

**Loopy Substructural Local Search for the Bayesian  
Optimization Algorithm**

**Claudio F. Lima, Martin Pelikan,  
Fernando G. Lobo, and David E. Goldberg  
IlliGAL Report No. 2010003  
March, 2010**

Illinois Genetic Algorithms Laboratory  
University of Illinois at Urbana-Champaign  
117 Transportation Building  
104 S. Mathews Avenue Urbana, IL 61801  
Office: (217) 333-2346  
Fax: (217) 244-5705

# Loopy Substructural Local Search for the Bayesian Optimization Algorithm

Claudio F. Lima<sup>1</sup>, Martin Pelikan<sup>2</sup>, Fernando G. Lobo<sup>1</sup>, David E. Goldberg<sup>3</sup>

<sup>1</sup>Department of Electronics and Computer Science Engineering  
University of Algarve, Campus de Gambelas, 8000-117 Faro, Portugal  
{clima.research,fernando.lobo}@gmail.com

<sup>2</sup>Missouri Estimation of Distribution Algorithm Laboratory (MEDAL)  
Department of Mathematics and Computer Science  
University of Missouri at St. Louis, St. Louis MO 63121  
pelikan@cs.umsl.edu

<sup>3</sup>Illinois Genetic Algorithms Laboratory (IlliGAL)  
Department of Industrial and Enterprise Systems Engineering  
University of Illinois at Urbana-Champaign, Urbana IL 61801  
deg@illinois.edu

March 25, 2010

## Abstract

This paper presents a local search method for the Bayesian optimization algorithm (BOA) based on the concepts of substructural neighborhoods and loopy belief propagation. The probabilistic model of BOA, which automatically identifies important problem substructures, is used to define the topology of the neighborhoods explored in local search. On the other hand, belief propagation in graphical models is employed to find the most suitable configuration of conflicting substructures. The results show that performing loopy substructural local search (SLS) in BOA can dramatically reduce the number of generations necessary to converge to optimal solutions and thus provides substantial speedups.

## 1 Introduction

The Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantú-Paz, 1999; Pelikan, 2005) replaces the standard crossover and mutation operators of evolutionary algorithms (EAs) by building a probabilistic model of promising solutions and sampling from the corresponding probability distribution. This feature allows BOA and other estimation of distribution algorithms (EDAs) (Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002) to automatically identify the problem decomposition and important problem substructures, leading to superior performance for many problems when compared to EAs with fixed, problem-independent variation operators.

Although EDAs are effective at exploring the search space to find promising regions, they inherit a common drawback from traditional EAs: slower convergence to optimal solutions when compared

with appropriate local searchers that start the search within the basin of attraction of the optima. This observation has led to the combination of EAs with local search methods known as hybrid EAs or memetic algorithms (Moscato, 1989; Hart, 1994). In this context EDAs are no exception and many applications in real-world optimization have been accomplished with the help of some sort of local search. However, systematic methods for hybridizing and designing competent global and local-search methods that automatically identify the problem decomposition and important problem substructures are still scarce. For instance, the probabilistic models of EDAs contain useful information about the underlying problem structure that can be exploited to speedup the convergence of EDAs to optimal solutions.

This paper makes use of the concept of substructural neighborhoods (Sastry & Goldberg, 2004; Lima, Pelikan, Sastry, Butz, Goldberg, & Lobo, 2006)—where the structure of the neighborhoods is defined by learned probabilistic models—to perform local search in BOA. The local search method proposed is inspired on loopy belief propagation, that is often used for obtaining the most probable state of a Bayesian network. To guide the propagation of beliefs, we use a surrogate fitness model that also relies on substructural information. Experiments are performed for a boundedly difficult problem with both non-overlapping and overlapping subproblems. The results show that incorporating loopy substructural local search (SLS) in BOA leads to a significant reduction in the number of generations, providing relevant speedups in terms of number of evaluations.

The next section briefly reviews the Bayesian optimization algorithm, while Section 3 details the notion of substructural local search in EDAs. Section 4 introduces belief propagation in graphical models and its potential for function optimization. A new substructural local search method based on loopy belief propagation is then presented in Section 5. Section 6 presents and discusses empirical results. The paper ends with a brief summary and conclusions.

## 2 Bayesian Optimization Algorithm

Estimation of distribution algorithms (Larrañaga & Lozano, 2002; Pelikan, 2005) replace traditional variation operators of EAs by building a probabilistic model of promising solutions and sampling the corresponding probability distribution to generate the offspring population. The Bayesian optimization algorithm (Pelikan, Goldberg, & Cantú-Paz, 1999; Pelikan, 2005) uses Bayesian networks as the probabilistic model to capture important problem regularities.

BOA starts with an initial population of candidate solutions that is usually randomly generated. In each iteration, selection is performed to obtain a population of promising solutions. This population is then used to build the probabilistic model for the current generation. After the model structure is learned and its parameters estimated, the offspring population is generated by sampling from the distribution of modeled individuals. The new solutions are then evaluated and incorporated into the original population by using any standard replacement method. The next iteration proceeds again from the selection phase until some stopping criterion is satisfied. Here, we use a simple replacement scheme where new solutions fully replace the original population.

