Hybrid Evolutionary Algorithms and Clustering Search

Alexandre C. M. Oliveira[1] and Luiz A. N. Lorena[2]

[1] Universidade Federal do Maranhão - UFMA, Departamento de Informática, São Luís MA, Brasil
acmo@deinf.ufma.br
[2] Instituto Nacional de Pesquisas Espaciais - INPE, Laboratório Associado de Computação e Matemática Aplicada, São José dos Campos SP, Brasil.
lorena@lac.inpe.br

Summary. A challenge in hybrid evolutionary algorithms is to employ efficient strategies to cover all the search space, applying local search only in actually promising search areas. The inspiration in nature has been pursued to design flexible, coherent and efficient computational models. In this chapter, the Clustering Search (*CS) is proposed as a generic way of combining search metaheuristics with clustering to detect promising search areas before applying local search procedures. The clustering process aims to gather similar *information* about the search space into groups, maintaining a representative solution associated to this information. Two applications are examined for combinatorial and continuous optimization problems, presenting how to develop hybrid evolutionary algorithms based on *CS.

Keywords: Hybrid search metaheuristics; Combinatorial and continuous optimization.

1 Introduction

Modern search methods for optimization consider hybrid evolutionary algorithms (HEA) those employing evolutionary algorithm (EA) and local optimizers working together. The hybridism comes from the balancing of global and local search procedures. The inspiration in nature has been pursued to design flexible, coherent and efficient computational models. The main focuses of such models are real-world problems, considering the known little effectiveness of canonical genetic algorithms (GAs) in dealing with them.

Investments have been made in new methods, which the evolutionary process is only part of the whole search process. Due to their intrinsic features as global solver, GAs are employed as a generator of search areas (subspaces), which are more intensively inspected by a heuristic component. This scenario

comes to reinforce the parallelism of EAs, as well a collaborative perspective between solver components.

This chapter proposes the Clustering Search (*CS): the generalized way of detecting promising search areas by clusters of solutions. This generalized approach is achieved both by the possibility of employing any metaheuristic and by also applying to combinatorial and continuous optimization problems. The remainder of this chapter is organized as follows. Section 2 discusses some ideas related to clustering and social behaviour inspiring hybrid methods found in literature. Section 3 describes the basic ideas and conceptual components of *CS. Section 4 and 5 examine two applications for unconstrained continuous optimization and pattern sequencing, respectively. The findings and conclusions are summarized in section 6.

2 Related works

Several natural processes can be simulated by evolutionary algorithms, enriching their capabilities, extending their applicability and enhancing their performance. The migration of individuals between search areas can be taken with a social behaviour interpretation. A society corresponds to a cluster of points in the search space and the set of all societies composes a civilization. Each society has its set of leaders that help others individuals to improve through an intrasociety information exchange.

The intrasociety information exchange is analogous to an intensified local search around a better performing point, resulting in the shift of points in the cluster. Leaders of a society can improve only through an intersociety information exchange that results in the migration of a leader from a society to another. The intersociety information exchange is analogous to a search around globally promising regions in the search space [2].

Another interpretation for clusters and migration of individuals comes from the analogy with ecosystems in nature. Each search area can be seen as a geographical region where distinct species or societies evolve at the same time. Each group of individuals lives and reproduces in their respective region. They search for natural resources by their particular strategy of exploration. Sometimes, the region is inappropriate to be explored by long periods and the resources become scarce, obliging the individuals to migrate to another region or even to die on it. The migration of groups of individuals allows discovering other areas, keeping inhabited the most resource-full of them.

Local search methods have been combined with metaheuristics in different ways to solve particular problems more efficiently. Gradient as well as direct search methods have been employed as exploitation tool in continuous optimization. In the cases where there are no derivatives available for fitness function, pattern search methods are useful to provide more robust convergence [3].

Hill-climbing procedures are largely employed in the so called memetic algorithms (MA) as a Lamarckian learning process [4]. For example, a simple crossover can work as a local search around the parents, hill-climbing by repeatedly generating some number of offspring and replacing the worst parent [5].

The main challenge in such hybrid methods is to define efficient strategies to cover all search space, applying local search only in actually promising areas. Elitism plays an important role towards achieving this goal, since the best solutions represent promising neighborhood. The Simplex Genetic Algorithm Hybrid (SGAH), proposed in [6], applies a probabilistic version of Nelder and Mead Simplex (NMS)[7] in the elite of population. However, such well-evaluated solutions can be concentrated in few areas and thus the exploitation moves are not rationally applied.

More recently, a different strategy have been used, demonstrating concern about rationally employing local search in the evolutionary solver fashion. The evolutionary process runs normally until a promising area is detected. The promising area is detected when the highest distance between the best individual and the other individuals of the population is smaller than a given radius, i.e., when population diversity is lost. Thereafter, the search domain is reduced, an initial simplex is built inside this area and a local search based on Nelder and Mead Simplex is started. This continuous hybrid algorithm, called CHA, perform very well, reaching good results [8]. The exploitation moves are started using the simplex method around the best individual found in the exploration cycle. With respect to detection of promising areas, the CHA has a limitation. The exploitation is started once, after diversity loss, and the evolutionary process can not be continued afterwards, unless a new population takes place.

Another approach attempting to find out relevant areas for continuous optimization is a parallel hill-climber, called Universal Evolutionary Global Optimizer (UEGO) by its authors [1]. The separated hill-climbers work in restricted search regions (or clusters) of the search space. The volume of the clusters decreases as the search proceeds, resulting in a cooling effect similar to simulated annealing. Each cluster center represents diversity and quality, since it is result of hill-climbing procedures in separated search subspaces [1]. UEGO do not work so well as CHA for high dimension functions.

The scatter search (SS), proposed in [9], by another way, separates diversified and improved solutions in two sets: the reference set, containing the best solutions found so far and the diversity set, containing the solutions most distant from the solutions of the reference set. The solutions in these two sets are improved by local search. Thus, SS employs systematic exploration/exploitation moves, combining quality and representative solutions [9].

The idea behind all these methods is to explore the most promising search areas by strategies beyond those resulting from the regular sampling of the fitness landscape. In EAs, promising search areas can be detected by fit or frequency merits [10]. By fit merits, the fitness of the solutions can be used

to say how good their neighborhoods are. On the other hand, in frequency merits, the evolutionary process naturally privileges the good search areas by a more intensive sampling in them. Good solutions are more frequently sampled. Fig. 1 shows the 2-dimensional contour map of a test function known as *Langerman*. The points are candidate solutions over fitness surface in a certain generation. One can note their agglomeration over the promising search areas.

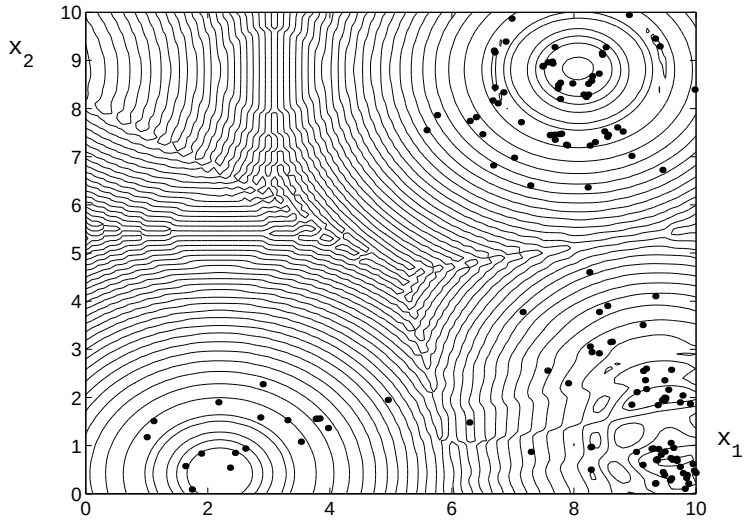

Fig. 1. Convergence of typical GA into fitter areas.

