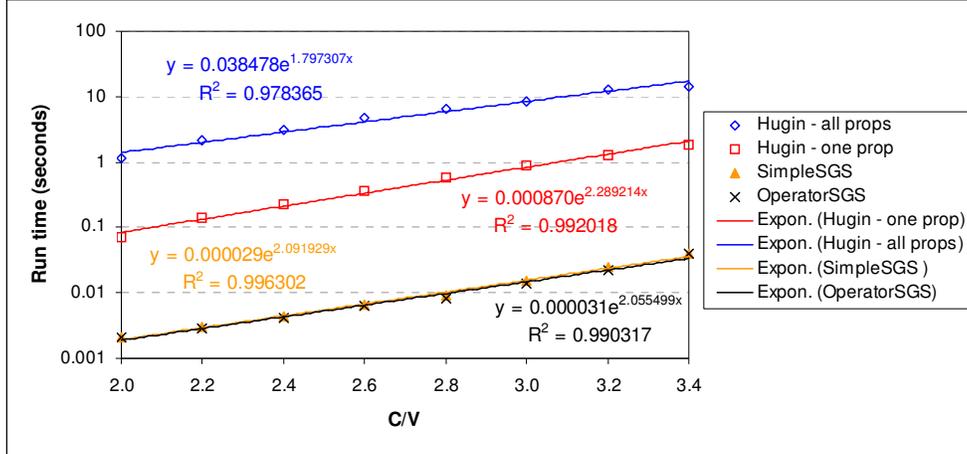


Figure 9: A comparison of the performance of HUGIN, SIMPLESGS, and OPERATORSGS as  $C/V$  is varied. In general, HUGIN needs multiple clique tree propagations in order to find an MPE, hence results for one propagation (“HUGIN - one prop”) as well as multiple propagations (“HUGIN - all props”) are shown. On average, both SIMPLESGS and OPERATORSGS are more than one order of magnitude faster than one HUGIN propagation, and more than two orders of magnitude faster than all HUGIN propagations.

| Bayesian network | Nodes $n$ | States avg. | Conditional probability table (CPT) values |                |                      |                      |                |       |
|------------------|-----------|-------------|--|----------------|----------------------|----------------------|----------------|-------|
|                  |           |             | 0  | $(0, 10^{-3}]$ | $(10^{-3}, 10^{-2}]$ | $(10^{-2}, 10^{-1}]$ | $(10^{-1}, 1)$ | 1     |
| Munin1           | 189       | 5.26        | 56.0%                                      | 3.64%          | 3.87%                | 7.99%                | 18%            | 10.4% |
| Munin2           | 1003      | 5.36        | 55.53%                                     | 4.74%          | 5.08%                | 9.46%                | 17.42%         | 7.74% |
| Mildew           | 35        | 17.6        | 93.06%                                     | 0.01%          | 0.75%                | 2.07%                | 3.93%          | 0.15% |
| Water            | 32        | 3.63        | 51.7%                                      | 0.04%          | 1.4%                 | 12.7%                | 31.8%          | 2.35% |

Table 3: Information on Bayesian networks from applications. Parameters characterizing the BNs are the number of nodes  $n$  and the average number of states per node. Conditional probability table values are also shown. These BNs have a large number of zeros in CPTs, thus for a large number of explanations  $\mathbf{x}$  have  $\Pr(\mathbf{x}) = U_B(\mathbf{x}) = 0$ . In these highly deterministic Bayesian networks, the additive approach turns out to reduce run time when used in stochastic local search.