### 2.1 Modeling Variable Interactions in BOA

Bayesian networks (Pearl, 1988) are powerful graphical models that combine probability theory with graph theory to encode probabilistic relationships between variables of interest. A Bayesian network is defined by a structure and corresponding parameters. The structure is represented by a directed acyclic graph where the nodes correspond to the variables of the data to be modeled and the edges correspond to conditional dependencies. The parameters are represented by the conditional

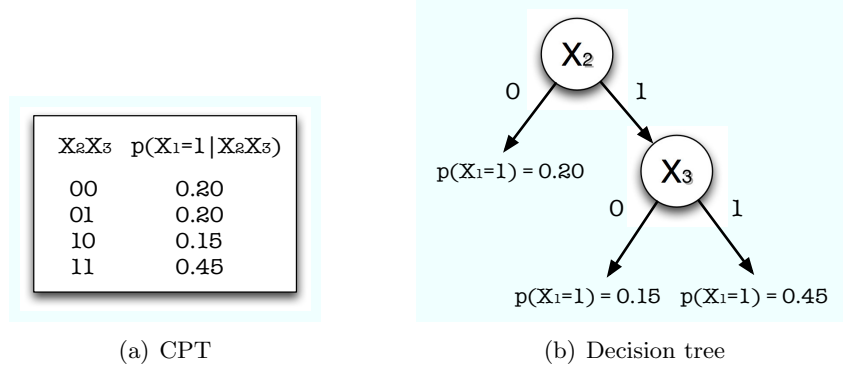


Figure 1: Example of a conditional probability table for  $p(X_1|X_2X_3)$  using the (a) traditional representation and a (b) decision tree. The decision tree allows a more efficient and flexible representation of the conditional probabilities.

probabilities for each variable given any instance of the variables that this variable depends on. More formally, a Bayesian network encodes the following joint probability distribution,

$$p(X) = \prod_{i=1}^{\ell} p(X_i|\Pi_i), \quad (1)$$

where  $X = (X_1, X_2, \dots, X_\ell)$  is a vector of all the variables of the problem,  $\Pi_i$  is the set of *parents* of  $X_i$  (nodes from which there exists an edge to  $X_i$ ), and  $p(X_i|\Pi_i)$  is the conditional probability of  $X_i$  given its parents  $\Pi_i$ .

In BOA, both the structure and the parameters of the probabilistic model are searched and optimized to best fit the data (set of promising solutions). To learn the most adequate structure for the Bayesian network a greedy algorithm is usually used for a good compromise between search efficiency and model quality.

The parameters of a Bayesian network are represented by a set of conditional probability tables (CPTs) specifying the conditional probabilities for each variable given all possible instances of the parent variables  $\Pi_i$ . Alternatively, these conditional probabilities can be stored in the form of local structures such as decision trees or decision graphs, allowing a more efficient and flexible representation of local conditional distributions. Figure 1 shows a simple example. In this work, decision trees are used to encode the parameters of the Bayesian network.

## 2.2 Modeling Fitness in BOA

Pelikan and Sastry (2004) extended the Bayesian networks used in BOA to encode a surrogate fitness model that is used to estimate the fitness of a proportion of the population, thereby reducing the total number of function evaluations. For each possible value  $x_i$  of every variable  $X_i$ , an estimate of the marginal fitness contribution of a subsolution with  $X_i = x_i$  is stored for each instance  $\pi_i$  of  $X_i$ 's parents  $\Pi_i$ . Therefore, in the binary case, each row in the CPT is extended by two additional entries. The fitness of an individual can then be estimated as

$$f_{est}(X_1, X_2, \dots, X_\ell) = \bar{f} + \sum_{i=1}^{\ell} (\bar{f}(X_i|\Pi_i) - \bar{f}(\Pi_i)), \quad (2)$$

where  $\bar{f}$  is the average fitness of all solutions used to learn the surrogate,  $\bar{f}(X_i|\Pi_i)$  is the average fitness of solutions with  $X_i$  and  $\Pi_i$ , and  $\bar{f}(\Pi_i)$  is the average fitness of all solutions with  $\Pi_i$ .

Fitness information can also be incorporated in Bayesian networks with decision trees or graphs in a similar way. In this case, the average fitness of each instance for every variable must be stored in every leaf of the decision tree or graph. The fitness averages in each leaf are now restricted to solutions that satisfy the condition specified by the path from the root of the tree to the leaf.

### 3 Substructural Local Search

One of the key requirements for designing an efficient mutation operator is to ensure that it searches in the correct neighborhood. This is often accomplished by exploiting and incorporating domain- or problem-specific knowledge in the design of neighborhood operators. While these neighborhood operators are designed for a particular search problem, oftentimes on an ad-hoc basis, they do not generalize their efficiency beyond a small number of applications. On the other hand, simple bitwise hillclimbers are frequently used as local search methods with more general applicability, providing inferior but still competitive results, especially when combined with population-based search procedures. Clearly, there is a tradeoff between generalization and efficiency for neighborhood operators with fixed structure. Therefore, it is important to study systematic methods for designing neighborhood operators that can solve a broad class of search problems.

The exploration of neighborhoods defined by the probabilistic models of EDAs is an approach that exploits both the underlying problem structure while not losing the generality of application. The resulting mutation operators explore a more *global*, problem-dependent neighborhood than traditional local, purely representation-dependent search procedures. Sastry and Goldberg (2004) showed that a selectomutative algorithm that performs hillclimbing in the substructural space can successfully solve problems of bounded difficulty with subquadratic scalability.

Lima, Pelikan, Sastry, Butz, Goldberg, and Lobo (2006) introduced the concept of substructural neighborhoods to the Bayesian optimization algorithm. The parental neighborhood, which considers all possible values for a given variable  $X_i$  and its corresponding parents  $\Pi_i$ , was adopted to perform local search in the subsolution space (Lima, Pelikan, Sastry, Butz, Goldberg, & Lobo, 2006). The substructural local search is performed for a proportion of the population in BOA to speedup convergence to good solutions, as in traditional hybrid EAs or memetic algorithms (Moscato, 1989; Hart, 1994). The SLS procedure essentially explores all substructural neighborhoods in a random order, choosing the best subsolution for each neighborhood according to  $f(X_i, \Pi_i)$ .

Although this type of SLS succeeds in reducing the number of generations necessary to converge to optimal solutions for problems of bounded difficulty, the results do not carry over for problems with highly conflicting subsolutions (Lima, 2009). This will become clear from the results presented in Section 6.