Clusters of mutually close solutions hopefully can correspond to relevant areas of attraction in the most of search metaheuristics, including EAs. Relevant search areas can be treated with special interest by the algorithm as soon as they are discovered. The clusters work as sliding windows, framing the search areas and giving a reference point (center) to problem-specific local search procedures. Furthermore, the cluster center itself is always updated by a permanent interaction with inner solutions, called assimilation [11, 12].

This basic idea was employed to propose the Evolutionary Clustering Search (ECS) early applied to unconstrained continuous optimization [11]. Posteriorly, the search guided by clustering was extended to a GRASP (Greedy Randomized Adaptive Search Procedure[13]) with VNS (Variable Neighborhood Search[14]), and applied to Prize Collecting Traveling Salesman Problem (PCTSP) [12], a generalization of the Traveling Salesman Problem, which the salesman collects a prize in each city visited and pays a penalty for each city not visited [12]. The later is the first attempt of replacing the evolutionary metaheuristic by another.

3 Clustering Search foundations

The Clustering Search (*CS) employs clustering for detecting promising areas of the search space. It is particularly interesting to find out such areas as soon as possible to change the search strategy over them. An area can be seen as a search subspace defined by a neighborhood relationship in metaheuristic coding space.

The *CS attempts to locate promising search areas by framing them by clusters. A cluster can be defined as a tuple $\mathcal{G} = \{c, r, s\}$, where c and r are the *center* and the *radius* of the area, respectively. The radius of a search area is the distance from its center to the edge. There can exist different *search strategies* s associated to the clusters.

Initially, the center c is obtained randomly and progressively it tends to slip along really promising points in the close subspace. The total cluster volume is defined by the radius r and can be calculated, considering the problem nature. It is important that r must define a search subspace suitable to be exploited by the search strategies associated to the cluster.

For example, in unconstrained continuous optimization, it is possible to define r in a way that all Euclidean search space is covered depending on the maximum number of clusters[11]. In combinatorial optimization, r can be defined as the number of movements needed to change a solution into another. In both cases, the neighborhood is function of some distance metric related to the search strategy s.

3.1 Components

*CS can be splitted off in four conceptually independent parts. Fig. 2 brings its conceptual design.

1. a search metaheuristic (SM);
2. an iterative clustering (IC) component;
3. an analyzer module (AM); and
4. a local searcher (LS).

SM component works as a full-time solution generator, according to its specific search strategy, performing independently of the remaining parts, and manipulating a set of $|P|$ solutions ($|P| > 1$ for EAs). In an evolutionary algorithm fashion, individuals are selected, crossed over, and updated for the next generations. This entire process works like an infinite loop, in which solutions are generated along the iterations.

IC component aims to gather similar solutions into groups, maintaining a representative cluster center for them. To avoid extra computational effort, IC is designed as an online process, in which the clustering is progressively fed by solutions generated in each regular iteration of SM. A maximum number of clusters $\mathcal{NC}$ is an upper bound value that prevents an unlimited cluster

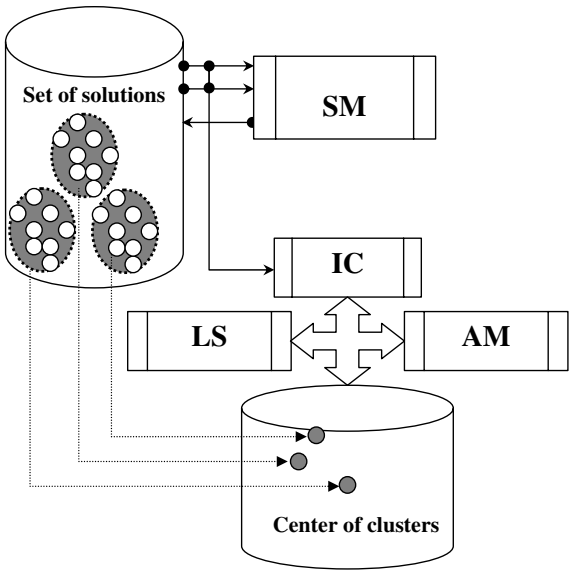

Fig. 2. *CS components.

creation. For a $n-$dimensional problem, the IC complexity is, at most, $O(\mathcal{NC} \cdot n)$ when all cluster centers are allocated. A *distance metric*, $\wp$, must be defined, *a priori*, allowing a similarity measure for the clustering process.

AM component examines each cluster, in regular intervals, indicating a probable promising cluster. A *cluster density*, δ_i, is a measure that indicates the activity level inside the cluster i. For simplicity, δ_i counts the number of solutions generated by SM (selected solutions, in the EA case[11]). Whenever δ_i reaches a certain *threshold*, meaning that some information template becomes predominantly generated by SM, such information cluster must be better investigated to accelerate the convergence process on it. Clusters with lower δ_i are eliminated, as part of a mechanism that will allow creating other centers of information, keeping framed the most active of them. The cluster elimination does not affect the set of $|P|$ solutions in SM. Only the center of information is considered irrelevant for the process.

At last, the LS component is an internal searcher module that provides the exploitation of a supposed promising search area, framed by cluster. This process can happen after AM having discovered a target cluster or it can be a continuous process, inherent to IC, being performed whenever a new point is grouped. LS can be considered as the particular search strategy s associated with the cluster, i.e., a problem-specific local search to be employed into the cluster.

3.2 The clustering process

The clustering process described here is based on Yager's work, which says that a system can learn about an external environment with the participation of previously learned beliefs of the own system [15, 16]. The IC is the *CS's core, working as a classifier, keeping in the system only relevant information, and driving search intensification in the promising search areas. To avoid propagation of unnecessary information, the local search is performed without generating other intermediary points of search space.

Solutions s_k generated by SM are passed to IC that attempts to group as known information, according to $\wp$. If the information is considered sufficiently new, it is kept as a center in a new cluster, c_{new}. Otherwise, redundant information activates the closest center c_i (cluster center that minimizes $\wp(s_k, c_{j=1,2,\cdots,})$), causing some kind of perturbation on it. This perturbation means an *assimilation process*, in which the previously learned knowledge (center of the cluster) is updated by the received information. Considering $\mathcal{G}_{j\ (j=1,2,\cdots)}$ as all current detected clusters:

- $c_{new} = s_k$, if $\wp(s_k, c_j) > r_j, \forall \mathcal{G}_j$; or
- $c_i' =$ assimilation of s_k by c_i, otherwise.

3.3 Assimilation

The assimilation process is applied over the closest center c_i, considering the new generated solution s_k. The general assimilation form is [10]:

$$c_i' = c_i \oplus \beta(s_k \ominus c_i) \tag{1}$$

where $\oplus$ and $\ominus$ are abstract operations over c_i and s_k meaning, respectively, addition and subtraction of solutions. The operation $(s_k \ominus c_i)$ means the vector of differences between each one of the n variables compounding the solutions s_k and c_i, considering the distance metric. A certain percentage β of the vector is the update step for c_i, giving c_i'. According to β, the assimilation can assume different forms. The three types of assimilation are shown in Fig. 3.

Simple assimilation

In simple assimilation, $\beta \in [0, 1]$ is a constant parameter, meaning a deterministic move of c_i in the direction of s_k. Only one internal point is generated more or less closer to c_i, depending on β, to be evaluated afterwards. The greater β, the less conservative the move is. This type of assimilation can be employed only with real-coded variables, where percentage of intervals can be applied to. Its specific form is:

$$c_i' = c_i + \beta(s_k - c_i) \tag{2}$$

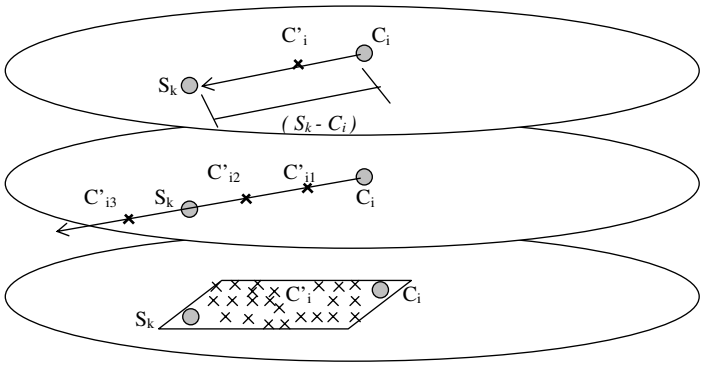

Fig. 3. Simple, path and crossover assimilations, respectively.

