ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms

Jérémie Humeau, Arnaud Liefooghe, El-Ghazali Talbi, Sébastien Verel

ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms

Jérémie Humeau, Arnaud Liefooghe, El-Ghazali Talbi, Sébastien Verel

ISSN 0249-6399 ISRN INRIA/RR--7871--FR+ENG

RESEARCH REPORT
N° 7871
June 2013
Project-Team Dolphin

ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms

Jérémie Humeau*, Arnaud Liefooghe†, El-Ghazali Talbi‡, Sébastien Verel§

Project-Team Dolphin

Research Report n° 7871 — June 2013 — 36 pages

Abstract: This document presents a general-purpose software framework dedicated to the design, the analysis and the implementation of local search algorithms: ParadisEO-MO. A substantial number of single-solution based local search metaheuristics has been proposed so far, and an attempt of unifying existing approaches is here presented. Based on a fine-grained decomposition, a conceptual model is proposed and is validated by regarding a number of state-of-the-art methodologies as simple variants of the same structure. This model is then incorporated into the ParadisEO-MO software framework. This framework has proven its efficiency and high flexibility by enabling the resolution of many academic and real-world optimization problems from science and industry.

Key-words: local search; metaheuristic; fitness landscapes; conceptual unified model; algorithm design, analysis and implementation; software framework

* École des Mines de Douai, département IA, 941 rue Charles Bourseul, BP 10838, 59508 Douai, France (jeremie.humeau@mines-douai.fr)
† Université Lille 1, Laboratoire LIFL, UMR CNRS 8022, Cité scientifique, Bât. M3, 59655 Villeneuve d'Ascq cedex, France (arnaud.liefooghe@univ-lille1.fr)
‡ Université Lille 1, Laboratoire LIFL, UMR CNRS 8022, Cité scientifique, Bât. M3, 59655 Villeneuve d'Ascq cedex, France (talbi@lifl.fr)
§ Université Nice Sophia Antipolis, Laboratoire I3S, UMR CNRS 6070, 2000 route des Lucioles, BP 121, 06903 Sophia Antipolis cedex, France (verel@i3s.unice.fr)

RESEARCH CENTRE
LILLE – NORD EUROPE

Parc scientifique de la Haute-Borne
40 avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq

ParadisEO-MO: De l'analyse du paysage de fitness à des algorithmes de recherche locale efficaces

Résumé : Ce document présente une plateforme logicielle polyvalente dédiée à la conception, l'analyse et l'implémentation d'algorithmes de recherche locale: ParadisEO-MO. Un nombre substantiel de métaheuristiques basées sur une solution unique a été proposé jusqu'à présent, et une tentative d'unifier les approches existantes est ici présentée. Basé sur une décomposition fine, un modèle conceptuel est proposé et validé par l'instantiation d'un grand nombre de méthodologies classiques comme des variantes simples de la même structure. Ce modèle est ensuite incorporé dans la plateforme logicielle ParadisEO-MO. Cette plateforme a prouvé son efficacité et sa grande flexibilité en permettant la résolution de nombreux problèmes d'optimisation académiques et du monde réel, des domaines de la science et de l'industrie.

Mots-clés : recherche locale; métaheuristique; paysages de fitness; modèle conceptuel unifié; conception, analyse et implémentation d'algorithmes; plateforme logicielle

1 Introduction

The need of software frameworks is essential in the design and implementation of local search metaheuristics. Those frameworks enable the application of different search algorithms (*e.g.* hill-climbing, tabu search, simulated annealing, iterated local search) in a unified way to solve a large variety of optimization problems (single-objective/multi-objective, continuous/discrete) as well supporting the extension and adaptation of the metaheuristics for continually evolving optimization problems. A framework is different from a solver, since it does not implement a universal optimal resolution methodology but rather provides tools allowing a better development in terms of cost and effort. Hence, the user only has to focus on high-level design aspects. Indeed, a metaheuristic is *not* a heuristic. It requires a number of problem-specific components in order to be applied to a particular solving task. A metaheuristic is rather an upper-level general methodology that can be used as a guiding strategy in designing underlying heuristics to solve specific optimization problems.

In general, the efficient solving of a given optimization problem requires to experiment many solving methods, tuning the parameters of each metaheuristic, etc. The metaheuristic domain in terms of new algorithms is also evolving. More and more increasingly complex local search algorithms are developed. Moreover, it allows the design of complex hybrid and parallel models which can be implemented in a transparent manner on a variety of architectures (shared-memory such as multi-cores and GPUs, distributed memory such as clusters, and large-scale distributed architecture such as Grids and Clouds). Hence, there is a clear need to provide a ready-to-use implementation of such metaheuristics. It is important for application engineers to choose, implement and apply state-of-the-art algorithms without in-depth programming knowledge and expertise in optimization. For optimization experts and developers, it is useful for them to evaluate and compare fairly different algorithms, to transform ready-to-use algorithms, to design new algorithms, as well as to combine and parallelize algorithms. Frameworks may provide default implementation of classes. The user has to replace the defaults that is appropriate for his/her application. Indeed, software frameworks are not supposed to be universal implemented applications, but rather adaptable tools allowing a better implementation in terms of cost and effort.

ParadisEO is a software framework allowing the reusable design of metaheuristics. It is available at the following URL: `http://paradiseo.gforge.inria.fr`. Unlike black-box solvers, It is based on a conceptual separation between the search algorithm and the problem to be solved. ParadisEO is a free open-source white-box object-oriented software framework implemented in C++. This project has been downloaded more than 20000 times and more than 250 active users are registered in the mailing-list. It contains four interconnected modules: EO (Keijzer et al, 2001) for population-based metaheuristics, MO for single solution-based metaheuristics, MOEO (Liefooghe et al, 2011) for multi-objective optimization and PEO (Cahon et al, 2004) for parallel and distributed metaheuristics. In addition, the whole framework allows the implementation of hybrid approaches.

ParadisEO-MO (Moving Objects) is the module dedicated to the design of single solution-based metaheuristics (*i.e.* local search). An important aspect in ParadisEO-MO is that the common search concepts of local search metaheuristics are factored. All search components are defined as templates (generic classes). ParadisEO is based on the object-oriented programming and design paradigm in order to make those search mechanisms adaptable. The user designs and implements a local search algorithm by deriving the available templates that provide the functionality of different search components: problem-specific templates (*e.g.* representation, objective function) and problem-independent templates (*e.g.* neighborhood exploration, cooling schedule, stopping criteria, etc.). Moreover, some available components allow to trace statistics on local search execution describing the landscape of the problem. This paper presents the design, analysis and implementation of the ParadisEO-MO module, allowing to tackle an optimization problem as a whole, from its fitness landscape analysis to its resolution by means of local search metaheuristics.

The paper is organized as follows. In Section 2, a unified view of local search algorithms is presented. This section details the common search components for local search metaheuristics. It introduces, in an incremental way, the well-known local search algorithm and outlines the landscape analysis of optimization problems. Then, Section 3 discusses the design and implementation of the ParadisEO-MO framework. Some design and implementations of popular local search algorithms such as hill-climbing, simulated annealing, tabu search and iterated local search are illustrated. Finally, Section 4 outlines the main conclusions and perspectives of this work.

2 A Conceptual Model for Local Search

2.1 Local Search General Template

While solving optimization problems, single solution-based metaheuristics (or local search metaheuristics) improve a single solution. They could be viewed as "walks" through neighborhoods or search trajectories through the search space of the problem at hand (Talbi, 2009). The walks (or trajectories) are performed by iterative procedures that move from the current solution to another one in the search space. Local search metaheuristics show their efficiency in tackling various optimization problems in different domains.

Local search metaheuristics iteratively apply the generation and replacement procedures from the current single solution (Fig. 1). In the generation phase, a set of candidate solutions are generated from the current solution s. This set $C(s)$ is generally obtained by local transformations of the solution. A candidate solution is often a *neighboring solution*, and so, the set $C(s)$ is a subset of the *neighborhood* of solution s. In the replacement phase (also named transition rule, pivoting rule and selection strategy), a selection is performed from the candidate solution set $C(s)$ to replace the current solution, *i.e.* a solution $s' \in C(s)$ is selected to be the new solution. When s' is selected, it replaces the current solution according to an acceptance criterion. This process iterates until a given stopping criteria is satisfied. The generation and the replacement phases may be *memoryless*. In this case, the two procedures are based only on the current solution. Otherwise, some history of the search stored in a memory can be used in the generation of the candidate list of solutions and the selection of the new solution. Popular examples of such local search metaheuristics are hill-climbing, simulated annealing and tabu search. Algorithm 1 illustrates the high-level template of local search metaheuristics.

Algorithm 1 High-level template of local search metaheuristics.

Input: Initial solution s.
repeat
  Select one solution s' in the neighborhood of s
  if acceptance criterion is true **then**
    $s \longleftarrow s'$
  end if
until Stopping criteria satisfied
Output: Best solution found.

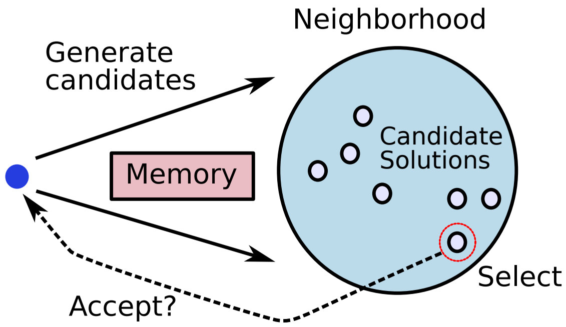

Figure 1: Template for local search metaheuristic: (i) generate candidate solutions from the neighborhood, (ii) select a neighbor, (iii) decide to replace the current solution by the selected neighbor.

2.2 Common Issues

The common search concepts for *all* local search metaheuristics are the definition of the *representation* of solutions, the *evaluation function*, the *neighborhood* structure, the *incremental evaluation* of neighbors, and the determination of the *initial solution*.

2.2.1 Representation

Designing any metaheuristic needs a representation which encodes the solutions of the search space S according to the target optimization problem. It is a fundamental design question in the development of metaheuristics. The representation plays a major role in the efficiency and effectiveness of any metaheuristic and then constitutes an essential step in designing a metaheuristic. The representation must be suitable and relevant to the tackled optimization problem. Moreover, the efficiency of a representation is also related to the search operators applied on solutions (Rothlauf, 2006). In fact, when defining a representation, one has to bear in mind how the solution will be evaluated and how the search operators which defines the neighborhood will operate.

Many straightforward representations may be applied for some traditional families of optimization problems. Indeed, there exist some classical representations that are commonly used to solve a large variety of optimization problems. Those representations may be combined or underly new representations. According to their structure, there are two main classes of representations: linear and non-linear. Linear representations may be viewed as strings of symbols of a given alphabet (*e.g.* binary, permutations, continuous, discrete). Non-linear representations are in general more complex structures. They are mostly based on graph structures. Among the traditional non-linear representations, trees are the most often used.

2.2.2 Evaluation

The objective function f, also defined as the fitness function, cost function, evaluation function or utility function, formulates the goal to achieve. It associates to each solution of the search space a real value which describes the quality or the fitness of the solution: $f : S \rightarrow \mathbb{R}$. Then, it represents an absolute value and allows a complete ordering of solutions from the search space.

The objective function is an important element in designing a metaheuristic. It will guide the search towards "good" solutions of the search space. If the objective function is improperly defined, it can lead to non-acceptable solutions whatever which metaheuristic is used.

2.2.3 Neighborhood

The definition of the neighborhood is a required common step for the design of any local search metaheuristic. The neighborhood structure plays a crucial role in the performance of a local search metaheuristic. If the neighborhood structure is not adequate to the problem, any local search will fail to solve the problem.