### 4 Loopy Belief Propagation

Belief propagation (BP) (Pearl, 1988) is a method for performing exact and approximate inference in graphical models, which has enjoyed increasing popularity over the last years. Although BP has been reinvented several times in different fields (Kschischang, Frey, & Loeliger, 2001; Mooij, 2008), it is mainly applied to two tasks: (1) obtaining marginal probabilities for some of the variables, or (2) finding the most probable explanation or instance for the graphical model. These two versions are known as the sum-product and max-product algorithms.

BP algorithms are typically applied to factor graphs (Kschischang, Frey, & Loeliger, 2001), which can be seen as a unifying representation for both Bayesian networks and Markov networks (Kindermann & Snell, 1980). Factor graphs explicitly express the factorization structure

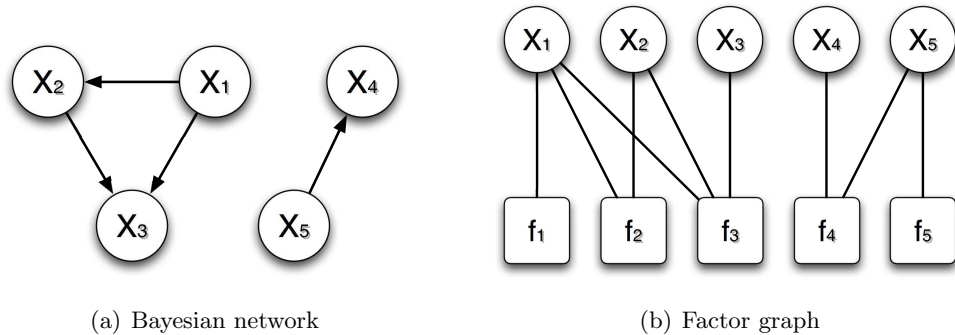


Figure 2: Example of a (a) Bayesian network and its equivalent representation as a (b) factor graph. Note that each factor corresponds to a conditional probability table, therefore the number of variable and factor nodes is the same.

of the corresponding probability distribution. Consider a function  $g(X)$  whose joint probability distribution can be factorized in several local functions, such that

$$g(x_1, x_2, \dots, x_\ell) = \frac{1}{Z} \prod_{I \in \mathcal{F}} f_I(x_{N_I}), \quad (3)$$

where  $Z = \sum_x \prod_{I \in \mathcal{F}} f_I(x_{N_I})$  is a normalization constant,  $I$  is the factor index,  $N_I$  is the subset of variable indices associated with factor  $I$ , and factor  $f_I$  is a nonnegative function. Note that for a Bayesian network each factor corresponds to a conditional probability table.

A factor graph is a bipartite graph consisting of variable nodes  $i \in \mathcal{V}$ , factor nodes  $I \in \mathcal{F}$ , and an undirected edge  $\{i, I\}$  between  $i$  and  $I$  if and only if  $i \in N_I$ , meaning that factor  $f_I$  depends on  $x_i$ . Factor nodes are typically represented as squares and variable nodes as circles.

An example of a Bayesian network, along with the corresponding representation as a factor graph, is presented in Figure 2. The factor graph represents the following factorization

$$g(x_1, x_2, x_3, x_4, x_5) = \frac{1}{Z} f_1(x_1) f_2(x_1, x_2) f_3(x_1, x_2, x_3) f_4(x_4, x_5) f_5(x_5). \quad (4)$$

If one substitutes the factor functions by the corresponding conditional probabilities, the joint probability distribution of a Bayesian network is obtained.

When BP is applied to cyclic graphs it is often referred as *loopy* belief propagation (LBP). In this situation, the convergence to exact beliefs can not be guaranteed as it is for acyclic graphs (without loops). However, empirical studies have shown that good approximate beliefs can be obtained for several domains (see Mooij (2008) for an extensive list).

The inference performed by BP is done by message-passing between the nodes of the graphical model. Each node sends and receives messages from its neighbors until a stable state is reached. The outgoing messages are functions of incoming messages at each node. This iterative process is repeated according to some schedule that describes the sequence of message updates in time (Mooij, 2008).

When performing BP in factor graphs, there are two types of messages: messages  $m_{I \rightarrow i}$ , sent from factors  $I \in \mathcal{F}$  to neighboring variables  $i \in N_I$ , and messages  $m_{i \rightarrow I}$ , sent from variables  $i \in \mathcal{V}$  to neighboring factors  $I \in N_i$ . The new messages  $m'$  are given in terms of the incoming messages by the following update rules:

$$m'_{i \rightarrow I}(x_i) = \prod_{J \in N_i \setminus I} m_{J \rightarrow i}(x_i) \quad \forall i \in \mathcal{V}, \quad \forall I \in N_i, \quad (5)$$

$$m'_{I \rightarrow i}(x_i) = \sum_{x_{N_I \setminus i}} f_I(x_{N_I}) \prod_{j \in N_I \setminus i} m_{j \rightarrow I}(x_j) \quad \forall I \in \mathcal{F}, \quad \forall i \in N_I, \quad (6)$$

$$m'_{I \rightarrow i}(x_i) = \max_{x_{N_I \setminus i}} \left( f_I(x_{N_I}) \prod_{j \in N_I \setminus i} m_{j \rightarrow I}(x_j) \right) \quad \forall I \in \mathcal{F}, \quad \forall i \in N_I, \quad (7)$$

where  $N_i \setminus I$  represents the set of neighboring factor nodes of variable node  $i$  excluding node  $I$ ,  $N_I \setminus i$  represents the set of neighboring variable nodes of factor node  $I$  excluding node  $i$ , and  $x_{N_I \setminus i}$  stands for a possible combination of values that all variables but  $X_i$  in  $X_{N_I}$  can take while variable  $X_i$  remains instantiated with value  $x_i$ .