Crossover assimilation

Despite its name, crossover assimilation is not necessarily associated with an evolutionary operator. In a general way, it means any random operation between two candidate solutions (parents), giving other ones (offsprings), similarly as a crossover operation in EAs. In this assimilation, β is an $n-$dimensional random vector and c_i' can assume a random point inside the hyper plane containing s_k and c_i. The crossover assimilation can be rewritten by:

$$c_i' = c_i + \overrightarrow{\beta} \cdot (s_k - c_i) \tag{3}$$

Since the whole operation is a crossover or other binary operator between s_k and c_i, it can be applied to any type of coding or even problem (combinatorial or continuous one). The $\overrightarrow{\beta}$ parameter is resulting from the type of crossover employed, not the crossover parameter itself. For example, in continuous optimization, a blend crossover (BLX-α) [17] can be usually inserted in (3), giving a final form to $\overrightarrow{\beta}$:

$$\overrightarrow{\beta} = -blx_\alpha + \overrightarrow{\beta'}(1 + 2blx_\alpha) \tag{4}$$

where the random $\overrightarrow{\beta'} \in [0, 1]$, blx_α is the BLX-α parameter and hence the random resulting $\overrightarrow{\beta} \in [-blx_\alpha, 1 + blx_\alpha]$.

Path assimilation

Simple and crossover assimilations generate only one internal point to be evaluated afterwards. Path assimilation, instead, can generate several internal points or even external ones, holding the best evaluated one to be the new center. It seems to be advantageous, but clearly costly. These exploratory moves are commonly referred in path relinking theory [18].

In this assimilation, β is a $\eta-$dimensional vector of constant and evenly spaced parameters, used to generate η samples taken in the path connecting c_i and s_k. Since each sample is evaluated by the objective function, the path assimilation itself is an intensification mechanism inside the clusters. The new center c'_i is given by:

$$\begin{array}{rl} c'_i = c'_V,\, f(c'_V) = \min\left\{f(c'_1), f(c'_2), \cdots, f(c'_\eta)\right\} \\ c'_j \quad = c_i + \beta_j(s_k - c_i) \\ \beta_j \quad \in \{\beta_1, \beta_2, \cdots, \beta_\eta\} \end{array} \tag{5}$$

where $\beta_j \in \{]0, 1[\bigcup]1, \infty[\}$, $f(c'_V)$ is the objective function of the best evaluated solution sampled in the path and min is concerned to minimization problems.

With respect to the infinite interval in (5), it means that external points can be sampled indefinitely while there are well-succeeded points beyond s_k. A well-succeeded point has an objective function value better than the previous point sampled, in a way that an evaluated worse point stops the sampling. In the Fig. 3, one can see the point c_{i3} evaluated after s_k. Such extrapolation move is suitable for path relinking [18] and it can intentionally shift the center cluster to a new promising search area.

4 ECS for unconstrained continuous optimization

The *CS was first applied to unconstrained continuous optimization, employing a steady-state genetic algorithm as SM component. The evolutionary algorithm acts as a generator of search points. Several other approaches can be connected to *CS. A detailed investigation must have place further. For a while, the next sections are still devoted to the evolutionary metaheuristic.

An evolutionary algorithm, employed as SM component in Clustering Search, is called ECS. A real-coded version of ECS for unconstrained continuous optimization was proposed in [11]. Here, some relevant aspects of that application are briefly explained to clarify the subsequent application.

The unconstrained continuous optimization problem can be presented by:

$$\min\ /\ \max\ \ f(x), x = (x_1, x_2, x_3, \ldots, x_n)^T \in R^n \text{ where } L_i < x_i < U_i \tag{6}$$

In test functions, the upper U_i and lower L_i bounds are defined *a priori* and they are part of the problem, bounding the search space over the challenger areas in function surface. Some of well-known test functions, such as *Michalewicz*, *Langerman*, *Shekel* [19], *Rosenbrock*, *Sphere* [20], *Schwefel*, *Griewank*, and *Rastrigin* [21] were used in the tests.

4.1 Implementation

The component SM was instanced by a steady-state real-coded genetic algorithm employing the well-known genetic operators: roulette wheel selection [22], blend crossover ($BLX_{0.25}$) [17], and non-uniform mutation [23]. Briefly explaining, in each generation, a fixed number of individuals $\mathcal{NS}$ are selected, crossed over, mutated and updated in the same original population, replacing the worst individual (steady-state updating). Parents and offspring are always competing against each other and the entire population tends to converge quickly.

A maximum number of clusters, $\mathcal{NC}$, was defined *a priori*. The i^{th} cluster has its own center c_i, but a common radius r_t.

$$r_t = \frac{x_{sup} - x_{inf}}{2 \cdot \sqrt[n]{|C_t|}} \tag{7}$$

where $|C_t|$ is the current number of clusters (initially, $|C_t| = \mathcal{NC}$), x_{sup} and x_{inf} are, respectively, the known upper and lower bounds of the domain of variable x, considering that all variables x_i have the same domain.

Whenever a selected individual s_k was *far away* from all centers (a distance above r_t), then a new cluster is created. $\mathcal{NC}$ works as a bound value that prevents the creation of an unlimited number of clusters. The simple assimilation was employed in [11] with $\beta = 0.05$, keeping the centers more conservative to new information. Tests with the other assimilation types for unconstrained continuous optimization can be found in [10].

At the end of each generation, the component AM has performed the *cooling* of all clusters, resetting the accounting of δ_i. Remembering that a cluster is considered inactive when no activity has occurred in the last generation. This mechanism is used to eliminate clusters that have lost the importance along the generations, allowing that other search areas can be framed.

The component LS has been activated, at once, if

$$\delta_i \geq \mathcal{PD} \cdot \frac{\mathcal{NS}}{|C_t|} \tag{8}$$

The pressure of density, $\mathcal{PD}$, allows controlling the sensibility of the component AM.

The component LS was implemented by a Hooke and Jeeves direct search (HJD) [24]. The HJD is an early 60's method that presents some interesting features: excellent convergence characteristics, low memory storage, and requiring only basic mathematical calculations. The method works by two types of move. At each iteration, there is an exploratory move with one discrete step size per coordinate direction. Supposing that the line gathering the first and last points of the exploratory move represents an especially favorable direction, an extrapolation is made along it before the variables are varied again individually. Its efficiency decisively depends on the choice of the initial step sizes $\mathcal{SS}$. In [11], $\mathcal{SS}$ was set to 5% of initial radius.

4.2 Results

The ECS was coded in ANSI C and it was run on Intel AMD (1.33 GHz) platform [11]. In the three experiments, ECS was compared with other approaches, taken from the literature, as the well-known Genocop III [23], the OptQuest Callable Library (OCL)[25] and the Continuous Hybrid Algorithm (CHA) [8]. The ECS has found promising results for unconstrained continuous optimization which have stimulated further applications [11].

Some results extracted of [11] are shown in Table 1. The average of function calls (FC) was considered to measure the algorithm performance. The average of execution time in seconds (ET) is only illustrative. The GAP of 0.001 was reached a certain number of times, giving the success rate (SR) obtained for each test function. The worst performance happens for *Michalewicz* and *Langerman*'s functions (SR about 65%).

Table 1. ECS results for some test functions.

