J Heuristics (2008) 14: 519–547
DOI 10.1007/s10732-007-9049-8

Combining variable neighborhood search and estimation of distribution algorithms in the protein side chain placement problem

Roberto Santana · Pedro Larrañaga · José A. Lozano

Received: 7 April 2006 / Revised: 28 September 2006 / Accepted: 12 January 2007 /
Published online: 23 October 2007

Abstract The aim of this work is to introduce several proposals for combining two metaheuristics: variable neighborhood search (VNS) and estimation of distribution algorithms (EDAs). Although each of these metaheuristics has been previously hybridized in several ways, this paper constitutes the first attempt to combine both optimization methods.

The different ways of combining VNS and EDAs will be classified into three groups. In the first group, we will consider combinations where the philosophy underlying VNS is embedded in EDAs. Considering different neighborhood spaces (points, populations or probability distributions), we will obtain instantiations for the approaches in this group. The second group of algorithms is obtained when probabilistic models (or any other machine learning paradigm) are used in order to exploit the good and bad shakes of the randomly generated solutions in a reduced variable neighborhood search. The last group of algorithms contains the results of alternating VNS and EDAs.

An application of the first approach is presented in the protein side chain placement problem. The results obtained show the superiority of the hybrid algorithm in comparison with EDAs and VNS.

Keywords VNS · EDAs · UMDA · Protein folding · Rotamers · Protein side chain placement

R. Santana (✉) · P. Larrañaga · J.A. Lozano
Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country, Paseo Manuel de Lardizábal 1, 20080, San Sebastián–Donostia, Spain
e-mail: rsantana@si.edu

P. Larrañaga
e-mail: pedro.larranaga@ehu.es

J.A. Lozano
e-mail: ja.lozano@ehu.es

When a known hard optimization problem has to be solved and no clue about the characteristics of the search space is available, a repertoire of optimization methods is usually tried in the hope that the best method for the problem is identified. In these situations, metaheuristics are one of the most employed optimization approaches. Even if there are a variety of such metaheuristics, sometimes the results obtained by each algorithm separately are not satisfactory. One alternative in such cases is the combination of those metaheuristics that have proven to be the best contenders, or that benefit from different search strategies. The study of possible ways to combine metaheuristics is therefore an important topic in optimization (Kovačević et al. 1999; Brimberg et al. 2000; Andreatta and Ribeiro 2002; Rodríguez et al. 2003).

Variable neighborhood search (VNS) (Mladenović 1995; Mladenović and Hansen 1997; Hansen and Mladenović 2002; 2003b) and estimation of distribution algorithms (EDAs) (Mühlenbein and Paaß 1996; Larrañaga and Lozano 2002) are among the class of metaheuristics that have provided optimal solutions in many different problem domains. They use different search strategies. On the one hand, VNS is based on the application of local search by systematically changing the neighborhood during the search. Different variants of this metaheuristic exist. On the other hand, EDAs are evolutionary algorithms that, at each generation, extract relevant information of the search space and represent this information using probabilistic models. The models are used to sample new points from the search space. Proofs of global convergence for VNS (Brimberg et al. 2003) and EDAs (González et al. 2002a) have been given. Parallel implementations of each of the methods have been proposed (Lozano et al. 2002; García et al. 2005; Mendiburu et al. 2005).

Recent research has focused on the improvement and extensions of VNS (Hansen and Mladenović 2001; 2003a; Davidović et al. 2004) and EDAs (Larrañaga and Lozano 2002; Pelikan et al. 2002; Lozano et al. 2006) to cope with problems where the classical variants of the algorithms face limitations. One of the possible extensions of VNS and EDAs is the design of hybrid algorithms with other methods. Although each of these metaheuristics has been previously hybridized in several ways, the combination of both methods has not been studied yet. This paper constitutes the first attempt to combine both optimization methods. We introduce a classification of three main ways to combine VNS and EDAs. Results on the application of a hybrid EDAs+VNS approach are presented in the solution of the protein side chain placement problem (Lee and Subbiah 1991).

The paper is organized as follows. In the next section, we review the main aspects of VNS. EDAs are briefly explained in Sect. 2. A number of proposals to combine VNS and EDAs are introduced in Sect. 3. In Sect. 4, the protein side chain placement problem is introduced. This section also presents the EDAs+VNS approach to solve this problem. Section 5 introduces the neighborhood structures used by the VNS approach to the protein side chain placement problem. Results of the experiments on the application of the EDAs+VNS approach are presented in Sect. 6. Finally, in Sect. 7 the conclusions of the paper are given.

1 Variable neighborhood search (VNS)

VNS (Mladenović 1995; Mladenović and Hansen 1997) is based on a simple principle: the systematic change of neighborhoods within the search. It explores increasingly distant neighborhoods of the current solution, jumping from this solution to a new one if and only if an improvement has been made. Working in this way, favorable characteristics of the current solution will be often kept and used to obtain promising solutions. Moreover, in order to obtain local optima, a local search routine is repeatedly applied to these neighboring solutions.

More formally, let $\mathcal{N}_k$ $(k = 1, \ldots, k_{\max})$ be a finite set of previously fixed neighborhood structures, and $\mathcal{N}_k(\mathbf{x})$ the set of solutions in the kth neighborhood of $\mathbf{x}$ that represents a possible solution of the optimization problem. The main steps of the VNS algorithm are presented in Algorithm 1.

The basic VNS is a descent, first-improvement method with randomization—notice that point $\mathbf{x}'$ is generated at random in step 6 of Algorithm 1. Modifications to the basic VNS can be done in several ways (Hansen and Mladenović 2003a). In the variable neighborhood descent (VND), steps 6 and 7 are replaced by finding the best neighbor of $\mathbf{x}$. If, in step 8, moving is done with some probability, making the selection of a solution feasible even if it is worse than the current one, the basic VNS is transformed into a descent–ascent method. Moving to the best neighbor among all $k_{\max}$ of them converts the basic VNS into a best-improvement method. Further modifications to basic VNS as well as applications of this metaheuristic to several interesting combinatorial optimization problems are discussed in (Hansen and Mladenović 2001). Comparisons of VNS with tabu search (Glover 1986) and genetic algorithms (GAs) (Goldberg 1989) can be found in (Davidović et al. 2004).

To stop, the algorithm applies a previously specified termination criterion. Several termination criteria can be used. The most common ones are to fix a maximum CPU time, a maximum number of iterations, or a maximum number of iterations between two contiguous improvements.

Algorithm 1 Main steps of the VNS algorithm

1 **Initialization.** Select the set of neighborhood structures $\mathcal{N}_k$ $(k = 1, \ldots, k_{\max})$.
2 Find an initial solution, $\mathbf{x}$.
3 **do** {
4 $k \leftarrow 1$.
5 **do** {
6 *Shaking*. Generate a point $\mathbf{x}' \in \mathcal{N}_k(\mathbf{x})$ at random.
7 *Local search*. Apply some local search method with $\mathbf{x}'$ as the initial solution. Denote with $\mathbf{x}''$ the so-obtained local optimum.
8 *Move or not*. If this local optimum is better than the current solution, $\mathbf{x} \leftarrow \mathbf{x}''$ and $k \leftarrow 1$.
Otherwise, $k \leftarrow k + 1$.
9 } **until** $k = k_{\max}$.
10 } **until** Termination criterion is met.

2 Estimation of distribution algorithms

EDAs are evolutionary algorithms that replace the traditional crossover and mutation operators used in GAs by learning and sampling probabilistic models. These algorithms construct, in each generation, a probabilistic model that is used to estimate the probability distribution of the selected solutions. The probabilistic model must be able to capture, in the form of statistical dependencies, a number of relevant relationships between the variables. Dependencies are then used to generate solutions during a simulation step.

We use X_i to represent a discrete random variable. A possible value of X_i is denoted x_i. Similarly, we use $\mathbf{X} = (X_1, \ldots, X_n)$ to represent an n-dimensional random variable and $\mathbf{x} = (x_1, \ldots, x_n)$ to represent one of its possible values. We will work with positive probability distributions denoted by $p(\mathbf{x})$. Similarly, $p(\mathbf{x}_S)$ will denote the marginal probability distribution for $\mathbf{X}_S$.

Probability distributions can be represented by Bayesian networks (Pearl 1988), which are graphical models based on directed acyclic graphs and discrete variables. They have been used for probabilistic inference in domains such as expert systems (Lauritzen and Spiegelhalter 1988; Dawid 1992), classification problems (Friedman et al. 1997; Blanco et al. 2002), and optimization (Etxeberria and Larrañaga 1999; Pelikan et al. 1999).

In a Bayesian network, where variable X_i has r_i possible values, $(x_i^1, \ldots, x_i^{r_i})$, the local distribution $p(x_i \mid \mathbf{pa}_i^{j,S}, \theta_i)$ is an unrestricted discrete distribution:

$$p(x_i^k \mid \mathbf{pa}_i^{j,S}, \theta_i) = \theta_{x_i^k \mid \mathbf{pa}_i^{j,S}} \equiv \theta_{ijk}, \tag{1}$$

where $\mathbf{pa}_i^{1,S}, \ldots, \mathbf{pa}_i^{q_i,S}$ denote the values of $\mathbf{Pa}_i^S$, the set of parents of X_i in the directed acyclic graph S. q_i is the number of possible different instances of the parent variables of X_i, hence $q_i = \prod_{X_g \in \mathbf{Pa}_i^S} r_g$.

The local parameters are given by $\theta_i = ((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i}$. Parameter θ_{ijk} is the conditional probability of variable X_i being in its kth state given that the set of parents is in its jth configuration.

The general scheme of the EDA approach is shown in Algorithm 2. The selection method employed can be any of those traditionally used by GAs. In the literature, truncation, Boltzmann, and tournament selection are commonly used with EDAs. The most commonly used termination criteria are to reach a maximum number of generations or function evaluations.

A main characteristic and crucial step of EDAs is the construction of the probabilistic model. These models may differ in the order and number of the probabilistic dependencies that they represent. The success of EDAs in the solution of different practical problems has been documented in the literature (Lozano et al. 2006).