Definition 1 *A neighborhood function N is a mapping $N : S \longrightarrow 2^S$ which assigns to each solution s of S a set of solutions $N(s) \subset S$.*

A solution $s' \in N(S)$ in the neighborhood of s is called a *neighbor* of s. In general, a neighbor is generated by the application of a *move* operator which performs a small perturbation to the solution s. The main property that must characterize a neighborhood is *locality*. Locality is the effect on the quality (fitness) when performing the move (perturbation) on the solution (Sendhoff et al, 1997). When small changes are made in the solution, the quality must reveal small changes. In this case, the neighborhood is said to have a strong locality. Hence, local search will perform a meaningful search in the landscape of the problem. Weak locality is characterized by a large effect on the quality when a small change is made in the solution. In the extreme case of weak locality, the search process tends to a random search.

The neighborhood definition depends strongly on the representation associated to the problem at hand. Some usual neighborhoods are associated to traditional representations (*e.g.* continuous, binary, discrete, permutations). Let us notice that for a given optimization problem, a local optimum for a neighborhood N_1 may not be a local optimum for a different neighborhood N_2. In designing a local search algorithm, there is often a compromise between the size (or diameter) and the quality of the neighborhood to use and the computational complexity to explore it. Designing large neighborhoods may improve the quality of the obtained solutions since more neighbors are considered at each iteration. However, this requires an additional computational time to generate and evaluate such a large neighborhood.

2.2.4 Incremental Evaluation

Often, the evaluation of the objective function is the most expensive part of any local search metaheuristic. A naive exploration of the neighborhood of a solution s is a *complete* evaluation of the objective function for every candidate neighbor s' of $N(s)$.

A more efficient way to evaluate the set of candidates is the *evaluation* $\Delta(s, m)$ of the objective function, where s is the current solution and m is the applied move. This is an important issue in terms of efficiency that must be taken into account in the design of a local search algorithm. It consists in evaluating only the transformation $\Delta(s, m)$ applied to a solution s rather than the complete evaluation of the neighbor solution $f(s') = f(s \oplus m)$. The definition of such an incremental evaluation and its complexity depends on the neighborhood used over the target optimization problem. It is a straightforward task for some problems and neighborhoods (*e.g.* TSP with 2-opt neighborhood) but it may be very difficult for other problems and/or neighborhood structures (*e.g.* VRP with the node exchange operator).

2.2.5 Initial Solution

The initial solution of a local search algorithm has a high impact on the final results. Two main strategies are used to generate the initial solution: a *random* or a *greedy* approach. There is always a trade-off between the use of random and greedy initial solutions in terms of the quality of solutions and the computational time required to generate the solution. The best answer to this trade-off will depend mainly on the efficiency and effectiveness of the random and greedy algorithms at hand, and the local search properties (Burke and Newall, 2002). For instance, the larger the neighborhood, the less sensitive the initial solution to the performance of the local search.

Generating a random initial solution is a quick operation but the local search metaheuristic may take a much larger number of iterations to converge. In order to speedup the search, a greedy heuristic may be used. Indeed, in most cases, greedy algorithms have a reduced polynomial-time complexity. Using greedy heuristics often leads to better quality local optima. Hence, the local search algorithm will require in general less iterations to converge towards a local optimum. Some approximation greedy algorithms may also be used to obtain a bound guarantee for the final solution. However, it does not mean that using better solutions as initial solutions will always lead to better local optima (Hoos and Stützle, 2004).

2.3 Fitness Landscapes Analysis

2.3.1 Parameter Setting

Generally speaking, additional information to the local search is called algorithm parameters, or simply *parameters*. The problem to choose efficient parameters for performing a particular task is *parameter setting*. Parameter setting has been extensively studied, and still one of the most critical issue in the design efficient local search algorithms. According to the taxonomy of Eiben et al (2007), there exists two types of parameter setting: the first one is *off-line*, before the actual run, often called *parameter tuning*, and the second one is *on-line*, during the run, called *parameter control*. Usually, parameter tuning is done by testing a sets of parameters, and selecting the combination of parameters that give a good performance with respect to a number of executions. In order to limit the number of executions, some parameter tuning methods have been developed. They include, among others, racing techniques (Birattari et al, 2002), CALIBRA (Adenso-Díaz and Laguna, 2006), REVAC (Nannen and Eiben, 2007), and ParamILS (Hutter et al, 2009). Obviously, such approach still may be time consuming. Another strategy consists in studying the fitness landscape of the problem under study, by computing a number of statistical measures. From those, designers may deduce the main properties of the problem under study in order to correctly tune the parameters.

2.3.2 Local Search Design using Fitness Landscape

The performance of local search algorithms is strongly related to the structure of the search space, such as the number and the distribution of local optima, the number and the size of plateaus, etc. The fitness landscape is the main model to analyze the structure of the search space. Different goals can be achieved by means of fitness landscapes analysis (Hoos and Stützle, 2004; Verel, 2009). First, an analysis can allow to compare the difficulty between different search spaces representations, local search operators, etc. Then a proper choice of the "right" search space can be made for a large class of local search algorithms, without an expensive experimental tests campaign. Second, the study of the global geometry of the landscape helps to decide the most appropriate algorithm. For example, if there is a lot of plateaus, and according to their features, we can decide to use a very explorative local search algorithm. Third, an off-line tuning of the parameters which define the local search algorithm can be guided by the fitness landscapes properties (Marmion et al, 2011a). For example, such parameters include the number of moves to be performed before a restart strategy. At last, the on-line control of parameters is the most challenging goal of fitness landscapes analysis. During the search process, the local geometry of fitness landscape can be used to control the search parameters, such as the maximum number of visited solutions in the neighborhood, or more generally the parameters which control the selection pressure. To summarize, learning about the problem structure using tools from fitness landscapes analysis leads to design better local search algorithms.

2.3.3 Definition

The definition of fitness landscapes follows the common issues for the design of local search algorithms. It provides a substantial number of tools in order to analyze the background of local search algorithms independently of the heuristic being used.

A fitness landscape (Stadler, 2002; Jones, 1995) is a triplet (S, f, N) where S is a set of *potential solutions* (also called search space), $N : S \rightarrow 2^S$, a *neighborhood* operator (see Definition 1), and $f : S \rightarrow \mathbb{R}$ is a fitness function that can be pictured as the "height" of the corresponding potential solutions. Often a topological concept of *distance* d can be associated to a neighborhood N. A distance $d : S \times S \mapsto \mathbb{R}^+$ is a function that associates with any two configurations in S a non-negative real number that satisfies well-known properties. For instance, for a binary-coded local search metaheuristic, the fitness landscape S is constituted by the boolean hypercube $B = \{0, 1\}^l$ consisting of the 2^l solutions for strings of length l and the associated fitness values. The neighborhood of a solution for the one-bit flip operator is the set of points $y \in B$ that are reachable from x by flipping one bit. A natural definition of distance for this landscape is the well-known *Hamming* distance.

Based on the neighborhood notion, one can define *local optima* as being configurations x for which (in the case of maximization): $\forall y \in N(x), f(y) \leq f(x)$. Global optima are defined as being the absolute maxima (or minima) in the whole search space S. Other features of a landscape such as basins, barriers, or neutrality can be defined likewise (Stadler, 2002).

Let us define the notion of *walk* on a landscape. A walk Γ from s to s' is a sequence $\Gamma = (s_0, s_1, \ldots, s_m)$ of solutions belonging to S where $s_0 = s$, $s_m = s'$ and $\forall i \in \{1, \ldots, m\}$, s_i is a neighbor of s_{i-1}. The walk can be random, for instance solutions can be chosen with uniform probability from the neighborhood, as in random sampling, or according to other weighted non-uniform distributions, as in Metropolis-Hasting sampling. It can also be obtained through the repeated application of a "move" operator, either stochastic or deterministic, defined on the landscape.

2.3.4 Density of States

Rosé et al (1996) developed the *density of states* approach (DOS) by plotting the number of sampled solutions in the search space with the same fitness value. Knowledge of this density allows to evaluate the performance of random search or random initialization of local search metaheuristics. DOS gives the probability of having a given fitness value when a solution is randomly chosen. The tail of the distribution at optimal fitness value gives a measure of the difficulty of an optimization problem: the faster the decay, the harder the problem.

2.3.5 Fitness Distance Correlation

Fitness distance correlation was first proposed by Jones (1995) with the aim of measuring the difficulty of problems with a single number. Jones' approach states that what makes a problem hard is the relationship between fitness and distance of the solutions from the optimum. This relationship can be summarized by calculating the *fitness-distance correlation coefficient* (FDC), which is the correlation coefficient between the fitness and the distance to the nearest global optimum for all solutions from the search space. It can be estimated based on a sample of the search space: given a sample of m solutions $\{s_1, s_2, ..., s_m\}$, the FDC is computed by:

$$FDC = \frac{cov(f(s_i)d(s_i))}{\sqrt{var(f(s_i))var(d(s_i))}}$$

where d gives the distance function to the nearest global optimum, $cov(f(s_i)d(s_i))$ is the covariance of f and d, and $var(f(s_i))$ and $var(d(s_i))$ are respectively the variance of f and d over the sample of m solutions. Thus, by definition, FDC $\in [-1, 1]$. As we hope that fitness increases as distance to a global optimum decreases (for maximization problems), we expect that, with an ideal fitness function, FDC will assume the value of -1. According to Jones (1995), search problems can be classified into three classes, depending on the value of the FDC coefficient:

- *Misleading* ($FDC \geq 0.15$), in which fitness increases with distance.
- *Difficult* ($-0.15 < FDC < 0.15$) in which there is virtually no correlation between fitness and distance.
- *Straightforward* ($FDC \leq -0.15$) in which fitness increases as the global optimum approaches.

The second class corresponds to problems for which the FDC coefficient does not bring any information. The threshold interval $[-0.15, 0.15]$ has been empirically determined by Jones. When FDC does not give a clear indication, *i.e.* in the interval $[-0.15, 0.15]$, examining the scatterplot of fitness versus distance can be useful.

The FDC has been criticized on the grounds that counterexamples can be constructed for which the measure gives wrong results (Altenberg, 1997; Quick et al, 1998; Clergue and Collard, 2002). Another drawback of FDC is the fact that it is not a *predictive* measure since it requires knowledge of the optima. Despite its shortcomings, we consider FDC here as another way of characterizing problem difficulty when we know some optima and we can predict whether it is easy to reach those local optima or not.

2.3.6 Autocorrelation Length and Autocorrelation Functions

Weinberger (1990, 1991) introduced the *autocorrelation function* and the *correlation length* of random walks to measure the correlation structure of fitness landscapes. Given a random walk $(s_t, s_{t+1}, \ldots)$, the autocorrelation function ρ of a fitness function f is the autocorrelation function of time series $(f(s_t), f(s_{t+1}), \ldots)$:

$$\rho(k) = \frac{E[f(s_t)f(s_{t+k})] - E[f(s_t)]E[f(s_{t+k})]}{var(f(s_t))}$$

where $E[f(s_t)]$ and $var(f(s_t))$ are the expected value and the variance of $f(s_t)$. Estimates $r(k)$ of autocorrelation coefficients $\rho(k)$ can be calculated with a time series $(s_1, s_2, \ldots, s_L)$ of length L :

$$r(k) = \frac{\sum_{j=1}^{L-k}(f(s_j) - \bar{f})(f(s_{j+k}) - \bar{f})}{\sum_{j=1}^{L}(f(s_j) - \bar{f})^2}$$

where $\bar{f} = \frac{1}{L}\sum_{j=1}^{L} f(s_j)$, and $L >> 0$. A random walk is representative of the entire landscape when the landscape is statistically isotropic. In this case, whatever the starting point of random walks and the selected neighbors during the walks, estimates of $r(n)$ must be nearly the same. The estimation error diminishes with the walk length.