Function	var	ET	GAP	FC	SR	Function	var	ET	GAP	FC	SR
Griewank	50	0.053	0.00010	5024.550	100.00	Rastrigin	10	0.100	0.00060	26379.950	100.00
Griewank	100	0.432	0.00000	24344.450	100.00	Rastrigin	20	0.339	0.00078	71952.667	90.00
Langerman	5	0.023	0.00000	5047.684	95.00	Schwefel	20	0.211	0.00035	39987.950	100.00
Langerman	10	0.075	0.00000	17686.692	65.00	Schwefel	30	0.591	0.00029	90853.429	70.00
Michalewicz	5	0.054	0.00035	12869.550	100.00	Michalewicz	10	0.222	0.00038	37671.923	65.00

The Fig. 4 depicts a sequence of contour maps taken at each 5 generations for ECS. The points are candidate solutions, the circles show the search areas framed by the clusters.

5 ECS for pattern sequencing

Now, *CS for combinatorial optimization is focused. A version of ECS for pattern sequencing problem is first presented in this chapter. Pattern sequencing problems may be stated by a matrix with integer elements where the objective is to find a permutation (or sequencing) of rows or patterns (client orders, or gates in a VLSI circuit, or cutting patterns) minimizing some objective function [26]. Objective functions considered here differ from traveling salesman-like problems because the evaluation of a permutation can not be computed by using values that only depend on adjacent patterns.

There are two similar pattern sequencing problems found in the literature: Minimization of Open Stacks Problem (MOSP) and Gate Matrix Layout Problem (GMLP) [27]. Theoretical aspects are basically the same for both problems. The difference between them resides only in their enunciation. Several test instances can be found in literature related to such problems [28, 27]. The ECS was tested only for GMLP instances once they are the large scale ever found in the literature.

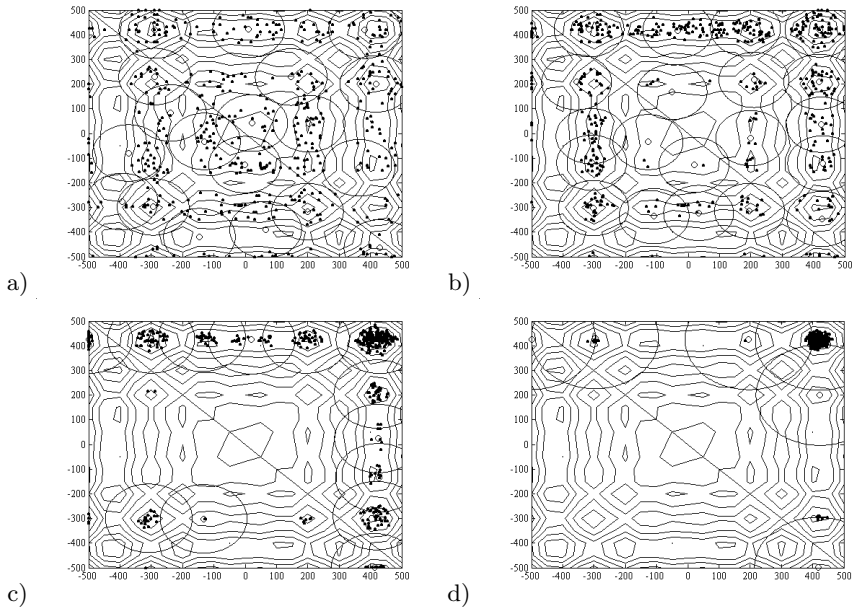

Fig. 4. Contour map of *Schwefel* function after a)2, b)7, c)12 and d)17 generations.

5.1 Theoretical issues of the GMLP

Gate matrix layout problems are related to one-dimensional logic arrays and programmable logic arrays folding [29, 30]. In very large scale integration design (VLSI design), the goal is to arrange a set of circuit gates (vertical wires) in an optimal sequence, such that the layout area is minimized, i.e., the number of tracks necessary to cover the gates interconnection is minimized. This can be achieved by placing non-overlapping nets in the same track.

A group of gates at the same connection is called a net (horizontal wires). Moreover, non-overlapping nets can be placed at the same connection track. To compute the number of tracks to cover all nets, it is enough to verify the maximum of overlapping nets. The number of tracks is an important cost factor of VLSI circuits. Therefore, this problem is due to limitations of physical space. A GMLP consists of determining a sequence of gates that minimizes the maximum of tracks (MOT) in the circuit.

The data for a GMLP are given by an $I \times J$ binary matrix $\mathbf{P}$, representing gates (rows) and nets (columns), where $\mathbf{P}_{ij} = 1$, if gate i belongs to net j, and $\mathbf{P}_{ij} = 0$ otherwise. The sequence of gates determines the number of tracks necessary to produce the VLSI circuit. Another binary matrix, here called matrix $\mathbf{Q}$, can be used to calculate the MOT for a certain gate permutation. It is derived from the input matrix $\mathbf{P}$, by the following rules:

- $\mathbf{Q}_{ij} = 1$ if there exist x and $y | \pi(x) \leq i \leq \pi(y)$ and $\mathbf{P}_{xj} = \mathbf{P}_{yj} = 1$;

- $\mathbf{Q}_{ij} = 0$, otherwise;

where $\pi(b)$ is the position of gate b in the permutation.

The $\mathbf{Q}$ shows the consecutive-ones property [31] applied to $\mathbf{P}$: in each column, 0's between 1's are replaced by 1's. The sum of 1's, by row, computes the number of open stacks when each pattern is processed. Fig. 5 shows an example of matrix $\mathbf{P}$, its corresponding matrix $\mathbf{Q}$, and the number of tracks in the circuit. At most, 7 tracks ($MOT = \max\{3, 3, 3, 5, 6, 7, 7, 5, 3\} = 7$) are needed to manufacture a circuit with permutation $\pi_0 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

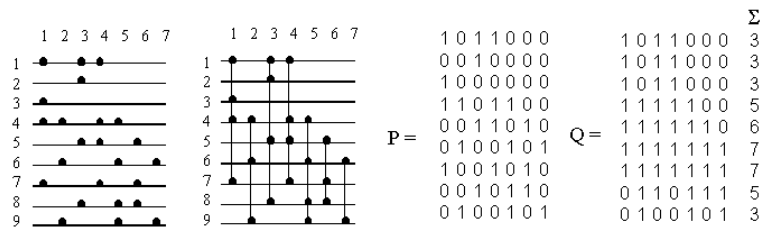

Fig. 5. GMLP (or MOSP) instance: matrix $\mathbf{P}$ and corresponding $\mathbf{Q}$.

Several aspects of the sequencing problems have been presented, including the NP-hardness of them [29, 30, 32]. The same example of Fig. 5 can be seen as a MOSP instance. The number of overlapping nets means the number of open stack in the MOSP context [27].

5.2 Implementation

The component SM is also a steady-state GA employing well-known genetic operators as roulette wheel selection [22], block-order crossover (BOX) [33], and 2-swap mutation [34]. In BOX, the *parent(A)* and *parent(B)* are mixed into only one offspring, by copying blocks of both parents, at random. Pieces copied from a parent are not copied from other, keeping the offspring feasible (Fig. 6a).

The component LS was implemented by a local search mutation which is applied to the center of promising cluster. The local search mutation explores a search tree, considering several 2-Opt neighborhoods as one can see in Fig. 6b). The best neighbor from a level (bold circle) is taken as starting point to the next, respecting a maximum width l (maximum number of 2-swap at each level) and height m (maximum number of levels).