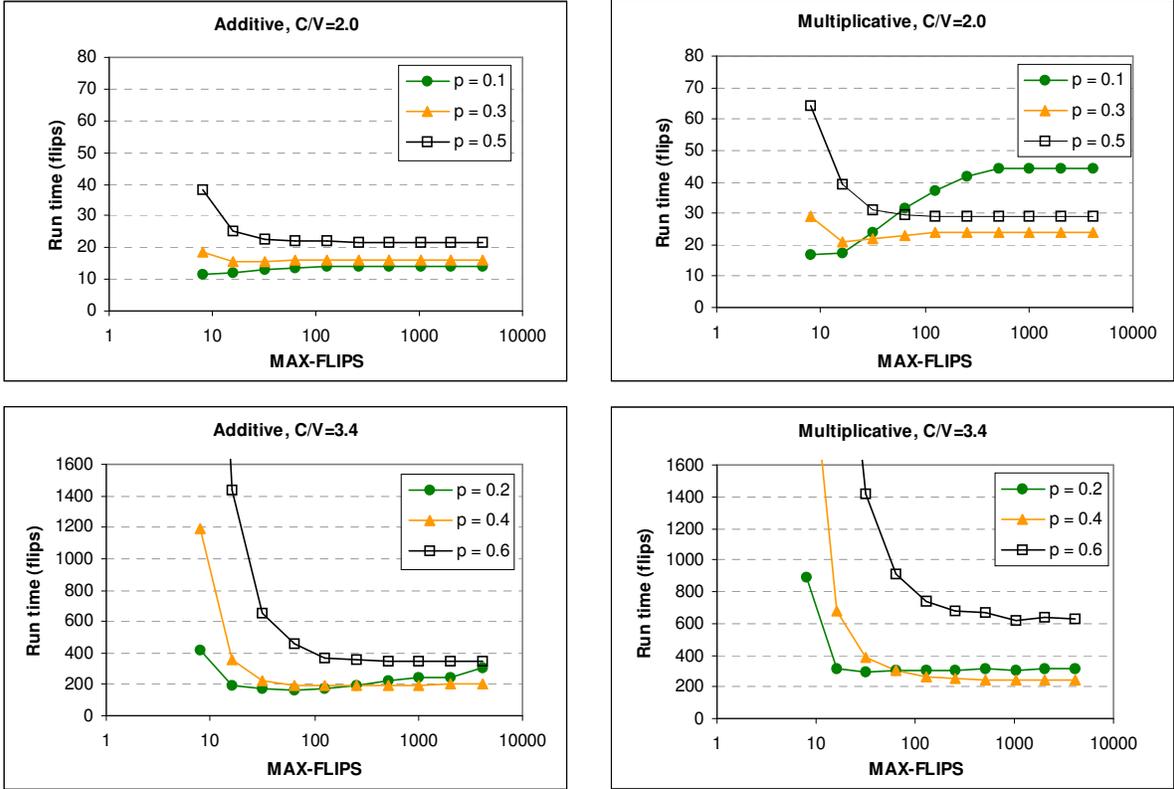


Figure 10: Empirical results for synthetic BNs under different experimental conditions for OPERATORSGS. The additive search operator AG is used in the column to the left, the multiplicative search operator MG is used in the column to the right. The top row is for  $C/V = 2.0$  and the bottom row is for  $C/V = 3.4$ . The restart parameter varies from  $\text{MAX-FLIPS} = 8$  to  $\text{MAX-FLIPS} = 4096$  as shown on the  $x$ -axis. The noise probability  $p = p_N$  is varied according to the labels. Each data point represents the sample mean of 10,000 runs: 100 runs for each of 100 BNs.

Results for OPERATORSGS are shown in Table 5. We use approximately optimal parameter settings for OPERATORSGS. Reflecting our analytical result in Theorem 58, OPERATORSGS uses homogenous initialization portfolios with the forward simulation (FS) and forward dynamic programming (FDP) initialization operators here. A more detailed analysis of OPERATORSGS can be found in Section 6.2.3 and Section 6.2.4.

Comparing the results for HUGIN and OPERATORSGS in Table 4 and Table 5 respectively, we make the following observations. First, OPERATORSGS is highly competitive with HUGIN for all these BNs. Particularly impressive is the performance of OPERATORSGS for Munin1 and Water, where the computation time of OPERATORSGS (measured in seconds) is substantially better than HUGIN’s. For Munin1, one HUGIN propagation takes 3,912.4 seconds while OPERATORSGS computation time is 0.02537 seconds. For Water, one HUGIN propagation takes 0.715 seconds while OPERATORSGS computation time is 0.0625 seconds. Second, for three of the BNs in Table 5, approximately optimal OPERATORSGS performance is obtained for  $p_A \geq 0.9$ , thus highlighting the power of our additive approach.

### 6.2.3 Application Networks: Different Search Operators

In this set of experiments with OPERATORSGS, we investigated different search operators by varying the search portfolio  $\Lambda_S$ . The purpose of these experiments was to investigate the effect of varying additive probability  $p_A$  and noise probability  $p_N$ . Appropriate use of noise has been found to be a powerful way to improve SLS performance [8, 17, 20, 33, 36, 49, 50]; however we know of no previous experiments where the effect of varying additive probability  $p_A$  is systematically investigated. To do so, we considered the following search portfolio.