Different classifications of EDAs can be used to analyze these algorithms. Relevant to our research is the classification according to the complexity of the models used to capture the interdependencies between the variables (Larrañaga and Lozano 2002). Regarding the way learning is done in the probability model, EDAs can be divided into two classes. One class groups the algorithms that only make a parametric learning of the probabilities, whereas the other class comprises those algorithms

Algorithm 2 Main scheme of the EDA approach

1 $D_0 \leftarrow$ Generate M individuals randomly.
2 $l \leftarrow 1$.
3 **do** {
4 $\quad D^s_{l-1} \leftarrow$ Select $N \leq M$ individuals from D_{l-1} according to a selection method.
5 $\quad p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D^s_{l-1}) \leftarrow$ Estimate the joint probability of selected individuals.
6 $\quad D_l \leftarrow$ Sample M individuals (the new population) from $p_l(\boldsymbol{x})$.
7 $\quad l \leftarrow l + 1$.
8 } **until** Termination criterion is met.

where also structural learning of the model is done. The univariate marginal distribution algorithm (UMDA) (Mühlenbein and Paaß 1996), which is the EDA chosen to approach the side chain placement problem, belongs to the former class of algorithms. Among others, EDAs, which use Bayesian networks (Etxeberria and Larrañaga 1999: Mühlenbein and Mahnig 2001; Pelikan 2005), belong to the latter.

3 Hybridization between VNS and EDAs

The combination of metaheuristics has been successfully used in several problems where the approach consisting of using a unique method provides solutions that are poor local optima. In the literature, some of such combinations concerning VNS, as well as EDAs, can be found. In order to analyze possible hybridization between VNS and EDAs, it is convenient to first review the way in which these two metaheuristics have been previously combined with other methods.

Combinations with VNS have been carried out in two different ways. A first type of hybridization consists of using another heuristic in a step of the VNS. This approach has been introduced for tabu search (Kovačević et al. 1999; Rodríguez et al. 2003) and for multistart search (Belacel et al. 2002). The second proposed combination of VNS has embedded it into a given metaheuristic. For instance, in Brimberg et al. (2000), VNS is combined in this way with tabu search, while Andreatta and Ribeiro (2002) present an application where VNS is embedded within a greedy adaptive search procedure.

There are publications that propose the application of EDAs together with other heuristics. Certain EDA proposals apply local search procedures to the solutions sampled from the probabilistic model. This is the case of the proposals presented in Mühlenbein and Mahnig (2002); Robles et al. (2002); Höns (2006). A second possibility is the alternation of the search using EDAs and other schemes. For instance, in Zhou et al. (2005); Robles et al. (2006), EDAs are used together with GAs.

We propose and analyze three main ways to combine VNS and EDAs:

1. To incorporate VNS within EDAs.
2. To use probabilistic models within VNS.
3. To alternate VNS and EDAs.

3.1 VNS within EDAs

Several ways to embed *VNS within EDAs* can be obtained depending on the space where the neighborhood is defined.

3.1.1 Neighborhood defined in the space of points

The simplest way is when the neighborhood is considered to be in the *space of points* of the search space and, consequently, the VNS heuristic is applied to each individual—representing a point of the search space—sampled from the probability distribution at each iteration of the EDAs. Solutions are evaluated using the original fitness function $f(\mathbf{x})$.

This proposal can be computationally very expensive. The frequency and extent of the application of VNS are elements that must be considered for an efficient implementation. To deal with this problem, solutions to which VNS will be applied can be selected according to a probability distribution defined on the individuals sampled by EDAs. This general approach allows a variety of alternatives to be applied that range from the use of random selection (uniform probability distribution) to fitness proportionate schemes (fitness proportional and exponential probability distributions). Other selection schemes can also be defined in terms of probability distributions. These include to select:

- A percentage of the population in each generation.
- The set of selected individuals in each generation.
- The best solution found by EDA in each generation.
- The best solution found by EDA during its evolution.

In Sect. 5, we present an algorithm that illustrates an application of the last alternative.

3.1.2 Neighborhood defined in population space

We can extend the former approach by considering neighborhoods defined in *population space*. In this case, the search space is formed by sets of points (or populations). The fitness of each population can be calculated as the average of the fitness of the solutions or taking the best fitness value of all the solutions.

A key point is how to define the neighborhoods. The definition of the neighborhood can be accomplished by manipulating the components and/or parameters of the EDA. These parameters implicitly determine which populations (the neighborhood) can be reached from the current one.

Several strategies can be taken into account to move in this set of neighborhood structures. A first strategy is to maintain the number of sampled individuals constant and to change the percentage of selected individuals. This will cause the probability distribution learned to be different. Therefore, the population generated during the simulation step will change accordingly. In this case, by manipulating the percentage of selected individuals, different neighborhood structures are accessed. This strategy is illustrated by Algorithm 3 where the neighborhood structure is implicitly determined by the truncation parameter T_k, $k < 10$ that could be defined to weaken the

Algorithm 3 VNS in population space within EDA

$D_0 \leftarrow$ Generate M individuals randomly.
$l \leftarrow 1$.
do {
  $k \leftarrow 1$.
  $D_l \leftarrow D_{l-1}$.
  do {
    $D^s_{l-1} \leftarrow$ Select $N \leq M$ individuals from D_{l-1} using truncation selection with parameter T_k.
    $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D^s_{l-1}) \leftarrow$ Estimate the joint probability of selected individuals.
    Sample a set R of M individuals from $p_l(\boldsymbol{x})$.
    If the average fitness of individuals in R is better than the current set of individuals solutions, $D_l \leftarrow R$ and $k \leftarrow 1$.
    Otherwise, $k \leftarrow k + 1$.
  } **until** $k = k_{\max}$.
  $l \leftarrow l + 1$.
} **until** Termination criterion is met.

selection pressure ($T_k = 0.1k$) or to make it stronger ($T_k = 1 - 0.1k$). Notice that in this particular implementation of EDA the population at generation l is initialized equal to its previous generation $D_l = D_{l-1}$. Population D_l is updated only if a population R with better average fitness is found by VNS.

A second strategy could be based on the use bootstrapping (Efron 1982). This is a method that allows resampling with replacement from the original sample. It can be applied to the set of selected individuals until the sampling of the learned probabilistic model produces an individual with an objective function better than the current best. Alternatively, when the average fitness of the individuals generated is better than the average of the current selected set. In this case, the bootstrap method provides a way to simulate a neighborhood in the population space.

A third strategy is to increase the number of sampled individuals, but maintaining the number of selected individuals constant. The objective of this strategy is to learn the probability distribution from a set of individuals selected with higher selective pressure. Selection pressure is the ratio of the best individual's selection probability to the average selection probability of all individuals in the selection pool. Its role in the performance of evolutionary algorithms has been investigated in Blickle and Thiele (1996). In this case, the increase in selection pressure makes it possible to obtain different neighborhoods.

3.1.3 Neighborhoods in probability distribution space

A different way to embed VNS in EDAs is by defining *neighborhoods in the probability distribution space*.

Once learned, the probabilistic model is used to generate new solutions, some of which are expected to be better than those already visited. However, sometimes the

probability model cannot fulfill this goal. This may be due, for instance, to the problem of overfitting, which refers to an overly accurate approximation of data which does not reflect more general features of the search space. In this case, and starting from the initial model learned, VNS can be used to search for a better model, able to generate solutions with a better fitness average.

The quality of the searched models can be evaluated by calculating the fitness average of a set of solutions sampled from it. The neighborhoods can be defined in the space of the parameters (in this case, the structure of the graphical model will be fixed) or in the space of the structures (the structure is modified and new parameters are estimated from the data). Neighborhood structures for problems defined on graphs have been proposed (Brimberg et al. 2005; Kochetov and Velikanova 2005).

A simple idea to move between neighborhoods is to change the complexity of the probabilistic models. When the neighborhood is defined exclusively in the space of parameters, credal networks (Zaffalon 2002) could be used instead of Bayesian networks. In credal networks, the estimation of the conditional probabilities is done by means of interval estimation instead of by means of usual point estimation. If the neighborhoods are defined in the space of structures, the change between neighborhoods can be done by varying the number of edges or arcs that can be added to and/or removed from the graphical model.

Finally, it is worth mentioning that the computational complexity of implementing VNS in the space of probabilistic models can be very high in comparison to VNS defined for individuals and populations.

3.2 Use of probabilistic models within VNS

EDAs can also be applied within a VNS scheme (e.g. they can be employed to find a solution that will be used as the starting point of VNS). However, in this section we analyze the question of hybridization from a more general perspective, considering the combination of VNS with probabilistic models—the main component of EDAs-. The analysis illustrates how the principles at the foundations of EDAs (i.e. learning and use of relevant search information by means of probabilistic models) can be translated to the VNS domain.

In some versions of VNS, solutions are generated at random from the kth neighborhood. In this case, probabilistic models, as well as other machine learning techniques, could be used to exploit the information given by the successful and unsuccessful shakes.

Probabilistic models can represent and illustrate relevant information about the search space and the neighborhood structure used. Examples of such information include:

- Which changes in the values of a given variable are statistically more likely to cause an improvement in fitness.
- Which subsets of variables are more likely to produce an improvement in fitness when their values are changed together.

The model can be constructed during an initial run of VNS and used later to improve the efficiency of the method. Alternatively, it can be updated adaptively, in a similar way in which EDAs periodically update the probability model that they employ.

3.3 Alternation of VNS and EDAs

EDAs keep one of the initial goals of GAs: the recombination of partial solutions. Sampling from probabilistic models aims at the non-disruption of partial solutions represented by the model, contrary to what usually takes places when applying crossover operator. We may expect that, if EDAs are applied starting from solutions improved by using VNS, the newly generated solutions will combine some of the features of the VNS-improved solutions. Similarly, the solutions found by EDAs can be further improved by applying VNS.

Therefore, a simple way to *alternate VNS and EDAs* is to iterate—until there is no improvement—the execution of M VNS with starting points corresponding to the points sampled by the probabilistic model learned by the EDA, with the execution of several iterations of the EDA.

