UEGO, an Abstract Clustering Technique for Multimodal Global Optimization*

Márk Jelasity
Research Group on Artificial Intelligence
MTA-JATE, Szeged, Hungary
jelasity@inf.u-szeged.hu
Pilar Martínez Ortigosa and Inmaculada García
Department of Computer Architecture and Electronics
University of Almería, Almería, Spain
ortigosa@ualm.es, igarcia@ualm.es

Abstract

In this paper, UEGO, a new general technique for accelerating and/or parallelizing existing search methods is suggested. The skeleton of the algorithm is a parallel hill climber. The separate hill climbers work in restricted search regions (or clusters) of the search space. The volume of the clusters decreases as the search proceeds which results in a cooling effect similar to simulated annealing. Besides this, UEGO can be effectively parallelized; the communication between the clusters is minimal. The purpose of this communication is to ensure that one hill is explored only by one hill climber. UEGO makes periodic attempts to find new hills to climb. Empirical results are also presented which include an analysis of the effects of the user-given parameters and a comparison with a hill climber and a GA.

1 Introduction

In this section a short introduction to the history and motivation behind developing UEGO is presented, but first let us state what the acronym means. UEGO stands for *Universal Evolutionary Global Optimizer*. However, it must be admitted from the start that this name is not over-informative, and the method is not even 'evolutionary' in the usual sense. In spite of this we have kept the name for historical reasons.

1.1 Roots

The predecessor of UEGO was GAS, a steady-state genetic algorithm with subpopulation support. For more details on GAS the reader should consult [10]. Let us note however that this work is self contained and does not assume any further knowledge about GAS; it will not be mentioned outside of this section only in connection with the empirical comparison results.

GAS has several attractive features. Perhaps the most important of them is that it offers a solution to the so-called niche radius problem which is a common problem of many simple niching techniques such as *fitness sharing* ([3] or [4]), *simple iteration* or the *sequential niching* [2]. This

*M. Jelasity, P. M. Ortigosa, and I. García. UEGO, an abstract clustering technique for multimodal global optimization. *Journal of Heuristics*, 7(3):215-233, May 2001. This work was supported by the Ministry of Education of Spain (CICYT TIC96-1125-C03-03), the Consejeria de Educación de la Junta de Andalucia (07/FSC/MDM) and the OTKA grant T25721.

problem is related to functions that have multiple local optima and whose optima are unevenly spread throughout the search space. With such functions the *niche radius* cannot be set correctly since if it is too small the search becomes ineffective and if it is too large those local optima that are too close to each other cannot be distinguished. The solution of GAS involves a 'cooling' technique which enables the search to focus on the promising regions of the space, starting off with a relatively large radius that decreases as the search proceeds.

However, the authors of GAS came in for a number of criticisms, one being that the algorithm was too much complex, and another that parallel implementation turned out to have many pitfalls associated with it.

Although UEGO is based on GAS there are two major differences that were motivated by the need for a better parallel implementation and the requirement of using domain specific knowledge in an effective way.

The structure of the algorithm has been greatly simplified. As a result the parallel implementation is much easier and the basic ideas become more accessible. This is important because, as the results of the paper will show, UEGO performs similarly or better than the GA and the simple stochastic hill climber (SHC) on our test problems, and at the same time it can be parallelized better than these methods [14, 15].

1.2 Basic Ideas

The basic idea is that in multimodal optimization problems where the objective function has multiple local optima and the structure of these optima should be discovered beside the global optimum, it may be useful to ensure that the optimizer does not waste its time exploring the same region multiple times but simultaneosly new and promising regions are found. This goal can be achieved by applying a non-overlapping set of clusters which define sub-domains for the applied optimizer. Based on the results of the optimizer, the search process can be directed towards smaller regions by creating a new set of non-overlapping clusters that consists of smaller sub-domains. This process is a kind of cooling method similar to simulated annealing. A particular cluster is not a fixed part of the search domain; it can move through the space as the search proceeds. The non-overlapping property of the set of clusters is maintained however.

UEGO is abstract in the sense that the 'cluster-management' and the cooling mechanism has been logically separated from the actual optimization algorithm, so it is possible to implement any kind of optimizers that work inside a cluster. This allows the adaptation of the method to a large number of possible search domains using existing domain specific optimizers while enjoying the advantages of having muliple non-overlapping clusters which ensures that search effort is focused on interesting regions.

In this paper an SHC is implemented as the optimizer algorithm. This choice is supported by results that show that the performance of the SHC is similar to that of the GA in many cases and sometimes may even be better (e.g. [13, 11, 19, 8]). In [5] a GA with very small population size (1) has been suggested for the graph coloring problem, which is in fact an SHC. Our results confirm that the SHC can indeed outperform the GA at least on the problems and parameter settings we considered.

1.3 A Note on Terminology

In the following sections the term *species* will be used instead of e.g. *cluster, zone, region, sub-domain* etc. This may be strange since in evolutionary computation this term normally means a population of similar individuals while here it denotes a subset of the search domain. This is not a major problem however; a species in our sense is nothing else but the set of all possible members that are similar according to some similarity measure which is in fact a function of e.g. the application domain. We think that the behavior of a species as will be defined later has strong biological analogies.