The correlation length τ measures how the autocorrelation function decreases and it summarizes the ruggedness of the landscape: the larger the correlation length, the smoother the landscape. Weinberger's definition $\tau = -\frac{1}{ln(\rho(1))}$ makes the assumption that the autocorrelation function decreases exponentially.

2.3.7 Sampling Local Optima by Adaptive Walks

Escaping from local optima is one of the main issue for local search algorithms. So, the number of local optima, the size of basins of attraction of local optima, and the network of local optima (Ochoa et al, 2008) should be estimated to understand the dynamics of local search and to design efficient search algorithms.

An *adaptive walk* is a walk $(s_0, s_1, \ldots, s_m)$ where the fitness values increase during the walk: $\forall i < m$, $f(s_i) < f(s_{i+1})$. An adaptive walk stops on a local optimum. Then, the sampling of the search space with adaptive walk can be used to estimate the fitness distribution of local optima, even if its estimation is biased by the size of basins. The number of local optima, the diameter, and then, the basin of attraction sizes can be estimated with the length of the adaptive walks. When the length of adaptive walks is large, the number of local optima is low, and the diameter of basins is large.

2.3.8 Neutrality

Neutrality is a particularly important issue in real-world optimization such as flow-shop scheduling (Marmion et al, 2011b), minimum linear arrangement (Rodriguez-Tello et al, 2008), etc. The notion of neutrality has been suggested by Kimura (1983) in his study of the evolution of molecular species. According to this view, most moves are neutral (their effect on fitness is small) or lethal.

A fitness landscapes is said to be *neutral* when many neighboring solutions have the same fitness value (Reidys and Stadler, 2001). The picture of such fitness landscapes is dominated by a lot of plateaus, also called *neutral networks*. More precisely, a neutral network is a graph where the nodes are the solutions with a given fitness value, and the edges are given by the neighborhood relation between those solutions. To study the neutrality of fitness landscapes, we should be able to measure and describe a few properties of neutral networks. The number of neutral networks, the *size*, and the *diameter* of neutral networks are basic information on the neutrality, but due to the size of the search space and of neutral networks, it is not always possible to measure information for real-world problems.

The *neutral degree* of a solution is the number of neighboring solutions with the same fitness value. The neutral degree shows the importance of neutrality in the landscapes. For example, the *neutral degree distribution* of solutions *i.e.* the degree distribution of the vertices in a neutral network, gives information which plays a role in the dynamics of local search metaheuristics (Van Nimwegen et al, 1999; Wilke, 2001).

Another way to describe a neutral network is given by the *autocorrelation of neutral degree* along a neutral random walk (Bastolla et al, 2003), *i.e.* a walk over a neutral network. From neutral degree collected along this neutral walk, its autocorrelation can be computed (see section 2.3.6). The autocorrelation measures the correlation structure of a neutral network. If the correlation is low, the variation of neutral degree is low ; and so, there is some areas in the neutral network of solutions which have nearby neutral degrees.

The percolation measure of neutral networks in the landscapes, the evolvability of solutions can be used. The evolvability of a solution is the ability to have better solutions in its neighborhood. From a solution with high evolvability, a local search can find a better solution in its neighborhood. The evolvability of solutions of a neutral network gives information on the surrounding of the neutral network. For instance, the average, minimal and maximal fitness value in the neighborhood of a solution can be used as an evolvability measure.

2.3.9 Fitness Cloud

In this section, we present the *fitness cloud* (FC) standpoint, first introduced by Verel et al (2003). The fitness cloud relative to the local search operator *op* is the conditional bivariate probability density $P_{op}(Y = \tilde{\varphi} \mid X = \varphi)$ of reaching a solution of fitness value $\tilde{\varphi}$ from a solution of fitness value φ applying the operator *op*. To visualize the fitness cloud in two dimensions, we plot the scatterplot $\{(f(s), f(s')) \mid s \in S \text{ and } s' \in N(s)\}$ where N is the neighborhood based on the operator *op*. Different statistics can be computed to describe this scatter plot such as: for fitness value $f(s) = \varphi$, the average, the standard deviation, the minimum and the maximum fitness value in the neighborhood.

In general, the size of the search space does not allow to consider all possible solutions, when trying to draw a fitness cloud. Instead, we need to use samples to estimate it. Two mains ways are used to sample the search space: the uniform random sampling, or the Metropolis-Hasting sampling (Madras, 2002) which gives more importance to the most interesting solutions of the search space.

2.4 Local Search Algorithms

This section describes the main local search metaheuristics.

2.4.1 Hill-Climbing Algorithm

The hill-climbing (HC) algorithm, also referred as descent, or iterative improvement, is likely the oldest and simplest local search metaheuristic (Aarts and Lenstra, 1997; Papadimitriou and Steiglitz, 1982). A pseudo-code is given in Algorithm 2 and follows the template of Algorithm 1. It starts with a given initial solution. At each iteration, the heuristic replaces the current solution by a neighbor that improves the objective function. The search process stops when all candidate neighbors are worse than the current solution, meaning a local optimum is reached. For large neighborhoods, candidate solutions may be a subset of the neighborhood. The main objective of this restricted neighborhood strategy is to speed-up the search. Variants of hill-climbing may be distinguished according to the order in which the neighboring solutions are generated (deterministic/stochastic), and the selection strategy (selection of the neighboring solution).

Algorithm 2 Template of Hill-Climbing (HC) algorithm.

Input: Initial solution s.
repeat
  Select one solution s' in the neighborhood of s
  if $f(s')$ is better than $f(s)$ **then**
    $s \longleftarrow s'$
  end if
until s is not a local optimum
Output: solution s

In addition to the definition of the initial solution and the neighborhood, designing a basic hill-climbing algorithm has to address the selection strategy of the neighbor which will determine the next current solution. Many strategies can be applied in the selection of a better neighbor:

- **Best improvement** (steepest descent): in this strategy, the best neighbor (*i.e.* the neighbor that improves the most the cost function) is selected. The neighborhood is evaluated in a fully and deterministic manner. Hence, the exploration of the neighborhood is *exhaustive*, all possibles moves are tried for a solution to select the best neighboring solution. This type of exploration may be time-consuming for large neighborhoods.

- **First improvement**: this strategy consists in choosing the first improving neighbor that is better than the current solution. Then, an improving neighbor is immediately selected to replace the current solution. This strategy involves a partial evaluation of the neighborhood. In a *cyclic* exploration, the neighborhood is evaluated in a deterministic way following a given order for generating the neighbors. In a *random* exploration, the neighborhood is evaluated in a random order, and then a random improving neighbor is selected. In the worst case (*i.e.* when no improvement is found), a complete evaluation of the neighborhood is performed.

A compromise in terms of quality of solutions and search time may consist in using the first improvement strategy when the initial solution is randomly generated, and the best improvement strategy when the initial solution is generated using a greedy procedure. In practice, on many applications, it has been observed that the first improving strategy leads to a same quality of solutions as the best improving strategy while using a smaller computational time. Moreover, the probability of premature convergence to a local optimum is less important in the first improvement strategy.

Another important point is the acceptance criterion used to define if a solution is "better" or not. The solution is better when the fitness is strictly higher (in a maximization problem): $f(s) < f(s')$. In this case, a local optimum is defined as follows: $\forall s' \in N(s), f(s') \leq f(s)$, and the stopping condition is well-defined. For problems with plateaus (neutral problems), one can define that a solution is better when the fitness value is higher or equal: $f(s) \leq f(s')$. The search process can then continue the exploration of plateaus to find an exit solution. In that case, plateaus are local optima, and then the stopping criterion can be based on the computational resources available.

2.4.2 Escaping from Local Optima

In general, hill-climbing is a very easy method to design and implement and gives fairly good solutions very quickly. This is why it is a widely used optimization method in practice. One of the main disadvantages of hill-climbing is that it converges towards local optima. Moreover, the algorithm can be very sensitive to the initial solution, *i.e.* a large variability of the quality of solutions may be obtained for some problems. At last, there is no mean to estimate the relative error from the global optimum and the number of iterations performed may not be known in advance. Even if the complexity in practice is acceptable, the worst case complexity of hill-climbing is exponential. Hill-climbing works well if there is not too many local optima in the search space or the quality of the different local optima is more or less similar. If the objective function is highly multi-modal, which is the case for the majority of optimization problems, hill-climbing is usually not an effective method to use.

As the main disadvantage of hill-climbing algorithms is the convergence towards local optima, many alternatives algorithms have been proposed to avoid becoming stuck at local optima. Those

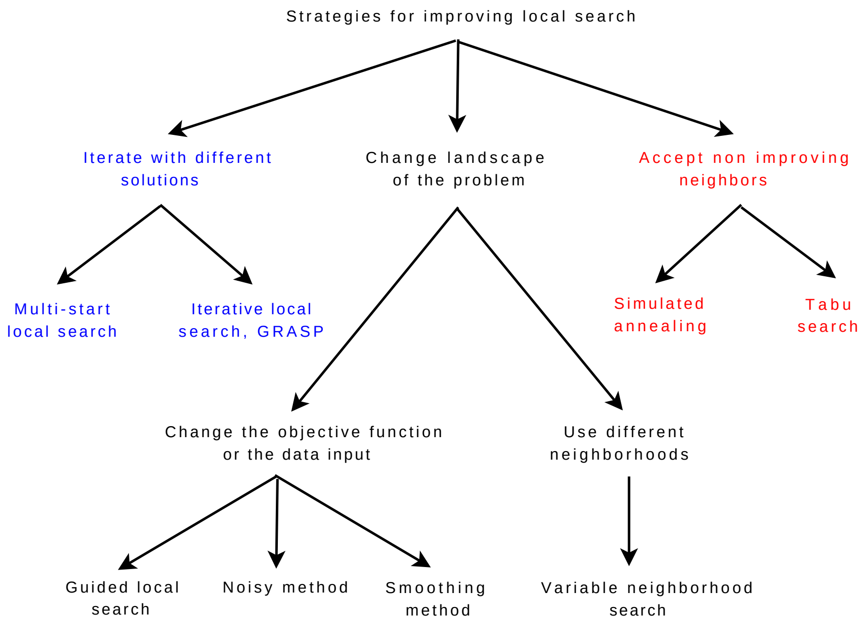

Figure 2: Local search family of algorithms for the improvement of hill-climbing and escaping from local optima.

algorithms became popular from the 1980's. Four different families of approaches can be used to escape from local optima (Fig. 2):

- **Iterating from different initial solutions:** this strategy is applied in multi-start local search (MLS), iterated local search (ILS), GRASP, and so forth.

- **Accepting non-improving neighbors:** those approaches enable moves that degrade the current solution. Then, it becomes possible to move out the basin of attraction of a given local optimum. Simulated annealing and tabu search are popular representatives of this class of algorithms. Simulated annealing was the first algorithm addressing explicitly the question "why should we consider only downhill moves?"

- **Changing the neighborhood:** this class of approaches consists in changing the neighborhood structure during the search process. For instance, this approach is used in variable neighborhood search strategies.

- **Changing the objective function or the input data of the problem:** in this class, the problem is transformed by perturbing the input data of the problem, the objective function or the constraints, in the hope to solve more efficiently the original problem. This approach has been implemented in the guided local search, the smoothing strategies and noising methods. The two last approaches may be viewed as approaches changing the fitness landscape of the problem to solve.

2.4.3 Simulated Annealing

Simulated annealing (SA) applied to optimization problems emerges from the work of Kirkpatrick et al (1983) and Cerny (1985). In those pioneering works, SA has been applied to graph partitioning and VLSI design. In the 1980's, SA had a major impact on the field of heuristic search for its simplicity and efficiency for solving combinatorial optimization problems. Then, it has been extended to deal with continuous optimization problems (Dekkers and Aarts, 1991; Ozdamar and Demirhan, 2000; Locatelli, 2000).