The application of VNS to points of the same population can lead to a population where solutions are very different from each other. This effect can be particularly evident in symmetric problems or problems with redundant representation. In these cases, solutions with similar fitness may be very different structurally and their combination (either by crossover or by classical probabilistic models) may not produce better solutions. A partial remedy to this case is to apply probabilistic models based on clustering the points according to their similarity. EDAs that use mixtures of probability distributions (Peña et al. 2005) have been applied to achieve this goal.

4 The side chain placement problem

Inferring the protein tertiary structure from its sequence is an important problem in molecular biology. The design of algorithms to predict the native structure of a protein from its amino acid sequence is an active research area. We approach the protein structure problem by focusing on a related problem, that of protein side chain placement.

An amino acid has a peptide backbone and a distinctive side chain. Assuming that the position of the backbone is fixed, and considering fixed bond lengths, the structure of the protein can be completely determined by the bond angles.

Figure 1 shows, from left to right, the complete native structure of the pdb1mrj protein,[1] only the backbone of the protein, and only the side chains.

The problem of finding an optimal positioning for the side chain residues is called side chain placement or side chain prediction (Lee and Subbiah 1991). Usually, the problem is addressed by constraining the search to the discrete space. This is done by employing discrete configurations of the angles, known as rotamers (Dunbrack 2002). Deterministic and stochastic methods have been proposed to cope with the side chain placement problem.

[1] All the proteins used in our research are referenced in this paper using their protein data bank identifier (PDB ID).

Fig. 1 *From left to right*: native structure of the pdb1mrj protein, backbone of the protein, side chains

4.1 Problem definition and fitness function

The backbone of the protein is the set of amino acid peptide backbones. The side chains can connect to the backbone in many different ways. A rotamer, short for rotational isomer, is a single side chain conformation represented as a set of discrete values, one for each dihedral angle degree of freedom (Dunbrack 2002). A rotamer library is a collection of rotamers for each residue type. The set of rotamers for an amino acid can be seen as a set of statistically significant conformations of the most probable configurations. In the side chain placement problem, the search for the protein structure is reduced to the search for a set of rotamers (one for each residue) that minimizes the objective function.

When the backbone is fixed, the energy of a sequence folded into a defined structure can be expressed (Voigt et al. 2000) as:

$$E(\mathbf{x}) = \sum_{i=1}^{n} E(x_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(x_i, x_j), \tag{2}$$

where x_i and x_j are two different rotamer configurations of residues i and j and n denotes the number of residues. $E(x_i)$ is the energy interaction between the rotamer and the backbone, and $E(x_i, x_j)$ represents the interaction energy between the rotamers pairs. The energies are estimated using probabilities calculated from a rotamer library.

Some algorithms, like dead-end elimination (DEE) (De Maeyer et al. 2000), take advantage of the pairwise decomposability of the fitness function to eliminate rotamer configurations that are proven not to be within the optimal solutions. One of simplest DEE implementations uses the Goldstein elimination criterion based on inequality 3 to iteratively eliminate rotamers.

$$E(x_i) - E(x_i') + \sum_{\substack{j=1 \\ j \neq i}}^{n} \min_{x_j} (E(x_i, x_j) - E(x_i', x_j)) > 0. \tag{3}$$

Equation 3 establishes a sufficient condition (De Maeyer et al. 2000) for rotamer configuration x_i to be absent from the optimal solution. If there exists a value x_i'

that satisfies Eq. 3 then value x_i can be eliminated. When no condition that further eliminates rotamers can be established, the algorithm stops. If the space of remaining configurations is small enough, the remaining combinations are searched using exhaustive enumeration. Unfortunately, this favorable scenario is not commonly found.

Other algorithms approach the search for side chain configurations as an optimization process. Particularly relevant for our work is the inference-based algorithm for structure prediction (SPRINT) (Yanover and Weiss 2003) which is one of the state of the art algorithms for protein side chain placement. In simple terms, it is a deterministic algorithm that associates a probabilistic model to the energy function and attempts to find the k most probable configurations of the model which in turn should correspond to the k solutions with lowest energy. The computation of the most probable configurations is done using belief propagation inference methods. To this end, exact (Nilsson 1998) and approximate inference methods (Yanover and Weiss 2004b) can be used. In the first case, the method is guaranteed to converge to the most probable configuration. However, the computational requirements of exact inference is usually unaffordable for medium and large problems. On the other hand, convergence is not guaranteed for approximate inference methods, not either that the solution found upon convergence is the optimal one.

5 An EDA-VNS approach to the side chain placement problem

In this section, we show that the combination of VNS with EDAs can improve the solutions found by only using EDAs or VNS. First, we describe the EDA approach to the side chain placement problem. Then, we propose a VNS scheme for this problem.

We use the following problem representation. Each residue will be represented by a random variable X_i. The number of values of each variable will correspond to the number of possible rotamer configurations for the corresponding residue i (i.e. $x_i \in \{1, \ldots, K_i\}$, where K_i is the number of feasible rotamer configurations for residue i).

The fitness evaluation function $f(\mathbf{x})$ decodes the solution $\mathbf{x}$ into the corresponding vector of rotamer configurations. Then, the energy function 2 is evaluated.

5.1 UMDA approach

UMDA uses a univariate model which is based on the assumption that all variables are independent. The configuration of variable X_i does not depend on the configuration of any other variable. In UMDA, $p(\mathbf{x})$ is factorized as follows:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i). \tag{4}$$

Using this particular factorization, the steps of the algorithm are as described in Algorithm 2. The pseudocode of the method used to solve the side chain placement problem is shown in Algorithm 4.

The algorithm starts by calculating the adjacency matrix of the protein considering the distance between the atoms in the backbone. Two residues are adjacent

Algorithm 4 UMDA-based algorithm for side chain placement

1 Construct the adjacency matrix of the protein using the backbone.
2 Calculate the energy interaction between neighboring rotamers.
3 Apply Goldstein elimination criterion to simplify the number of rotamer configurations.
4 Apply UMDA to find the candidate best solution.

(the corresponding value in the matrix is 1) if the distance between their respective atoms is below a given threshold. Otherwise, the value in the matrix is equal to 0. The calculation of the matrix simplifies the evaluation of the solutions by evaluating only those pairwise interactions that exist between adjacent residues in the graph. Then, the number of possible configurations for each residue is calculated using the backbone-dependent rotamer library of Dunbrack and Cohen (1997).

In the next step, we apply the Goldstein elimination criterion as shown in Eq. 3. This step can considerably contribute to reduce the dimension of the search space, but for medium and large proteins, search remains unaffordable for exact methods. When the application of the Goldstein elimination criterion cannot further reduce the number of values of the variables, we determine which residues have more than one rotamer configuration. The corresponding variables are the only ones to be considered in the optimization process. The probability model 4 will represent the probability of a given side chain configuration.

5.2 Definition of the VNS neighborhood and VNS schemes

In this section, we introduce two different VNS schemes in the context of EDAs to solve the protein side chain placement problem.

A crucial element of the VNS algorithm is the definition of the neighborhood. The neighborhood will be defined only for the points represented by those variables and values that remained after applying the Goldstein elimination criterion. As explained in Sect. 4.1, the Goldstein elimination criterion allows the number of variables and the range of values for each variable to be reduced. For the side chain placement problem, we define the k-neighborhood of a solution $\mathbf{x}$ as the set of solutions that are different from solution $\mathbf{x}$ in exactly k variables. More formally,

$$\mathcal{N}_k(\mathbf{x}) = \left\{ \mathbf{x}' \mid n - \sum_{i=1}^{n} I(x_i, x_i') = k \right\}, \tag{5}$$

where I is the indicator function, equal to one if both values are equal.

Clearly, given a point $\mathbf{x}$, for all $j \neq k, \mathcal{N}_j(\mathbf{x}) \cap \mathcal{N}_k(\mathbf{x}) = \emptyset$. Additionally, we use the protein structure information to constrain the neighborhood. Particularly, we use the information contained in the adjacency matrix. In the analysis of the k-neighborhood ($k > 1$), we only consider those sets of k variables for which each pair of variables has a non-zero entry in the adjacency matrix. This connection constraint naturally arises from the pairwise nature of the fitness function. Variables whose corresponding residues are not connected in the graph do not contribute together to the fitness

function. The independent contribution of the variables to the fitness function is covered by the 1-neighborhood.

In the 1-neighborhood of $\mathbf{x}$, all the values for each of the variables are tried in the exploration of the neighborhood. For $k > 1$, the algorithm calculates all possible sets of k variables with non-zero entries in the adjacency matrix. Each possible configuration of each set that is different in all k variables from $\mathbf{x}$ defines a neighbor solution. By requiring that the solution has to be different in all k variables, we guarantee that the neighborhoods do not overlap. On the other hand, by reducing the search of k-size sets to those interacting in the protein structure, the algorithm critically reduces the search space.

Given this neighborhood, we propose two alternative ways to define local search step 8 of Algorithm 1. Exhaustive and randomized procedures can be employed.

To select $\mathbf{x}''$ in the exhaustive schema, the $\mathcal{N}_k(\mathbf{x}')$ $(k = 1, \ldots, k_{\max})$ is completely searched to find a point where the fitness function is locally optimized. In the randomized scheme, point $\mathbf{x}''$ is selected using a random strategy. A local search is conducted by randomly selecting a solution in the neighborhood and moving to this solution if fitness is improved. A parameter, *maxtries*, defines the maximum number of points of the neighborhood that will be searched.

Obviously, the exhaustive search can be more computationally costly than the randomized one. However, in the second case, the cost depends on *maxtries*. To reduce the computational cost of the exhaustive scheme, we can constrain the set of neighborhood structures.

5.3 UMDA+VNS

We have applied a VNS algorithm to search for optimal solutions starting from the solutions found by UMDA. The VNS approach followed is the one mentioned in Sect. 1. The widest neighborhood which we consider has $k = 3$. Algorithm 5 describes the pseudocode of the resulting UMDA+VNS approach that uses the exhaustive procedure.

5.4 Computational complexity of UMDA+VNS

An important issue related with the efficiency of the algorithm is its computational complexity that translates in the time needed by the implementations to solve the problem. We present an analysis of the computational complexity of UMDA+VNS. First, the expressions that describe UMDA's complexity are shown. Then, we analyze the complexity of the variants of VNS proposed to address the problem.

