Local Search Based on Genetic Algorithms

Carlos García-Martínez[1] and Manuel Lozano[2]

[1] Dept. of Computing and Numerical Analysis, University of Córdoba, 14071 Córdoba, Spain.
cgarcia@uco.es

[2] Dept. of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain.
lozano@decsai.ugr.es

Abstract

Genetic Algorithms have been seen as search procedures that can quickly locate high performance regions of vast and complex search spaces, but they are not well suited for fine-tuning solutions, which are very close to optimal ones. However, genetic algorithms may be specifically designed to provide an effective *local search* as well. In fact, several genetic algorithm models have recently been presented with this aim. In this chapter, we call these algorithms *Local Genetic Algorithms*.

In this chapter, first, we review different instances of local genetic algorithms presented in the literature. Then, we focus on a recent proposal, the Binary-coded Local Genetic Algorithm. It is a *Steady-state Genetic Algorithm* that applies a *crowding replacement method* in order to keep, in the population, groups of chromosomes with high quality in different regions of the search space. In addition, it maintains an external solution (*leader chromosome*) that is crossed over with individuals of the population. These individuals are selected by using *Positive Assortative Mating*, which ensures that these individuals are very similar to the leader chromosome. The main objective is to orientate the search in the nearest regions to the leader chromosome.

We show an empirical study comparing a *Multi-start Local Search* based on the binary-coded local genetic algorithm with other instances of this metaheuristic based on local search procedures presented in the literature. The results show that, for a wide range of problems, the multi-start local search based on the binary-coded local genetic algorithm consistently outperforms multi-start local search instances based on the other local search approaches.

Key words: Local Genetic Algorithms, Local Search Procedures, Multi-start Local Search

1 Introduction

Local Search Procedures (LSPs) are optimisation methods that maintain a solution, known as *current solution*, and explore the search space by steps within its *neighbourhood*. They usually go from the current solution to a better *close* solution, which is used, in the next iteration, as current solution. This process is repeated till a stop

condition is fulfilled, e.g. there is no better solution within the neighbourhood of the current solution.

Three important LSPs are:

- *First Improvement Local Search* [3]: Replaces the current solution with a randomly chosen neighbouring solution with a better fitness value.
- *Best Improvement Local Search* [3]: Replaces the current solution with the best among all the neighbouring solutions.
- *Randomised K-opt LSP* (RandK-LS) [26, 27, 34]: Looks for a better solution by altering a variable number of k components of the current solution per iteration, i.e. the dimension of the explored neighbourhood is variable.

The interest on LSPs comes from the fact that they may effectively and quickly explore the *basin of attraction* of optimal solutions, finding an optimum with a high degree of accuracy and within a small number of iterations. In fact, these methods are a key component of metaheuristics that are *state-of-the-art* for many optimisation problems, such as *Multi-Start Local Search* (MSLS) [4], *Greedy Randomised Adaptive Search Procedures* (GRASP) [8, 41], *Iterated Local Search* (ILS) [30], *Variable Neighbourhood Search* (VNS) [35], and *Memetic Algorithms* (MAs) [36].

Genetic Algorithms (GAs) [17, 24] are optimisation techniques that use a *population of candidate solutions*. They explore the search space by *evolving* the population through four steps: *parent selection*, *crossover*, *mutation*, and *replacement*. GAs have been seen as search procedures that can locate high performance regions of vast and complex search spaces, but they are not well suited for fine-tuning solutions [14, 29]. However, the components of the GAs may be *specifically designed* and their parameters *tuned*, in order to provide an *effective local search behaviour*. In fact, several GA models have recently been presented with this aim [29, 31]. In this chapter, these algorithms are called *Local Genetic Algorithms* (LGAs).

LGAs have some advantages over classic LSPs. Most LSPs lack the ability to follow the proper path to the optimum on complex search landscapes. This difficulty becomes much more evident when the search space contains very narrow paths of arbitrary direction, also known as *ridges*. That is because LSPs attempt successive steps along orthogonal directions that do not necessarily coincide with the direction of the ridge. However, it was observed that LGAs are capable of following ridges of arbitrary direction in the search space regardless of their direction, width, or even, discontinuities [29]. Thus, the study of LGAs is a promising way to design more effective metaheuristics based on LSPs [13, 22, 29, 31, 40, 47].

The aim of this chapter is to analyse LGAs in depth. In order to do this:

- First, we introduce the LGA concept and identify its main properties.
- Second, we review different LGA instances presented in the literature.
- Finally, we focus on a recent LGA example, the *Binary-coded LGA* (BLGA) [12]. We describe an empirical study comparing a MSLS based on the BLGA with other instances of this metaheuristic based on LSPs proposed in the literature. The results show that, for a wide range of problems, the MSLS instance based on the BLGA consistently outperforms the MSLS instances based on the other local search approaches.

The chapter is organised as follows. In Sect. 2, we outline three LSPs that have been often considered in the literature to build metaheuristics based on LSPs. In Sect. 3, we introduce a brief overview about GAs. In Sect. 4, we inspect the LGA concept and review several LGA instances found in the literature. In Sect. 5, we describe an example of LGA, the BLGA. In Sect. 6, we show an empirical study comparing the performance of the MSLS based on the BLGA with other instances of the MSLS based on LSPs presented in Sect. 2. Finally, in Sect. 7, we provide some conclusions and future research directions.

2 Local Search Procedures in the Literature

LSPs are improvement heuristics that maintain a solution, known as *current solution* ($\boldsymbol{X^C}$), and search its *neighbourhood* ($N(\boldsymbol{X^C})$) for a better one. If a better solution $\boldsymbol{S} \in N(\boldsymbol{X^C})$ is found, $\boldsymbol{S}$ becomes the new $\boldsymbol{X^C}$ and the neighbourhood search starts again. If no further improvement can be made, i.e. $\nexists\, \boldsymbol{S} \in N(\boldsymbol{X^C})$ such as $\boldsymbol{S}$ improves $\boldsymbol{X^C}$, then, a local or global optimum has been found.

The interest in LSPs comes from the fact that they may effectively and quickly explore the *basin of attraction* of optimal solutions, finding an optimum with a high degree of accuracy and within a small number of iterations. The reasons for this high exploitative behaviour are:

- LSPs usually keep as $\boldsymbol{X^C}$ the best found solution so far, and
- $N(\boldsymbol{X^C})$ is composed of solutions with minimal differences from $\boldsymbol{X^C}$, i.e. LSPs perform a *local* refinement on $\boldsymbol{X^C}$.