**Definition 61 (Search portfolio)** *The search portfolio  $\Lambda_S(p_N, p_A)$  is parametrized by noise probability*

| BN     | HUGIN clique tree statistics |             |        | HUGIN execution times (sec) |         |         |         |
|--------|------------------------------|-------------|--------|-----------------------------|---------|---------|---------|
|        | Sum                          | Max         | Median | Compile                     | Execute | Total   | Prop.   |
| Munin1 | 384,620,599                  | 288,000,000 | 300    | 5099.5                      | 3,912.4 | 9,011.7 | 3,912.4 |
| Munin2 | 4,861,824                    | 504,000     | 320    | 2.089                       | 2.445   | 4.535   | 0.815   |
| Mildew | 9,566,232                    | 4,372,480   | 8,600  | 1.0671                      | 3.284   | 4.352   | 0.821   |
| Water  | 3,657,180                    | 1,769,472   | 2,187  | 0.616                       | 0.715   | 1.330   | 0.715   |

Table 4: Performance of HUGIN on application Bayesian networks. There are clique tree statistics, and the results in the Total column are for computing an MPE and are in seconds. The Compile and Execute columns give the time taken in each of these two phases.

| BN     | OPERATORSGS input parameters |  |           |       | Computation times |       |
|--------|------------------------------|--|-----------|-------|-------------------|-------|
|        | Initialization               | Search   | MAX-FLIPS | $p_A$ | Time (sec)        | Flips |
| Munin1 | $p_{FS} = 1.0$               | $p_{MN} = 0.05, p_{MG} = 0.05, p_{AG} = 0.45, p_{AN} = 0.45$     | 30        | 0.9   | 0.02537           | 25.37 |
| Munin2 | $p_{FS} = 1.0$               | $p_{AN} = 0.5, p_{AG} = 0.5$                                     | 70        | 1.0   | 1.385             | 245.5 |
| Mildew | $p_{FDP} = 1.0$              | $p_{MN} = 0.336, p_{MG} = 0.504, p_{AN} = 0.064, p_{AG} = 0.096$ | 200       | 0.16  | 0.2404            | 193.3 |
| Water  | $p_{FS} = 1.0$               | $p_{AN} = 0.1, p_{AG} = 0.9$                                     | 10        | 1.0   | 0.00625           | 61.71 |

Table 5: Performance of OPERATORSGS on application Bayesian networks. The values of approximately optimal input parameters, including initialization and search operators, as well as results for computing an MPE are shown. Note how  $p_A$ , or additive probability, is non-zero in all cases.

$p_N$  and additive probability  $p_A$  and is defined as

$$\Lambda_S(p_N, p_A) := \{(\text{AN}, p_{AP_N}), (\text{AG}, p_A(1 - p_N)), (\text{MN}, (1 - p_A)p_N), (\text{MG}, (1 - p_N)(1 - p_A))\}.$$

By introducing  $\Lambda_S(p_N, p_A)$ , we reduce the three-dimensional problem of varying probabilities in the portfolio  $\Lambda_S(p_{AN}, p_{AG}, p_{MN})$  from Definition 47 into a two-dimensional problem of varying probabilities  $p_N$  and  $p_A$ . Reflecting Theorem 58, a BN-specific optimal initialization algorithm — either forward simulation [15] or a randomized variant of the Viterbi algorithm [38, 56] — was used in  $\Lambda_I$  for each application BN. Forward simulation was used for Munin1, Munin2, and Water. Mildew was initialized using the forward variant of the randomized Viterbi algorithm. BN-specific values for MAX-FLIPS were also used: MAX-FLIPS = 30 for Munin1, MAX-FLIPS = 100 for Munin2, MAX-FLIPS = 10 for Water, and MAX-FLIPS = 200 for Mildew.

For each application BN, the probability of using an additive operator,  $p_A$ , was varied from  $p_A = 0$  to  $p_A = 1$  in increments of  $\Delta p_A = 0.1$ . Noise probability  $p_N$  was varied from  $p_N = 0.1$  to  $p_N = 0.9$  in increments of  $\Delta p_N = 0.2$ . We call these and similar run time curves generated by varying the probability  $p_A$  “additive response curves”, since they are similar to noise response curves generated by varying the probability  $p_N$ . Results, in the form of sample means and piecewise linear approximations for  $\hat{r}(p_A)$ , are reported in Figure 11. Each data point represents the mean of 1,000 experiments.