To begin, we consider the computational complexity of each generation of UMDA. The initialization step of UMDA consists of assigning the values to all the individuals in the initial population. It has complexity $O(nM)$. The computational complexity of the evaluation step depends on the number of the non-zero entries in the adjacency matrix which, in the worst case, can be quadratic in the number of variables. Then, the time complexity of this step is $O(n^2 M)$. The complexity of the UMDA selection steps depends on the selection method used. For truncation selection, complexity is related to the ordering of the solutions. It is $O(M \log(M))$.

Algorithm 5 UMDA+VNS

1 Set $t \leftarrow 0$. Generate M points randomly.
2 **do** {
3 Select a set S of $N \leq M$ points using truncation selection.
4 Compute the univariate marginal frequencies $p_i^s(x_i, t)$ of S.
5 Generate M new points according to the distribution $p(\mathbf{x}, t+1) = \prod_{i=1}^{n} p_i^s(x_i, t)$.
6 $t \leftarrow t + 1$
7 } **until** Termination criteria are met.
8 Start from the best solution $\mathbf{x}$ found by UMDA.
9 **do** {
10 $k \leftarrow 1$.
11 **do** {
12 Generate a point $\mathbf{x}' \in \mathcal{N}_k(\mathbf{x})$ at random.
13 Apply the exhaustive procedure with $\mathbf{x}'$ as the initial solution.
Denote with $\mathbf{x}''$ the so-obtained local optimum.
14 If this local optimum is better than the current solution, $\mathbf{x} \leftarrow \mathbf{x}''$ and $k \leftarrow 1$.
Otherwise, $k \leftarrow k + 1$.
15 }textbfuntil $k = k_{\max}$.
16 } **until** Termination criterion is met.

The complexity of the learning step is $O(Nn)$. This is the cost of inspecting the values of every variable of the N selected solutions. The complexity of the sampling step depends on the number of variables, their values and the number of points. It is $O((M - N)nr_{\mathrm{MAX}})$, where $r_{\mathrm{MAX}} = \max_{i \in \{1,\ldots,n\}} |K_i|$ is the highest cardinality among the variables. $|K_i|$ corresponds to the maximum number of rotamer configurations the ith residue can have.

The actual number of generations needed by UMDA to converge is problem dependent. In general, this parameter is very difficult to estimate, although theoretical results for some classes of functions are available (González et al. 2002b). Let G be the maximal number of allowed generations. The complexity of UMDA for the side chain problem can be roughly estimated as $O(GM(n^2 + \log(M) + nr_{\mathrm{MAX}}))$.

The computational complexity of the exhaustive VNS depends on number of evaluations E done at each point, its complexity $\mathrm{O}(n^2)$, and the number of transitions V. E depends on the number of variables, the number of values for each variable, and the neighborhood size. It can be calculated as $E = \sum_{i=1}^{k_{\max}} \binom{n}{k} {r_{\mathrm{MAX}}}^k$. On the other hand, it is difficult to estimate the number of transitions of the algorithm before no further improvement is possible. The computational complexity of the exhaustive VNS can be roughly estimated for the worst case ($k_{\max} = \frac{n}{2}$) as $O(2^{n-1} {r_{\mathrm{MAX}}}^{k_{\max}} V)$. Notice, that for the derivation we have assumed that all possible neighborhoods of k variables are considered. As analyzed in Sect. 5.2, this situation corresponds to the worst case scenario where all the k possible sets has non-zero entries in the adjacency matrix. Additionally, we have not analyzed the reduction that partial evaluation of the energy function would have in the complexity of the evaluation step. Unfortunately, the reduction in the computational complexity of the evaluation step due to partial evaluation is difficult to estimate.

Finally, we analyze the computational complexity for randomized VNS. For this algorithm, the number of evaluations depends on the maximum number of points of the neighborhood that will be searched: $E = \text{maxtries}$. The computational complexity of randomized VNS is $O(n^2 \text{maxtries} V)$.

6 Experiments

We begin this section by describing the database of proteins used to empirically evaluate the algorithms. Subsequently, an explanation of the parameters used for each one of the tested algorithms is presented. Additionally, the criteria used in the comparison of the algorithms are enumerated. We also explain the experimental framework and the statistical tests employed.

Not all variants of VNS can be applicable to all the instances (e.g. exhaustive VNS, $k_{\max} = 3$ is not a feasible alternative for protein instances with a large number of residues and rotamer configurations). Therefore, the presentation of the numerical results has been organized according to a classification of the instances based on their size, as explained in the next section. Nevertheless, for the sake of clarity, the final results of the statistical tests are presented for all the instances altogether.

6.1 Protein instances

To test our algorithms, we have used a set of 463 protein structures.[2] The dataset corresponds to 463 X-ray crystal structures with a resolution better than or equal to 2Å, an R factor below 20%, and a mutual sequence identity lower than 50%. Each protein consisted of 1–4 chains and up to 1000 residues. This data set was originally employed in (Yanover and Weiss 2003) to evaluate the SPRINT algorithm.

The original database of proteins is divided into three groups: small, large, and dimer proteins. As a pre-processing step, we have determined, for each group, the instances in which the Goldstein elimination criterion 3 eliminates all configurations but one, and those instances in which SPRINT that uses loopy belief propagation method (an approximate inference technique) converges.

For the small class of instances, the protein structures obtained from the instances for which SPRINT converged are known to be the optimal ones (Yanover and Weiss 2004a). On the other hand, even if SPRINT does not converge, a solution (calculated from inconsistent marginals) is output by the algorithm. The solution, which is always the same for a given protein instance, is very likely to be sub-optimal and this fact constitutes one of the main drawbacks of the algorithm. Therefore, we focus on the application of UMDA+VNS to those instances were SPRINT does not converge. By constraining the protein benchmark to a set of difficult instances we intend to focus our experimental work on the most challenging problems, increasing the likelihood of UMDA+VNS to obtain new best solutions.

[2] These instances have been obtained from Chen Yanover's page: http://www.cs.huji.ac.il/~cheny/proteinsMRF.html.

The application of the Goldstein elimination criterion can only solve instances in the group of small instances. Moreover, SPRINT does not converge for 3% of the instances in the small class, 31% of the instances in the large class, and 32% of the instances in the dimer class. Proteins that were solved using the Goldstein elimination criterion and those for which SPRINT converged have been removed from the original database. The reduced small, large, and dimer groups contain the most difficult instances that we used in our experiments. The total number of instances for the reduced small, large and dimer groups are, respectively, 11, 14 and 25.

6.2 Algorithms and settings

The different variants of the VNS applied are the following:

- Exhaustive VNS, $k_{\max} = 3$. Applied to small protein instances.
- Exhaustive VNS, $k_{\max} = 1$. Applied to small and large protein instances.
- Randomized VNS (maxtries = 5000). Applied to small, large, and dimer protein instances.

The parameters of UMDA have been set as follows:

- Population size $M = 5000$.
- Maximum number of generations = 2000.
- Truncation selection with parameter $T = 0.15$. In this selection scheme, the best TM individuals of the population are selected to construct the probabilistic model.
- Best elitism. This a replacement strategy where the selected population at generation t is incorporated into the population of generation $t + 1$ keeping the best individuals so far found and avoiding to reevaluate their fitness function.
- The stop conditions were: The maximum number of generations is reached or the selected population has become too homogeneous (no more than 10 different individuals).

To compare the results of the algorithms, we conduct 50 experiments for each instance and algorithm. The performance of the algorithms are evaluated using the following criteria:

1. Fitness of the best solution found in each experiment.
2. Best fitness among all the best solutions found and number of experiments in which it was found.
3. Average number of fitness evaluations.

To determine whether differences between the fitness of the solutions found by the algorithms are statistically significant the Kruskal-Wallis test (Hsu 1996) has been employed. This test is used to accept or reject the null hypothesis that the samples obtained from 50 experiments corresponding to the different algorithms compared have been taken from equal populations. To perform the test, all the samples are combined into one large sample, and are sorted from smallest to largest. Then ranks are assigned (assigning the average rank to any observation in a group of tied observations). To compare the samples, the average of the ranks of the observations for each of the samples are calculated, and the test is applied. The test significance level was 0.05.

In the particular case of the comparison between UMDA+VNS and SPRINT, and since SPRINT is a deterministic algorithm we use the sign test. This is a non-parametric test used to compare two related groups. It is computed noting the direction of the difference between pair of scores. In our application, the signs correspond to the times, regarding the total number of cases considered, that the stochastic algorithm achieves results better (+), equal (=), or worse (−) than the deterministic algorithm. We expect that, if there are not significant difference between algorithms, the number of positive and negative signs must be approximately the same. The binomial distribution of parameters (n_b, k_b, p_b), where n_b is the number of cases, k_b is the number of positive outcomes, and $p_b = 0.5$ is used to compute the test statistic.

To compare the algorithms according to the best fitness, we determine which is the best solution found by each algorithm in the 50 experiments. The number of evaluations is considered an auxiliary measure of the efficiency of the algorithms. The number of evaluations for a VNS run is the number of evaluations made before no further improvement of the current solution is possible or a maximum number of tries (maxtries) has been reached.

Let g be the number of generations of UMDA before one stop condition has been fulfilled. The total number of evaluations is:

$$e = M + (g - 1)(M - TN).$$

The number of evaluations of UMDA+VNS is the sum of the number of evaluations needed to find the best solution of UMDA and the number of evaluations made by VNS.

6.3 Analysis of the behavior exhibited by UMDA+VNS

We start with an experiment that illustrates the behavior of UMDA+VNS. We have selected the dimer instance pdb1e6p which has 365 residues. Figure 2 shows the best energy obtained by UMDA+VNS$^{\text{randomized}}$ and VNS$^{\text{randomized}}$ as a function of the number of evaluations. The results are the average of 50 independent runs of each algorithm. The parameter maxtries of VNS$^{\text{randomized}}$ and UMDA+VNS$^{\text{randomized}}$ was set to 20000 to augment the exploratory capacity of the algorithm. The rest of the parameters remained with the values previously described. In each run, the number of evaluations needed to achieve the current best solution is stored every time that a solution with better energy is found. The experiment gives an idea of the efficiency of each method.