Three important LSPs are:

- *First Improvement Local Search* (First-LS) [3]: Works by comparing $\boldsymbol{X^C}$ with neighbouring solutions. When a neighbouring solution appears better, $\boldsymbol{X^C}$ is replaced and the process starts again. If all the neighbouring solutions are worse than $\boldsymbol{X^C}$, then, the algorithm stops. In First-LS, $N(\boldsymbol{X^C})$ is usually defined as the set of solutions with minimal differences from $\boldsymbol{X^C}$, i.e. in binary-coded problems, $\boldsymbol{S}$ differs from $\boldsymbol{X^C}$ only in one bit, $\forall \boldsymbol{S} \in N(\boldsymbol{X^C})$.
- *Best Improvement Local Search* (Best-LS) [3]: Generates and evaluates all the neighbouring solutions of $\boldsymbol{X^C}$. Then, the best one replaces $\boldsymbol{X^C}$ if it is better. Otherwise, the algorithm stops. In Best-LS, $N(\boldsymbol{X^C})$ is usually defined as in First-LS.
- *Randomised K-opt LSP* (RandK-LS) [26, 27, 34]: Is a variation of the *K-opt* LSP presented in [33]. That was specifically designed to tackle binary-coded problems. Its basic idea is to find a solution by flipping a variable number of k bits in the solution vector per iteration. In each step, n (n is the dimension of the problem) solutions ($\boldsymbol{X^1}, \boldsymbol{X^2}, \ldots, \boldsymbol{X^n}$) are generated by flipping one bit of the previous solution, i.e. solution $\boldsymbol{X^{i+1}}$ is obtained by flipping one bit of the solution $\boldsymbol{X^i}$ ($\boldsymbol{X^1}$ is generated from $\boldsymbol{X^C}$). A candidate set is used to assure that each bit is flipped no more than once. Then, the best solution in the sequence is accepted as $\boldsymbol{X^C}$ for the next iteration, if it is better, otherwise the algorithm stops and returns $\boldsymbol{X^C}$.

LSPs are a key component of many metaheuristics. In order to perform a *global search*, these metaheuristics look for *synergy* between the exploitative behaviour of

the LSP and *explorative components*. In this way, while the explorative components ensure that different promising search zones are focused upon, the LSP obtains the best possible accurate solutions within those regions. Examples of metaheuristics based on LSPs are:

- *MSLS* [4]: Iteratively applies the LSP to random solutions.
- *GRASP* [8,41]: Generates randomised heuristic solutions to the specific problem, and applies the LSP to them.
- *ILS* [30]: The LSP is initially applied to a random solution. In the following iterations, the LSP is applied to solutions generated by altering previous ones.
- *VNS* [35]: The idea is similar to that of ILS. The main difference is that the solutions are lightly or strongly altered depending on whether or not the new solutions improve the best one so far.
- *MAs* [36]: They are evolutionary algorithms that apply LSPs in order to refine the individuals of the population.

3 Genetic Algorithms

GAs are general purpose search algorithms that use principles inspired by *natural genetic populations* to evolve solutions to problems [17,24]. The basic idea is to maintain a *population of chromosomes* that represent candidate solutions to the concrete problem. The GA evolves the population through a process of competition and controlled variation. Each chromosome in the population has an associated *fitness* to determine which ones are used to form new chromosomes in the competition process, which is called *parent selection*. The new ones are created using genetic operators such as *crossover* and *mutation*.

GAs have had a great measure of success in search and optimisation problems. The reason for a great part of their success is their ability to exploit the information accumulated about an initially unknown search space in order to bias subsequent searches into useful subspaces. This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical search tools (enumerative, heuristic, ...) are inappropriate, offering a valid approach to problems requiring efficient and effective search techniques.

Two of the most important GA models are the *Generational* GA and the *Steady-state* GA:

- The Generational GA [17] creates new offspring from the members of an old population, using the genetic operators, and places these individuals in a new population that becomes the old population when the whole new population is created.
- The Steady-state GA (SSGA) [44,49] is different from the generational model in that there is typically one single new member inserted into the new population at any one time. A *replacement/deletion* strategy defines which member in the current population is forced to perish (or vacate a slot) in order to make room for the new offspring to compete (or, occupy a slot) in the next iteration. The basic algorithm step of the SSGA is shown in Fig. 1.

```
SSGA()
   initialise P;
   evaluate P;

   while (stop-condition is not fulfilled)
      parents ← select two chromosomes from P;
      offspring ← combine and mutate the chromosomes in parents;
      evaluate offspring;
      R ← select an individual from P; //replacement strategy
      decide if offspring should replace R;
```

Fig. 1 Structure of a SSGA.

4 Local Genetic Algorithms

There are two primary factors in the search carried out by a GA [49]:

- *Selection pressure*. In order to have an effective search there must be a search criterion (the fitness function) and a selection pressure that gives individuals with higher fitness a higher chance of being selected for reproduction, mutation, and survival. Without selection pressure, the search process becomes random and promising regions of the search space would not be favoured over non-promising regions.
- *Population diversity*. This is crucial to a GA's ability to continue fruitful exploration of the search space.

Selection pressure and population diversity are inversely related:

- increasing selection pressure results in a faster loss of population diversity, while
- maintaining population diversity offsets the effect of increasing selection pressure.

Traditionally, GA practitioners have carefully designed GAs in order to obtain a balanced performance between selection pressure and population diversity. The main objective was to obtain their beneficial advantages simultaneously: to allow the most promising search space regions to be reached (*reliability*) and refined (*accuracy*).

Due to the flexibility of the GA architecture, it is possible to design GA models specifically aimed to provide *effective local search*. In this way, their unique objective is to obtain *accurate* solutions. In this chapter, these algorithms are named *Local Genetic Algorithms*.

LGAs present some advantages over classic LSPs. Most LSPs lack the ability to follow the proper path to the optimum on complex search landscapes. This difficulty becomes much more evident when the search space contains very narrow paths of arbitrary direction, also known as *ridges*. This is because LSPs attempt successive steps along orthogonal directions that do not necessarily coincide with the direction of the ridge. However, it was observed that LGAs are capable of following ridges of arbitrary direction in the search space regardless of their direction, width, or even, discontinuities [29]. Thus, the study of LGAs becomes a promising way to allow the design of

more effective metaheuristics based on LSPs. In fact, some LGAs were considered for this task [13, 22, 29, 31, 40, 47].

In the following sections, we explain the features of different LGA instances that may be found in the literature. In addition, we cite the corresponding metaheuristic models in which LGAs were integrated:

- μGAs [29] (Section 4.1).
- The Crossover Hill-climbing [31] (Section 4.2).
- LGAs based on female and male differentiation [13] (Section 4.3).
- LGAs as components of distributed GAs [22, 40, 47] (Section 4.4).

4.1 μGAs Working as LGAs

In [29], a *Micro-Genetic Algorithm* (μGA) (GA with a small population and short evolution) is used as LGA within a memetic algorithm. Its mission is to refine the solutions given by the memetic algorithm. It evolves a population of *perturbations* ($\boldsymbol{P}^i$), whose aptitude values depend on the solution given by the memetic algorithm.

Its main features are the following:

- It is an *elitist* Generational GA that uses roulette wheel parent selection, a ten-point crossover, and bit mutation with adaptive probabilities. In addition, by using a small population (five individuals), the μGA may achieve high selection pressure levels, which allows accurate solutions to be reached.
- The perturbation space is defined in such a way that the μGA explores a small region centred on the given solution. Thus, it offers local improvements to the given solution.

The memetic algorithm based on the μGA was tested against 12 different evolutionary algorithm models, which include a simple GA and GAs with different hill-climbing operators, on five hard constrained optimisation problems. The simulation results revealed that this algorithm exhibits good performance, outperforming the competing algorithms in all test cases in terms of solution accuracy, feasibility rate, and robustness.