SA is a stochastic algorithm which enables, under some conditions, the degradation of a solution. The goal is to escape from local optima, and so to delay the convergence. SA is a memoryless algorithm in the sense that the algorithm does not use any information gathered during the search. From an initial solution, SA proceeds in several iterations. At each iteration, a random neighbor is generated. Moves that improve the cost function are always accepted. Otherwise, the neighbor is selected with a given probability which depends on the current temperature and the amount of degradation ΔE of the objective function. ΔE represents the difference in the objective value (energy) between the current solution and the generated neighbor solution. As the algorithm progresses, the probability that such moves are accepted decreases. In general, this probability follows the Boltzmann distribution:

$$P(\Delta E, T) = e^{-\frac{\Delta E}{T}}$$

It uses a control parameter, called temperature, to determine the probability of accepting non-improving solutions. At a particular level of temperature, many trials are explored. Once an equilibrium state is reached, the temperature is gradually decreased according to a cooling schedule such that few non-improving solutions are accepted at the end of the search. Algorithm 3 gives the template of the SA algorithm for maximization problems.

Algorithm 3 Template of Simulated Annealing (SA) algorithm.

Input: Initial solution s.
Set the temperature T to the initial value
repeat
  Select one random solution s' in the neighborhood of s
  $\Delta E \longleftarrow f(s) - f(s')$
  if $f(s) \leq f(s')$ or $rnd(0,1) \leq e^{\frac{-\Delta E}{T}}$ **then**
    $s \longleftarrow s'$
  end if
  Update temperature T according to the cooling schudele
until Stopping criteria satisfied
Output: Best solution found

In addition to the common design issues for hill-climbing algorithms such as the definition of the neighborhood and the generation of the initial solution, the main design issues which are specific to SA are:

- *The acceptance probability function:* it is the main element of SA which enables non-improving neighbors to be selected.
- *The cooling schedule:* the cooling schedule defines the temperature at each step of the algorithm. It has an essential role in the efficiency and the effectiveness of the algorithm.

Other similar methods to simulated annealing have been proposed in the literature such as threshold accepting, great deluge algorithm, record-to-record travel and demon algorithms (Talbi, 2009). The main objective in the design of those SA-inspired algorithms is to speedup the search of the SA algorithm without sacrificing the quality of solutions.

2.4.4 Tabu Search

Glover (1986) points out the controlled randomization in SA to escape from local optima, and proposed a deterministic algorithm. In a parallel work, a similar approach named "steepest ascent/mildest descent" has been proposed by Hansen (1986). In the 1990's, the tabu search algorithm became very popular in solving optimization problems in an approximate manner. Nowadays, it is one of the most widespread local search metaheuristic. The use of memory, which stores information related to the search process, represents the particular feature of tabu search. A comprehensive book on tabu search is (Glover and Laguna, 1997).

TS behaves like a steepest LS algorithm but it accepts non-improving solutions in order to escape from local optima when all the neighbors are non-improving solutions. Usually, the whole neighborhood is explored in a deterministic manner, whereas in SA a random neighbor is selected. As in hill-climbing, when a better neighbor is found, it replaces the current solution. When a local optimum is reached, the search carries on by selecting a candidate worse than the current solution. The best solution in the neighborhood is selected as the new current solution even if it is not improving the current solution. Tabu search may be viewed as a dynamic transformation of the neighborhood. This policy may generate cycles, *i.e.* previous visited solutions could be selected again.

To avoid cycles, TS discards the neighbors that have been previously visited. It memorizes the recent search trajectory. Tabu search manages a memory of the solutions or moves recently applied, which is called the *tabu list*. This tabu list constitutes the short-term memory. At each iteration of TS, the short-term memory is updated. Storing all visited solutions is time and space consuming. Indeed, we have to check at each iteration if a generated solution does not belong to the list of all visited solutions. Then, the tabu list often contains a constant number of tabu moves. Usually, the attributes of the moves are stored in the tabu list.

By introducing the concept of solution features or moves features in the tabu list, one may lose some information about the search memory. Then, we can reject solutions which have not yet been generated. If a move is "good", but it is tabu, do we still reject it? The tabu list may be too restrictive; a non-generated solution may be forbidden. Yet, for some conditions, called *aspiration criteria*, tabu solutions may be accepted. Then, the admissible neighbor solutions are those which are non-tabu or hold the aspiration criteria.

In addition to the common design issues for local search metaheuristics such as the definition of the neighborhood and the generation of the initial solution, the main design issues which are specific to a simple TS are:

- **Tabu list:** the goal of using the short-term memory is to prevent the search from revisiting previously visited solutions. As mentioned, storing the list of all visited solutions is not practical for efficiency issues.

- **Aspiration criterion:** a commonly used aspiration criteria consists in selecting a tabu move if it generates a solution that is better than the best found solution. Another aspiration criteria may be a tabu move that yields a better solution among the set of solutions possessing a given attribute.

Furthermore, some advanced mechanisms are commonly introduced in tabu search to deal with the intensification and the diversification of the search:

Table 1: The different search memories of tabu search.

Search memory	Role	Popular representation
Tabu list	Prevent cycling	Visited solutions, moves attributes Solutions attributes
Medium-term memory	Intensification	Recency memory
Long-term memory	Diversification	Frequency memory

- **Intensification (medium-term memory):** the medium-term memory stores the elite (*e.g.* best) solutions found during the search. Then, the idea is to give a priority to attributes of the set of elite solutions, usually based on a weighted probability. The search is biased by those attributes.
- **Diversification (long-term memory):** the long-term memory stores informations on the visited solutions along the search. Then, it explores the unvisited areas of the search space. For instance, it will discourage the attributes of elite solutions in the generated solutions in order to diversify the search to other areas of the search space.

Algorithm 4 describes the template of the TS algorithm. In addition to the search components of hill-climbing such as representation, neighborhood, initial solution, we have to define the following concepts which compose the search memory of TS: the tabu list (short-term memory), the intensification (medium-term memory), and the diversification (long-term memory), as detailed in Table 1.

Algorithm 4 Template of Tabu Search (TS) algorithm.

Input: Initial solution s.
Initialize the tabu list
Initialize the medium- and long-term memories of the intensification and the diversification procedures
repeat
  Perform intensification procedure on s
  Perform diversification procedure on s
  Select s' either, the best non-tabu solution in the neighborhood of s, or the best solution if it verifies the aspiration criterium
  if one solution s' is selected **then**
    $s \longleftarrow s'$
  end if
  Update the tabu list
  Update the medium- and long-term memories of the intensification and the diversification procedures
until Stopping criteria satisfied
Output: Best solution found

2.4.5 Iterated Local Search

The quality of the local optima obtained by a hill-climbing method depends of the initial solution. As we can generate local optima with high variability, iterated local search (ILS), also known

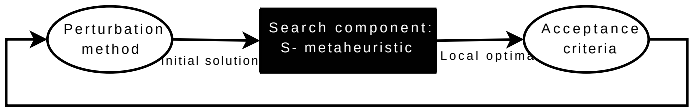

Figure 3: The search component is seen as a black-box for the ILS algorithm.

as iterated descent, large-step Markov chains, and chained local optimization, may be used to improve the quality of successive local optima. This kind of strategy has been applied first by Martin et al (1991), and then generalized by Stutzle (1999) and Lourenco et al (2002).

In *multi-start local search*, the initial solution is always chosen randomly, and then is unrelated to the generated local optima. ILS improves the classical multi-start local search by perturbing the local optima and reconsidering them as initial solutions.

ILS is based on a simple principle which has been used in many specific heuristics such as the iterated Lin-Kernighan heuristic for the traveling salesman problem (Johnson, 1990), and the adaptive tabu search for the quadratic assignment problem (Talbi et al, 1998). First a local search is applied to an initial solution (a hill-climbing algorithm or any other local search metaheuristic). Then, at each iteration, a *perturbation* of the obtained local optima is carried out. A local search is then applied on the perturbed solution. The generated solution is accepted as the new current solution under some conditions. This process iterates until a given stopping criterion. Algorithm 5 describes the ILS algorithm.

Algorithm 5 Template of the Iterated Local Search (ILS) algorithm.

Input: Initial solution s.
Initialize perturbation
repeat
  Perform perturbation on s
  Apply local search on s
  if acceptance criterium is verified **then**
    $s \longleftarrow s'$
  end if
  Update perturbation
until Stopping criteria satisfied
Output: Best solution found

Three basic elements compose an ILS:

- *Local search:* any local search metaheuristic (deterministic or stochastic) can be used within the ILS framework such as a simple hill-climbing algorithm, a tabu search or simulated annealing. The search procedure is treated as a black-box (Fig. 3). In the literature, population-based metaheuristics are excluded to be candidate in the search procedure as they manipulate populations. However, some population-based metaheuristics integrate the concept of perturbation of the (sub)population to encourage the search diversification.

- *Perturbation method:* the perturbation operator may be seen as a large random move of the current solution. The perturbation method should keep some parts of the solution and perturb strongly another part of the solution to move hopefully to another basin of attraction.

- *Acceptance criteria:* the acceptance criterion defines the conditions the new local optima must satisfy to replace the current solution.

Once the local search metaheuristic involved in the ILS framework is specified, the design of ILS will depend mainly on the used perturbation method and the acceptance criterion. Many different designs may be defined according to the various choices for implementing the perturbation method and the acceptance criterion.

2.4.6 Other Local Search Metaheuristics

Some existing local search algorithms use other strategies to escape from local optima. They are briefly described below.

- Variable Neighborhood Search (VNS) (Mladenovic and Hansen, 1997). The basic idea of VNS is to successively explore a set of predefined neighborhoods to provide a better solution. It explores either at random or systematically a set of neighborhoods to get different local optima and to escape from local optima. VNS exploits the facts that using various neighborhoods in local search may generate different local optima and that the global optimum is a local optimum for a given neighborhood. Indeed, different neighborhoods generate different fitness landscapes.

- Guided Local Search (GLS) is a deterministic local search metaheuristic which has been mainly applied to combinatorial optimization problems. Its adaptation to continuous optimization problems is straightforward given that GLS sits on top of a local search algorithm (Voudouris, 1998). The basic principle of GLS is the dynamic changing of the objective function according to the already generated local optima (Voudouris and Tsang, 1999). The features of the obtained local optima are used to transform the objective function. It allows the modification of the fitness landscape structure to be explored by a local search metaheuristic to escape from the obtained local optima.

- Search space smoothing consists in modifying the landscape of the target optimization problem (Glover and Millan, 1986; Gu and Huang, 1994). The smoothing of the landscape associated to the problem reduces the number of local optima and the depth of the basins of attraction without changing the location region of the global optimum of the original optimization problem. The search space associated to the landscape remains unchanged; only the objective function is modified. Once the landscape is smoothed by “hiding” some local optima, any local search metaheuristic (or even a population-based metaheuristic) can be used in conjunction with the smoothing technique.

- The noisy method (NM) is another local search metaheuristic which is based on the landscape perturbation of the problem to solve (Charon and Hudry, 1993). Instead of taking the original data into account directly, the NM considers that they are the outcomes of a series of fluctuating data converging towards the original ones. Some random noise is added to the objective function f. At each iteration of the search, the noise is reduced. For instance, the noise is initially randomly chosen into an interval $[-r, +r]$. The range of the interval r decreases during the search process until a value of 0. Different ways may be used to decrease the noise rate r.