Since the first 5000 evaluations of UMDA+VNS$^{\text{randomized}}$ correspond to the solutions randomly generated by UMDA, VNS$^{\text{randomized}}$ is able to achieve better solutions with the same number of evaluations at the beginning. This fact can be clearly appreciated in Fig. 3 which offers a detailed view of the behavior of both algorithms during the first evaluations. As evolution continues, UMDA+VNS$^{\text{randomized}}$ quickly obtains better solutions than those achieved by VNS$^{\text{randomized}}$ with the same number of evaluations. As it can be seen in Fig. 2, even if a great number of evaluations are allowed to both algorithms, UMDA+VNS$^{\text{randomized}}$ consistently outperforms VNS$^{\text{randomized}}$.

We have also studied the efficiency of both algorithms in terms of the time needed to improve the current best solution. For the same runs analyzed before, we have also

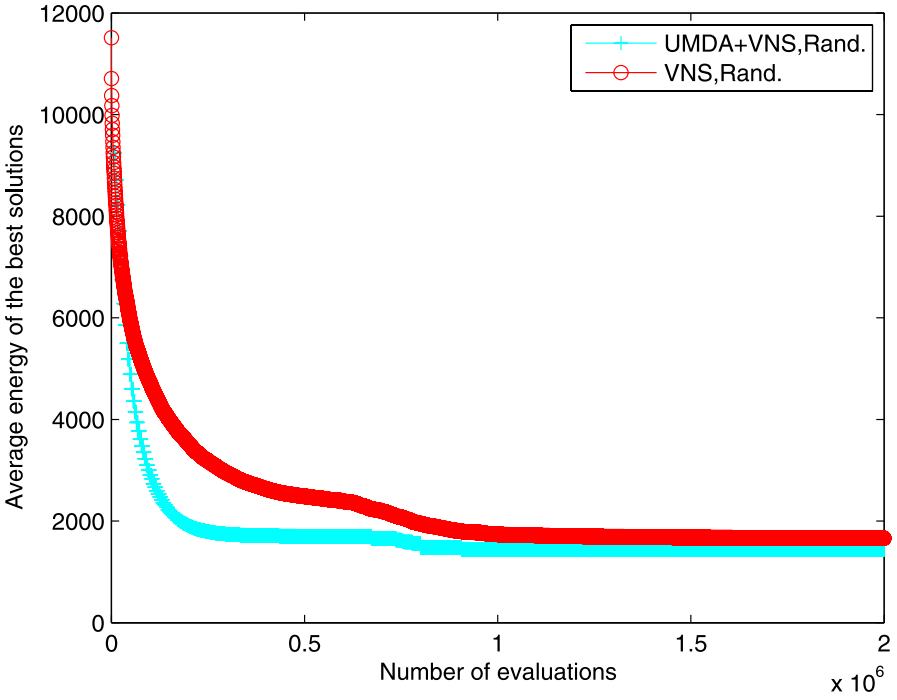

Fig. 2 Energy of the best solutions found by UMDA+VNS$^{\text{randomized}}$ and VNS$^{\text{randomized}}$ as the number of evaluations increases

stored the time at which a new best solution is found. Results are shown in Fig. 4. It can be seen that VNS$^{\text{randomized}}$ needs a short time to achieve relatively good solutions. However, it converges later to solutions with higher energies than those obtained by UMDA+VNS$^{\text{randomized}}$.

An interesting issue that illustrates the behavior of VNS$^{\text{randomized}}$ is the discontinuity in the pattern exhibited by the algorithm which corresponds to an important improvement in the quality of solutions (from solutions with energies higher than 2000 to energies below this threshold). This pattern can be explained by the effect of passing from a 1-neighborhood to a 2-neighborhood. After the capacity of the search to find better solutions in the 1-neighborhood has been exhausted, the search of the 2-neighborhood improves the solutions in a short time. Nevertheless, the search of the 3-neighborhood does not produce the same effect.

The main objective of the following experiments is to evaluate whether UMDA+VNS is able to improve the solutions obtained by UMDA and VNS. Additionally, we compare the performance of the exhaustive and randomized schemes introduced in Sect. 5.2.

6.4 Numerical results for small instances

Initial experiments are conducted using the small set of instances. We apply different variants of VNS and UMDA+VNS with the parameters described in Sect. 6.2.

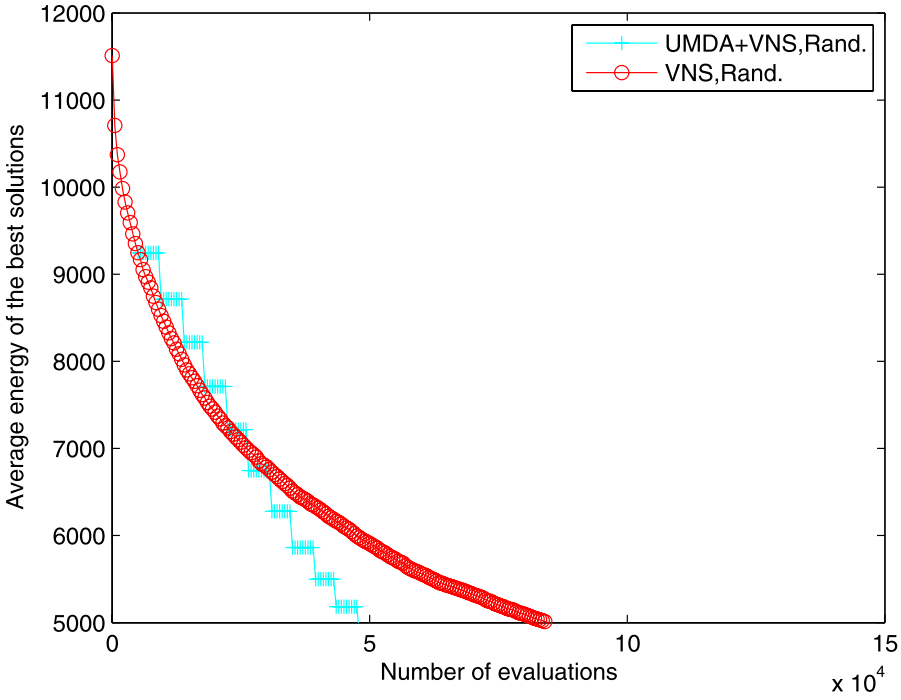

Fig. 3 Energy of the best solutions found by UMDA+VNS$^{\text{randomized}}$ and VNS$^{\text{randomized}}$ as the number of evaluations increases: Detail

The UMDA+VNS is applied to those solutions found by UMDA in the 50 experiments conducted for each instance. On the other hand, VNS is applied to 50 randomly generated solutions for each instance.

Table 1 shows the results achieved by UMDA and the different variants of VNS used. The table shows, for each algorithm, the fitness of the best solution found by the algorithm (best), the number of times it was found in the 50 experiments (S), and the mean of the fitness values calculated from the best solutions found in each of the 50 experiments (mean). Results corresponding to the algorithm that found the best solution among all the algorithms appear in bold. When the best solution is found by more than one algorithm, the algorithm that has found it more times is the one marked in bold. Most of the best solutions are achieved by exhaustive VNS, $k_{\max} = 3$. The average number of function evaluations needed by the algorithms is shown in Fig. 5. It can be appreciated that the number of evaluations needed by exhaustive VNS, $k_{\max} = 3$ is huge. However there is a great variability in the behavior of the algorithms depending on the instances. It is remarkable that the application of UMDA+VNS$^{\text{exhaustive}}$, $k_{\max} = 3$ requires in some cases less function evaluations than VNS$^{\text{exhaustive}}$, $k_{\max} = 3$.

Table 2 presents the results of UMDA+VNS using the same variants of VNS shown in Table 1. As expected, the best results are achieved by UMDA+VNS$^{\text{exhaustive}}$, $k_{\max} = 3$. However, the computational cost associated with neighborhood structures $k = \{2, 3\}$ makes it impractical to use the exhaustive search of the whole neighbor-

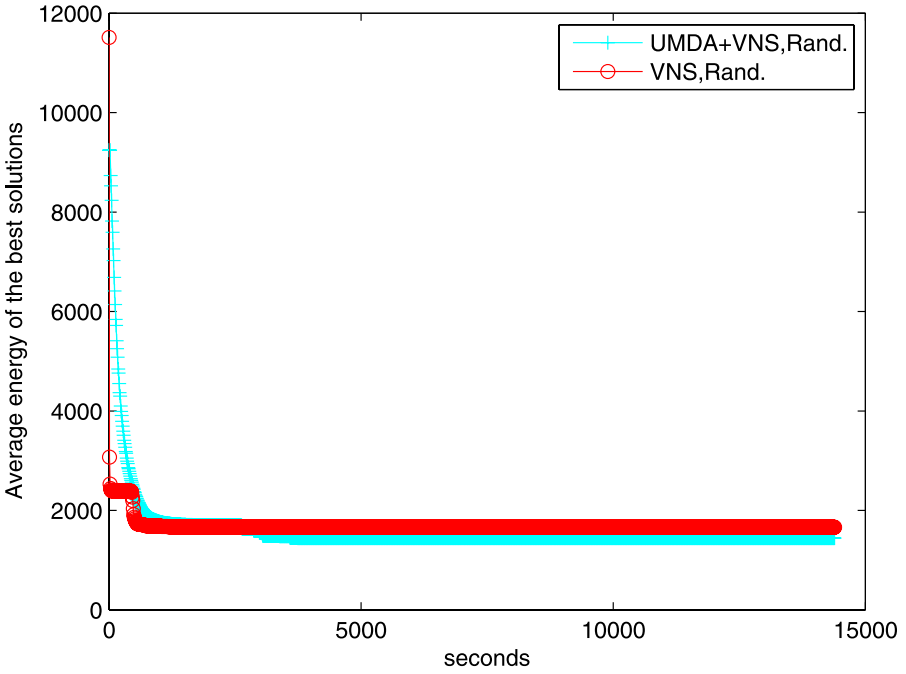

Fig. 4 Energy of the best solutions found by UMDA+VNS$^{\text{randomized}}$ and VNS$^{\text{randomized}}$ as a function of the time spent by the optimization algorithms