For the sum-product algorithm, equations 5 and 6 are used, while for the max-product algorithm equations 5 and 7 should be used instead. When messages stop changing over time, the BP algorithm has converged and marginal functions (sum-product) or max-marginals (max-product) can be obtained as the normalized product of all messages received for  $X_i$ :

$$g_i(x_i) \propto \prod_{I \in N_i} m_{I \rightarrow i}(x_i). \quad (8)$$

For the max-product algorithm, the most probable configuration (MPC) for each variable  $X_i$  is obtained by assigning the value associated with the highest probability at each max-marginal.

When applying BP algorithms, three types of parameters need to be defined (Mendiburu, Santana, Lozano, & Bengoetxea, 2007): message scheduling, stopping criteria, and initial settings. For more details about parameter setting in BP algorithms the reader is referred elsewhere (Mendiburu, Santana, Lozano, & Bengoetxea, 2007; Mooij, 2008; Pearl, 1988; Kschischang, Frey, & Loeliger, 2001).

#### 4.1 Message-Passing Techniques for Optimization

Several message-passing algorithms have been developed and applied to different optimization problems. The idea is to associate a probability distribution to the function to be optimized in such a way that the most probable value of the distribution is reached for the solution(s) that optimize the function (Mendiburu, Santana, Lozano, & Bengoetxea, 2007). Recent applications have been used to solve satisfiability problems (Braunstein, Mezard, & Zecchina, 2005; Feige, Mossel, & Vilenchik, 2006) and for finding the maximum weight matching in a bipartite graph (Bayati, Shah, & Sharma, 2008).

Recognizing the potential of BP for Bayesian EDAs, Mendiburu, Santana, and Lozano (2007) introduced belief propagation to the estimation of Bayesian networks algorithm (EBNA) (Etxeberria & Larrañaga, 1999), which is very similar to BOA. The idea is to combine probabilistic logic sampling (PLS) (Henrion, 1988) with loopy belief propagation to sample the offspring population. Specifically,  $n - 1$  individuals are sampled through PLS and the remaining individual is instantiated with the most probable configuration for the current Bayesian network. The Bayesian network is mapped into an equivalent factor graph so that the max-product algorithm can be applied to obtain the new individual. Although the authors concluded that this modification allowed an improvement in the optimization capabilities of EBNA, the results fail to demonstrate great improvements both in solution quality and number of function evaluations required (Mendiburu, Santana, & Lozano, 2007).

While the calculation of the most probable configuration of the Bayesian network at each generation is expected to generate a good solution, its relative quality is strongly dependent upon the current stage of the search. It seems clear that high-quality solutions can only be generated by LBP

when BOA starts focusing on more concrete regions of the search space. On the other hand, instead of performing loopy belief propagation based on the conditional probabilities, substructural fitness information can be used for the factor nodes. Although probabilities represent likely substructures, using the associated fitness provides more direct information when looking for solutions with high quality. That is what is proposed in the next section.

## 5 BOA with Loopy Substructural Local Search

This section describes a substructural local searcher based on loopy belief propagation that can be incorporated in BOA. The resulting method which is named as loopy substructural local search (loopy SLS) uses substructural fitness information  $\bar{f}(X_i, \Pi_i)$  to guide the max-product algorithm in finding the MPC, which is the solution that is expected to maximize fitness based on the contribution of its substructures.

Regarding the parameterization of BP, the maximum number of iterations that the algorithm is allowed to run is set to  $2\ell$ , while the allowed difference when comparing two messages is of at least  $10^{-6}$  (otherwise messages are considered to be similar). These are typical parameter values from the literature. The update schedule used is the maximum residual updating (Elidan, McGraw, & Koller, 2006), which calculates all residuals (difference between updated and current messages) and updates only the message with the largest residual. Consequently, only the residuals that depend on the updated message need to be recalculated.

If the factor graph is acyclic, BP will converge towards a unique fixed point within a finite number of iterations, while the beliefs can be shown to be exact. However, if the factor graph contains loops, which is the typical situation when translating a Bayesian network from BOA, the result can be only interpreted as an approximate solution. Therefore, two different situations can arise when performing loopy SLS: (1) the max-product algorithm might not converge to a stable point and (2) even in case of convergence, the solution can present ties for certain positions.

If the LBP algorithm does not converge to a stable state, the configuration found after the maximum number of iterations ( $2\ell$ ) is still used as the result of loopy SLS. While this solution is not guaranteed to be the MPC, it is likely a local optimum and therefore should be inserted in the population. Another situation that can happen is the presence of ties for certain variables, where the MPC can not be decided between 0 and 1. Typically, for problems tested with BOA, this occasionally occurs but for very few variables. Therefore, when the MPC presents ties, the loopy SLS enumerates all possible configurations and insert them in the population as the result of local search. To account for the rare case where the number of ties  $n_t$  is beyond reasonable, the maximum number of possible configurations/individuals returned by local search is set to  $\ell$ , in which case the configurations chosen are randomly selected from all possible  $2^{n_t}$ .

The loopy SLS method presents several differences from the proposal by Mendiburu, Santana, and Lozano (2007). The most significative difference is that the factor nodes use fitness information instead of the traditional approach in BP which is to use the conditional probabilities stored in CPTs. The motivation for doing so is discussed later with a detailed example. By using fitness information, the algorithm becomes a local search method based on loopy message-passing principles— therefore the name of loopy substructural local search. Another important innovation is the selection of relevant factor nodes to perform loopy SLS. Essentially, factor nodes (and corresponding edges) whose variable set is a subset of another factor are removed. Consider the previous example of a factor graph in Figure 2. The relevant factor nodes are  $f_3$  and  $f_4$  because the variable domain of the remaining factors is already included in these factors. Note that this simplification of the BN is possible because  $\bar{f}(X_1, X_2, X_3)$  (stored in factor  $f_3$ ) is more informative than both

---

**Loopy Substructural Local Search (Loopy SLS)**

---

- (1) Map the current Bayesian network  $B$  into a factor graph  $F$ , where factor nodes store substructural fitness information  $\bar{f}(X_i, \Pi_i)$ .
  - (2) Remove factor nodes (and corresponding edges) whose variable set is a subset of another factor in  $F$ .
  - (3) Perform loopy belief propagation in  $F$ . Return the most probable configuration MPC and possible number of tied positions  $n_t$ .
  - (4) **If**  $n_t = 0$ , instantiate an individual with the values from MPC;  
**Else If**  $2^{n_t} \leq \ell$ , enumerate all possible  $2^{n_t}$  configurations and instantiate them in  $2^{n_t}$  different individuals;  
**Else** enumerate  $\ell$  randomly chosen configurations out of  $2^{n_t}$  and instantiate them in  $\ell$  different individuals.
  - (5) Evaluate the resulting individuals.
- 

Figure 3: Pseudocode of the loopy substructural local search in BOA.