The actual number of solutions in a species is given by the applied optimizer. In the case of SHC it is one but e.g. in the case of a GA it may be larger.

1.4 Outline of the Paper

Section 2 describes UEGO; the basic concepts, the general algorithm and the theoretical tools that are used to set the parameters of the system based on a few user-given parameters. Sections 3 and 5 discuss the experimental results that describe the effects of these parameters of the algorithm on the quality of the results and compare UEGO with a simple GA (GAS), a stochastic hill climber (SHC) and a multistart hill climber (MHC) using a set of test functions. Section 6 then provides a short summary.

2 Description of UEGO

In this section the basic concepts, the algorithm, and the setting of the parameters are outlined. In UEGO, a domain specific optimizer has to be implemented. Wherever we refer to 'the optimizer' in the paper we mean this optimizer.

2.1 Basic Concepts

In the following it will be assumed that the parameters of the function take values from the same interval. This is easy to achieve for any function via normalization.

A key notion in UEGO is that of a *species*. A species can be thought of as a window on the whole search space. This window is defined by its *center* and a *radius*. The center is a solution, and the radius is a positive number. Of course, this definition assumes a *distance* defined over the search space. The role of this window is to 'localize' the optimizer which is always called by a species and can 'see' only its window, so every new sample is taken from there. This means that the largest step made by the optimizer in a given species is no larger than the radius of the given species. If the value of a new solution is better than that of the old center, the new solution becomes the center and the window is moved.

The radius of a species is not arbitrary; it is taken from a list of decreasing radii, the *radius list*. The radii decrease in a regular fashion in geometrical progression. The first element of this list is always the diameter of the search space which will ensure that the largest species always contains the whole space independently of its center. The diameter is given by the largest distance between any two possible solutions according to the distance mentioned above. If the radius of a species is the ith element of the list, then we say that the *level* of the species is i.

During the optimization process, a list of species is kept by UEGO. The algorithm is in fact a method for managing this *species-list* (i.e. creating, deleting and optimizing species); it will be described in Section 2.2.

2.2 The Algorithm

Firstly, some parameters of UEGO will be very briefly mentioned more details of which can be found in Section 2.3.

As we mentioned earlier, every species has a fixed level during its lifetime. Species-level operators may change this level however as will be described. The maximal value for the level is given by a parameter called `levels`. Every valid level i (i.e. for levels from [1,`levels`]) has a radius value (r_i) and two function evaluation numbers. One is used when new species are created at a given level (new_i) while the other is used when optimizing individual species (n_i). To define the algorithm fully, one more parameter is needed: the maximal length of the above-mentioned species list (`max_spec_num`).

The basic algorithm is shown in Figure 1. Now the procedures called by UEGO will be described.

`Init_species_list.` Create a species list consisting of one species with a random center at level 1.

```
uego
        init_species_list()
        optimize_species( n[1] )
        for i = 2 to levels
                create_species( new[i]/length(species_list) )
                fuse_species( r[i] )
                shorten_species_list( max_spec_num )
                optimize_species( n[i]/max_spec_num )
                fuse_species( r[i] )
        rof
ogeu
```

Figure 1: The basic algorithm of UEGO.

`Create_species(evals).` For every species in the list, create random pairs of solutions in the 'window' of the species, and for every such pair take the middle of the *section* connecting the pair. If the objective function value of the middle is worse than the values of the pair, then the members of the pair are inserted in the species list. Every new inserted species is assigned the actual level value (`i` in Figure 1).

The motivation behind this method is simple: to create species that are on different 'hills' so ensuring that there is a valley between the new species. Of course this is a heuristic only. In higher dimensions it is possible (in fact typical) that many species are created even if the function is unimodal. This is an unlucky effect which is handled by the cooling process to ensure that at the beginning the algorithm does not create too many species capturing only the rough structure of the landscape. The parameter of this procedure is an upper bound of the function evaluations. Note that this algorithm needs a definition of *section* in the search space.

`Fuse_species(radius).` If the centers of any pair of species from the species list are closer to each other than the given radius, the two species are fused. The center of the new species will be the one with the best function value while the level will be the minimum of the levels of the original species to be fused. Of course this method does not ensure that no species will overlap after fusion though the amount of the overlapping regions is typically highly decreased.

`Shorten_species_list(max_spec_num).` Delete species to reduce the list length to the given value. Higher level species are deleted first.

`Optimize_species(evals).` Start the optimizer for every species with the given evaluation number (i.e. every single species in the actual list receives the given number of evaluations). See Section 2.1.

It is clear that if for some level i the species list is shorter than the allowed maximal length, `max_spec_num`, the overall number of function evaluations will be smaller than n_i (see Figure 1, `optimize_species`). In our implementation we use the difference of the actual number of function evaluations and n_i to find more species. This technique has no effect when there are many species but if the number of species is small, a lot of extra effort is devoted to finding new ones.