- The GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic is an iterative greedy heuristic to solve combinatorial optimization problems. It has been introduced by Feo and Resende (1989). Each iteration of the GRASP algorithm contains two steps: construction and local search (Feo and Resende, 1995). In the construction step, a feasible solution is built using a randomized greedy algorithm, while in the next step a local search heuristic is applied from the constructed solution. A similar idea, known as the *semi-greedy heuristic*, was presented by Hart and Shogan (1987), where a multi-start greedy approach is proposed but without the use of local search. The greedy algorithm must be randomized to be able to generate various solutions. Otherwise, the local search procedure can be applied only once. This schema is repeated until a given number of iterations and the best found solution is kept as the final result. We notice that the iterations are completely independent, and so there is no search memory. This approach is efficient if the constructive heuristic samples different promising regions of the search space which makes the different local searches generating different local optima of "good" quality.

2.5 Summary

In addition to the representation, the objective function and constraint handling which are common search concepts to all metaheuristics, the common concepts for local search metaheuristics are (Fig. 4):

- **Initial solution:** an initial solution may be specified randomly or by a given heuristic.
- **Neighborhood:** the main concept of local search metaheuristics is the definition of the neighborhood. The neighborhood has an important impact on the performances of this class of metaheuristics. The interdependency between representation and neighborhood must not be neglected. The main design question in local search metaheuristics is the trade-off between the efficiency of the representation/neighborhood and its effectiveness (*e.g.* small versus large neighborhoods).
- **Incremental evaluation of the neighborhood:** this is an important issue for the efficiency aspect of a local search metaheuristic.
- **Stopping criteria.**

Hence, most of the search components will be reused by different local search algorithms (Fig. 4). Moreover, an incremental design and implementation of different local search metaheuristics can be carried out. In addition to the common search concepts of local search metaheuristics, the following main search components have to be defined for designing the following local search metaheuristics:

- *Hill-climbing:* neighbor selection strategy.
- *Simulated annealing, demon algorithms, threshold accepting, great deluge and record-to-record travel:* annealing schedule.
- *Tabu search:* tabu list, aspiration criteria, medium and long term memories.
- *Iterated local search:* perturbation method, acceptance criteria.
- *Variable Neighborhood search:* neighborhoods for shaking and neighborhoods for local search.

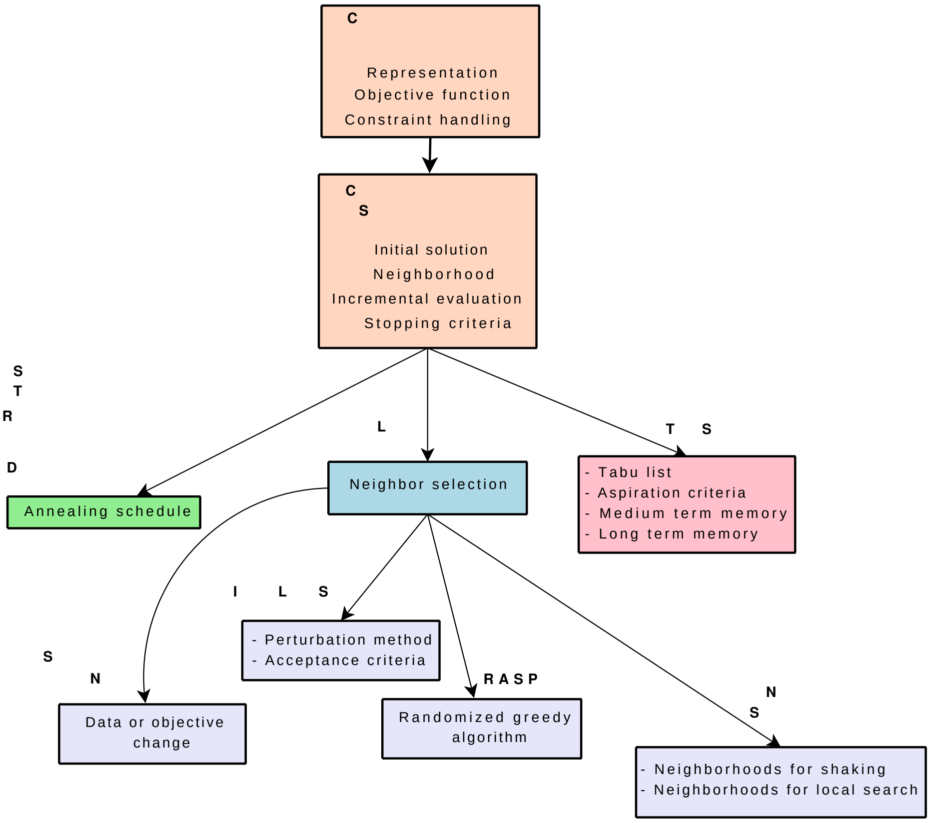

Figure 4: Common concepts and relationships in local search metaheuristics.

- *Guided local search, smoothing method, noisy method:* function changing the input data or the objective.
- *GRASP:* randomized greedy heuristic.

Moreover, there is a high flexibility to transform a local search metaheuristic to another one reusing most of the design and implementation work.

3 Design and Implementation of Local Search Algorithms under ParadisEO-MO

This sections gives a general presentation of ParadisEO, with a particular interest on the ParadisEO-MO module, dedicated to the design of local search metaheuristics and of fitness landscape analysis components.

3.1 The ParadisEO Software Framework

ParadisEO (`http://paradiseo.gforge.inria.fr`) is a white-box object-oriented software framework dedicated to the flexible design of metaheuristics for optimization problems of continuous

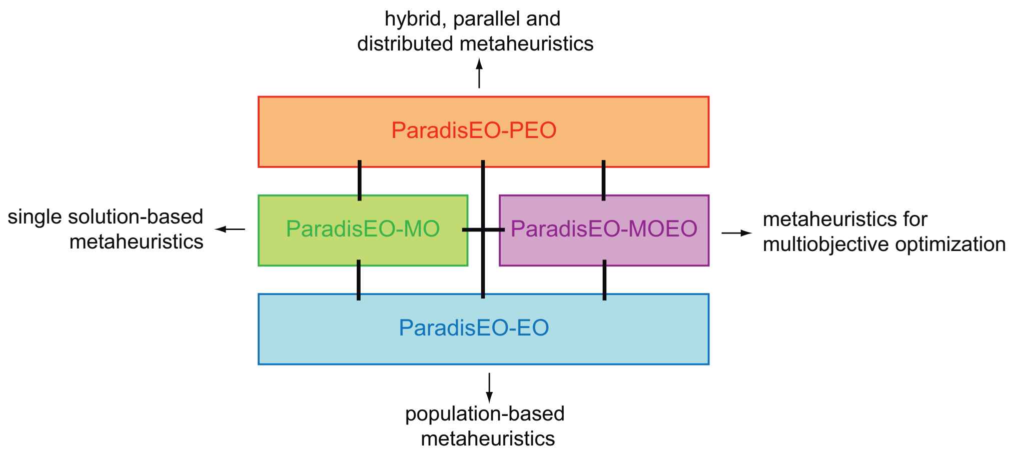

Figure 5: Interacting modules of the ParadisEO software framework.

and combinatorial nature. Based on EO (Evolving Objects, `http://eodev.sourceforge.net`) (Keijzer et al, 2001), this template-based C++ computation library is portable across both Unix-like and Windows systems. This software is governed by the CeCILL license under French law and abiding by the rules of distribution of free software (`http://www.cecill.info`). ParadisEO tends to be used both by non-specialists and optimization experts. As illustrated in Fig. 5, it is composed of four connected modules that constitute a global framework. Each module is based on a clear conceptual separation of the solution methods from the problems they are intended to solve. This separation confers a maximum code and design reuse to the user. The first module, ParadisEO-EO (Keijzer et al, 2001), provides a broad range of classes for the development of population-based metaheuristics, including evolutionary algorithms or particle swarm optimization techniques. Second, ParadisEO-MO, which is of our interest in this paper, contains a set of tools for single solution-based metaheuristics, *i.e.* hill-climbing, simulated annealing, tabu search, iterative local search, etc. Next, ParadisEO-MOEO (Liefooghe et al, 2011) is specifically dedicated to the reusable design of metaheuristics for multi-objective optimization. Finally, ParadisEO-PEO (Cahon et al, 2004) provides a powerful set of classes for the design of parallel and distributed metaheuristics: at the algorithmic-level, the iteration-level and the solution-level. In the frame of this paper, we exclusively focus on the ParadisEO-MO module.

3.1.1 Motivations

In practice, there exists a large diversity of optimization problems to be solved, engendering wide possibilities in terms of models to handle in the frame of a metaheuristic solution method. Moreover, a growing number of general-purpose search methods are proposed in the literature, with evolving complex mechanisms. From a practitioner point of view, there is a popular demand to provide a set of ready-to-use metaheuristic implementations, allowing a minimum programming effort. On the other hand, an expert generally wants to design new algorithms, to integrate new elements into an existing method, or even to combine different search mechanisms. Moreover, such a tool is of large interest in order to be able to evaluate and to compare different algorithms fairly.

Hence, as pointed out by Cahon et al (2004) and Talbi (2009), three major approaches exist for the development of metaheuristics: *from scratch* or *no reuse*, *code reuse only* and *both design* *and code reuse.* Firstly, programmers are tempted to develop and implement their own code from scratch. However, it requires time and energy and the resulting code is generally error-prone and difficult to maintain and evolve. The second approach consists of reusing a third-party source code available on the web, either as individual programs or as libraries. Individual programs often have application-dependent sections that are to be extracted before a new application-dependent code is to be inserted. Similarly, modifying these sections is often time-consuming and error-prone. Code reuse through libraries is obviously better because they are often well tried, tested, documented, and thus more reliable. However, libraries do not allow the reuse of the complete invariant part of the algorithms related to the design. Therefore, the code effort remains important. At last, both design and code reuse allow to overcome this problem. As a consequence, an approved approach for the development of metaheuristics is the use of frameworks.

A metaheuristic software framework may be defined by a set of building-blocks based on a strong conceptual separation of the invariant part and the problem-specific part of metaheuristics. Thus, each time a new optimization problem is to be tackled, both code and design can directly be reused in order to redo as little code as possible. Hence, the implementation effort is minimal with regards to the problem under investigation. Generally speaking, the constant part is encapsulated in generic or abstract skeletons that are implemented in the framework. The variable part, which is problem-specific, is fixed in the framework but must be supplied by the user. These user-defined functions are thus to be called by the framework. To do so, the design of the framework must be based on a clear conceptual separation between the resolution methods and the problem to be solved. Object-oriented design and programming is generally recommended for such a purpose. But another way to perform this separation is to provide a set of modules for each part, and to make them cooperate thought text files. However, this allows less flexibility than the object-oriented approach, and the execution is generally much more time consuming. Besides, note that two types of software frameworks can be distinguished: white-box frameworks and black-box solvers.

3.1.2 Main Characteristics

A framework is usually intended to be exploited by a large number of users. Its exploitation could only be successful if a range of user criteria are satisfied. Therefore, the main goals of the ParadisEO software framework are the following ones (Cahon et al, 2004; Talbi, 2009):

- *Maximum design and code reuse.* The framework must provide a whole architecture design for the metaheuristic approach to be used. Moreover, the programmer may redo as little code as possible. This aim requires a clear and maximal conceptual separation of the solution methods and the problem to be solved. The user might only write the minimal problem-specific code and the development process might be done in an incremental way, so that it will considerably simplify the implementation and reduce the development time and cost.

- *Flexibility and adaptability.* It must be possible to easily add new features or to modify existing ones without involving other algorithmic elements. Users must have access to source code and use inheritance or specialization concepts of object-oriented programming to derive new objects from base or abstract classes. Furthermore, as existing problems evolve and new others arise, the framework must be conveniently specialized and adapted.

- *Utility.* The framework must cover a broad range of metaheuristics, problems, parallel and distributed models, hybridization mechanisms, etc. Of course, advanced features must not add any difficulty for users wanting to implement classical algorithms.