We should point out that μGAs have been considered as LGAs by other authors:

- Weicai et al. [48] propose a *Multi-agent GA* that makes use of a μGA.
- Meloni et al. [32] insert a μGA in a *multi-objective evolutionary algorithm* for a class of sequencing problems in manufacturing environments.
- Papadakis et al. [38] use a μGA within a GA-based fuzzy modelling approach to generate TSK models.

4.2 A Real-coded LGA: Crossover Hill-climbing

Lozano et al. [31] propose a *Real-coded Memetic Algorithm* that uses *Crossover Hill-climbing* (XHC) as LGA. Its mission is to obtain the best possible accuracy levels to lead the population towards the most promising search areas, producing an effective refinement on them. In addition, an adaptive mechanism is employed to determine the probability with which solutions are refined with XHC.

The XHC is a Real-coded Steady-state LGA that maintains a pair of parents and performs crossover repeatedly on this pair until some number of offspring, n_{off}, is reached. Then, the best offspring is selected and replaces the worst parent, only if it is better. The process iterates n_{it} times and returns the two final current parents.

The XHC proposed may be conceived as a *Micro Selecto-Recombinative Real-coded LGA* model that employs the minimal population size necessary to allow the crossover to be applicable, i.e. two chromosomes. Although XHC can be instantiated with any crossover operator, the authors used a *self-adaptive real-parameter operator* that generates offspring according to the current distribution of the parents. If the parents are located close to each other, the offspring generated by the crossover might be distributed densely around them. On the other hand, if the parents are located far away from each other, then the offspring will be sparsely distributed.

Experimental results showed that, for a wide range of problems, the real-coded memetic algorithm with XHC operator consistently outperformed other real-coded memetic algorithms appearing in the literature.

Other studies have considered some variants of the XHC algorithm [6,7,37].

4.3 LGAs Based on Female and Male Differentiation

Parent-Centric Crossover Operators (PCCOs) is a family of real-parameter crossover operators that use a probability distribution to create offspring in a restricted search region marked by one of the parent, *the female parent*. The range of this probability distribution depends on the distance among the female parent and the other parents involved in the crossover, *the male parents*.

Traditionally, PCCO practitioners have assumed that every chromosome in the population may become either a female parent or a male parent. However, it is very important to emphasise that female and male parents have two differentiated roles [13]:

- female parents *point* to the search areas that will receive sampling points, whereas
- male parents are used to determine the *extent* of these areas.

With this idea in mind, García-Martínez et al. [13] propose applying a *Female and Male Differentiation* (FMD) process before the application of a PCCO. The FMD process creates two different groups according to two tuneable parameters (N_F and N_M):

- G_F with the N_F best chromosomes in the population, which can be female parents; and
- G_M with the N_M best individuals, which can be selected as male parents.

An important feature of this FMD process is that it has a strong influence on the degree of *selection pressure* kept by the GA:

- On the one hand, when N_F is low, high selection pressure degrees are achieved, because the search process is very focused in the best regions.
- On the other hand, if N_F is high, the selection pressure is softened, providing extensive sampling on the search areas represented in the current population.

The authors argue that the two parameters associated with the FMD process, N_F and N_M, may be adequately adjusted in order to design *Local Real-coded GAs* that reach *accurate* solutions:

- On the one hand, with low N_F values ($N_F = 5$), the GA keeps the best solutions found so far in a similar way to which LSPs keep the best solution found so far in $\boldsymbol{X}^{\boldsymbol{C}}$.
- On the other hand, PCCOs sample the neighbourhood of the N_F best solutions as LSPs sample the neighbourhood of $\boldsymbol{X}^{\boldsymbol{C}}$, i.e. PCCOs perform a *local* refinement on the N_F best solutions.

In addition, the authors argue that *Global Real-Coded GAs* can also be obtained by adequately adjusting N_F and N_M. Global Real-coded GAs offer *reliable* solutions when they tackle multimodal and complex problems.

Finally, with the aim of achieving *robust operation*, García-Martínez et al. followed a simple *hybridisation technique* to put together a Global Real-coded GA and a Local Real-coded GA.

Empirical studies confirmed that this hybridisation was very competive with *state-of-the-art on metaheuristics for continuous optimisation*.

4.4 LGAs as Components of Distributed GAs

Distributed GAs keep in parallel, several independent *subpopulations* that are processed by a GA [45]. Periodically, a *migration mechanism* produces a chromosome exchange between the subpopulations. Making distinctions between the subpopulations by applying GAs with different configurations, we obtain *Heterogeneous Distributed Genetic Algorithms* (HDGAs). These algorithms represent a promising way for introducing correct exploration/exploitation balance in order to avoid premature convergence and reach accurate final solutions.

Next, we describe three HDGA models that assign to every subpopulation a different exploration or exploitation role. In this case, the exploitative subpopulations are LGAs whose mission is to refine the solutions that have been migrated from explorative subpopulations:

- *Gradual Distributed Real-coded GAs* [22] (Sect. 4.4.1).
- *GA Based on Migration and Artificial Selection* [40] (Sect. 4.4.2).
- *Real Coded GA with an Explorer and an Exploiter Population* [47] (Sect. 4.4.3).

4.4.1 Gradual Distributed Real-coded GAs

The availability of crossover operators for real-coded GAs [21, 23] that generate different exploration or exploitation degrees makes the design of Heterogeneous Distributed Real-coded GAs based on these operators feasible. Herrera et al. [22] propose *Gradual Distributed Real-coded GAs* (GD-RCGAs) that apply a different crossover operator to each subpopulation. These operators are differentiated according to their associated exploration and exploitation properties and the degree thereof. The effect achieved is a *parallel multiresolution* with regard to the action of the crossover operators. Furthermore, subpopulations are suitably connected to exploit this multiresolution in a *gradual* way.

GD-RCGAs are based on a hypercube topology with three dimensions (Fig. 2). There are two important sides to be differentiated:

- The *front side* is devoted to exploration. It is made up of four subpopulations $E_1, \ldots, E_4$, to which exploratory crossover are applied. The exploration degree increases clockwise, starting at the lowest E_1, and ending at the highest E_4.
- The *rear side* is for exploitation. It is composed of subpopulations $e_1, \ldots, e_4$ that undergo exploitative crossover operators. The exploitation degree increases clockwise, starting at the lowest e_1, and finishing at the highest e_4. Notice that the $e_1, \ldots, e_4$ populations are LGAs that achieve different exploitation levels.

The connectivity of the GD-RCGA allows a *gradual refinement* when migrations are produced from an exploratory subpopulation toward an exploitative one, i.e., from E_i to e_i, or between two exploitative subpopulations from a lower degree to a higher one, i.e. from e_i to e_{i+1}.

Experimental results showed that the GD-RCGA consistently outperformed sequential real-coded GAs and homogeneous distributed real-coded GAs, which are equivalent to them, and other real-coded evolutionary algorithms reported in the literature.

4.4.2 GA Based on Migration and Artificial Selection

In [40], a distributed GA, called GAMAS, was proposed. GAMAS uses four subpopulations, denoted as *species I–IV*, which supply different exploration or exploitation levels by using different mutation probabilities:

- Species II is a subpopulation used for exploration. For this purpose, it uses a high mutation probability ($p_m = 0.05$).
- Species IV is a subpopulation used for exploitation. This way, its mutation probability is low ($p_m = 0.003$). Species IV is an LGA that attempts to achieve high exploitation by using a low mutation probability.
- Species III is an exploration and exploitation subpopulation with $p_m = 0.005$.