$\bar{f}(X_1, X_2)$  and  $\bar{f}(X_1)$  (stored in factors  $f_2$  and  $f_1$ ). In the same way,  $\bar{f}(X_4, X_5)$  already contains information from  $\bar{f}(X_5)$ . This straightforward procedure simplifies and improves the information exchange between nodes in the factor graph. In addition, the method for dealing with ties is also a novel contribution.

Figure 3 presents the pseudocode for the loopy substructural local searcher in BOA. The algorithm starts by mapping the current Bayesian network to a factor graph which stores fitness information in the factor nodes. The method proceeds by removing all factor nodes that are not relevant to local search, simplifying the search complexity for the MPC. The max-product algorithm is then applied to the resulting graph and the result is inserted in the population. Depending upon the number of possible ties for certain variable positions, up to  $\ell$  different individuals can be inserted in the population. This is a reasonable number in terms of population-sizing which is known to scale as  $\Theta(\ell \log \ell)$  (Yu, Sastry, Goldberg, & Pelikan, 2007). Finally, it is important to mention that the loopy local search takes place after the offspring population is generated.

## 6 Results and Discussion

This section presents and discusses the results obtained for the standard BOA, BOA with loopy SLS, BOA with standard LBP (as proposed by Mendiburu, Santana, and Lozano (2007)), and BOA with the simpler SLS (Lima, Pelikan, Sastry, Butz, Goldberg, & Lobo, 2006).

### 6.1 Experimental Setup

The test problem considered is the  $m - k$  trap function, where  $m$  stands for the number of concatenated  $k$ -bit trap functions. Trap functions (Deb & Goldberg, 1993; Goldberg, 2002) are relevant to test problem design because they bound an important class of nearly decomposable problems (Goldberg, 2002). The trap function used is defined as follows

$$f_{trap}(u) = \begin{cases} k, & \text{if } u = k \\ k - 1 - u, & \text{otherwise} \end{cases} \quad (9)$$

where  $u$  is the number of ones in the string,  $k$  is the size of the trap function. Note that for  $k \geq 3$  the trap function is fully deceptive (Deb & Goldberg, 1993) which means that any lower than  $k$ -order statistics will mislead the search away from the optimum. In this problem the accurate identification and exchange of the building-blocks (BBs) is critical to achieve success, because processing substructures of lower order leads to exponential scalability (Thierens & Goldberg, 1993). Note that no information about the problem is given to the algorithm; therefore, it is equally difficult for BOA if the variables correlated are closely or randomly distributed along the chromosome string. A trap function with size  $k = 5$  is used in our experiments.

Overlapping difficulty is an important problem feature because many problems can have different interactions that share common components. The difficulty of overlap is addressed by considering an overlapping version of  $m$ - $k$  trap problem, where each variable index set corresponding to a subfunction shares  $o$  variables with two other neighboring subproblems. More precisely, a trap- $k$  subfunction  $f_j(x_i, x_{i+1}, \dots, x_{i+k-1})$  will overlap with  $f_{j-1}(x_{i-k+o}, x_{i-k+o+1}, \dots, x_{i+o-1})$  and  $f_{j+1}(x_{i+k-o}, x_{i+k-o+1}, \dots, x_{i+2k-o-1})$ .

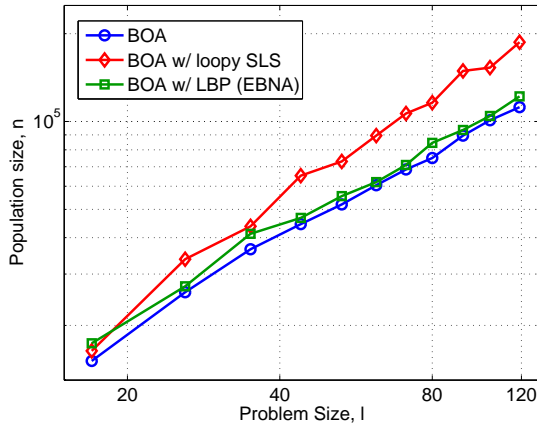
For all experiments, we use the population size that minimizes the total number of function evaluations required to solve the problem in 10 out of 10 independent runs. The population size adjustment is performed for 10 independent runs using a modified bisection method (Lima, 2009; Pelikan, 2005). Therefore, the total number of function evaluations is averaged over 100 ( $10 \times 10$ ) runs.

## 6.2 Loopy SLS *versus* Standard LBP

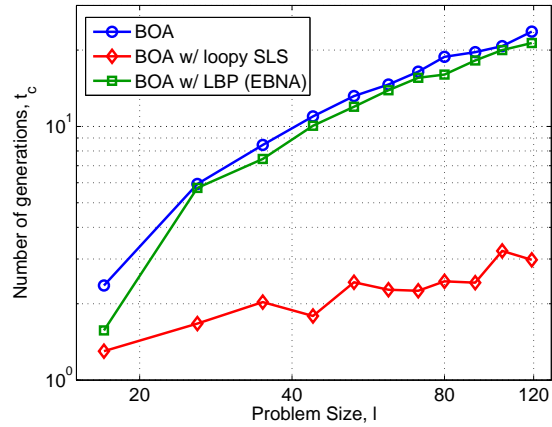
This section compares the proposed loopy SLS with the proposal by Mendiburu, Santana, and Lozano (2007). It should be clear that the only difference between the two alternatives is that loopy SLS uses (1) substructural fitness information instead of conditional probabilities for the factor nodes and (2) removes non-relevant factors. Other aspects such as message-scheduling, ties management, and parameters are set similarly (as described for loopy SLS), to focus the experimental comparison on the capability for generating high-quality solutions, rather than comparing different configurations of the max-product algorithm. The results for BOA with both standard loopy BP (as proposed for EBNA) and loopy SLS are presented in Figure 4.