Concerning to component IC, in the continuous optimization case, the radius of the cluster was calculated in the Euclidean space, considering the whole search space. In sequencing problems, one can employ the 2-swap distance metric, i.e., the number of 2-swap needed to move a point (solution), along the search space, to another. Identical solutions need no changes to

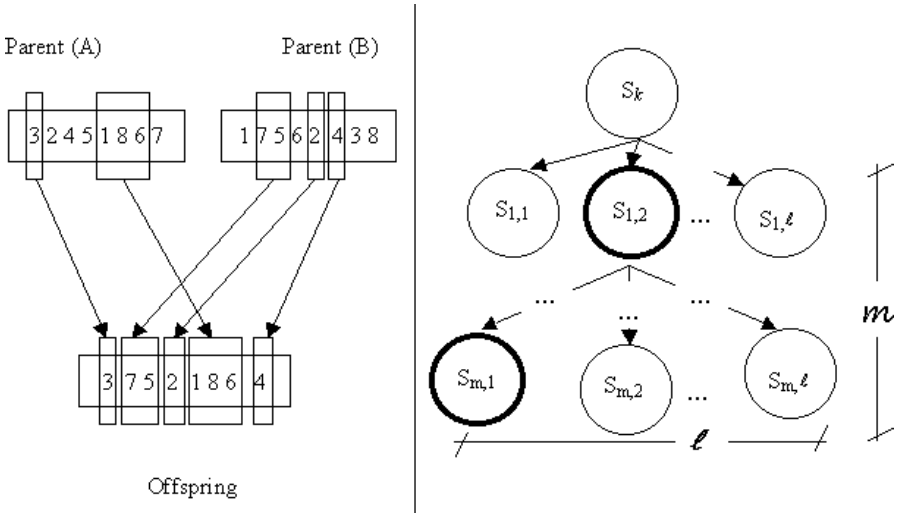

Fig. 6. a) Block order crossover and b) local search mutation tree.

turn one into other. By the other side, completely distinct solutions may need about I−1 2-swap moves to lead a point to another. The radius of a cluster could be given by:

$$r_t = \left\lceil \frac{I-1}{2\sqrt[I]{|C_t|}} \right\rceil \tag{9}$$

where $|C_t|$ is the current number of *clusters*, I is the number of patterns of the problem at hand which is taken to estimate the greater 2-swap distance found in the search space.

Examining (9), with $|C_t| = 20$ and $20 < I < 140$ (common values found in the problem instances tested in this application), one can calculate that $r_t \simeq 0.5I$. In other words, a point s_k belongs to the cluster c_i where the half of pattern sequencing matches. However, (9) have not performed well, causing over clustering. Hence, the radius becomes:

$$r_t = \lceil 0.9I \rceil \tag{10}$$

i.e., a relatively greater radius, because it requires only 10% of labels matching for a sequencing to be considered sufficiently close to a cluster center, belonging to it. Whenever a selected individual s_k is *far away* from all centers (a distance above r_t), then a new cluster must be created.

As there already have been said, the cluster assimilation is a foreseen step that can be implemented by different techniques. In this application, the path assimilation was chosen. This is a typical assimilation process for combinatorial problems. The more distance $\wp(c_i, s_k)$, the more potential solutions exist between c_i and s_k. The sampling process, depending on the number of instance

variables, can be costly, since each solution must be evaluated by objective function. In Table 2, a completely 2-swap path between two solutions, c_i and s_k, can be seen.

Table 2. Example of full path between center c_i and new point s_k.

c_i=	1	2	3	4	5	6	7	8	9	comparison	swap	evaluation
1)	4	2	3	1	5	6	7	8	9	1	1	1
2)	4	8	3	1	5	6	7	2	9	1	1	1
3)	4	8	5	1	3	6	7	2	9	1	1	1
4)	4	8	5	9	3	6	7	2	1	1	1	1
5)	4	8	5	9	1	6	7	2	3	1	1	1
6)	4	8	5	9	1	7	6	2	3	1	1	1
7)	4	8	5	9	1	7	6	2	3	1		
8)	4	8	5	9	1	7	6	2	3	1		
s_k=	4	8	5	9	1	7	6	2	3	8	6	6

Each comparison means one iteration in the assimilation algorithm. Besides, a swap and evaluation of the intermediary solutions can occur. At last, the center will be shifted to the best point evaluated in this path. Actually, there have been occurred 6 pattern swaps and, consequently, 6 objective function calls.

The distance $\wp(c_i, s_k)$ is not necessarily 6. Other paths with distance less than 6 could be found. However, *ECS* requires computing such distance to associate the point with a particular center during the clustering process. Therefore, $\wp(c_i, s_k)$ is estimated considering the number of patterns in different positions in each permutation (variables that do not match). This value is still decremented by one, because even if all I patterns were in different positions in each permutation, it would be generated at most $I - 1$ intermediary solutions.

In Fig. 7, some paths from the center $\{1, 2, 3, 4\}$ are shown. The solutions in white boxes are candidates to be assimilated in a possible path. The arrows form examples of patches. Solutions in gray belong to the 2-swap neighborhood (only one swap) of the center. The implemented algorithm for path assimilation is a non-deterministic one, since it chooses random positions to make swaps. The maximum number of intermediary points evaluated may be bounded by a running parameter.

The component AM manages the clusters that must be investigated by the component LS and remove the inactive ones alike previously mentioned application.

5.3 Computational experiments

The ECS was coded in ANSI C and it was run on Intel AMD (1.33 GHz) platform. Table 3 presents the instance problems considered, along with the

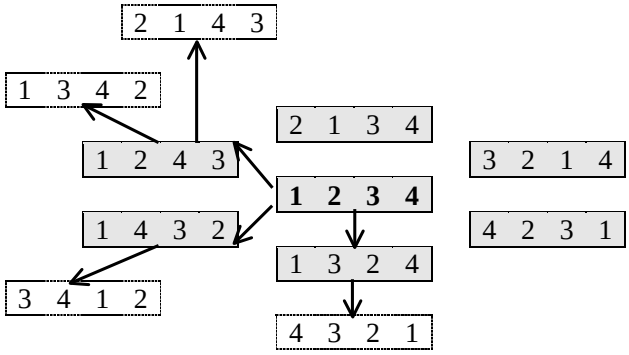

Fig. 7. Examples of 2-swap paths in a 4−pattern instance.

corresponding number of nets/gates and the best-known number of tracks (expected solution). Chosen problems were the largest ones found in the literature [28].

Table 3. GMLP instances.

instance	gates	nets	best-known solution
v4090	27	23	10
x0	48	40	11
v4470	47	37	9
w2	33	48	14
w3	70	84	18
w4	141	202	27

The results were examined from two viewpoints: the best setting for ECS and comparison against other approaches. Several settings of $\mathcal{NC}/|P|$ were tested to achieve the best performance. The number of clusters ($\mathcal{NC}$) was varied from 10 to 40 clusters. The population size ($|P|$) was varied between 100 and 1300 individuals.

Best setting

Each test consists of 20 trials, allowing ECS to perform $10,000,000$ objective function calls (bound for failed search) in each trial, at most. It is especially interesting high success rate (SR) with low number of objective function calls (FC). Such tuning performance had not been considered in early work [11].

The results for each problem instance are summed up in tables 4-6. The tables show the following information: the tried setting $\mathcal{NC}/|P|$, the number of well-succeeded runnings (WS), as well its percentage (SR) in 20 trials, the average of objective function calls for well-succeeded trials (WS-FC), and

the average of the maximum of tracks found (MOT), as well the equivalent error percentage (ER). One can note the ill-succeeded trials were not taken in account for calculating the average of function calls.

Table 4. Results for $x0$ and $v4090$ instances.

	x0					v4090				
$\mathcal{NC}$ / $\|P\|$	WS	(%SR)	WS-FC	MOT	(%ER)	WS	(%SR)	WS-FC	MOT	(%ER)
10/100	15	75	32,985	11.3	2.3	20	100	2,288	10.0	0.0
10/500	15	75	100,272	11.4	3.2	20	100	7,446	10.0	0.0
10/900	15	75	165,851	11.3	2.3	20	100	9,940	10.0	0.0
10/1300	17	85	279,504	11.3	2.3	20	100	16,554	10.0	0.0
20/300	16	80	78,367	11.4	3.2	20	100	7,525	10.0	0.0
20/500	17	85	101,012	11.2	1.4	20	100	9,104	10.0	0.0
20/700	16	80	185,515	11.2	1.8	20	100	16,130	10.0	0.0
20/900	14	70	163,723	11.4	3.2	20	100	17,739	10.0	0.0
30/500	20	100	119,296	11.0	0.0	20	100	10,887	10.0	0.0
30/700	18	90	146,819	11.1	0.9	20	100	18,459	10.0	0.0
30/900	17	85	195,685	11.2	1.8	20	100	19,417	10.0	0.0
40/100	15	75	37,376	11.4	3.6	20	100	6,277	10.0	0.0
40/300	18	90	63,472	11.1	0.9	20	100	8,875	10.0	0.0
40/500	15	75	100,096	11.3	2.3	20	100	14,736	10.0	0.0
40/700	17	85	123,973	11.2	1.4	20	100	21,626	10.0	0.0
40/900	17	85	237,124	11.2	1.4	20	100	21,690	10.0	0.0
40/1300	20	100	239,983	11.0	0.0	20	100	32,551	10.0	0.0

The Table 4 shows the results for instances $x0$ and $v4090$. All ECS settings were able to solve instance $v4090$. Although, the computational effort has varied from few $2,288.30$ to $32,550.70$ FCs. Hence, the setting $\mathcal{NC} = 10/|P| = 100$ was enough to obtain good results for this instance probably because the instance size.