The case  $p_N = p_G = 0.5$  represents the situation where greedy and noise search operators are applied with equal probability. Further, at  $p_A = 0$  only the operators MN and MG are applied, while at  $p_A = 1$  only AN and AG are used. Previous experimental results are at  $p_A = 0$  if they only use multiplicative gain and not additive gain in their search heuristics; we will shortly see the disadvantage of this traditional approach.

Some of the main results found in Figure 11 are as follows. Our main novel result here is that varying  $p_A$  has, in many cases, a striking impact on SLS run time. The impact depends on the particular BN and varies dramatically with noise level  $p_N$ . Furthermore, by using  $p_A = 0$  as is employed in most traditional SLS approaches to BN inference, one may obtain far from optimal performance.

The shapes of all additive response curves in Figure 11 are all convex or close to convex, in line with previous results for noise response curves [36]. This is supporting our hypothesis of underlying convex additive response curves, meaning that an underlying expected hitting time  $h(p_A)$  is rational and convex.

In Figure 11, the shapes of the additive response curves for Water, Munin1, and Munin2 are quite similar: Optimal levels of  $p_A$ ,  $p_A^*$ , appear to be found at high values of  $p_A$ , and for such values the impact of noise is relatively minor. Combining high noise probability  $p_N$  and low additive probability  $p_A$  has, on the other hand, a very negative impact on run time. Examples of this can be found for  $p_N = 0.7$  and  $p_A = 0.1$  in Figure 11 for Water, Munin1, and Munin2. Robust performance, with respect to variations in  $p_N$ , is ensured by using a high value for  $p_A$ .

The shapes of the run time curves for Mildew are somewhat different than those for Water, Munin1, and Munin2. Here we find, in all cases, monotonic decreases in run times for small values of  $p_A$  and monotonic