- *Transparent and easy access to performance and robustness.* As the optimization applications are often time-consuming, the performance issue is crucial. Parallelism and distribution are two important ways to achieve high performance execution. Moreover, the execution of the algorithms must be robust in order to guarantee the reliability and the quality of the results. Hybridization mechanisms generally allow to obtain robust and better solutions.

- *Portability.* In order to satisfy a large number of users, the framework must support many material architectures (sequential, parallel, distributed) and their associated operating systems (Windows, Linux, MacOS).

- *Easy-of-use and efficiency.* The framework must be easy to use and must not contain any additional cost in terms of time or space complexity in order to keep the efficiency of a special-purpose implementation. On the contrary, the framework is intended to be less error-prone than a specifically developed metaheuristic.

3.1.3 Existing Software Frameworks for Local Search Algorithms

Several white-box frameworks for local search metaheuristics have been proposed in the literature. Most of them have the following limitations:

- Non unified view of local search algorithms: most of exiting frameworks focus only on a given local search metaheuristic or family of local search metaheuristics such as hill-climbing, *e.g.* EasyLocal++ (Gaspero and Schaerf, 2003), Localizer (Michel and Hentenryck, 2001), Opt4j (Lukasiewycz et al, 2011), or Tabu Search, *e.g.* OpenTS (COIN-OR). Only few frameworks are dedicated to the design of both families of local search metaheuristics in an incremental and unified way.

- Optimization problems: most of the software frameworks are too narrow, *i.e.* they have been designed for a given family of optimization problems: non-linear continuous optimization, combinatorial optimization (*e.g.* iOpt), single-objective optimization (*e.g.* Eva2), multi-objective optimization (*e.g.* PISA by Bleuler et al (2003)), or specific problem classes (*e.g.* Google OR-tools).

- Parallel and hybrid metaheuristics: most of the existing frameworks do not provide hybrid and parallel local search algorithms at all.

- Architectures: it is seldom to find a framework which can target many types of sequential or parallel and distributed architectures: shared-memory (*e.g.* multi-core, GPUs), distributed-memory (*e.g.* clusters, network of workstations), large-scale distributed architectures (*e.g.* desktop grids and high-performance grids). Some software frameworks are dedicated to a given type of parallel architectures, *e.g.* MALLBA (Alba et al, 2002), MAFRA (Krasnogor and Smith, 2000), TEMPLAR (Jones et al, 1998; Jones, 2000).

- Fitness landscapes: Only two frameworks, ParadisEO and EasyAnalyzer (Di Gaspero et al, 2007), which is a plug-in to EasyLocal++, propose tools for fitness landscape analysis. We can also mention Viz (Halim et al, 2007), that allows to visually analyze local search metaheuristics, but do *not* propose tools for fitness landscape analysis.

Table 2 illustrates the characteristics of the main white-box software frameworks for metaheuristics. Of course, we do not claim an exhaustive comparison. For a more detailed review of software frameworks and libraries for metaheuristics, the reader may refer to Voss and Woodruff (2002)

Table 2: Main characteristics of some white-box software frameworks for metaheuristics (S-meta: single solution-based metaheuristics, P-meta: population-based metaheuristics, COP: Combinatorial optimization, Cont: Continuous optimization, Mono: Mono-objective optimization, Multi: Multi-objective optimization, HC: Hill-climbing, TS: Tabu Search, GA: Genetic algorithm, CP: Constraint Programming, Algo-level: Algorithmic-level of parallel model, Ite-level: Iteration-level of parallel models, Sol-level: Solution-level of parallel models).

Framework or library	Metaheuristics available	Optimization problems	Parallel models	Fitness landscapes
EasyLocal++ & EasyAnalyzer	S-meta	Mono	-	yes
Eva2	SA	Mono	-	-
FOM	S-meta	Mono	-	-
Google OR-tools	S-meta	Mono	-	-
Hotframe	S-meta	Mono	-	-
iOpt	S-meta, GA, CP	Mono, COP	-	-
Localizer++	S-meta	Mono	-	-
MALLBA	HC	Mono	Algo-level Ite-level	-
MAFRA	HC	Mono	-	-
MAGMA	S-meta	Mono	-	-
OpenTS	TS	Mono	-	-
Opt4J	SA	Mono	-	-
OptQuest	HC	Mono	-	-
TEMPLAR	HC, SA	Mono, COP	Algo-level	-
ParadisEO	S-meta P-meta	Mono, Multi COP, Cont	Algo-level Ite-level Sol-level	yes

or Parejo et al (2012). Most of the available frameworks or libraries are not maintained anymore (*e.g.* Hotframe, MALLBA, MAFFRA, TEMPLAR). Very few frameworks are widely used and organized into social networks (*e.g.* ParadisEO). There are also some frameworks for what an executable version or source code could not be obtained (*e.g.* iOpt, MAGMA, OptQuest).

It is also worth mentioning two black-box local search-based solvers. First, LocalSolver (Benoist et al, 2011) is a black-box local search solver for 0-1 integer models. A mathematical modeling language is proposed, and an adaptive simulated annealing algorithm is used as the main search heuristic. Despite of being a black-box solver, it provides an object-oriented application programming interface in different programming languages (C++, java, .NET). Second, Comet (Michel et al, 2009) is a commercial programming language used to solve combinatorial optimization problems in areas such as resource allocation and scheduling. Comet combines mathematical programming, constraint programming, and local search algorithm to solve combinatorial optimization problems.

With respect to the 'Programming by Optimization' (PbO) framework from Hoos (2012), ParadisEO falls in the third level of compliance. As pointed out by the author, one key issue to solve challenging optimization problem lies in the combination of design choices; that is, in the context of local search metaheuristics, the choice of representation, neighborhood, and so on. The PbO approach is based on *"the idea of avoiding premature commitment of certain design choices and actively developing promising alternatives for parts of the design"* (Hoos, 2012). PbO seeks at optimizing the performance of a software over a large design space of programs accomplishing a given computational task (this task may or may not relate to the context of optimization problem solving). Hoos (2012) identifies five levels of PbO: at level 0,

Algorithm 6 General Local Search Algorithm

searchExplorer.**initParam**(solution)
continuator.**init**(solution)
repeat
  searchExplorer.**generateSelect**(solution)
  if searchExplorer.**accept**(solution) **then**
    searchExplorer.**move**(solution)
  end if
  searchExplorer.**updateParam**(solution)
until (continuator(solution) **AND** searchExplorer.**continue**(solution))
searchExplorer.**terminate**(solution)

the parameters are set to a specific context; at level 1, the design choices hardwired into a given code are explicitly exposed; at level 2, the design choices are considered and actively kept and exposed to the user; at level 3, the software-development process is structured and carried out in a way that seeks to provide design choices and alternatives in many performance-relevant components; at level 4, all the design choices cannot be made prematurely, and can possibly be set during the optimization process by the user. ParadisEO provides design choices and alternatives at *many* parts of a metaheuristic development project, specifically for performance-related components. This corresponds to level 3 in PbO. Moreover, as argued in PbO, it is worth to mention that ParadisEO has been recently used in conjunction with a racing algorithm to automatically identify a well-performing local search metaheuristic configuration for solving a combinatorial optimization problem from scheduling; see Marmion et al (to appear) for details.

3.2 Algorithmic Components

Technical details on the implementation of local search algorithms under ParadisEO-MO can be found at the following URL: `http://paradiseo.gforge.inria.fr`. In addition, a complete documentation and many examples of use are provided. The high flexibility of the framework and its modular architecture based on the main local search design issues allows to implement efficient algorithms in solving a large diversity of problems. The granular decomposition of ParadisEO-MO is based on the conceptual model introduced in the previous section. ParadisEO is an object-oriented application, so that its components can be specified by the UML standard. UML (Unified Modeling Language) is a standard modeling language in object-oriented software engineering.

3.2.1 Local Search

The general local search algorithm as implemented in ParadisEO-MO is given in Algorithm 6. Existing approaches require specific parameters than can be set independently from the local search process. An iteration of the algorithm consists in exploring the neighborhood of the current solution and selecting one neighbor. Next, the acceptance criteria is tested, and the current solution is modified accordingly. Then, the possible local search parameters are updated with respect to the current state of the search process and a continuation condition is checked. The search explorer is based on the definition of a specific neighborhood for the problem under study, as well as an evaluation function. It is driven by a specific strategy, so that local search algorithms can now be viewed as simple instances of this conceptual model.

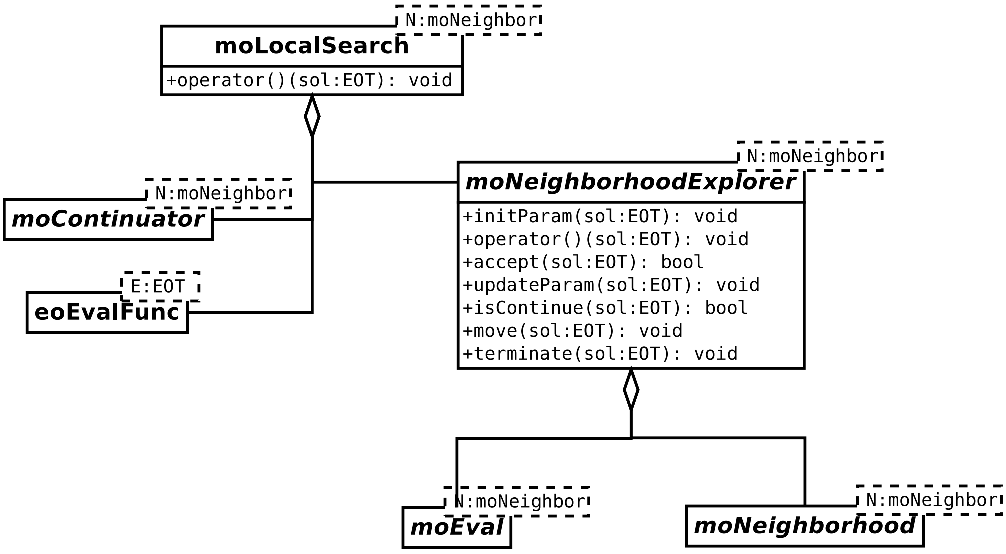

Figure 6: Simplified UML diagram for the design of local search algorithms.

Main UML classes. In order to instantiate a given local search approach for the problem under study, the main classes to be implemented are:

- `EO` for solution representation, coming from the EO module (Keijzer et al, 2001).
- `eoEvalFunc` and `moEval` for evaluation of solutions and neighbors (complete and incremental), respectively.
- `moNeighbor` and `moNeighborhood` for defining a neighbor and a neighborhood, respectively.

Those classes follow the main design issues identified in Section 2, The UML diagram of local search algorithms as implemented in the ParadisEO-MO framework is given in Fig .6. The UML diagram of the whole ParadisEO-MO software framework is omitted due to space limitation, but is available on the website. `moLocalSearch` is the main class which implements Algorithm 6. Different local search approaches can be defined by means of the `moNeighborhoodExplorer` abstract class. The different local search variants as defined below are implemented as specific implementations of `moNeighborhoodExplorer`.

Local search algorithms available. Based on this very general algorithm, a large number of local search strategies is included in ParadisEO-MO:

- Hill-climbing algorithms (best-improvement HC, first-improvement HC, random first-improvement HC, neutral HC)
- Walk-like algorithms to sample the search space (random walk, random neutral walk and Metropolis-Hasting)
- Tabu search (including medium-term and long-term memories)
- Simulated annealing (including multiple cooling scheduling strategies)

- Iterated local search,
- Variable neighborhood search.

These algorithms are based on a simple combination of the ParadisEO-MO building-blocks. They are implemented in such a way that a minimum number of problem- or algorithm-specific parameters are required. These easy-to-use algorithms also tends to be used as references for a fair performance comparison in the academic world, even if they are also well-suited for a straight use to solve real-world optimization problems. In comparison to the previous version of the framework, the modularity has been largely improved, together with an easier reuse of basic components. Different operators can be experimented without engendering significant modifications in terms of code writing. A wide range of strategies are already provided, but this list is not exhaustive as the framework perpetually evolves and offers all that is necessary to develop new ones with a minimum effort. Indeed, ParadisEO is a white-box framework that tends to be flexible while being as user-friendly as possible.