For the instance $x0$, the best result ($SR = 100$ %) was obtained with $\mathcal{NC} = 30/|P| = 500$, performing about $119,296$ FCs. Another good result was obtained with $\mathcal{NC} = 40/|P| = 1300$, but a considerable computational effort (about $239,983.3$ FCs, in average) was needed, especially when compared with other WS-FC, as $63,472.2$ for setting $\mathcal{NC} = 40/|P| = 300$, for example.

The Table 5 shows the results for instances $w2$ and $w3$. ECS was able to solve instance $w2$ performing about WS-FC= $8,757.06$ (SR= 90%). To obtain (SR= 100%), the best setting was $\mathcal{NC} = 20/|P| = 300$, performing WS-FC= $26,185.20$. The ECS setting $\mathcal{NC} = 30/|P| = 500$ has needed about twice function evaluations. For instance $w3$, ECS has obtained at most SR= 60% ($\mathcal{NC} = 40/|P| = 700$) with WS-FC= $583,429.33$. This instance appears to be more difficult for ECS.

The Table 6 shows the last two instance problems: $v4470$ and $w4$. In the best setting, it was possible to achieve the best known solution in a half of trials (SR about 50) for both. Together $w3$, these are good instances to

Table 5. Results for $w2$ and $w3$ instances.

	w2					w3				
$\mathcal{NC}$ / $\|P\|$	WS	(%SR)	WS-FC	MOT	(%ER)	WS	(%SR)	WS-FC	MOT	(%ER)
10/100	18	90	8,757	14.1	0.7	10	50	445,116	19.3	7.2
10/500	19	95	61,332	14.1	0.4	6	30	270,106	20.0	10.8
10/900	20	100	58,685	14.0	0.0	8	40	398,604	19.5	8.3
10/1300	20	100	58,685	14.0	0.0	4	20	645,862	19.7	9.2
20/300	20	100	26,185	14.0	0.0	10	50	745,618	19.5	8.3
20/500	20	100	40,909	14.0	0.0	6	30	464,372	19.5	8.3
20/700	20	100	79,751	14.0	0.0	8	40	340,648	19.3	7.2
20/900	20	100	76,747	14.0	0.0	7	35	902,890	19.5	8.1
30/500	20	100	50,938	14.0	0.0	10	50	540,893	19.2	6.4
30/700	20	100	63,769	14.0	0.0	6	30	364,486	19.7	9.2
30/900	20	100	89,137	14.0	0.0	8	40	818,991	19.1	6.1
40/100	17	85	23,860	14.2	1.1	5	25	867,674	19.9	10.6
40/300	19	95	33,255	14.1	0.4	8	40	213,753	19.3	6.9
40/500	20	100	39,389	14.0	0.0	5	25	278,447	20.0	10.8
40/700	20	100	65,893	14.0	0.0	12	60	583,429	19.2	6.4
40/900	20	100	70,831	14.0	0.0	11	55	535,272	18.8	4.2
40/1300	20	100	123,245	14.0	0.0	5	25	696,676	19.6	8.9

Table 6. Results for $v4470$ and $w4$ instances.

	v4470					w4				
$\mathcal{NC}$ / $\|P\|$	WS	(%SR)	WS-FC	MOT	(%ER)	WS	(%SR)	WS-FC	MOT	(%ER)
10/100	4	20	51,961	10.1	12.2	2	10	1,947,715	29.4	8.7
10/500	9	45	140,951	9.7	7.8	7	35	1,678,860	28.2	4.4
10/900	5	25	169,861	10.0	10.6	5	25	1,513,475	28.9	6.9
10/1300	7	35	303,022	9.8	8.3	3	15	1,161,946	28.8	6.5
20/300	2	10	59,704	10.2	12.8	7	35	1,971,477	28.1	4.1
20/500	4	20	129,967	10.2	12.8	8	40	1,334,679	28.3	4.8
20/700	2	10	137,840	10.1	12.2	8	40	1,554,822	28.3	4.6
20/900	4	20	184,022	10.0	11.1	4	20	2,415,212	28.8	6.7
30/500	12	60	169,136	9.5	5.6	11	55	1,695,924	27.9	3.1
30/700	2	10	170,157	10.1	12.2	2	10	2,560,380	28.6	5.9
30/900	3	15	227,527	10.2	13.3	5	25	1,799,301	28.1	3.9
40/100	7	35	109,720	9.9	10.0	4	20	2,099,928	28.3	4.8
40/300	6	30	561,623	10.0	11.1	4	20	1,250,663	28.2	4.3
40/500	4	20	129,457	10.0	10.6	7	35	1,632,453	28.4	5.2
40/700	5	25	233,898	10.2	13.3	8	40	2,167,944	27.9	3.1
40/900	3	15	288,405	10.1	11.7	7	35	1,557,529	28.3	4.6
40/1300	8	40	345,202	9.8	8.9	4	20	1,144,752	28.8	6.5

estimate an efficient setting for the algorithm. One can note that the best setting was $\mathcal{NC} = 30/|P| = 500$ only for $v4470$ and $w4$. For instance $w3$, this setting has obtained the third best SR, but still good. Hence, for the 3 hardest instances in this work, there have been found the best known solution at least

in SR= 50% of the trials, setting the algorithm with 30 maximum clusters and 500 individuals.

For the 4 large instances, the setting $\mathcal{NC} = 30/|P| = 500$ has been the more successful. It is due to, probably, the range of size of the instances. By the early experience with the unconstrained continuous optimization, the maximum number of clusters could mean that no more than 30 promising search areas, along the evolution, should be concurrently investigated, by the local searcher (LS), once detected. The capability of concurrently detection ($\mathcal{NC}$) allows exploitation moves in such search areas before the diversity loss, commonly observed in evolutionary algorithms, which can be properly delayed by an adequate number of individuals. In this case, no more than 500 individuals were enough.

Comparison against other approaches

In the second experiment, ECS is compared against other approach found in literature that was applied to the same instances: the Parallel Memetic Algorithm (PMA). Besides a parallel algorithm, employing a suitable migration policy, PMA presents a new 2-swap local search with a reduction scheme, which discards useless swaps, avoiding unnecessary objective function calls. The PMA results were considered so far the best ones obtained in the literature, specifically with large GMLP instances [34].

The Fig. 8-11 gives a close up in WS-FC and SR, also showing the best result found by PMA for the 4 hardest GMLP instances: $x0$, $v4470$, $w3$ and $w4$. The figures show the comparison between the best PMA result, represented by dashed lines in top and bottom of the next figures, and all ECS settings for each hard instance. The PMA results were obtained in 10 trials, for each instance.

Instance $v4090$ was not tested by PMA authors. For instance $w2$, the best PMA setting was able to obtain SR= 100% with $3,523$ objective function calls, in average, i.e., about a half of ECS function calls ($8,757.06$), in its fast setting.