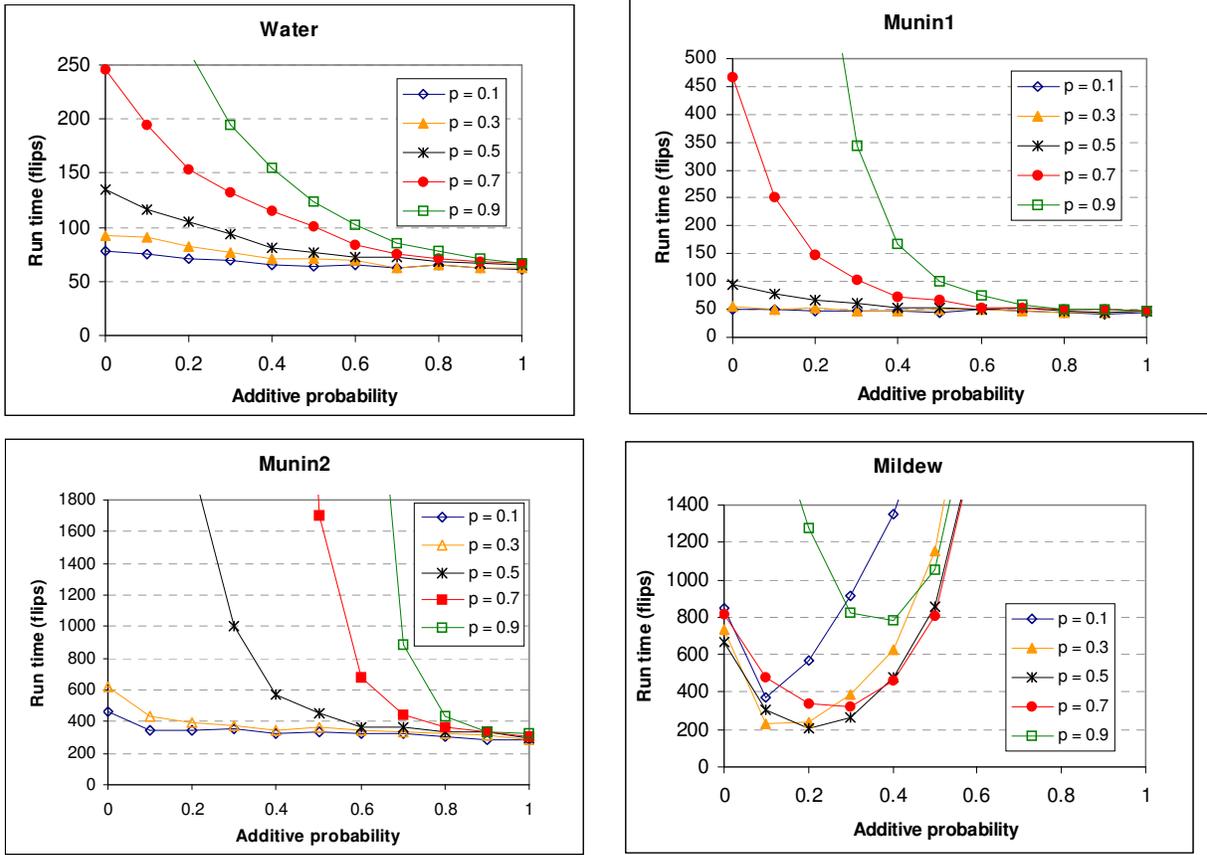


Figure 11: Empirical results for OPERATORSGS for the BNs Munin1, Munin2, Water, and Mildew under different experimental conditions. The additive probability  $p_A$ , or the probability of applying an additive search operator, varies from  $p_A = 0$  to  $p_A = 1$  as shown on the  $x$ -axis. The noise probability  $p = p_N$  is varied from  $p_N = 0.1$  to  $p_N = 0.9$  according to the labels. Each data point represents the sample mean of 1000 runs. Clearly, varying  $p_A$  and  $p_N$  has a significant impact on run time.

increases in run times for large values of  $p_A$ . These convex empirical curves suggest an optimal value for  $p_A^*$  that is increasing with  $p_N$ . A relatively high level of noise improves the run time on Mildew;  $p_N = 0.5$  gives best results in these experiments, with an average run time of 209.8 flips at  $\hat{p}_A^* = 0.2$ .

#### 6.2.4 Application Networks: Impact of Restart

In this section we investigate the impact, on OPERATORSGS performance, of varying the restart parameter MAX-FLIPS along with  $p_A$  and  $p_N$ . We experiment with Mildew since it has the most challenging (and interesting) additive response curve in Figure 11. Similar to in Section 6.2.3, we employ the search portfolio  $\Lambda_S(p_N, p_A)$ . In order to further investigate the region in which minimal average run time occurred in Figure 11, we varied additive probability  $p_A$  from  $p_A = 0.1$  to  $p_A = 0.3$  in increments of  $\Delta p_A = 0.02$ . Noise probability  $p_N$  was varied from  $p_N = 0.3$  to  $p_N = 0.6$  in increments of  $\Delta p_N = 0.1$ .

Results, in the form of sample means and piecewise linear approximations for  $\hat{r}(p_A)$ , are reported in Figure 12. Each data point represents the mean of 1,000 runs. In each panel of Figure 12, varying MAX-FLIPS has a substantial impact. Generally, MAX-FLIPS = 50 gives the fastest run time. Further, careful optimization of  $p_A$ ,  $p_N$ , and MAX-FLIPS, gives significant benefit, and the approximate performance optimum is at  $p_A = 0.17$ ,  $p_N = 0.4$ , and MAX-FLIPS = 50. We also note that most of these curves are convex or almost convex, lending support to a hypothesis that the underlying analytical curves are convex rational functions.