GAMAS selects the best individuals from species II–IV, and introduces them into species I whenever those are better than the elements in this subpopulation. The mis-

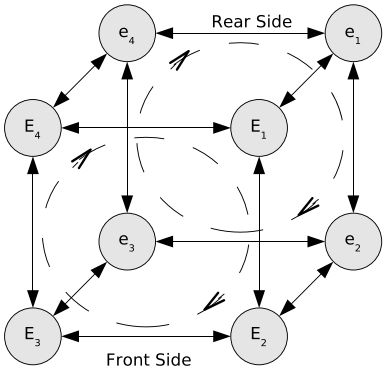

Fig. 2 Structure of a GD-RCGA

sion of species I is to preserve the best chromosomes appearing in the other species. At predetermined generations, its chromosomes are reintroduced into species IV by replacing all of the current elements in this species.

Experimental results showed that GAMAS consistently outperforms simple GAs and alleviates the problem of *premature convergence*.

4.4.3 Real-coded Genetic Algorithm with an Explorer and an Exploiter Population

Tsutsui et al. [47] propose a GA with two populations whose missions are well differentiated: one is aimed to explore the search space, whereas the other is an LGA that searches the neighbourhood of the best solution obtained so far. Both of them are generational GAs. However, the LGA uses a *fine-grained mutation* and a population of half the size of the explorer population. This way, the LGA performs a high exploitation over the best solution so far.

The proposed technique exhibited performance significantly superior to standard GAs on two complex highly multimodal problems.

5 Binary-coded Local Genetic Algorithm

In this section, we describe a recent LGA example, the Binary-coded LGA (BLGA) [12] that may be used to design metaheuristics based on LSPs. The aim of BLGA is two-fold:

- On the one hand, BLGA has been specifically designed to perform an effective local search in a similar way to LSPs. BLGA optimises locally the solutions given by the metaheuristic, by steps within their neighbourhoods.
- On the other hand, while BLGA performs the local search, its population (P) acquires information about the location of the best search regions. Then, BLGA can make use of the knowledge in P in order to guide the search. This kind of information cannot be used by LSPs.

BLGA is a SSGA (Sect. 3) that uses a *crowding replacement method* (*restricted tournament selection* [19]) that favours the formation of *niches* (groups of chromosomes of high quality located in different and scattered regions of the search space) in P. In addition, BLGA maintains an external chromosome, the *leader chromosome* ($\boldsymbol{C^L}$), which plays the same role as $\boldsymbol{X^C}$ in classical LSPs:

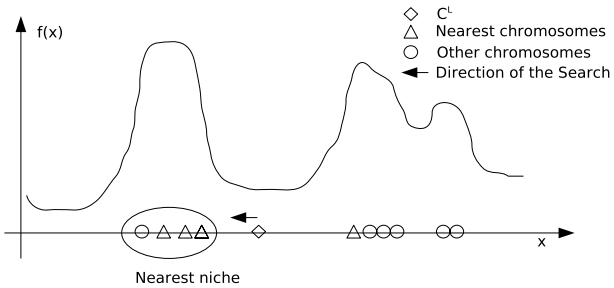

Fig. 3 Niches considered to guide the local search

- BLGA samples new solutions within the neighbourhood of $\boldsymbol{C^L}$ in a similar way to LSPs with $\boldsymbol{X^C}$, by means of a *multi-parent* version of the *uniform crossover operator* [44]. In addition, BLGA directs the sampling operation towards the closest niches to $\boldsymbol{C^L}$ (Fig. 3) by selecting parents with *positive assortative mating* [9].
- BLGA keeps the best sampled solution in $\boldsymbol{C^L}$ just as LSPs keep the best solution obtained so far in $\boldsymbol{X^C}$.

5.1 General Scheme of the BLGA

Let's suppose that a particular metaheuristic applies the BLGA as LSP. When the metaheuristic calls the BLGA to refine a particular solution, the BLGA considers this solution as $\boldsymbol{C^L}$. Then, the following steps are carried out during each iteration (Fig. 4):

1. *Mate selection.* m chromosomes ($\boldsymbol{Y^1}, \boldsymbol{Y^2}, ..., \boldsymbol{Y^m}$) are selected from the population by applying the positive assortative mating m times (Sect. 5.2).
2. *Crossover.* $\boldsymbol{C^L}$ is crossed over with $\boldsymbol{Y^1}, \boldsymbol{Y^2}, ..., \boldsymbol{Y^m}$ by applying the multi-parent uniform crossover operator, generating an offspring $\boldsymbol{Z}$ (Sect. 5.3).
3. *To update the leader chromosome and replacement.* If $\boldsymbol{Z}$ is better than $\boldsymbol{C^L}$, then $\boldsymbol{C^L}$ is inserted into the population using the restricted tournament selection (Sect. 5.4) and $\boldsymbol{Z}$ becomes the new $\boldsymbol{C^L}$. Otherwise, $\boldsymbol{Z}$ is inserted in the population using this replacement scheme.

All these steps are repeated until a termination condition is achieved (Sect. 5.5).

5.2 Positive Assortative Mating

Assortative mating is the natural occurrence of mating between individuals of similar phenotype more or less often than expected by chance. Mating between individuals with similar phenotype more often is called positive assortative mating and less often is called negative assortative mating. Fernandes et al. [9] implement these ideas to design two mating selection mechanisms. A first parent is selected by the roulette wheel method and n_{ass} chromosomes are selected with the same method (in BLGA all the candidates are selected at random). Then, the similarity between each of these

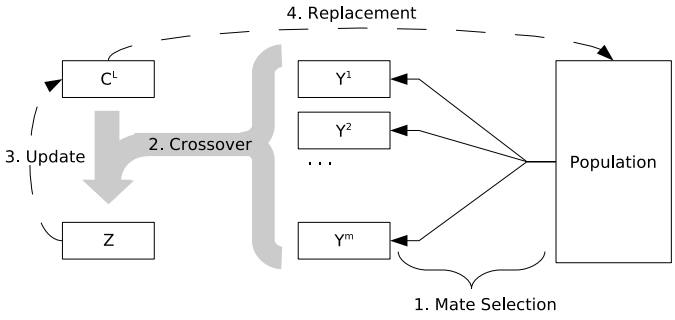

Fig. 4 Model of the BLGA

chromosomes and the first parent ($\boldsymbol{C^L}$ in the BLGA) is computed (similarity between two binary-coded chromosomes is defined as the Hamming distance between them). If assortative mating is negative, then the one with less similarity is chosen. If it is positive, the genome more similar to the first parent is chosen to be the second parent. In the case of BLGA, the first parent is $\boldsymbol{C^L}$ and the method is repeated m times, in order to obtain m parents.

Since positive assortative mating selects similar individuals to $\boldsymbol{C^L}$, it helps BLGA to achieve the two main objectives:

- Positive assortative mating helps BLGA to perform a local refinement on $\boldsymbol{C^L}$ because similar parents make the crossover operator to sample near to $\boldsymbol{C^L}$.
- Positive assortative mating probabilistically guides the search according to the information kept in P, because it probabilistically selects chromosomes from the nearest niches to $\boldsymbol{C^L}$ (see Fig. 3).