Table 1 Results achieved by UMDA and different variants of VNS for the subset of small instances for which SPRINT does not converge

pdb id	UMDA			Exhaustive, $k_{\max} = 3$ VNS			Exhaustive, $k_{\max} = 1$ VNS			Randomized VNS		
	Best	S	Mean	Best	S	Mean	Best	S	Mean	Best	S	Mean
pdb1buu	200.24	12	200.71	**200.24**	40	200.52	200.24	1	237.62	**200.24**	40	200.52
pdb1bv1	**132.54**	50	132.54	**132.54**	50	132.54	132.63	1	140.52	**132.54**	50	132.54
pdb1ema	**286.40**	48	286.47	286.40	16	287.83	288.71	1	333.33	286.40	15	289.37
pdb1et9	227.15	16	227.66	**226.80**	38	226.84	227.92	1	285.82	226.80	21	227.52
pdb1h6h	97.95	15	98.44	**97.95**	50	97.95	98.71	1	116.77	97.95	46	98.00
pdb1hh8	374.63	1	381.04	**369.57**	10	370.04	370.73	1	408.69	369.57	5	379.46
pdb1mrj	232.66	3	239.23	**232.66**	38	232.99	236.18	1	284.76	232.66	4	235.52
pdb2fcr	**236.28**	50	236.28	236.28	26	242.01	236.28	1	284.07	236.28	12	248.93
pdb2ilk	**138.55**	50	138.55	138.55	29	145.75	138.55	1	162.73	138.55	31	145.06
pdb2tir	92.61	50	92.61	**92.44**	15	92.56	92.61	6	106.98	92.44	10	92.62
pdb3kvt	163.96	33	164.15	**160.50**	50	160.50	163.96	2	235.16	160.50	14	167.93

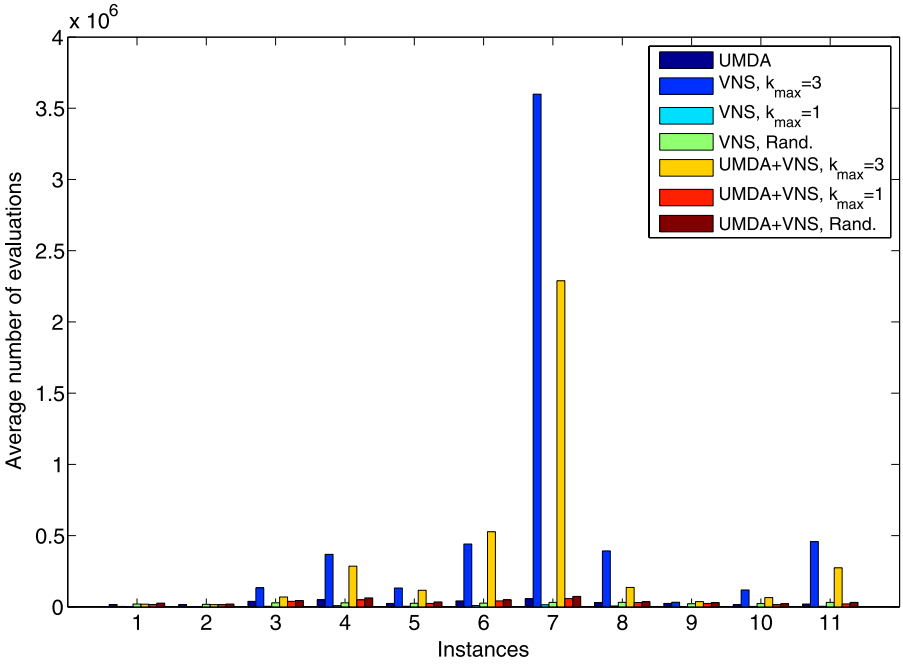

Fig. 5 Average number of evaluations of different algorithms for the set of small protein instances

Table 2 Results achieved by different variants of the UMDA+VNS for the subset of small instances for which SPRINT does not converge

pdb id	Exhaustive, $k_{\max} = 3$ UMDA+VNS			Exhaustive, $k_{\max} = 1$ UMDA+VNS			Randomized UMDA+VNS		
	Best	S	Mean	Best	S	Mean	Best	S	Mean
pdb1buu	**200.24**	50	200.24	200.24	17	200.68	**200.24**	50	200.24
pdb1bv1	**132.54**	50	132.54	**132.54**	50	132.54	**132.54**	50	132.54
pdb1ema	**286.40**	49	286.41	**286.40**	49	286.41	**286.40**	49	286.41
pdb1et9	**226.80**	22	226.90	227.15	20	227.63	**226.80**	20	226.95
pdb1h6h	**97.95**	50	97.95	97.95	18	98.40	97.95	39	98.10
pdb1hh8	**370.16**	50	370.16	370.95	1	380.95	370.16	1	380.70
pdb1mrj	**232.66**	36	232.86	232.66	6	235.01	232.66	14	234.69
pdb2fcr	**236.28**	50	236.28	**236.28**	50	236.28	**236.28**	50	236.28
pdb2ilk	**138.55**	50	138.55	**138.55**	50	138.55	**138.55**	50	138.55
pdb2tir	**92.61**	50	92.61	**92.61**	50	92.61	**92.61**	50	92.61
pdb3kvt	**160.50**	50	160.50	163.96	41	164.03	160.50	12	163.06

hood structure for larger instances. Since UMDA+VNS$^{\text{exhaustive}}$, $k_{\max} = 3$ is not a feasible alternative in general, we constrain the statistical analysis to the determination of whether UMDA+VNS$^{\text{randomized}}$ improves the results achieved by randomized

Table 3 Instances for which the average results achieved by UMDA+VNS$^{\text{randomized}}$ were significantly better or worse than those obtained with UMDA and VNS$^{\text{randomized}}$

Database	Better than UMDA	Better than VNS	Worse than VNS
Small	1, 4, 5, 11	1, 3, 7–9	6
Large	2–14	2, 4–14	1
Dimer	1–4, 6–26	1, 2, 4, 6–25	

Table 4 Results achieved by UMDA and different variants of VNS for the subset of large instances for which SPRINT does not converge

pdb id	UMDA			Exhaustive, $k_{\max} = 1$ VNS			Randomized VNS		
	Best	S	Mean	Best	S	Mean	Best	S	Mean
pdb1crz	626.41	1	627.25	628.71	1	720.08	**626.12**	4	627.20
pdb1ddt	754.93	1	760.02	764.39	1	843.83	**753.38**	2	777.34
pdb1dpe	727.37	2	750.51	792.09	1	902.58	**725.50**	6	746.04
pdb1e39	**545.30**	24	545.82	566.13	1	640.10	545.30	21	553.22
pdb1f5n	585.78	4	595.87	646.43	1	790.30	**585.78**	7	620.73
pdb1gsk	939.94	1	947.77	974.96	1	1088.01	**934.01**	1	963.16
pdb1h3n	1626.09	1	1639.00	1863.94	1	2129.65	**1623.20**	1	1722.27
pdb2jy1	861.92	1	870.34	883.11	1	981.23	**858.70**	1	878.87
pdb2kmo	925.90	1	943.09	930.50	1	1060.22	**889.09**	1	939.17
pdb2kwh	972.11	1	988.21	1011.46	1	1230.97	**960.73**	1	1010.49
pdb3n5u	860.67	1	877.39	899.46	1	1128.55	**858.89**	5	899.36
pdb2nqe	570.29	1	593.49	627.17	1	742.87	**569.04**	1	611.55
pdb2nr0	913.87	5	922.96	940.33	1	1115.18	**908.02**	4	946.14
pdb3nap	1108.79	1	1134.36	1101.21	1	1137.23	**1101.21**	1	1135.72

VNS and UMDA. The first row of Table 3 shows the instances for which the Kruskall-Wallis test found statistically significant differences.

6.5 Numerical results for large instances

Table 4 shows the results achieved by UMDA, exhaustive VNS, $k_{\max} = 1$, and randomized VNS in the optimization of the large protein instances used for our experiments. In terms of the best solution found, randomized VNS is clearly superior to the other two algorithms.

Table 5 presents the results of UMDA+ VNS$^{\text{exhaustive}}$, $k_{\max} = 1$ and UMDA+ VNS$^{\text{randomized}}$. Results corresponding to the best solution found appear in bold. Additionally, results where UMDA+VNS$^{\text{randomized}}$ produced the best known solution are underlined. The second row of Table 3 shows the instances for which the Kruskall-Wallis test found statistically significant differences in the results achieved by the different algorithms. The average number of evaluations of all the algorithms are

Table 5 Results achieved different by variants of UMDA+VNS for the subset of large instances for which SPRINT does not converge

pdb id	Exhaustive, $k_{max} = 1$ UMDA+VNS			Randomized UMDA+VNS		
	Best	*S*	Mean	Best	*S*	Mean
pdb1crz	**626.41**	1	627.20	**626.41**	1	627.18
pdb1ddt	754.93	1	759.68	**753.38**	1	755.34
pdb1dpe	727.37	12	745.13	**725.50**	7	739.35
pdb1e39	545.30	25	545.74	**545.30**	40	545.57
pdb1f5n	585.78	4	595.57	**585.78**	34	589.34
pdb1gsk	938.97	1	942.89	**935.65**	1	939.09
pdb1h3n	1623.17	1	1634.37	**1620.39**	1	1627.07
pdb2jy1	**856.84**	4	859.58	**856.84**	4	858.39
pdb2kmo	917.36	1	935.28	**901.88**	1	918.15
pdb2kwh	961.21	2	973.55	**960.73**	1	972.45
pdb3n5u	860.67	1	876.39	**858.89**	3	869.73
pdb2nqe	565.37	2	587.35	**565.37**	3	580.76
pdb2nr0	913.87	13	919.69	**912.54**	24	916.43
pdb3nap	1101.67	1	1111.30	**1101.21**	6	1107.90

shown in Fig. 6. Also in this case there is a great variability in the behavior of the algorithms depending on the instances. It can be seen the increment in the number of evaluations due to the application of VNS is not significant.