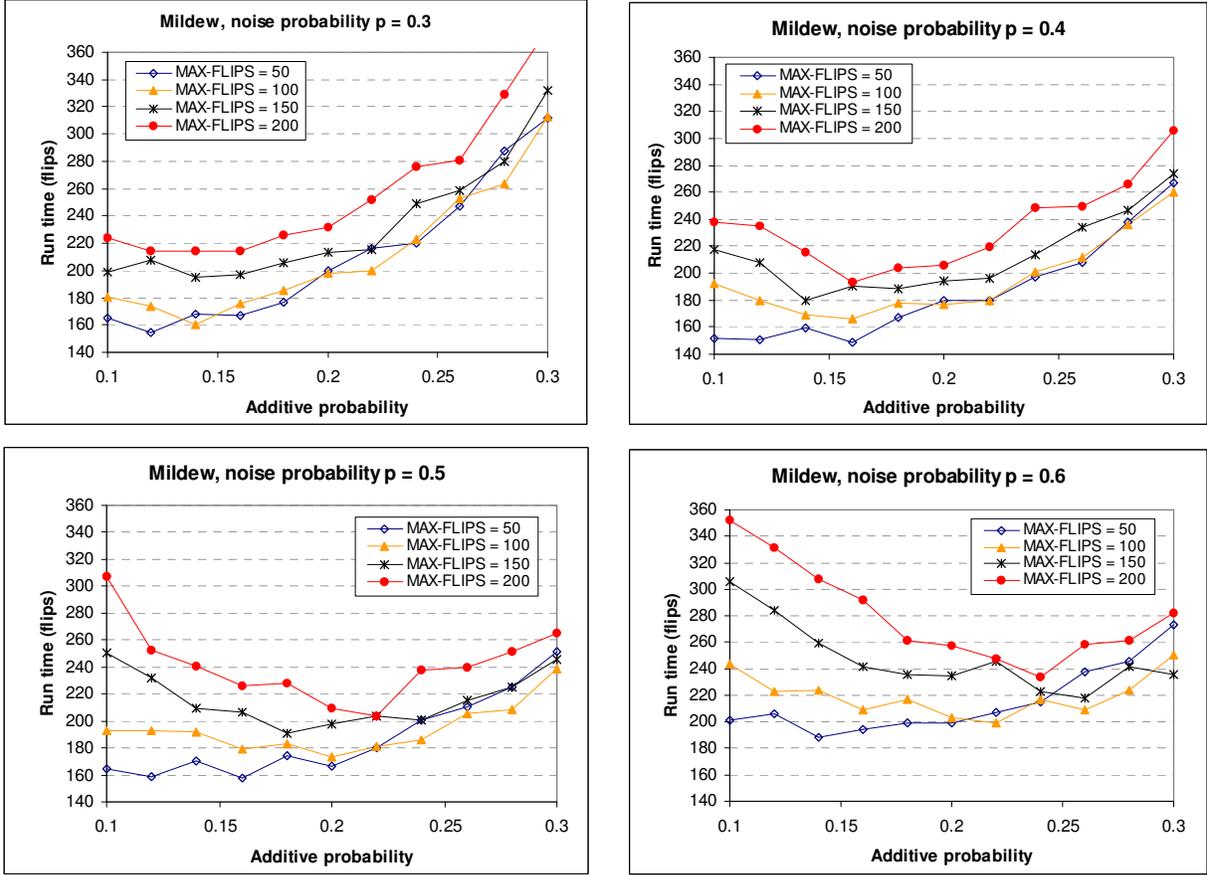


Figure 12: Empirical results for the Mildew BN under different experimental conditions for OPERATORSGS. The probability  $p_A$  of applying an additive search operator, varying from  $p_A = 0.1$  to  $p_A = 0.3$ , is shown on the  $x$ -axis. The noise probability  $p = p_N$  is systematically varied from  $p_N = 0.3$  (top left) to  $p_N = 0.6$  (bottom right), and the restart parameter MAX-FLIPS is varied according to the labels. Each data point represents the sample mean of 1,000 runs. Clearly, varying  $p_A$ ,  $p_N$ , and MAX-FLIPS has a significant impact on run time.

## 7 Conclusion and Future Work

Stochastic local search has proven to be powerful algorithms for finding satisfying assignments in satisfiability (SAT) instances [14, 21, 33, 48, 50, 51]. In this article, we discuss the generalization of stochastic local search from the logical case of SAT to the probabilistic case of Bayesian networks (BNs). The main contributions of this article are our portfolio approach and our additive measure of utility and gain. We have incorporated these techniques in the SGS stochastic local search approach, which heuristically computes a most probable explanation (MPE) in a Bayesian network. SGS combines greedy search, stochastic search, and stochastic initialization algorithms. We have presented two SGS algorithms, namely simple SGS (SIMPLESGS) and operator-based SGS (OPERATORSGS); the latter is more flexible and general than the former due to its initialization and search portfolios. The initialization and search portfolios of OPERATORSGS contain operators, which are algorithms for initialization or search respectively, along with probabilities that control their selection and execution. The initialization algorithms suggest good starting points to the stochastic local search based on a complete solution of a randomly simplified BN. The initialization algorithms perform in time linear in the number of nodes in the BN. Our OPERATORSGS framework, for which we provide both theoretical and experimental results, extends previous research on hybrid algorithms for MPE and MAP computation [26, 41, 42] as well as other computational problems [12, 13, 24, 25, 37, 57].