5.3 Multi-parent Uniform Crossover Operator

Since the main aim of the BLGA is to *fine-tune* $\boldsymbol{C^L}$, it should sample new points near it. *Uniform crossover* (UX) [44] creates an offspring from two parents by choosing the genes of the first parent with the probability p_f. If it uses a high p_f value, it will generate the offspring near to the first parent. The BLGA uses a *multi-parent UX* that will be defined below.

During application of the crossover operator, the BLGA uses a *short term memory* mechanism to avoid the generation of any offspring previously created. It remembers the genes of $\boldsymbol{C^L}$ that have been flipped when generating an offspring $\boldsymbol{Z^k}$. Then, it avoids flipping those genes of $\boldsymbol{C^L}$, in order to prevent the creation of $\boldsymbol{Z^k}$ once again. In order to do that, this mechanism maintains a mask, $\boldsymbol{M} = (M_1, \ldots, M_n)$, where $M_i = 1$ indicates that the ith gene of $\boldsymbol{C^L}$ (C_i^L) cannot be flipped in order to create a new offspring. Initially, and when $\boldsymbol{C^L}$ is updated with a better solution, any gene can be flipped, so M_i is set to 0 for all $i \in \{1, \ldots, n\}$.

The pseudocode of the crossover operator with short term memory is shown in Fig. 5, where $U(0,1)$ is a random number in $[0,1]$, $RI(1,m)$ is a random integer in $\{1,2,\ldots,m\}$, and p_f is the probability of choosing genes from $\boldsymbol{C^L}$. It creates the offspring $\boldsymbol{Z}$ as follows:

- Z_i is set to C_i^L for all $i = 1, \ldots, n$ with $M_i = 1$.
- If $M_i = 0$, then Z_i is set to C_i^L with probability p_f. Otherwise, Z_i is set to the ith gene of a randomly chosen parent $\boldsymbol{Y^j}$. The mask is updated if Z_i is different from C_i^L.
- Finally, if the $\boldsymbol{Z}$ obtained is equal to $\boldsymbol{C^L}$, then a gene i with $M_i = 0$ chosen at random, is flipped and the mask is updated.

Tabu Search [15] also uses a short term memory. Tabu search stores in that memory the last movements that were used to generate the current solution. It forbides those movements in order to avoid sampling previous solutions. In tabu search, each forbidden movement in the short term memory has a *tabu tenure* that indicates when it should be removed from the memory. When the tabu tenure expires, the movement is permitted again.

```
multiparent_UX(C^L, Y^1, ..., Y^m, M, p_f)

   For (i = 1, ..., n)

      If (M_i = 1  OR  U(0,1) < p_f) //short term memory mechanism
         Z_i ← C_i^L;

      Else
         k ← RI(1, m);
         Z_i ← Y_i^k;

         If (Z_i ≠ C_i^L)
            M_i ← 1  ; //update the mask

   If (Z = C^L)
      j ← RI(1, n)  such as  M_i = 0;
      M_i ← 1  ; //update the mask
      Z_i ← 1 – Z_i;

   Return Z;
```

Fig. 5 Pseudocode of the Multi-parent Uniform Crossover Operator with short term memory

5.4 Restricted Tournament Selection

BLGA considers *Restricted Tournament Selection* (RTS) [19] as its *crowding replacement method*. The application of RTS together with the use of high population size may favour the creation of groups of chromosomes with high quality in P, which become located in different and scattered regions of the search space (*niches*). In this way, the population of the BLGA acquires knowledge about the location of the best regions of the search space. The aim of the BLGA is to use this information to guide future searches.

The pseudocode of the RTS is shown in Fig. 6. Its main idea is to replace the closest chromosome $\boldsymbol{R}$ to the one being inserted in the population, from a set of n_T randomly selected ones.

```
RTS(Population, solution)

   G_T ← Select randomly n_T individuals from Population;
   R ← Choose from G_T the most similar
      chromosome to solution;

   If (solution is better than R)
      replace R with solution;
```

Fig. 6 Pseudocode of restricted tournament selection

5.5 Stop Condition

It is important to notice that when every component of the mask of the short term memory (Sect. 5.3) is equal to 1, then, $\boldsymbol{C^L}$ will not be further improved, because the crossover operator will create new solutions exactly equal to $\boldsymbol{C^L}$. Thus, this condition will be used as the stop condition for the BLGA, and the BLGA will return $\boldsymbol{C^L}$ to the metaheuristic.

6 Experiments: Comparison with Other LSPs

This section reports on an empirical comparative study between the BLGA method and other LSPs for binary-coded problems presented in the literature: First-LS [3]; Best-LS [3]; and RandK-LS [26, 27, 34].

The study compares four instances of the simplest LSP based metaheuristic, the *Multi-start Local Search* [4], each one with a different LSP. The pseudocode of the MSLS metaheuristic is shown in Fig. 7.

The four MSLS instances are defined as follows:

- MS-First-LS: MSLS with the First-LS.
- MS-Best-LS: MSLS with the Best-LS.
- MS-RandK-LS: MSLS with the RandK-LS.
- MS-BLGA: MSLS with the BLGA.

We have chosen the MSLS metaheuristic in order to avoid possible synergies between the metaheuristic and the LSP. In this way, comparisons among the LSPs are fairer. All the algorithms were executed 50 times, each with a maximum of 100,000 evaluations.

The BLGA uses 500 individuals as the population size, $p_f = 0.95$ and $m = 10$ mates for the crossover operator, $n_{ass} = 5$ for the positive assortative mating, and

```
multistart_LS (LSP)
   S^best ← generate random solution;

   While (stop-condition is not fulfilled)
      S ← generate a random solution;
      S' ← perform LSP on S;

      If (S' is better than S^best)
         S^best ← S';

   Return S^best;
```

Fig. 7 Pseudocode of the MSLS metaheuristic

$n_T = 15$ for restricted tournament selection (parameter values from [12]). The population of the BLGA does not undergo initialisation after the iterations of the MSLS, i.e. the initial population of the BLGA at the jth iteration of the MS-BLGA is the last population of the $(j-1)$th iteration. On the other hand, the leader chromosome is given by the MSLS, i.e. it is generated at random, at the beginning of the iterations of the metaheuristic.

We used the 19 test functions described in Appendix A. Table 1 indicates their name, dimension, optimisation criteria, and optimal fitness value.