6.6 Numerical results for dimer instances

Table 6 shows the results achieved by UMDA, randomized VNS, and UMDA+VNS$^{\text{randomized}}$ in the optimization of the dimer protein instances used for our experiments. The third row of Table 3 shows the dimer instances for which the Kruskall-Wallis test found statistically significant differences in the results achieved by the different methods. The average number of evaluations of all the algorithms are shown in Fig. 7. The behavior of the number of evaluations needed by the algorithms for the set of dimer instances is similar that for the set of large instances.

6.7 Comparison with SPRINT algorithm

We also compare the results achieved by SPRINT and UMDA+VNS$^{\text{randomized}}$. Since SPRINT is a deterministic algorithm, we use the sign test. First, we compare the solution given by SPRINT with the best solution achieved by UMDA+VNS$^{\text{randomized}}$. Considering the 50 protein instances used in our experiments, UMDA+VNS$^{\text{randomized}}$ achieved better results than SPRINT in 62% of the cases, and worse results in only 10% of the cases. For the remaining 28% of instances, both algorithm obtained identical best structures. This means that UMDA+VNS$^{\text{randomized}}$ was successful in finding 31 protein side chain structures better than those previously known. The application

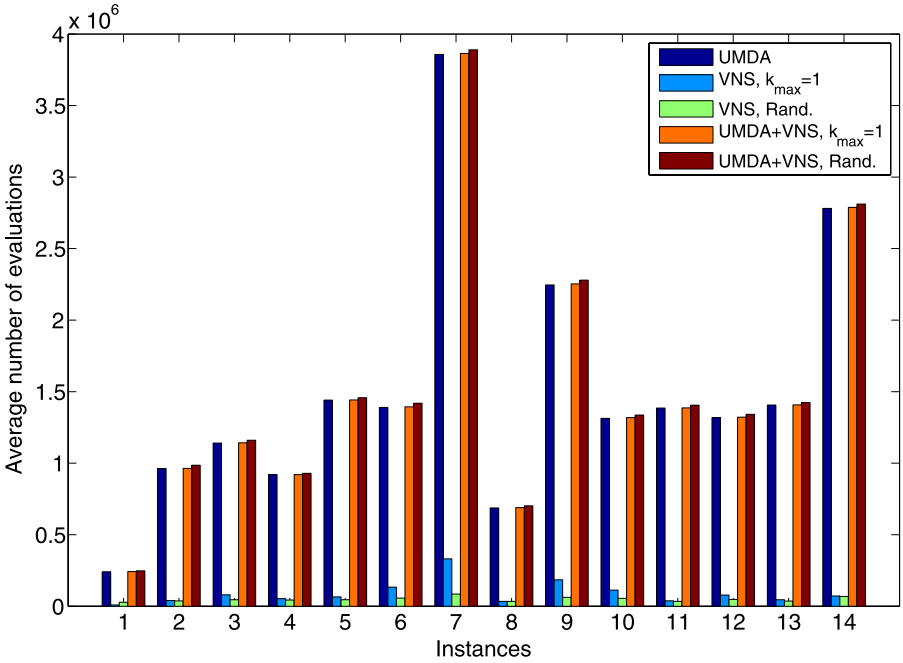

Fig. 6 Average number of evaluations of different algorithms for the set of large protein instances

of the sign test shown that the probability that the difference between the two algorithms arising by chance is only 0.059.

If results are analyzed taking into consideration the membership of the instances to the three different sets, we conclude that for the set of small instances, UMDA+VNS$^{\text{randomized}}$ and SPRINT obtained identical results in 100% of cases. For the set of large instances however, the algorithm introduced in this paper achieved better results than SPRINT in 57% of the cases, and worse results in 21% of the cases. The application of the sign test shown that the probability that the difference between algorithms arising by chance for this set is only 0.059. For the most complex set of instances, the dimer set, UMDA+VNS$^{\text{randomized}}$ achieved better results than SPRINT in 92% of the cases, and worse results in 8% of the cases. This analysis shows that, as the complexity of the sequences increases, the performance of UMDA+VNS$^{\text{randomized}}$ with respect to SPRINT is also improved.

6.8 Discussion of the results

A first conclusion of the experiments is that randomized VNS can achieve remarkably good results for the protein side chain placement problem. As Tables 1, 4 and 6 reveal, the best instances found by randomized VNS have, in most of cases, better fitness than those obtained by UMDA.

A second observation is that exhaustive VNS, $k_{\max} = 3$ is not a feasible alternative for protein instances with a high number of residues. The number of fitness evalua-

Table 6 Results achieved by UMDA, VNS and UMDA+VNS (randomized scheme) for the subset of dimer instances for which SPRINT does not converge

pdb id	UMDA			Randomized VNS			Randomized UMDA+VNS		
	Best	*S*	Mean	Best	*S*	Mean	Best	*S*	Mean
pdb1b25	4788.89	1	4820.76	4771.97	1	4916.25	**4725.57**	1	4769.04
pdb1d2e	1839.67	1	1847.65	1824.81	2	1839.53	**1826.88**	7	1829.41
pdb1dxr	1703.73	1	1722.79	**1695.16**	1	1713.68	**1695.16**	1	1704.83
pdb1dz4	875.77	1	884.79	**867.01**	2	886.33	867.01	1	874.41
pdb1e61	1936.92	1	1958.89	1944.16	1	2004.35	**1926.36**	3	1935.03
pdb1e6p	1681.67	1	1694.86	1678.30	1	1705.68	**1673.27**	1	1684.97
pdb1f60	537.42	3	540.78	537.04	1	541.48	**537.04**	8	539.85
pdb1fmj	1100.51	1	1121.42	**1088.80**	1	1150.22	**1088.80**	1	1098.54
pdb1fn9	989.51	3	993.92	987.47	1	1014.70	**987.47**	3	991.68
pdb1fnn	735.75	1	749.53	732.90	2	748.01	**732.01**	1	737.26
pdb1giq	806.53	1	823.58	**800.67**	1	824.62	**800.67**	1	813.38
pdb1h0h	4848.93	1	4913.00	4798.25	1	4910.64	**4755.80**	1	4809.89
pdb1h3f	785.56	1	795.11	786.16	1	803.61	**782.98**	5	788.34
pdb1h4r	825.64	1	830.12	815.84	9	819.23	**815.84**	16	817.08
pdb1h80	1036.90	1	1040.45	1034.83	2	1041.42	**1034.77**	9	1035.27
pdb1hhs	2627.41	1	2651.59	2596.78	1	2637.82	**2584.92**	1	2606.17
pdb1iqc	1538.37	1	1546.85	1530.18	1	1562.17	**1530.18**	2	1534.45
pdb1j3b	1600.16	1	1625.67	1594.03	1	1624.35	**1586.47**	2	1602.66
pdb1j8f	957.08	1	964.50	942.62	5	956.90	**942.62**	21	943.69
pdb1jmx	1518.10	1	1545.51	**1508.42**	1	1589.82	1515.11	1	1534.48
pdb1lqa	1017.56	1	1029.34	1018.76	1	1083.92	**1015.88**	1	1022.04
pdb1lqt	935.95	1	967.32	926.76	2	966.23	**926.16**	1	955.82
pdb1lsh	1125.04	1	1135.00	1118.69	1	1127.22	**1118.28**	1	1125.32
pdb1np7	1783.09	1	1799.91	1787.93	1	1848.19	**1765.25**	1	1779.11
pdb1tki	858.67	1	867.24	**855.56**	5	860.52	855.56	2	858.11

tions grows exponentially in this case. On the other hand, exhaustive VNS, $k_{\max} = 1$ achieves very poor results in comparison with randomized VNS and UMDA.

The main conclusion of our experiments, as shown in Tables 2, 3, 5, and 6 is that the combination of UMDA and VNS can improve the solutions achieved by each one of these algorithms. These results confirm that UMDA+VNS can obtain state-of-the-art solutions for the side chain placement problem.

Finally, we point to the fact that UMDA needs a very high number of function evaluations in comparison with randomized VNS. Although the parameters of UMDA were not tuned to minimize the number of function evaluations, for the parameters used in this paper it is clear that VNS is able to find better solutions with fewer evaluations. However, it is not clear whether a higher number of evaluations will allow VNS to reach better solutions than those achieved by UMDA+VNS. This remains as a question for further research.

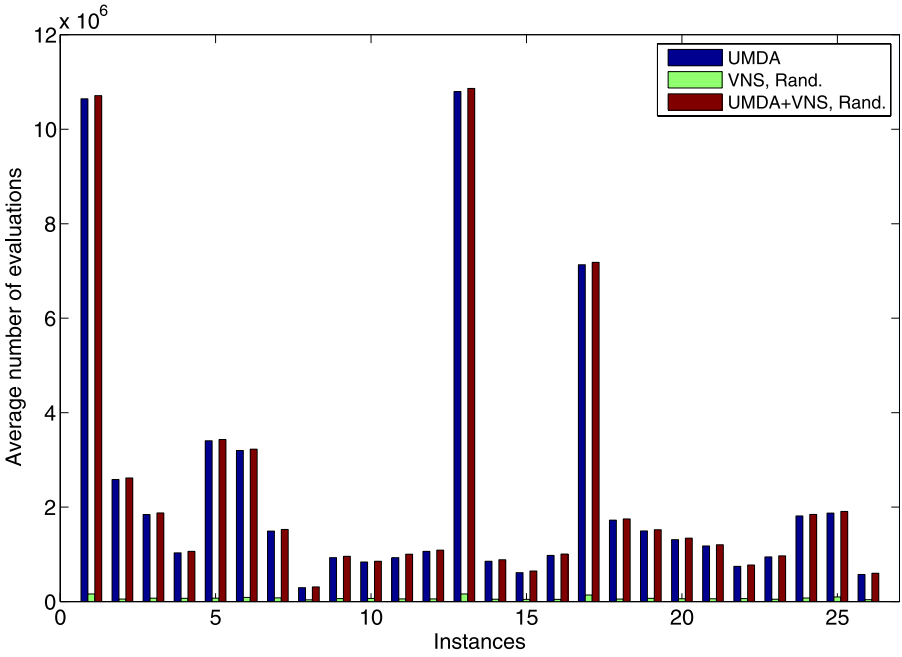

Fig. 7 Average number of evaluations of different algorithms for the set of dimer protein instances

7 Conclusions

A considerable amount of work has been produced showing that VNS and EDAs are suitable heuristics that can provide solutions in many different problem domains.