The two alternatives present a very different behavior. While the LBP algorithm behaves similarly to the original BOA, preferring smaller population sizes but taking more iterations and consequently evaluations, the loopy SLS seems to take advantage from using larger population sizes. Note that both alternatives use the same method to tune the population size. Nevertheless, increasing the population size for standard LBP does not reduce the number of generations necessary to solve the problem. Although minor gains are obtained with LBP for some problem instances, the corresponding speedup is very close to one.

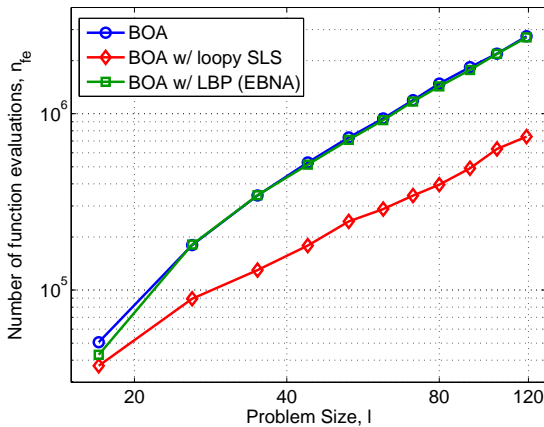
Loopy SLS can effectively take advantage from larger populations to gather more accurate information to speedup the solution of the problem. More accurate fitness information allows the loopy local searcher to converge faster to optimal solutions. If conditional probabilities are used instead (as in standard LBP), the algorithm requires a certain number of generations for selecting and propagating the best substructures until their sampling probability becomes significant enough. For example, consider the trap-5 function with the local optimum at 00000 and the global optimum at 11111. Initially, the local optimum will dominate the population because it's much easier to climb. Later on, when both optima are the most frequent alternatives, the selection process starts propagating 11111 over 00000. Only at this stage, the max-product algorithm based of conditional probabilities is expected to return 11111 as the most probable configuration. On the other hand, when using substructural fitness information, once the fitness surrogate is accurate enough to



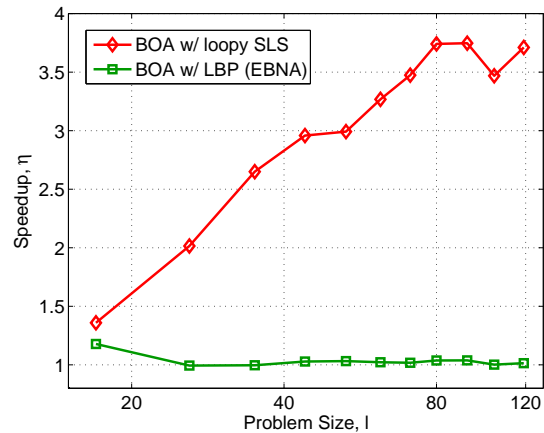
(a) Population size



(b) Num. of generations



(c) Num. of function evaluations



(d) Speedup

Figure 4: Results for BOA with both standard LBP and loopy SLS when solving the trap-5 problem with two overlapping variables ( $o = 2$ ).

identify 11111 as a better alternative than 00000, the MPC is expected to return the optimal solution. Consequently, BOA with loopy SLS takes advantage from using larger populations by building a more accurate fitness surrogate model.

Another important difference between the two approaches is the removal of factors which are not relevant to the MPC search. This is directly related to the dependency structure used by Bayesian networks to represent interactions among several variables. While this structure is required to be able to sample new instances with PLS, it is not necessary or even desirable when using BP methods. Given that  $\prod_{i=1}^k p(X_i | X_{i+1}, X_{i+2}, \dots, X_k) = p(X_1, X_2, \dots, X_k)$ , if a factor node relating  $k$  interacting variables stores the joint probability  $p(X_1, X_2, \dots, X_k)$ , there is no need of having  $k$  factors, one for each conditional probability in the product above. In this case, the presence of  $k$  factors can be even prejudicial if the lower-order statistics are misleading, which is the case for deceptive problems. The factors corresponding to lower-order statistics will make judgments based on local/deceptive information, somehow discrediting the information sent by the factor nodes with  $k$ -order statistics. Only when lower-order statistics start guiding the search towards 11111