According to the authors, as it is shown in Fig. 8, PMA was able to find the best known solution for instance $x0$ in all 10 trials, evaluating the objective function $43,033$, in average [34]. Best setting of ECS has reached the same solution in all 20 trials, but evaluating about $119,296$ ($\mathcal{NC} = 30/|P| = 500$), at least. Working with smaller population (100 individuals), in general, ECS has performed less WS-FC, but not reaching the same number of well-succeeded runnings (about 75%).

ECS also has not surpassed PMA, neither by WS-FC nor by WS, for instance $w3$, with any setting. Actually, ECS has a poor performance for this instance (Fig. 9).

For instance $v4470$ and $w4$, ECS has reached better results (Figs. 10 and 11). For $v4470$, the best setting ($\mathcal{NC} = 30/|P| = 500$) has obtained the same WS percentage of PMA (60%) but with a little bit of advantage with respect to objective function calls: $169,136$ against $176,631$ of PMA.

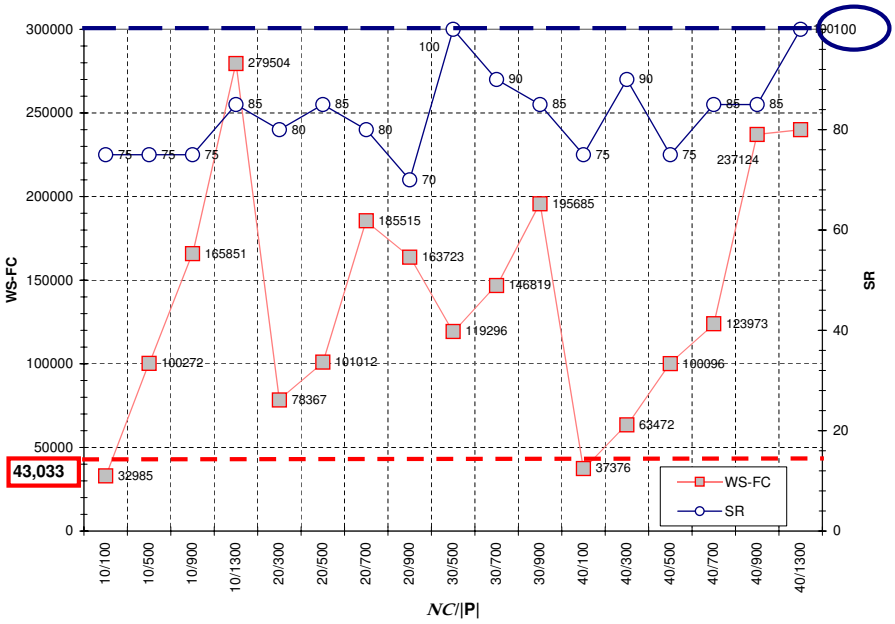

Fig. 8. Best PMA result against all ECS settings for x0 instance.

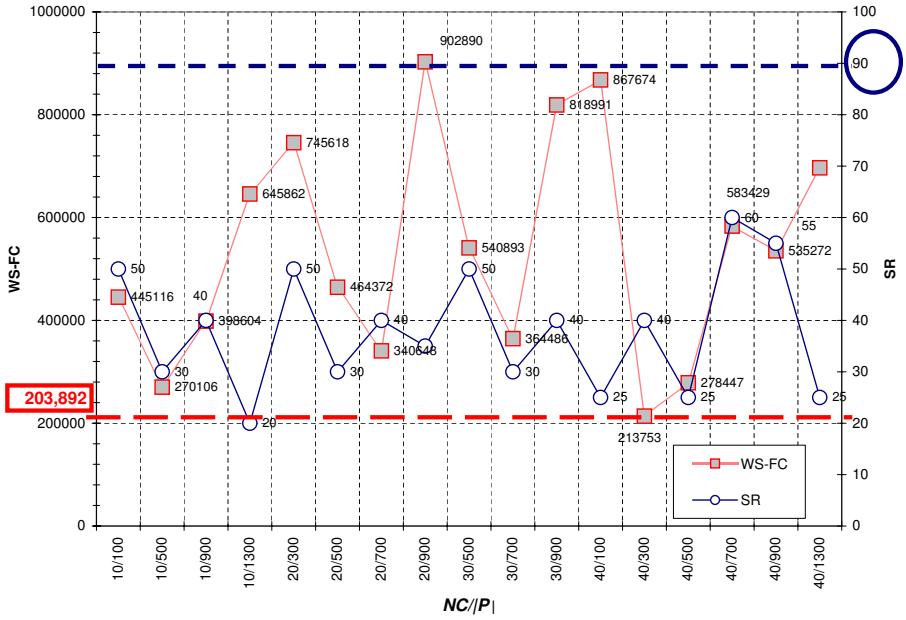

Fig. 9. Best PMA result against all ECS settings for w3 instance.

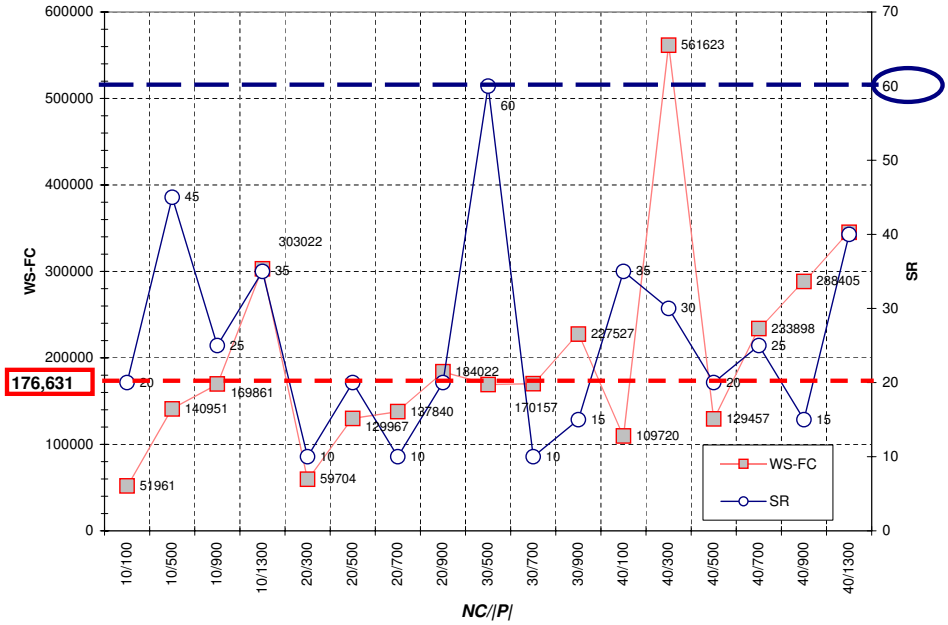

Fig. 10. Best PMA result against all ECS settings for v4470 instance.

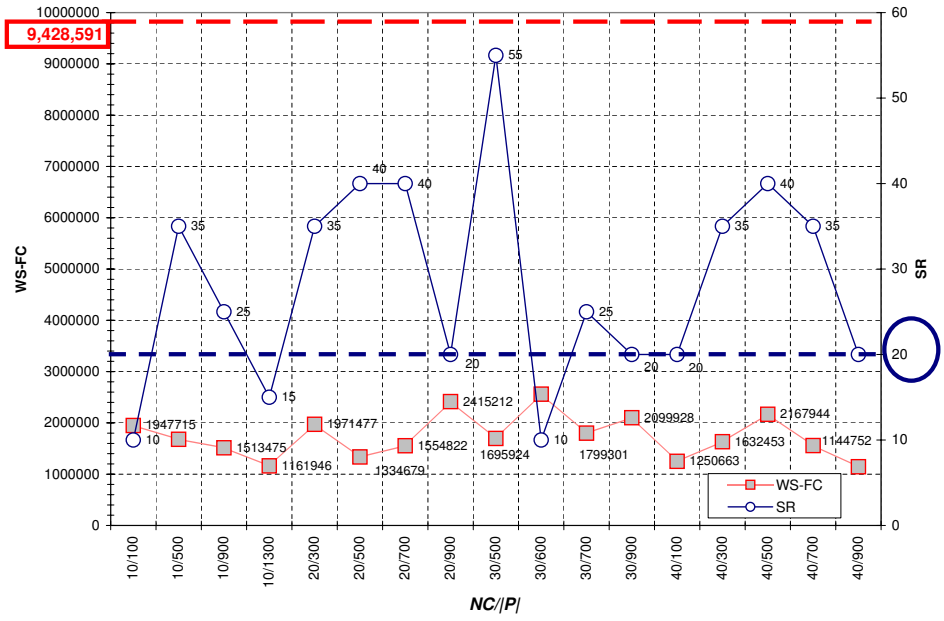

Fig. 11. Best PMA result against all ECS settings for w4 instance.