We carefully formalized the concepts of SLS utility and gain in the context of MPE computation, and emphasized our additive approach. The additive measure, for which we provided several new results, is a generalization of GSAT’s and WALKSAT’s utility and gain measures to the probabilistic setting. Application

BNs often have many deterministic nodes, and the additive measure turned out to be very powerful in such BNs. This measure has, to our knowledge, not been extensively studied in previous research on MPE computation using stochastic local search, and we also investigated how it relates to traditional multiplicative gain as used in other SLS algorithms for MPE. In addition, we introduced a novel Markov chain model, augmented random walk, where the behavior of OPERATORSGS’s initialization and search operators is explicitly represented as Markov chain states. Using a probabilistic argument, we also showed that the optimal OPERATORSGS initialization portfolio is homogenous.

Two sets of experiments have been conducted. We have shown that SGS outperforms HUGIN on hard BNs constructed from satisfiability instances, and also outperforms HUGIN on application BNs with a high degree of determinism. In one set of experiments, we have utilized an approach to constructing synthetic Bayesian networks of varying hardness [39]. This approach helps in developing an understanding of the suitability of different algorithms for different classes of BNs. In these experiments, we found that SGS was quite effective on hard instances, and in particular the algorithm consistently performed orders of magnitude better than the state-of-the-art exact algorithm HUGIN as the  $C/V$ -ratio was varied.

Experiments with OPERATORSGS on application BNs have also been done. We have found that the algorithm is quite effective on these networks too, and performs comparably to HUGIN. Key factors in the success of OPERATORSGS are the initialization operators, the additive measure, and the approach of using a portfolio of operators for computation. OPERATORSGS’s varying approximately optimal parameter values, including values for the selection probabilities of the initialization and search operators, highlights the importance of the portfolio approach to stochastic local search.

Areas for current and future work include the following. First, the important role of portfolios and different measures of gain, especially in application networks, highlights the opportunity of adaptively tuning the probabilities (similar to noise adaptation [8, 17, 33]) of portfolio operators when computing MPE or MAP for a given BN. Both analytical and experimental work on portfolio adaptation would be of great interest. Second, in local search, an essential question is how to escape from local minima. In this article, our answer has been the use of noise, which is a local escape mechanism. Another approach is to use crossover from genetic algorithms. In the MPE or MAP context, crossover would take place between two explanations, thus providing a more global escape mechanism than noise. More generally, there is potential for additional hybridization, for example combining stochastic local search with clique tree clustering. Third, additional investigation of when to terminate, other than by using a Las Vegas approach or time limits, is needed in order to increase the easy-of-use of the SGS approach in applications. Fourth, there is an opportunity to generalize additional successful SAT and constraint satisfaction research results, similar to what we have done for the additive approach above, into the setting of Bayesian networks.

## Acknowledgments

This material is based upon work supported in part by NASA under awards NCC2-1426 and NNA07BB97C. 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.

Vadim Bulitko, David Fried, Song Han, and Brent Spillner are acknowledged for comments on early presentations of this research. David Fried and Song Han are also acknowledged for their co-development of the Raven software, with which much of the experimental work reported here was performed.

## References

- [1] A. M. Abdelbar and S. M. Hedetnieme. Approximating MAPs for belief networks is NP-hard and other theorems. *Artificial Intelligence*, 102:21–38, 1998.
- [2] S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen. HUGIN—a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 2, pages 1080–1085, Detroit, MI, August 1989.
- [3] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.
- [4] F. G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

- [5] A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.
- [6] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, 125(1-2):93–118, 2001.
- [7] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987.
- [8] A. Fukunaga, G. Rabideau, and S. Chien. Robust local search for spacecraft operations using adaptive noise. In *Proceedings of the 4th International Workshop on Planning and Scheduling for Space (IWSS-04)*, Darmstadt, Germany, 2004.
- [9] S. F. Galan and O. J. Mengshoel. Constraint handling using tournament selection: Abductive inference in partly deterministic Bayesian networks. *Accepted for publication, Evolutionary Computation*, 2008.
- [10] R. G. Gallager. Low density parity check codes. *IRE Transactions on Information Theory*, 8:21–28, Jan 1962.
- [11] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):721–741, 1984.
- [12] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1–2):43–62, 2001.
- [13] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437, Madison, WI, 1998.
- [14] P. W. Gu, J. Purdom, J. Franco, and B. W. Wah. *Satisfiability Problem: Theory and Applications*, chapter Algorithms for the Satisfiability SAT Problem: A Survey, pages 19–152. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1997.
- [15] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence 2*, pages 149–163. Elsevier, Amsterdam, 1988.
- [16] J. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1996.
- [17] H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 655–660, Edmonton, Alberta, Canada, 2002.
- [18] H. H. Hoos. A mixture-model for the behaviour of SLS algorithms for SAT. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 661–667, Edmonton, Alberta, Canada, 2002.
- [19] H. H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112(1-2):213–232, 1999.
- [20] H. H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.
- [21] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, 2005.
- [22] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and D. Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 235–244, Seattle, WA, 2001.
- [23] E. J. Horvitz, H. J. Suermondt, and G. F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, pages 182–193, Windsor, Ontario, 1989. Morgan Kaufmann.
- [24] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275(3):51–54, 1997.