The results for all the algorithms are included in Table 2. The performance measure is the average of the best fitness function found over 50 executions. In addition, a two-sided *t-test* at 0.05 level of significance was applied in order to ascertain if the differences in performance of the MS-BLGA are significant when compared with those for the other algorithms. We denote the direction of any significant differences as follows:

Table 1 Test problems

Name	Dimension	Criterion	f^*
Onemax(400)	400	minimisation	0
Deceptive(13)	39	minimisation	0
Deceptive(134)	402	minimisation	0
Trap(1)	36	maximisation	220
Trap(4)	144	maximisation	880
Maxcut(G11)	800	maximisation	572.7[1]
Maxcut(G12)	800	maximisation	621[2]
Maxcut(G17)	800	maximisation	Not known
Maxcut(G18)	800	maximisation	1063.4[1]
Maxcut(G43)	1000	maximisation	7027[2]
M-Sat(100,1200,3)	100	maximisation	1[3]
M-Sat(100,2400,3)	100	maximisation	1[3]
NkLand(48,4)	48	maximisation	1[3]
NkLand(48,12)	48	maximisation	1[3]
BQP('gka')	50	maximisation	3414[4]
BQP(50)	50	maximisation	2098[4]
BQP(100)	100	maximisation	7970[4]
BQP(250)	250	maximisation	45,607[4]
BQP(500)	500	maximisation	116,586[4]

[1] Upper bounds presented in [11].
[2] Upper bounds presented in [10].
[3] 1 is the maximum possible fitness value, however an optimal solution with that fitness value may not exist, depending on the current problem instance.
[4] Best known values presented in [1].

Table 2 Comparison of the MS-BLGA with other MSLS instances

	MS-First-LS	MS-Best-LS	MS-RandK-LS	MS-BLGA
Onemax(400)	0 ~	0 ~	0 ~	0
Deceptive(13)	8.68 ~	3.36–	14.32+	8.68
Deceptive(134)	177.6–	128.4–	201.6+	185.84
Trap(1)	213.12+	219.1 ~	201.86+	218.38
Trap(4)	790.08+	828.92+	781.78+	869.3
Maxcut(G11)	437.36+	349.6+	441+	506.64
Maxcut(G12)	425.6+	335.16+	431.32+	497.36
Maxcut(G17)	2920.82+	2824.66+	2946.58+	2975.7
Maxcut(G18)	849.86+	628.32+	873.82+	898.08
Maxcut(G43)	6427.44+	5735.84+	6463.1 ~	6463.18
M-Sat(100,1200,3)	0.9551+	0.9526+	0.9563 ~	0.9566
M-Sat(100,2400,3)	0.9332 ~	0.9314+	0.9335 ~	0.9338
NkLand(48,4)	0.7660+	0.7647+	0.7694+	0.7750
NkLand(48,12)	0.7456 ~	0.7442 ~	0.7493 ~	0.7468
BQP("gka")	3414 ~	3414 ~	3414 ~	3414
BQP(50)	2098 ~	2094.08 ~	2096.72 ~	2098
BQP(100)	7890.56+	7831.7+	7881.52+	7927.56
BQP(250)	45,557.16 ~	45,171.38+	45,504.22+	45,510.96
BQP(500)	115,176.88 ~	108,588.26+	115,335.34 ~	115,256.3

- A plus sign (+): the performance of MS-BLGA is better than that of the corresponding algorithm.
- A minus sign (−): the algorithm improves the performance of MS-BLGA.
- An approximate sign (~): no significant differences.

We have introduced Fig. 8 in order to facilitate the analysis of these results. It shows the percentage improvements, reductions, and non-differences, according to the t-test, obtained when comparing MS-BLGA with the other algorithms on all the test problems.

From Fig. 8, we can say that MS-BLGA performs better than all the other algorithms for more than the 50% of the test problems, and better than or equivalent to

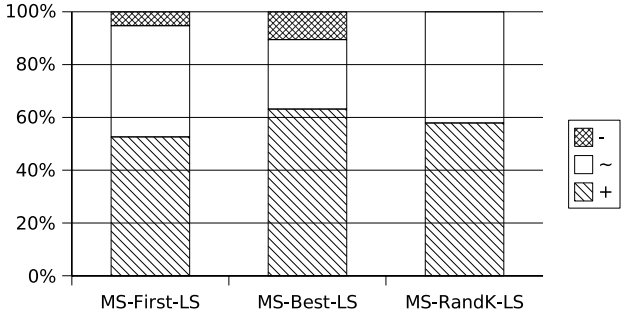

Fig. 8 Comparison of MS-BLGA with other algorithms

almost 90%. Thus, we may conclude that the BLGA is a very promising algorithm for dealing with binary-coded optimisation problems.

On the other hand, Fig. 9 shows the percentage improvements, reductions and non-differences obtained when using MS-BLGA for each test problem (with regard to the other algorithms). Two remarks are worth mentioning regarding Fig. 9:

- MS-BLGA is one of the best algorithms for almost 90% of the test functions. Specifically, MS-BLGA achieves better or equivalent results to those of the other algorithms for all functions, except the two Deceptive ones.
- MS-BLGA returns the best results on four of the five Max-Cut problems.

To sum up, we may conclude that the BLGA, working within the MSLS metaheuristic, is very competitive with classic LSPs, because it obtains better or equivalent results for almost all the test problems considered in this study.

7 Conclusions

In this chapter, we have shown that GAs may be specifically designed with the aim of performing an effective local search: we called these GAs Local GAs. First, we surveyed different LGA instances appearing in the literature. Then, we focused on the BLGA, a recent LGA proposal. BLGA incorporates a specific mate selection mechanism, the crossover operator, and a replacement strategy to direct the local search towards promising search regions represented in the proper BLGA population.

An experimental study, including 19 binary-coded test problems, has shown that when we incorporate the BLGA into a MSLS metaheuristic, this metaheuristic improves results compared with the use of other LSPs that are frequently used to implement it. The good performance of the LGAs reviewed and the satisfactory results

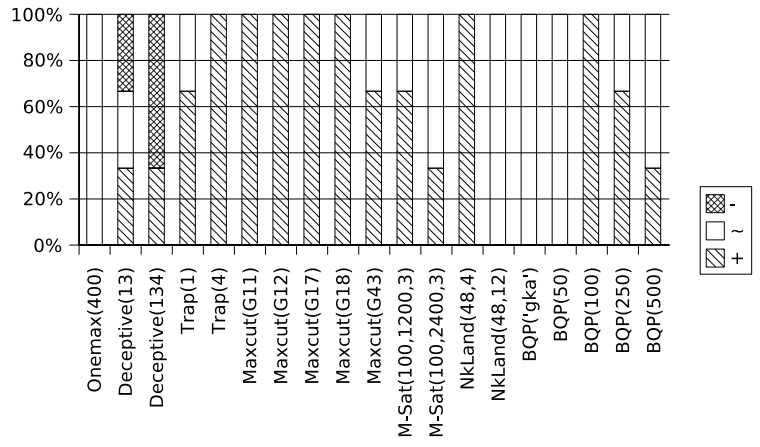

Fig. 9 Performance of MS-BLGA on each test problem

given by the BLGA indicate that further study of these GAs is a topic of major interest. We currently intend to:

- analyse the behaviour of LGAs when they are used by different metaheuristics based on LSPs [4, 8, 30, 35, 36, 41]. Specifically, we are interested in the BLGA.
- extend our investigation to different test-suites (other coding schemes) and real-world problems.

Acknowledgement. This research has been supported by the Spanish MEC project TIN2005-08386-C05-01.

A Appendix. Test Suite

The test suite used for the experiments consists of 19 binary-coded test problems (n is the dimension of the problem). They are described in the following sections.

A.1 Onemax Problem

This is a minimisation problem that applies the following formula:

$$f(\boldsymbol{X}) = n - \sum_{i=1}^{n} X_i \tag{1}$$

We have denoted as Onemax(n) an instance of the Onemax problem with n decision variables: we used Onemax(200).