For instance $w4$, the largest one found in literature, ECS has reached meaningful results. While PMA has found the best known solution in 20% of the trials, ECS got it in 55% of the trials, performing only $1,695,924$ FCs ($\mathcal{NC} = 30/|P| = 500$) against $9,428,591$ FCs, in average, by PMA.

6 Conclusion

This chapter proposes a new way of detecting promising search areas based on clustering: the Clustering Search (*CS). The *CS can be splitted off in four conceptually independent parts: a search metaheuristic (SM); an iterative clustering (IC) component; an analyzer module (AM); and a local searcher (LS). When SM is an evolutionary algorithm, the resulting approach is called Evolutionary Clustering Search (ECS).

In a general way, *CS attempts to locate promising search areas by cluster of solutions. The clusters work as sliding windows, framing the search areas and giving a reference point (center) to problem-specific local search procedures. Furthermore, the cluster center itself is always updated by a permanent interaction with inner solutions, in a process called assimilation.

A real-coded version of ECS for unconstrained continuous optimization was early proposed, however some relevant aspects of that application were examined in this chapter, as a recipe for further applications. Besides, ECS also was applied to pattern sequencing problems and its results were here first presented. Pattern sequencing problems arise in scenarios involving arrangement of a set of client orders, gates in VLSI circuits, cutting patterns, etc.

The computational results were examined from two viewpoints: the best setting for ECS and comparison against other approaches. Several algorithm settings were tested to achieve the best performance. For the 3 hardest instances in literature, there have been found the best known solution at least SR= 50% of the trials, setting the algorithm with 30 maximum clusters and 500 individuals. In comparison against the best results so far known, ECS has achieved similar and sometimes superior performance in both applications here presented.

For further work, it is intended to build new algorithms based on *CS, including other metaheuristics as Ant Colony System, Immune Systems, and Evolution Strategies.

References

1. Jelasity M, Ortigosa P, García I (2001) UEGO, an Abstract Clustering Technique for Multimodal Global Optimization, Journal of Heuristics 7(3):215-233.
2. Ray, T. and Liew, K.: Society and civilization: An optimization algorithm based on the simulation of social behavior, IEEE Transaction on Evolutionary Computation 7(4)386-396.

3. Hart W E, Rosin C R, Belew R K, Morris G M (2000) Improved evolutionary hybrids for flexible ligand docking in AutoDock. Optimization in Computational Chemistry and Molecular Biology 209-230.
4. Moscato, P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M, Glover F (eds) New Ideas in Optimization. McGraw-Hill London 219-234.
5. Lozano M, Herrera F, Krasnogor N, Molina D (2004) Real-coded memetic algorithms with crossover hill-climbing. Evolutionary Computation 12(3):273-302.
6. Yen J, Lee B (1997) A Simplex Genetic Algorithm Hybrid, In: IEEE International Conference on Evolutionary Computation ICEC97 175-180.
7. Nelder J A, Mead R (1956) A simplex method for function minimization. Computer Journal 7(23):308-313.
8. Chelouah R, Siarry P (2003) Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. European Journal of Operational Research 148(2):335-348.
9. Glover F (1998) A template for scatter search and path relinking. Selected Papers from the Third European Conference on Artificial Evolution. Springer-Verlag 3-54.
10. Oliveira A C M (2004) Algoritmos Evolutivos Híbridos com Detecção de Regiões Promissoras em Espaços de Busca Contínuos e Discretos. PhD Thesis. Instituto Nacional de Pesquisas Espaciais INPE São José dos Campos Brasil.
11. Oliveira A C M, Lorena L A N (2004) Detecting promising areas by evolutionary clustering search. In: Brazilian Symposium on Artificial intelligence SBIA2004 Springer LNAI 3171:385-394.
12. Chaves A A, Lorena L A N (2005) Hybrid algorithms with detection of promising areas for the prize collecting travelling salesman problem In: V International Conference on Hybrid Intelligent Systems HIS2005 49-54.
13. Feo T, Resende M (1995) Greedy randomized adaptive search procedures. Journal of Global Optimization 6:109-133.
14. Mladenovic N, Hansen P (1997) Variable neighborhood search. Computers and Operations Research 24:1097-1100.
15. Yager R R (1990) A model of participatory learning. IEEE Transaction on Systems, Man and Cybernetics 20(5):1229-1234.
16. Silva L R S (2003) Aprendizagem Participativa em Agrupamento Nebuloso de Dados, MA Thesis, Faculdade de Engenharia Elétrica e de Computação UNICAMP Campinas Brasil.
17. Eshelman L J, Schawer J D (1993) Real-coded genetic algorithms and interval-schemata, In: Foundation of Genetic Algorithms-2, L. Darrell Whitley (eds), Morgan Kaufmann Pub 187-202.
18. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control and Cybernetics 39:653-684.
19. Bersini H, Dorigo M, Langerman S, Seront G, Gambardella L M (1996) Results of the First International Contest on Evolutionary Optimisation 1stICEO In: Proc IEEE-EC96 611-615.
20. De Jong K A (1975) An analysis of the behavior of a class of genetic adaptive systems, PhD Thesis, *University of Michigan Press* Ann Arbor.
21. Digalakis J, Margaritis K (2000) An experimental study of benchmarking functions for Genetic Algorithms. IEEE Systems Transactions 3810-3815.
22. Goldberg, D E (1989) Genetic algorithms in search, optimisation and machine learning. Addison-Wesley.

23. Michalewicz Z (1996) Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag New York.
24. Hooke R, Jeeves T A (1961) "Direct search" solution of numerical and statistical problems. Journal of the ACM 8(2):212-229.
25. Laguna M, Martí R (2002) The OptQuest Callable Library In Optimization Software Class Libraries, Stefan Voss and David L. Woodruff (eds), Kluwer Academic Pub 193-218.
26. Fink A, Voss S (1999) Applications of modern heuristic search methods to pattern sequencing problems, Computers and Operations Research 26(1):17-34.
27. Oliveira A C M, Lorena L A N (2005) Population training heuristics. In: European Conference on Evolutionary Computation in Combinatorial Optimization. Springer Berlin LNCS 3448:166-176.
28. Linhares A, Yanasse H, Torreão J (1999) Linear gate assignment: a fast statistical mechanics approach. IEEE Transactions on Computer-Aided Designed of Integrated Circuits and Systems, 18(12):1750-1758.
29. Möhring R (1990) Graph problems related to gate matrix layout and PLA folding. Computing 7:17-51.
30. Kashiwabara T, Fujisawa T (1979) NP-Completeness of the problem of finding a minimum clique number interval graph containing a given graph as a subgraph, In: Symposium of Circuits and Systems.
31. Golumbic M (1980) Algorithmic graph theory and perfect graphs. Academic Press New York.
32. Linhares A (2002) Industrial pattern sequencing problems: some complexity results and new local search models. PhD Thesis, Instituto Nacional de Pesquisas Espaciais INPE São José dos Campos Brasil.
33. Syswerda G (1991) Schedule optimization using genetic algorithms. Handbook of Genetic Algorithms, Van Nostrand Reinhold New York 332-349.
34. Mendes A, Linhares A (2004) A multiple population evolutionary approach to gate matrix layout, International Journal of Systems Science, Taylor & Francis 35(1):13-23.

Index