Problem-related components available. ParadisEO-MO also provides many components for classical problem representations, like bit-strings and permutations. As well, many neighborhood structures are defined for such problems, *i.e.* k-flip for bit-strings; k-swap, k-exchange, two-opt, and insertion for permutations. Moreover, both a complete and an incremental evaluation functions are provided for many academic optimization problems, including OneMax, MaxSAT, traveling salesman problem, quadratic assignment problem, permutation flowshop scheduling problem, NK-landscapes, etc. For instance, to instantiate a local search algorithm for a new permutation-based problem, it is possible to use standard operators for representation, initialization and neighborhood so that the evaluation function is the single component to be implemented. Once this is provided, the user can use any algorithm (HC, SA, TS, ILS, VNS) or any tool from fitness landscapes for his/her own optimization problem.

3.2.2 Fitness Landscapes

Another feature of the ParadisEO-MO software framework relates to sampling and statistical tools for fitness landscape analysis. Indeed, many checkpointing mechanisms have been introduced and clearly adapted to local search principles. This checkpointing process is called at each iteration of the local search algorithm through the component related to the stopping condition (Algorithm 6, Line 9). Statistical tools include neighborhood-related statistics (minimum, maximum, mean and standard deviation of neighboring solutions, probability to increase, neutral degree, and so on), general-purpose statistics (fitness of the current solution, number of iterations, evaluations, best found so far, etc.). The evaluation of all these values can now be printed onto output files. Thanks to all those statistical values, it is now possible to sample the fitness landscape in order to compute the density of states, the ruggedness by autocorrelation, the fitness-distance correlation, the fitness distribution of local optima, the length of adaptive walks, the fitness cloud, the neutral degree distributions and other statistics based on random neutral walks.

Only the main principle of fitness landscape analysis in ParadisEO-MO are reported in the paper. The technical details are explained in the tutorial available on the website of ParadisEO. In particular, one lesson explains how to easily perform a fitness landscape analysis with all the tools available within the framework using the same components than the local search metaheuristic presented in other lessons.

3.3 Discussion

We believe that the aforementioned characteristics make from ParadisEO a valuable tool for both researchers and practitioners, and a unique software framework in comparison to existing ones. Indeed, it includes many state-of-the-art local search algorithms. The rich set of ParadisEO modular ingredients has serve as building-blocks to implement these methods. The related source code of ParadisEO, that contains more than 50000 lines of code, is maintained and regularly updated by the developers. Since October 2006, ParadisEO has been downloaded more than 20000 times, and more than 250 users are registered on the mailing-list (`paradiseo-users@lists.gforge.inria.fr`). Moreover, many examples and tutorials of local search algorithms and fitness landscapes analysis, as well as a complete documentation of the application programming interface, are available on the ParadisEO website (`http://paradiseo.gforge.inria.fr`). The tutorials related to ParadisEO-MO available on the website goes from the implementation of an hill-climbing algorithm to fitness landscape analysis, and help the user to incrementally incorporate advanced features related to neighborhood, simulated annealing, tabu search, iterated local search and even hybridization between local search and evolutionary algorithms.

According to a recent survey on software frameworks for metaheuristics (Parejo et al, 2012), ParadisEO is competitive in terms of supported metaheuristics, problem adaptation/encoding, advanced metaheuristic characteristics, design, implementation and licensing, as well as documentation, samples and popularity. Overall, ParadisEO ranks second over ten *selected* software frameworks (Parejo et al, 2012), behind ECJ (White, 2012). However, let us emphasize that ECJ does not provide any local search metaheuristic, but is specialized onto evolutionary computation algorithms only.

ParadisEO gives the possibility to design and implement a wide number of new resolution methods, either sequential or parallel, just by combining existing elements in an innovative way, or by implementing original ones. Moreover, it can serve as a reference implementation in order to compare different algorithms fairly. For instance, whenever a new algorithm is proposed, its efficiency can be experimentally demonstrated by comparing its behavior with existing ones. On the other hand, ParadisEO is also a practical tool that can be used to tackle an original optimization problem. The implementation of efficient programs is highly facilitated so that the user only has to focus on problem-related issues of representation, initialization, evaluation and neighborhood. The implementation effort is even more reduced when a classical solution representation can be applied for the problem under consideration, *i.e.* a binary or a permutation-based encoding. For such problems, the development and time cost is reduced to minimum since the evaluation function is the single element to be implemented. Of course, this cost is always related to the proficiency of the programmer in charge of the implementation. Once this evaluation function is available, the user only has to instantiate any local search algorithm (hill-climbing, simulated annealing, tabu search...) to obtain a powerful resolution program that is able to run on a large range of material architectures (sequential, cluster, grid, GPU) and their associated operating systems (Windows, Linux, MacOS). Though, for more sophisticated solution encodings, the development cost remains substantial with respect to the complexity of the underlying representation and to the level of expertise of the programmer. But it will always be lower than implementing a whole specific algorithm from scratch. At last, hybrid metaheuristics like memetic algorithms (Talbi, 2009) can be conveniently designed by combining components from the different modules of ParadisEO. Moreover, starting from a single-objective optimization problem implemented within ParadisEO, it is a commonplace to investigate a multiobjective variant by means of the ParadisEO-MOEO module (Liefooghe et al, 2011). In particular, multiobjective local search algorithms are also provided (Liefooghe et al, 2012).

Finally, the ParadisEO-MO tools for fitness landscape analysis and local search algorithms have been validated on a large range of optimization problems from both academic and real-world fields, including vehicle routing (Lecron et al, 2010), scheduling (Marmion et al, 2011b), packing (Khanafer et al, 2012), NK-landscapes (Ochoa et al, 2010), quadratic assignment problem (Daolio et al, 2010), and bio-informatics (Boisson et al, 2011), among many others.

4 Conclusions

Designing software frameworks for local search algorithms is primordial. In practice, there is a large diversity of optimization problems. Moreover, there is a continual evolution of the models associated to optimization problems. The problem may change or needs further refinements. Some objectives and/or constraints may be added, deleted or modified. In general, the efficient solving of a problem needs to experiment many solving methods, tuning the parameters of each metaheuristic, etc. Moreover, the metaheuristic domain is also evolving in terms of new algorithms. More and more increasingly complex local search algorithms are developed (*e.g.* hybrid strategies, parallel models).

There is a clear need to provide a ready-to-use implementation of metaheuristics. It is important for application engineers to choose, implement and apply state-of-the-art algorithms without in-depth programming knowledge and expertise in optimization. For optimization experts and developers, it is useful for them to evaluate and compare fairly different algorithms, transform ready-to-use algorithms, design new algorithms, combine and parallelize algorithms.

ParadisEO-MO has been completely designed in order to provide, at the same time, *a priori*, *a posteriori* and on-line tools of analysis and efficient local search implementations. This makes from ParadisEO a unique software framework in the metaheuristics community. All these features have been documented, tested and validated on various problems from routing, assignment, packing, and scheduling. A number of tutorials with many examples of use are available on the website. In future works, we plan to extend the framework to adaptive search metaheuristics based on on-line fitness landscape analysis.

Once a local search algorithm is designed, the ParadisEO-MO software framework allows to implement it easily. The architecture modularity reduces the time and the complexity of designing local search metaheuristics. An expert user can, without difficulty, extend the already available building-blocks in order to more suit to the problem, and then to obtain better performance. Nevertheless, ParadisEO-MO can be used by newbies with a minimum of code to produce in order to implement diverse search strategies. A natural perspective is to evolve the open-source software by integrating more search components, heuristics and problem solving environments (*e.g.* logistics, transportation, energy production). Moreover, the ParadisEO-MO module has been recently extended to run under GPU (Melab et al, 2011).

The fitness landscape analysis of optimization problems is an important aspect in designing a local search algorithm. It is one of the most challenging problem in the theory of heuristic search algorithms. Indeed, the properties of the landscape has an important impact on the performance of local search metaheuristics. They have a major role in describing, explaining and predicting the behavior of local search metaheuristics. One of the main lessons to learn is to analyze and exploit the structural properties of the landscape associated to a problem class. One can also modify the landscape by changing the representation/neighborhood structure or the guiding function so that it becomes "easier" to solve (*e.g.* deep valley landscape).

One of the most important perspective is the automatic parameter setting. Indeed, many parameters have to be tuned for any local search algorithm. Parameter setting may allow a larger flexibility and robustness, but requires a careful initialization. Those parameters may have a great influence on the efficiency and effectiveness of the search. It is not obvious to define off-line or on-line which parameter setting should be used. The optimal values for the parameters depend mainly on the problem and even the instance to deal with and on the search time that the user wants to spend in solving the problem.

Acknowledgements. We would like to thank the Inria research institute for their support on the DOLPHIN project. Thanks also to all the members of the DOLPHIN research group for their collaboration in the development of the ParadisEO framework.

References

Aarts EHL, Lenstra JK (1997) Local search in combinatorial optimization. John Wiley

Adenso-Díaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research 54(1):99–114

Alba E, Almeida F, Blesa M, Cotta C, Díaz M, Dorta I, Gabarró J, González J, León C, Moreno L, Petit J, Roda J, Rojas A, Xhafa F (2002) MALLBA: A library of skeletons for combinatorial optimisation. In: Parallel Processing Conference (Euro-Par 2002), LNCS, vol 2400, Springer-Verlag, Berlin Heidelberg, pp 927–932

Altenberg L (1997) Fitness distance correlation analysis: an instructive counterexemple. In: Bäck T (ed) Seventh Int. Conf. on Genetic Algorithms, Morgan Kaufmann, pp 57–64

Bastolla U, Porto M, Roman HE, Vendruscolo M (2003) Statiscal properties of neutral evolution. Journal Molecular Evolution 57(S):103–119

Benoist T, Estellon B, Gardi F, Megel R, Nouioua K (2011) LocalSolver 1.x: a black-box local-search solver for 0-1 programming. 4OR: A Quarterly Journal of Operations Research 9:299–316

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, GECCO '02, pp 11–18

Bleuler S, Laumanns M, Thiele L, Zitzler E (2003) PISA - a platform and programming language independent interface for search algorithms. In: Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Faro, Portugal, pp 494–508

Boisson JC, Jourdan L, Talbi EG (2011) Metaheuristics based de novo protein sequencing: A new approach. Applied Soft Computing 11(2):2271–2278

Burke E, Newall J (2002) Enhancing timetable solutions with local search methods. In: Practise and Theory of Automated Timetabling IV (PATAT 2002, Gent, Belgium). LNCS, Vol. 2740, IEEE Press, Springer, pp 195–206

Cahon S, Melab N, Talbi EG (2004) ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10(3):357–380

Cerny V (1985) A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications 45:41–51

Charon I, Hudry O (1993) The noising method: A new method for combinatorial optimization. Operations Research Letters 14:133–137

Clergue M, Collard P (2002) GA-hard functions built by combination of trap functions. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), IEEE Press, pp 249–254

Daolio F, Verel S, Ochoa G, Tomassini M (2010) Local optima networks of the quadratic assignment problem. In: Proceeding of IEEE world conference on computational intelligence (WCCI), Barcelona, Spain, pp 3145 – 3152

Dekkers A, Aarts E (1991) Global optimization and simulated annealing. Mathematical Programming 50:367–393

Di Gaspero L, Roli A, Schaerf A (2007) Easyanalyzer: an object-oriented framework for the experimental analysis of stochastic local search algorithms. In: International conference on Engineering stochastic local search algorithms (SLS 2007), Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, pp 76–90