A.2 Deceptive Problem

In deceptive problems [16] there are certain schemata that guide the search towards a solution that is not globally competitive. It is due to, the schemata that have the global optimum do not bear significance and so, they may not proliferate during the genetic process. The deceptive problem used consists of the concatenation of k subproblems of length 3. The fitness for each 3-bit section of the string is given in Table 3. The overall fitness is the sum of the fitnesses of these deceptive subproblems. To obtain an individual's fitness, the value of this function is subtracted from the maximum value ($30k$). Therefore, the optimum has a fitness of zero.

We denoted as Deceptive(k) an instance of the Deceptive problem with k subproblems of length 3. We used two instances: Deceptive(13) and Deceptive(134).

Table 3 Deceptive order-3 problem

Chromosomes	000	001	010	100	110	011	101	111
Fitness	28	26	22	14	0	0	0	30

A.3 Trap Problem

Trap problem [46] consists of misleading subfunctions of different lengths. Specifically, the fitness function $f(\boldsymbol{X})$ is constructed by adding subfunctions of length 1 (F_1), 2 (F_2), and 3 (F_3). Each subfunction has two optima: the optimal fitness value is obtained for an all-ones string, while the all-zeroes string represents a local optimum. The fitness of all other strings in the subfunction is determined by the number of zeroes: the more zeroes the higher the fitness value. This causes a large basin of attraction towards the local optimum. The fitness values for the subfunctions are specified in Table 4, where the columns indicate the number of ones in the subfunctions F_1, F_2, and F_3. The fitness function $f(\boldsymbol{X})$ is composed of four subfunctions F_3, six subfunctions F_2, and 12 subfunctions F_1. The overall length of the problem is thus 36. $f(\boldsymbol{X})$ has 2^{10} optima of which only one is the global optimum: the string with all ones having a fitness value of 220.

$$f(\boldsymbol{X}) = \sum_{i=0}^{3} F_3(X_{[3i:3i+2]}) + \sum_{i=0}^{5} F_2(X_{[2i+12:2i+13]}) + \sum_{i=0}^{11} F_1(X_{24+i}) \tag{2}$$

We used two instances of the Trap problem:

- Trap(1), which coincides exactly with the previous description. And,
- Trap(4), which applyies Trap(1) to a chromosome with four groups of 36 genes. Each group is evaluated with Trap(1), and the overall fitness of the chromosomes is the sum of the fitnesses of each group.

A.4 Max-Sat Problem

The satisfiability problem in propositional logic (SAT) [42] is the task of deciding whether a given propositional formula has a model. More formally, given a set of m clauses $\{C_1, \ldots, C_m\}$ involving n Boolean variables $X_1, \ldots, X_n$ the SAT problem is to decide whether an assignment of values to variables exists such that all clauses are simultaneously satisfied.

Max-Sat is the optimisation variant of SAT and can be seen as a generalisation of the SAT problem: given a propositional formula in conjunctive normal form (CNF), the Max-Sat problem then is to find a variable assignment that maximises the number of satisfied clauses. It returns the percentage of satisfied clauses.

We used two sets of instances of the Max-Sat problem with 100 variables, three variables by clause, and 1200 and 2400 clauses, respectively. They were obtained using

Table 4 Fitness values of the subfunctions F_i of length i; the columns represent the number of bits in the subfunction that are equal to one

	0	1	2	3
F_3	4	2	0	10
F_2	5	0	10	
F_1	0	10		

the random generator in [43] ([5]). They are denoted as M-Sat(n, m, l, *seed*), where l indicates the number of variables involved in each clause, and *seed* is a parameter needed to randomly generate the Max-Sat instance. Each execution of each algorithm used a different *seed*, i.e. the ith execution of every algorithm used the same $seed_i$, whereas the jth execution used $seed_j$.

A.5 NK-Landscapes

In the NK model [28], N represents the number of genes in a haploid chromosome and K represents the number of linkages each gene has to other genes in the same chromosome. To compute the fitness of the entire chromosome, the fitness contribution from each locus is averaged as follows:

$$f(\boldsymbol{X}) = \frac{1}{N}\sum_{i=1}^{N} f(locus_i) \tag{3}$$

where the fitness contribution of each locus, $f(locus_i)$, is determined by using the (binary) value of gene i together with values of the K interacting genes as an index into a table T_i of size 2^{K+1} of randomly generated numbers uniformly distributed over the interval $[0, 1]$. For a given gene i, the set of K linked genes may be randomly selected or consists of the immediately adjacent genes.

We used two sets of instances of the NK-Landscape problem: one with $N = 48$ and $K = 4$, and another with $N = 48$ and $K = 12$. They are denoted as NKLand (N, K, *seed*), where *seed* is a parameter needed to randomly generate the NK-Landscape instance. They were obtained using the code offered in [39] ([5]). Each execution of each algorithm used a different *seed*, i.e. the ith executions of all the algorithms used the same $seed_i$, whereas the jth executions used $seed_j$.

A.6 Max-Cut Problem

The Max-Cut problem [25] is defined as follows: let an undirected and connected graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ and $E \subset \{(i, j) : 1 \leq i < j \leq n\}$, be given. Let the edge weights $w_{ij} = w_{ji}$ be given such that $w_{ij} = 0\ \forall (i, j) \notin E$, and in particular, let $w_{ii} = 0$. The Max-Cut problem is to find a bipartition (V_1, V_2) of V so that the sum of the weights of the edges between V_1 and V_2 is maximised.

We used five instances of the Max-Cut problem (G11, G12, G17, G18, G43), obtained by means of the code in [50] ([20]).

A.7 Unconstrained Binary Quadratic Programming Problem

The objective of the Unconstrained Binary Quadratic Programming (BQP) [1,18] is to find, given a symmetric rational $n \times n$ matrix $Q = (Q_{ij})$, a binary vector of length n that maximises the following quantity:

$$f(\boldsymbol{X}) = \boldsymbol{X}^{\boldsymbol{t}} Q \boldsymbol{X} = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij} X_i X_j, \quad X_i \in \{0, 1\} \tag{4}$$

We used five instances with different values for n. They were taken from the OR-Library [2]. They are the first instances of the BQP problems in the files 'bqpgka', 'bqp50', 'bqp100', 'bqp250', 'bqp500'. They are called BQP('gka'), BQP(50), BQP(100), BQP(250), and BQP(500), respectively.