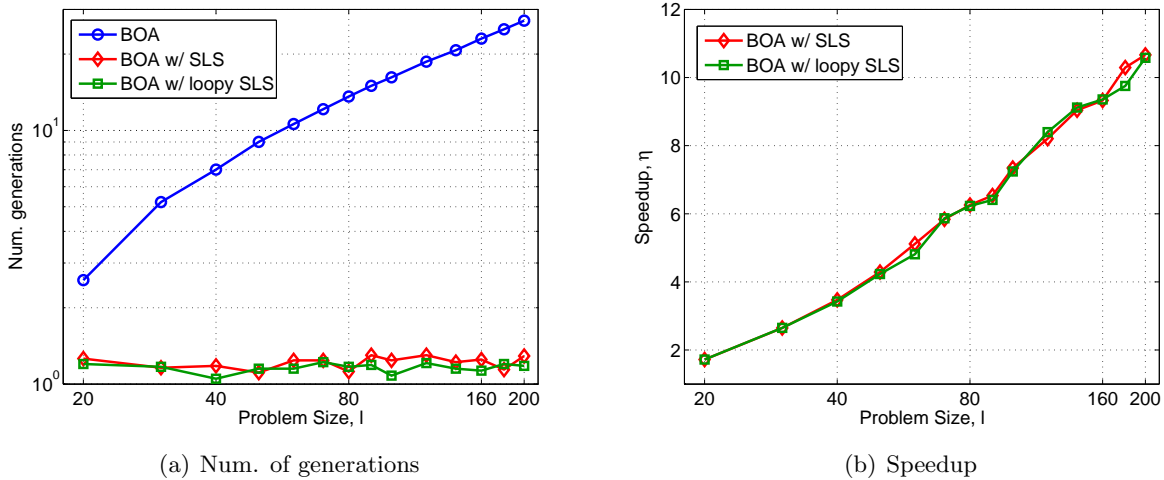


Figure 5: Results for BOA with both simple SLS and loopy SLS when solving the non-overlapping trap-5 problem. The corresponding speedup scales approximately as  $\Theta(\sqrt{\ell})$ .

(as mentioned above), this configuration will get enough recommendations to set the MPC with the optimal substructure.

### 6.3 Loopy SLS *versus* Simple SLS for Increasing Overlap

Figure 5 details the performance of BOA with both substructural local searchers for the trap-5 problem without overlap. Clearly, both SLS versions succeed in reducing the number of generations required to solve the problem. Consequently, the total number of function evaluations required is significantly reduced, providing speedups superior to 10. This translates into an order of magnitude less evaluations to solve the same problem. More importantly, the speedup consistently increases with problem size approximately as  $\Theta(\sqrt{\ell})$ .

Figure 6 presents the results for the trap-5 problem with several degrees of overlapping ( $o = \{1, 2, 3\}$ ). By using loopy substructural local search the savings in function evaluations are much greater than those obtained by the previous local searcher. For the simpler SLS, when the degree of overlapping between different subfunctions increases, the efficiency of performing local search reduces drastically. These results are not surprising given the nature of the local searcher. When searching for the best substructure at a given subproblem, the decision-making does not take into account the corresponding context. Because different subproblems are solved in a particular sequence, the best subsolution for a subproblem considered in isolation might not be the best choice when considering other subproblems that overlap with the first. While this is not the case for the overlapping trap-5 problem, because all subproblems have the same global optimum at 11111, the local searcher can still be deceived.

Consider the following example, where two different trap-5 subproblems overlap in two variables ( $X_4$  and  $X_5$ ), being the total problem size  $\ell = 8$ . When performing local search, the initial solution 00000000 has fitness  $f = 4 + 4 = 8$ , but when considering the best substructure for the first partition 11111000 the corresponding total fitness decreases to  $f = 5 + 2 = 7$ . While locally the best substructure is identified, the decrease in the overall fitness will not accept the move (see (Lima, Pelikan, Sastry, Butz, Goldberg, & Lobo, 2006) for further details). Even if the order of visit for the neighborhoods is randomly shuffled each time local search is performed, there is no guarantee that all possibilities are covered for highly overlapping problems.

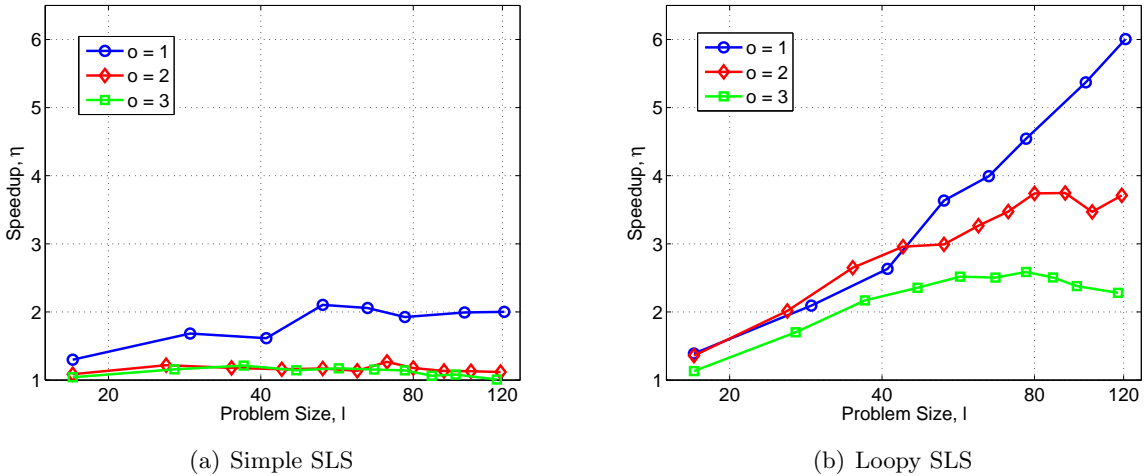


Figure 6: Speedup obtained for BOA with the simple SLS and the loopy SLS when solving the trap-5 problem with overlap of  $o = \{1, 2, 3\}$ .

With loopy SLS the context for each variable is now taken into account. For 1-variable overlap ( $o = 1$ ), the speedup grows up to 6, behaving very similarly to the non-overlapping case. For 2-variable overlap ( $o = 2$ ), the speedup also increases with the problem size but with a more moderate slope. Finally, for 3-variable overlap, the speedup grows with even a more moderate slope, while for larger problem instances the speedup seems to stagnate. Notice that for a trap subfunction with  $k = 5$  and  $o = 3$ , three out of five variables (60%) are shared with each of the two neighboring subfunctions, and each subfunction overlaps with another four on at least one variable. This translates into a considerable amount of noise at the decision-making for each subproblem, when looking for the best subsolution. Although the effect of overlapping variable interactions is similar to that of exogenous noise (Goldberg, 2002), which is known to be extremely hard for local search (Sastry & Goldberg, 2004), the speedups obtained with loopy SLS for problems with overlap are still substantial for considerable proportions of overlap. Speedups of 6, 3.75, and 2.5 were obtained for proportions of overlap of 20%, 40%, and 60%, respectively.