Eiben AE, Michalewicz Z, Schoenauer M, Smith JE (2007) Parameter control in evolutionary algorithms. In: Lobo FG, Lima CF, Michalewicz Z (eds) Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, vol 54, Springer, pp 19–46

Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8:67–71

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. Journal of Global Optimization 6:109–133

Gaspero LD, Schaerf A (2003) EasyLocal++: An object-oriented framework for flexible design of local search algorithms. Software — Practice & Experience 33(8):733–765

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Ops Res 13(5):533–549

Glover F, Laguna M (1997) Tabu Search. Kluwer Academic, Dordrecht

Glover F, Millan CM (1986) The general employee scheduling problem: An integration of MS and AI. Computers and Operations Research 13(5):563–573

Gu J, Huang X (1994) Efficient local search with search space smoothing: a case study of the traveling salesman problem. IEEE Transactions on Systems Man and Cybernetics 24(5):728–735

Halim S, Yap RHC, Lau HC (2007) An integrated white+black box approach for designing and tuning stochastic local search. In: 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), Springer, Lecture Notes in Computer Science, vol 4741, pp 332–347

Hansen P (1986) The steepest ascent mildest descent heuristic for combinatorial programming, congress on Numerical Methods in Combinatorial Optimization, Capri, Italy

Hart JP, Shogan AW (1987) Semi-greedy heuristics: An empirical study. Operations Research Letters 6(3):107–114

Hoos H, Stützle T (2004) Stochastic Local Search: Foundations and Applications. Morgan Kaufmann

Hoos HH (2012) Programming by optimization. Communications of the ACM 55(2):70–80, DOI 10.1145/2076450.2076469

Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: an automatic algorithm configuration framework. J Artif Int Res 36(1):267–306

Johnson DS (1990) Local optimization and the travelling salesman problem. In: 17th Colloquium on Automata, Languages and Programming, LNCS No.443, Springer, Berlin, pp 446–461

Jones M (2000) A object-oriented framework for the implementation of search techniques. PhD thesis, University of East Anglia

Jones M, McKeown G, Rayward-Smith V (1998) Templar: A object-oriented framework for distributed combinatorial optimization. In: Proc. of the UNICOM Seminar on Modern Heuristics for Decision Support, UNICOM Ltd, Brunel university, UK

Jones T (1995) Evolutionary algorithms, fitness landscapes and search. PhD thesis, University of New Mexico, Albuquerque

Keijzer M, Merelo JJ, Romero G, Schoenauer M (2001) Evolving objects: A general purpose evolutionary computation library. In: 5th International Conference on Artificial Evolution (EA 2001), Le Creusot, France, pp 231–244

Khanafer A, Clautiaux F, Hanafi S, El-Ghazali T (2012) The min-conflict packing problem. Computers & Operations Research 39:2122–2132

Kimura M (1983) The Neutral Theory of Molecular Evolution. Cambridge University Press, Cambridge, UK

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680

Krasnogor N, Smith J (2000) MAFRA: A Java memetic algorithms framework. In: Data mining with evolutionary algorithms, Las Vegas, Nevada, USA, pp 125–131

Lecron F, Manneback P, Tuyttens D (2010) Exploiting grid computation for solving the vehicle routing problem. In: 2010 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), pp 1–6

Liefooghe A, Jourdan L, Talbi EG (2011) A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. European Journal of Operational Research 209(2):104–112

Liefooghe A, Humeau J, Mesmoudi S, Jourdan L, Talbi EG (2012) On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics 18(2):317–352

Locatelli M (2000) Simulated annealing algorithms for continuous global optimization: convergence conditions. Journal of Optimization Theory and Applications 29(1):87–102

Lourenco HR, Martin O, Stutzle T (2002) Handbook of metaheuristics, Operations Research and Management Science, vol 57, Kluwer Academic Publishers, chap Iterated local search, pp 321–353

Lukasiewycz M, Glaß M, Reimann F, Teich J (2011) Opt4J - A Modular Framework for Metaheuristic Optimization. In: Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011), Dublin, Ireland

Madras N (2002) Lectures on Monte Carlo Methods. American Mathematical Society, Providence, Rhode Island

Marmion ME, Dhaenens C, Jourdan L, Liefooghe A, Verel S (2011a) NILS: a Neutrality-based Iterated Local Search and its application to Flowshop Scheduling. In: Merz P, Hao JK (eds) Evolutionary Computation in Combinatorial Optimization, Springer, Turino, Italie, Lecture Notes in Computer Science, vol 6622, pp 191–202

Marmion ME, Dhaenens C, Jourdan L, Liefooghe A, Verel S (2011b) On the neutrality of flowshop scheduling fitness landscapes. In: 5th Learning and Intelligent OptimizatioN Conference (LION 5), Springer, Rome, Italy, Lecture Notes in Computer Science, vol 6683, pp 238–252

Marmion ME, Mascia F, López-Ibáñez M, Stützle T (to appear) Automatic design of hybrid stochastic local search metaheuristics. In: Hybrid Metaheuristics (HM 2013), Springer, Lecture Notes in Computer Science

Martin O, Otto S, Felten EW (1991) Large-step markov chains for the traveling salesman problem. Complex Systems 5(3):299–326

Melab N, Luong TV, Karima B, Talbi EG (2011) Towards ParadisEO-MO-GPU: a Framework for GPU-based Local Search Metaheuristics. In: 11th International Work-Conference on Artificial Neural Networks, Springer, Torremolinos-Málaga, Espagne, Lecture Notes in Computer Science, vol 6691

Michel L, Hentenryck PV (2001) Localizer++: An open library for local search. Tech. Rep. CS-01-02, Brown University, Computer Science

Michel L, See A, Hentenryck PV (2009) Parallel and distributed local search in COMET. Computers & Operations Research 36(8):2357–2375

Mladenovic M, Hansen P (1997) Variable neighborhood search. Computers and Operations Research 24:1097–1100

Nannen V, Eiben AE (2007) Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proceedings of the 20th international joint conference on Artifical intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'07, pp 975–980

Ochoa G, Tomassini M, Verel S, Darabos C (2008) A Study of NK Landscapes' Basins and Local Optima Networks. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM New York, NY, USA, Atlanta, United States, pp 555–562, DOI 10.1145/1389095.1389204

Ochoa G, Verel S, Tomassini M (2010) First-improvement vs. best-improvement local optima networks of nk landscapes. In: Proceedings of the 11th International Conference on Parallel Problem Solving From Nature, Krakow, Poland, pp 104–113

Ozdamar L, Demirhan M (2000) Experiments with new stochastic global optimization search techniques. Computers and Operations Research 27(9):841–865

Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization: algorithms and complexity. Prentice-Hall, Inc.

Parejo JA, Ruiz-Cortés A, Lozano S, Fernández P (2012) Metaheuristic optimization frameworks: A survey and benchmarking. Soft Computing 16(3):527–561

Quick R, Rayward-Smith V, Smith G (1998) Fitness distance correlation and ridge functions. In: et al AEE (ed) Fifth Conference on Parallel Problems Solving from Nature (PPSN'98), Springer-Verlag, Heidelberg, Lecture Notes in Computer Science, vol 1498, pp 77–86

Reidys CM, Stadler PF (2001) Neutrality in fitness landscapes. Applied Mathematics and Computation 117(2–3):321–350

Rodriguez-Tello E, Hao JK, Torres-Jimenez J (2008) An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Computers & Operations Research 35(10):3331 – 3346

Rosé H, Ebeling W, Asselmeyer T (1996) The density of states - a measure of the difficulty of optimisation problems. In: Parallel Problem Solving from Nature (PPSN 1996), pp 208–217

Rothlauf F (2006) Representations for genetic and evolutionary algorithms, 2nd edn. Springer-Verlag

Sendhoff B, Kreutz M, von Seelen W (1997) A condition for the genotype-phenotype mapping: Causality. In: Proc. of the 7th Int. Conf. on Genetic Algorithms, East Lansing, MI, pp 73–80

Stadler PF (2002) Fitness landscapes. In: Biological Evolution and Statistical Physics, Springer-Verlag, Heidelberg, Lecture Notes Physics, vol 585, pp 187–207

Stutzle T (1999) Local search algorithms for combinatorial problems - analysis, algorithms and new applications. PhD thesis, DISKI - Dissertationen zur Kunstliken Intelligenz., Sankt augustin, Germany

Talbi EG (2009) Metaheuristics: from design to implementation. Wiley

Talbi EG, Hafidi Z, Geib JM (1998) A parallel adaptive tabu search approach. Parallel computing 24(14):2003–2019

Van Nimwegen E, Crutchfield J, Huynen M (1999) Neutral evolution of mutational robustness. In: Proc. Nat. Acad. Sci. USA 96, pp 9716–9720

Verel S (2009) Fitness landscapes and graphs: multimodularity, ruggedness and neutrality. In: 11th annual conference companion on Genetic and evolutionary computation conference (GECCO 2009), ACM, Montreal, Canada, pp 3593–3656

Verel S, Collard P, Clergue M (2003) Where are bottleneck in NK fitness landscapes? In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), IEEE Press, Canberra, pp 273–280

Voss S, Woodruff DL (2002) Optimization software class librairies. Kluwer

Voudouris C (1998) Guided local search - an illustrative example in function optimization. BT Technology Journal 16(3):46–50

Voudouris C, Tsang E (1999) Guided local search. European Journal of Operational Research 113(2):469–499

Weinberger ED (1990) Correlated and uncorrelatated fitness landscapes and how to tell the difference. In: Biological Cybernetics, pp 63:325–336

Weinberger ED (1991) Local properties of Kauffman's NK model, a tuneably rugged energy landscape. Physical Review A 44(10):6399–6413

White DR (2012) Software review: the ECJ toolkit. Genetic Programming and Evolvable Machines 13(1):65–67

Wilke CO (2001) Adaptative evolution on neutral networks. Bull Math Biol 63:715–730

Contents

1 Introduction **3**

2 A Conceptual Model for Local Search **4**
2.1 Local Search General Template . . . 4
2.2 Common Issues . . . 5
2.2.1 Representation . . . 5
2.2.2 Evaluation . . . 5
2.2.3 Neighborhood . . . 6
2.2.4 Incremental Evaluation . . . 6
2.2.5 Initial Solution . . . 6
2.3 Fitness Landscapes Analysis . . . 7
2.3.1 Parameter Setting . . . 7
2.3.2 Local Search Design using Fitness Landscape . . . 7
2.3.3 Definition . . . 8
2.3.4 Density of States . . . 8
2.3.5 Fitness Distance Correlation . . . 8
2.3.6 Autocorrelation Length and Autocorrelation Functions . . . 9
2.3.7 Sampling Local Optima by Adaptive Walks . . . 10
2.3.8 Neutrality . . . 10
2.3.9 Fitness Cloud . . . 11
2.4 Local Search Algorithms . . . 11
2.4.1 Hill-Climbing Algorithm . . . 11
2.4.2 Escaping from Local Optima . . . 12
2.4.3 Simulated Annealing . . . 14
2.4.4 Tabu Search . . . 15
2.4.5 Iterated Local Search . . . 16
2.4.6 Other Local Search Metaheuristics . . . 18
2.5 Summary . . . 19

3 Design and Implementation of Local Search Algorithms under ParadisEO-MO **20**
3.1 The ParadisEO Software Framework . . . 20
3.1.1 Motivations . . . 21
3.1.2 Main Characteristics . . . 22
3.1.3 Existing Software Frameworks for Local Search Algorithms . . . 23
3.2 Algorithmic Components . . . 25
3.2.1 Local Search . . . 25
3.2.2 Fitness Landscapes . . . 27
3.3 Discussion . . . 28

4 Conclusions **29**

RESEARCH CENTRE
LILLE – NORD EUROPE

Parc scientifique de la Haute-Borne
40 avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399