It is, therefore, a promising research trend to investigate possible ways to combine these two metaheuristics. In this paper, we have introduced a number of general alternatives for hybridizing VNS and EDAs. The different ways to combine VNS and EDAs have been classified into three main groups. This classification has been based on the different roles that both metaheuristics can play during a combined application, as well as on the possible definitions of the neighborhood space for VNS.

We have shown that, for the side chain placement problem, a hybrid approach between VNS and EDAs can improve the results achieved by using only EDAs and VNS. What is more, UMDA+VNS has obtained new protein structures with energy values better than those previously reported.

While our experiments have focused on the UMDA, we foresee that hybridization between VNS and other EDAs can also lead to improvements of the final solutions. Furthermore, the other alternatives introduced in this paper for combining VNS and EDAs should also be investigated.

Acknowledgements The authors thank Chen Yanover for useful comments on the protein side chain placement problem and for providing the set of instances used in our experiments. The authors are also grateful to Alex Mendiburu and Jose Miguel for providing part of the computational resources used in our experiments. This work was supported by the SAIOTEK-Autoinmune (II) 2006 and Etortek research projects from the Basque Government. It has been also supported by the Spanish Ministerio de Ciencia

y Tecnología under grant TIN2005-03824. The SGI/IZO-SGIker UPV/EHU (supported by the Spanish Program for the Promotion of Human Resources within the National Plan of Scientific Research, Development and Innovation—Fondo Social Europeo and MCyT) is gratefully acknowledged for their generous allocation of computational resources.

References

Andreatta, A., Ribeiro, C.: Heuristics for the phylogeny problem. J. Heuristics **8**, 429–447 (2002)

Belacel, N., Hansen, P., Mladenović, N.: Fuzzy J-means: a new heuristic for fuzzy clustering. Pattern Recognit. **35**(10), 2193–2200 (2002)

Blanco, R., Inza, I., Merino, M., et al.: Feature selection in Bayesian classifiers for the prognosis of survival of cirrhotic patients treated with TIPS. J. Biomed. Inform. **38**(5), 376–388 (2005)

Blickle, T., Thiele, L.: A comparison of selection schemes used in evolutionary algorithms. Evol. Comput. **4**(4), 361–394 (1996)

Brimberg, J., Hansen, P., Mladenović, N., et al.: Improvements and comparison of heuristics for solving the multisource weber problem. Oper. Res. **48**(3), 444–460 (2000)

Brimberg, J., Hansen, P., Mladenović, N.: Convergence of variable neighborhood search. Technical Report G–2003–45, Les Cahiers du GERAD (2003)

Brimberg, J., Mladenović, N., Urošević, D.: Variable neighborhood search for the k-cardinality subgraph problem. In: Hansen, P., Mladenović, N., Pérez, J.A.M., Batista, B.M., Moreno-Vega, J.M. (eds.) Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, 2005

Davidović, T., Hansen, P., Mladenović, N.: Permutation-based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communications delays. Technical Report G–2004–19, Les Cahiers du GERAD (2004)

Dawid, A.P.: Applications of a general propagation algorithm for probabilistic expert systems. Stat. Comput. (2), 25–36 (1992)

De Maeyer, M., Desmet, J., Lasters, I.: The dead-end elimination theorem: mathematical aspects, implementation, optimization, evaluation, and performance. Methods Mol. Biol. **143**, 265–304 (2000)

Dunbrack, R.L.: Rotamer libraries in the 21st century. Curr. Opin. Struct. Biol. **12**, 431–440 (2002)

Dunbrack, R.L., Cohen, F.E.: Bayesian statistical analysis of protein side-chain rotamer preferences. Protein Sci. **6**(8), 1661–1681 (1997)

Efron, B.: The jackknife, the bootstrap, and other resampling plans. In: CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 38, 1982

Etxeberria, R., Larrañaga, P.: Global optimization using Bayesian networks. In: Proceedings of the Second Symposium on Artificial Intelligence CIMAF-99, pp. 151–173, Habana, Cuba, 1999

Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29**(2–3), 131–163 (1997)

García, C.G., Pérez, D., García, F.C.: Parallel variable neighborhood search for the linear ordering problem. In: Hansen, P., Mladenović, N., Pérez, J.A.M., Batista, B.M., Moreno-Vega, J.M. (eds.) Proceedings of the 18th Mini Euro Conference on Variable Neighborhood, 2005

Glover, F.: Future paths for Integer programming and links to artificial intelligence. Comput. Oper. Res. **5**, 533–549 (1997)

Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison–Wesley, Reading (1989)

González, C., Lozano, J.A., Larrañaga, P.: Mathematical modeling of discrete estimation of distribution algorithms. In: Larrañaga, P., Lozano, J.A. (eds.) Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, pp. 143–162. Kluwer Academic, Boston (2002a)

González, C., Lozano, J.A., Larrañaga, P.: Mathematical modeling of UMDAc algorithm with tournament selection. Behaviour on linear and quadratic functions. Int. J. Approx. Reason. **31**(4), 313–340 (2002b)

Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–467 (2001)

Hansen, P., Mladenović, N.: Variable neighborhood search. In: Pardalos, P., Resende, M. (eds.) Handbook of Applied Optimization, pp. 221–234. Oxford University Press, London (2002)

Hansen, P., Mladenović, N.: Tutorial on variable neighborhood search. Technical Report G–2003–46, Les Cahiers du GERAD (2003a)

Hansen, P., Mladenović, N.: Variable neighborhood search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 145–184. Kluwer Academic, Dordrecht (2003b)
Höns, R.: Estimation of distribution algorithms and minimum relative entropy. Ph.D. thesis, University of Bonn, Bonn, Germany (2006)
Hsu, J.C.: Multiple Comparisons: Theory and Methods. Chapman & Hall, London (1996)
Kochetov, Y., Velikanova, Y.: Variable neighborhood search for the 2D orthogonal packing. In: Hansen, P., Mladenović, N., Pérez, J.A.M., Batista, B.M., Moreno-Vega, J.M. (eds.) Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, 2005
Kovačević, V., Čangalović, M., Ašić, M., et al.: Tabu search methodology in global optimization. Comput. Math. Appl. **37**, 125–133 (1999)
Larrañaga, P., Lozano, J.A. (eds.): Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic, Boston (2002)
Lauritzen, S., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems (with discussion). J. R. Stat. Soc. Ser. B **50**, 157–224 (1988)
Lee, C., Subbiah, S.: Prediction of protein side-chain conformation by packing optimization. J. Mol. Biol. **217**, 373–388 (1991)
Lozano, J.A., Sagarna, R., Larrañaga, P.: Parallel estimation of distribution algorithms. In: Larrañaga, P., Lozano, J.A. (eds.) Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, pp. 125–142. Kluwer Academic, Boston (2002)
Lozano, J.A., Larrañaga, P., Inza, I., et al.: Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Springer, Berlin (2006)
Mendiburu, A., Lozano, J., Miguel-Alonso, J.: Parallel implementation of EDAs based on probabilistic graphical models. IEEE Trans. Evol. Comput. **9**(4), 406–423 (2005)
Mladenović, N.: A variable neighborhood algorithm—a new metaheuristics for combinatorial optimization. In: Abstracts of Papers Presented at Optimization Days, Montréal, p. 112, 1995
Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
Mühlenbein, H., Mahnig, T.: Evolutionary synthesis of Bayesian networks for optimization. In: Patel, M., Honavar, V., Balakrishnan, K. (eds.) Advances in Evolutionary Synthesis of Intelligent Agents, pp. 429–455. MIT Press, Cambridge (2001)
Mühlenbein, H., Mahnig, T.: Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. Int. J. Approx. Reason. **31**(3), 157–192 (2002)
Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (eds.) Parallel Problem Solving from Nature—PPSN IV, LNCS, vol. 1141, pp. 178–187. Springer, Berlin (1996)
Nilsson, D.: An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. Stat. Comput. **2**, 159–173 (1998)
Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo (1988)
Pelikan, M.: Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms. Springer, Berlin (2005)
Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999, vol. I, pp. 525–532. Morgan Kaufmann, Orlando (1999)
Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Comput. Optim. App. **21**(1), 5–20 (2002)
Peña, J., Lozano, J.A., Larrañaga, P.: Globally multimodal problem optimization via an estimation of distribution algorithm based on unsupervised learning of Bayesian networks. Evol. Comput. **13**(1), 43–66 (2005)
Robles, V., de Miguel, P., Larrañaga, P.: Solving the traveling salesman problem with EDAs. In: Larrañaga, P., Lozano, J.A. (eds.) Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, pp. 227–238. Kluwer Academic, Boston (2002)
Robles, V., Peña, J.M., Pérez, M.S., et al.: GA-EDA: a new hybrid cooperative search evolutionary algorithm. In: Lozano, J.A., Larrañaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms, pp. 187–200. Springer, Berlin (2006)
Rodríguez, I., Moreno, J.M., Moreno, J.A.: Variable neighborhood tabu search and its application to the median cycle problem. Eur. J. Oper. Res. **151**(2), 365–378 (2003)

Voigt, C.A., Gordon, D.B., Mayo, S.L.: Trading accuracy for speed: a quantitative comparison of search algorithms in protein sequence design. J. Mol. Biol. **299**(3), 799–803 (2000)

Yanover, C., Weiss, Y.: Approximate inference and protein-folding. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems, vol. 15, pp. 1457–1464. MIT Press, Cambridge (2003)

Yanover, C., Weiss, Y.: Approximate inference and side-chain prediction (2004a, submitted for publication). Available online from: http://www.leibniz.cs.huji.ac.il/tr/963.pdf

Yanover, C., Weiss, Y.: Finding the M most probable configurations using loopy belief propagation. In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems, vol. 16, MIT Press, Cambridge (2004b)

Zaffalon, M.: The naive credal classifier. J. Stat. Plan. Inference **105**, 5–21 (2002)

Zhou, A., Zhang, Q., Jin, Y., et al.: A model-based evolutionary algorithm for bi-objective optimization. In: Proceedings of the 2005 Congress on Evolutionary Computation CEC-2005, pp. 2568–2575. IEEE Press, Edinburgh (2005)