- [25] F. Hutter, H. H. Hoos, and T. Stützle. Efficient stochastic local search for MPE solving. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 169–174, Edinburgh, Scotland, 2005.
- [26] K. Kask and R. Dechter. Stochastic local search for Bayesian networks. In *Proceedings Seventh International Workshop on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, Jan 1999. Morgan Kaufmann.
- [27] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*, pages 511–518. Morgan Kaufmann, San Francisco, CA, 2000.
- [28] S. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B*, 50(2):157–224, 1988.
- [29] Z. Li and B. D’Ambrosio. Efficient inference in Bayes nets as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1):55–81, 1994.
- [30] R. Lin, A. Galper, and R. Shachter. Abductive inference using probabilistic networks: Randomized search techniques. Technical Report KSL-90-73, Knowledge Systems Laboratory, Stanford, CA, November 1990.
- [31] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2002.
- [32] A. Madsen. Computing MPEs in Hugin. Personal communication, April 2003.
- [33] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 321–326, Providence, RI, 1997.
- [34] O. J. Mengshoel. *Efficient Bayesian Network Inference: Genetic Algorithms, Stochastic Local Search, and Abstraction*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, April 1999.
- [35] O. J. Mengshoel. Macroscopic models of clique tree growth for Bayesian networks. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*, pages 1256–1262, Vancouver, British Columbia, 2007.
- [36] O. J. Mengshoel. Understanding the role of noise in stochastic local search: Analysis and experiments. *Accepted for publication, Artificial Intelligence Journal*, 2007.
- [37] 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, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, February 2000.
- [38] O. J. Mengshoel, D. Roth, and D. C. Wilkins. Initialization and restart in stochastic local search: Computing a most probable explanation in Bayesian networks. *Submitted for publication*, 2007.
- [39] O. J. Mengshoel, D. C. Wilkins, and D. Roth. Controlled generation of hard and easy Bayesian networks: Impact on maximal clique tree in tree clustering. *Artificial Intelligence*, 170(16-17):1137–1174, 2006.
- [40] D. Mitchell, B. Selman, and H. J. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, San Jose, CA, 1992.
- [41] J. D. Park and A. Darwiche. Approximating MAP using local search. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 403–410, Seattle, WA, 2001.
- [42] J. D. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research (JAIR)*, 21:101–133, 2004.
- [43] A. J. Parkes and J. P. Walser. Tuning local search for satisfiability testing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 356–362, Portland, OR, 1996.

- [44] J. Pearl. A constraint - propagation approach to probabilistic reasoning. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 357–369. Elsevier, Amsterdam, Netherlands, 1986.
- [45] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
- [46] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.
- [47] Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: A dynamic programming approach. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 573–586, Ithaca, NY, 2002.
- [48] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. *Artificial Intelligence*, 132(2):121–150, 2001.
- [49] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 290–295, Chambéry, France, 1993.
- [50] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, Seattle, WA, 1994.
- [51] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, 1992.
- [52] W. F. Sharpe. *Portfolio Theory and Capital Markets*. McGraw-Hill, New York, 1970.
- [53] P. P. Shenoy. A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 5(3):383–411, 1989.
- [54] E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.
- [55] M.A. Shwe, B. Middleton, D.E. Heckerman, M. Henrion, E.J. Horvitz, H.P. Lehmann, and G.F. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base: I. The probabilistic model and inference algorithms. *Methods of Information in Medicine*, 30(4):241–255, 1991.
- [56] A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.
- [57] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla-07: The design and analysis of an algorithm portfolio for sat. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-07)*, 2007.
- [58] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.