## 7 Summary and Conclusions

This paper presents a substructural local searcher for the Bayesian optimization algorithm based on the principles of loopy belief propagation in graphical models. The concept of substructural neighborhoods is used to perform local search in BOA. The local search method proposed makes use of a message-passing algorithm to find the optimal assignment of variable values, adequate for solving problems with highly conflicting subsolutions. Experiments are performed for different instances of the trap problem.

For the non-overlapping trap problem, substructural local search is shown to substantially reduce the number of function evaluations, providing speedups superior to 10 for a problem size of  $\ell = 200$ . This translates into one order of magnitude less evaluations to solve the same problem. More importantly, the speedup consistently increases with problem size approximately as  $\Theta(\sqrt{\ell})$ .

For the overlapping trap problem, BOA with loopy SLS maintains the substantial speedups from the non-overlapping case, but as the dimension of overlapping increases (making the problem more noisy), its efficiency is reduced. Nevertheless, local search still succeeds in saving a significant

number of function evaluations when compared to standard BOA. Speedups of 6, 3.75, and 2.5 were obtained for proportions of overlap of 20%, 40%, and 60%, respectively.

### 7.0.1 Acknowledgements

This work was sponsored by the Portuguese Foundation for Science and Technology under grants SFRH-BD-16980-2004 and PTDC-EIA-67776-2006, by the National Science Foundation under CAREER grant ECS-0547013, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, and by the University of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services, and the Research Award and Research Board programs.

The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Air Force Office of Scientific Research, or the U.S. Government.

## References

- Bayati, M., Shah, D., & Sharma, M. (2008, March). Max-product for maximum weight matching: Convergence, correctness, and LP duality. *Information Theory, IEEE Transactions on*, 54(3), 1241–1251.
- Braunstein, A., Mezard, M., & Zecchina, R. (2005). Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2), 201–226.
- Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. *Foundations of Genetic Algorithms 2*, 93–108.
- Elidan, G., McGraw, I., & Koller, D. (2006). Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proceedings of the Proceedings of the Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)* (pp. 165–173). Arlington, Virginia: AUAI Press.
- Etxeberria, R., & Larrañaga, P. (1999). Global optimization using Bayesian networks. In Rodriguez, A. A. O., et al. (Eds.), *Second Symposium on Artificial Intelligence (CIMAF-99)* (pp. 332–339). Habana, Cuba.
- Feige, U., Mossel, E., & Vilenchik, D. (2006). Complete convergence of message passing algorithms for some satisfiability problems. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, LNCS 4110* (pp. 339–350). Springer.
- Goldberg, D. E. (2002). *The design of innovation - lessons from and for competent genetic algorithms*. Norwell, MA: Kluwer Academic Publishers.
- Hart, W. E. (1994). *Adaptive global optimization with local search*. Doctoral dissertation, University of California, San Diego, San Diego, CA.
- Henrion, M. (1988). Propagation of uncertainty in Bayesian networks by logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Elsevier.
- Kindermann, R., & Snell, J. L. (1980). *Markov random fields and their applications*. Providence, RI: American Mathematics Society.

- Kschischang, F., Frey, B., & Loeliger, H.-A. (2001, Feb). Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2), 498–519.
- Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: a new tool for evolutionary computation*. Boston, MA: Kluwer Academic Publishers.
- Lima, C. F. (2009). *Substructural local search in discrete estimation of distribution algorithms*. Doctoral dissertation, University of Algarve, Faro, Portugal.
- Lima, C. F., Pelikan, M., Sastry, K., Butz, M., Goldberg, D. E., & Lobo, F. G. (2006). Substructural neighborhoods for local search in the Bayesian optimization algorithm. In Runarsson, T. P., et al. (Eds.), *PPSN IX: Parallel Problem Solving from Nature, LNCS 4193* (pp. 232–241). Springer.
- Mendiburu, A., Santana, R., & Lozano, J. A. (2007). *Introducing belief propagation in estimation of distribution algorithms: A parallel approach* (Technical Report EHU-KAT-IK-11-07). Department of Computer Science and Artificial Intelligence, University of the Basque Country.
- Mendiburu, A., Santana, R., Lozano, J. A., & Bengoetxea, E. (2007). A parallel framework for loopy belief propagation. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation* (pp. 2843–2850). New York, NY, USA: ACM.
- Mooij, J. M. (2008). *Understanding and improving belief propagation*. Doctoral dissertation, Radboud University Nijmegen, Nijmegen, Netherlands.
- Moscato, P. (1989). *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms* (Technical Report C3P 826). Pasadena, CA: Caltech Concurrent Computation Program, California Institute of Technology.
- Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.
- Pelikan, M. (2005). *Hierarchical Bayesian Optimization Algorithm: Toward a new generation of evolutionary algorithms*. Springer.
- Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm. In Banzhaf, W., et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99* (pp. 525–532). San Francisco, CA: Morgan Kaufmann.
- Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1), 5–20.
- Pelikan, M., & Sastry, K. (2004). Fitness inheritance in the bayesian optimization algorithm. In Deb, K., et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Part II, LNCS 3103* (pp. 48–59). Springer.
- Sastry, K., & Goldberg, D. E. (2004). Let’s get ready to rumble: Crossover versus mutation head to head. In Deb, K., & et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Part II, LNCS 3103* (pp. 126–137). Springer.
- Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.
- Yu, T.-L., Sastry, K., Goldberg, D. E., & Pelikan, M. (2007). Population sizing for entropy-based model building in genetic algorithms. In Thierens, D., et al. (Eds.), *Proceedings of the ACM*

*SIGEVO Genetic and Evolutionary Computation Conference (GECCO-2007)* (pp. 601–608).  
ACM Press.