References

1. Beasley JE (1998) Heuristic algorithms for the unconstrained binary quadratic programming problem. Tech. Rep., Management School, Imperial College, London, UK.
2. Beasley JE (1990) The Journal of the Operational Research Society 41(11):1069–1072. (http://people.brunel.ac.uk/ mastjjb/jeb/info.html)
3. Blum C, Roli A (2003) ACM Computing Surveys 35(3):268–308
4. Boese KD, Muddu S (1994) Operations Research Letters 16:101–113
5. De Jong K, Potter MA, Spears WM (1997) Using problem generators to explore the effects of epistasis. In: Bäck T (ed) Proc. of the Seventh International Conference on Genetic Algorithms. Morgan Kaufmann
6. Dietzfelbinger M, Naudts B, Van Hoyweghen C, Wegener I (2003)IEEE Transactions on Evolutionary Computation 7(5):417–423
7. Elliott L, Ingham DB, Kyne AG, Mera NS, Pourkashanian M, Wilson CW (2004) An informed operator based genetic algorithm for tuning the reaction rate parameters of chemical kinetics mechanisms. In: Deb K, Poli R, Banzhaf W, Beyer H-G, Burke EK, Darwen PJ, Dasgupta D, Floreano D, Foster JA, Harman M, Holland O, Lanzi PL, Spector L, Tettamanzi A, Thierens D, Tyrrell AM (eds) Proc. of the Genetic and Evolutionary Computation Conference, LNCS 3103. Springer, Berlin Heidelberg
8. Feo T, Resende M (1995) Journal of Global Optimization 6:109–133.
9. Fernandes C, Rosa A (2001) A study on non-random mating and varying population size in genetic algorithms using a royal road function. Proc. of the 2001 Congress on Evolutionary Computation, IEEE Press, Piscataway, New Jersey
10. Festa P, Pardalos PM, Resende MGC, Ribeiro CC (2002) Optimization Methods and Software 17(6):1033–1058
11. Fischer I, Gruber G, Rendl F, Sotirov R (2006) Mathematical Programming 105(2–3): 451–469
12. García-Martínez C, Lozano M, Molina D (2006) A Local Genetic Algorithm for Binary-coded Problems. In: Runarsson TP, Beyer H-G, Burke E, Merelo-Guervós JJ, Whitley LD, Yao X (eds) 9th International Conference on Parallel Problem Solving from Nature, LNCS 4193. Springer, Berlin Heidelberg
13. García-Martínez C, Lozano M, Herrera F, Molina D, Sánchez AM (2007) Global and local real-coded genetic algorithms based on parent-centric crossover operators. European Journal of Operational Research. In Press, Corrected Proof, Available online 18 October 2006
14. Gendreau M, Potvin J-Y (2005) Annals of Operations Research 140(1):189–213
15. Glover F, Laguna M (1999) Operational Research Society Journal 50(1):106–107
16. Goldberg DE, Korb B, Deb K (1989) Complex Systems 3:493–530
17. Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.
18. Gulati VP, Gupta SK, Mittal AK (1984) European Journal of Operational Research 15:121–125

19. Harik G (1995) Finding multimodal solutions using restricted tournament selection. In: Eshelman LJ (ed) Proc. of the 6th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, California
20. Helmberg C, Rendl F (2000) Siam Journal of Optimization 10(3):673–696
21. Herrera F, Lozano M, Verdegay JL (1998) Artificial Intelligence Revue 12(4):265–319
22. Herrera F, Lozano M (2000) IEEE Trans. on Evolutionary Computation 4(1):43–63
23. Herrera F, Lozano M, Sánchez AM (2003) International Journal of Intelligent Systems 18(3):309–338
24. Holland JH (1992) Adaptation in Natural and Artificial Systems. The MIT Press Cambridge, MA, USA
25. Karp RM (1972) Reducibility among combinatorial problems. In: Miller R, Thatcher J (eds), Complexity of Computer Computations. Plenum Press, New York
26. Katayama K, Tani M, Narihisa H (2000) Solving large binary quadratic programming problems by effective genetic local search algorithm. In: Whitley D, Goldberg D, Cantu-Paz E, Spector L, Parmee I, Beyer H-G (eds) Proc. of the 2000 Genetic and Evolutionary Computation Conference. Morgan Kaufmann
27. Katayama K, Narihisa H (2001) Trans. IEICE (A) J84-A(3):430–435
28. Kauffman SA (1989) Lectures in the Sciences of Complexity 1:527–618
29. Kazarlis SA, Papadakis SE, Theocharis JB, Petridis V (2001) IEEE Transactions on Evolutionary Computation 5(3):204–217
30. Lourenço HR, Martin O, Stützle T (2002) Iterated local search. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics. Kluwer Academic, Boston, MA, USA
31. Lozano M, Herrera F, Krasnogor N, Molina D (2004) Evolutionary Computation Journal 12(3):273–302
32. Meloni C, Naso D, Turchiano B (2003) Multi-objective evolutionary algorithms for a class of sequencing problems in manufacturing environments. Proc. of the IEEE International Conference on Systems, Man and Cybernetics 1
33. Merz P (2002) Nk-fitness landscapes and memetic algorithms with greedy operators and k-opt local search. In: Krasnogor N (ed) Proc. of the Third International Workshop on Memetic Algorithms (WOMA III)
34. Merz P, Katayama K (2004) Bio Systems 79(1–3):99–118
35. Mladenovic N, Hansen P (1997) Computers in Operations Research 24:1097–1100
36. Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F (eds), New Ideas in Optimization. McGraw-Hill, London
37. Nasimul N, Hitoshi I (2005) Enhancing differential evolution performance with local search for high dimensional function optimization. In: Beyer HG, O'Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E, Dasgupta D, Deb K, Foster JA, De Jong ED, Lipson H, Llora X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson J-P, Zitzler E (eds) Proc. of the Genetic and Evolutionary Computation Conference. ACM Press, New York
38. Papadakis SE, Theocharis JB (2002) Fuzzy Sets and Systems 131(2):121–152
39. Potter MA. http://www.cs.uwyo.edu/~wspears/nk.c
40. Potts JC, Giddens TD, Yadav SB (1994) IEEE Transactions on Systems, Man, and Cybernetics 24:73–86
41. Resende MGC, Ribeiro CC (2003) International Series in Operations Research and Management Science 57:219–250
42. Smith K, Hoos HH, Stützle T (2003) Iterated robust tabu search for MAX-SAT. In: Carbonell JG, Siekmann J (eds) Proc. of the 16th conference on the Canadian Society for Computational Studies of Intelligence, LNCS 2671. Springer, Berlin Heidelberg

43. Spears WM. http://www.cs.uwyo.edu/~wspears/epist.html
44. Sywerda G (1989) Uniform crossover in genetic algorithms. In: Schaffer JD (ed) Proc. of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA, USA
45. Tanese R (1987) Parallel genetic algorithms for a hypercube. In: Grefenstette JJ (ed) Proc. of the Second International Conference on Genetic Algorithms Applications. Hillsdale, NJ, Lawrence Erlbraum
46. Thierens D (2004) Population-based iterated local search: restricting neighborhood search by crossover. In: Deb K, Poli R, Banzhaf W, Beyer H-G, Burke EK, Darwen PJ, Dasgupta D, Floreano D, Foster JA, Harman M, Holland O, Lanzi PL, Spector L, Tettamanzi A, Thierens D, Tyrrell AM (eds) Proc. of the Genetic and Evolutionary Computation Conference, LNCS 3103. Springer, Berlin Heidelberg
47. Tsutsui S, Ghosh A, Corne D, Fujimoto Y (1997) A real coded genetic algorithm with an explorer and an exploiter population. In: Bäck T (ed) Proc. of the Seventh International Conference on Genetic Algorithms. Morgan Kaufmann Publishers, San Francisco
48. Weicai Z, Jing L, Mingzhi X, Licheng J (2004) IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 34(2):1128–1141
49. Whitley D (1989) The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: Schaffer JD (ed) Proc. of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA, USA
50. Ye Y. http://www.stanford.edu/ yyye/yyye/Gset/