Finally, let us make a remark about a possible parallel implementation. The most time-consuming parts of the basic algorithm is the creation and optimization of the species. Note that these two steps can be done independently for every species, so each species can be assigned to a different processor. Note also that a species is defined by its center and its level, so the amount of information used in communications is really small. In GAS algorithm a species is a set of individuals and there are relations among species and even among individuals, so it is quite difficult to send a species or an individual to another processor. The complexity of the possible parallel approach would be high enough. As our experimental results will clearly show, sequential UEGO performs slightly better than the SHC and the GA even when the number of species is as high as 200.

2.3 Parameters of UEGO

The most important parameters are those that belong to the different levels: the radii and two function evaluation numbers for species creation and optimization (see Figure 1). In this section a method is described which sets these parameters using a few easy-to-understand parameters set by the user. The experimental sections will provide further guidelines on the meaning and setting of these remaining user-given parameters.

We will now make use of the notation introduced in Section 2.2. The user-given parameters are listed below. Short notations (in brackets) that will be used in equations in the subsequent sections are also given.

`evals` (N)**:** The maximal number of function evaluations the user allows for the whole optimization process. Note that the actual number of function evaluations may be less than this value.

`levels` (l)**:** The maximal level value (see Figure 1).

`threshold` (ν)**:** The meaning of this parameter will be explained later.

`max_spec_num` (M)**:** The maximal length of the species list.

`min_r` (r_l)**:** The radius that is associated with the maximal level, i.e. `levels`.

The parameter setting algorithm to be described can use any four of the above five values while the remaining parameters are set automatically.

Speed of the optimizer. Before presenting the parameter setting method, the notion of the *speed* of the optimizer must be introduced. As explained earlier, the optimizer cannot make a larger step in the search space than the radius of the species it is working in. Furthermore if the center of a species is far from every local optimum then these steps will be larger while if the center is already close to a local optimum then the steps will be very small. Given a certain number of evaluations, it is possible to measure the distance the given species moves during the optimization process assuming that the species is suboptimal. This distance can be approximated (as a function of the radius and evaluations) for certain optimizers using ideal landscapes (such as linear functions) with the help of mathematical models or experimental results. This naturally leads to a notion of speed that will characterize a given domain (assuming e.g. a linear landscape) and will depend on the species radius. Speed will be denoted by $v(r)$. As we will not give any actual approximations here, the reader should refer to [10].

The parameter-setting method is based on intuitive and reasonable *principles*. These principles are now described below.

Principle of equal chance. At a level, every species moves a certain distance from its original center due to optimization. This principle ensures that every species will receive the number of evaluations that is enough to make it move at least a fixed distance assuming that the speed of this motion is $v(r)$. A species will not necessarily move that far but the definition of speed is such that if the species is far from the local optima then it will move approximately the given distance. This common distance is defined by $r_1\nu$. The meaning of `threshold` can now be given: it directly controls the distance a species is allowed to cover, so it actually controls the probability that they will eventually represent a local optimum: the further a species can go the higher the probability of reaching a local optimum is (and the more expensive the optimization is). Recall that r_1 is always the diameter of the search space. Now the principle can be formalized:

$$\frac{v(r_i)n_i}{M} = r_1\nu \qquad (i = 2, \ldots, l) \tag{1}$$

Principle of exponential radius decreasing. This principle is quite straightforward; given the smallest radius and the largest one (r_l and r_1) the remaining radii are expressed by the exponential function

$$r_i = r_1(\frac{r_l}{r_1})^{\frac{i-1}{l-1}} \qquad (i = 2, \ldots, l). \tag{2}$$

Principle of constant species creation chance. This principle ensures that even if the length of species list is maximal, there is a chance of creating at least two more species for each old species. It also makes a strong simplification, namely that all the evaluations should be set to the same constant value.

$$new_i = 3M \qquad (i = 2, \ldots, l) \tag{3}$$

Decomposition of N. Let us define $new_1 = 0$ for the sake of simplicity since new_1 is never used by UEGO. The decomposition of N results in the trivial equation

$$\sum_{i=1}^{l}(n_i + new_i) = (l-1)3M + \sum_{i=1}^{l} n_i = N \tag{4}$$

making use of (3) in the process. One more simplification is possible; set $n_1 = 0$ whenever $l > 1$. Note that if $l = 1$ then UEGO reduces to the optimizer it uses for optimizing the species.

Expressing n_i from (1) and substituting it into (4) we can write

$$(l-1)3M + \sum_{i=2}^{l} \frac{Mr_1\nu}{v(r_i)} = N \tag{5}$$

Using (2) as well, it is quite evident that the unknown parameters in (5) are just the user given parameters and due to the monotony of this equation in every variable, any of the parameters can be given using effective numerical methods provided the other parameters are known. Using the above principles the remaining important parameters (n_i, new_i and r_i) can be evaluated as well. Note however that some of the configurations set by the user may be infeasible.

3 Experiments with Real Functions

In this section experimental results on real functions will be presented. Due to the stochastic nature of UEGO, all the numerical results given in this work are average values of fifty executions, obtaining an enough statistic sample of experiments. From this data set the corresponding confidence intervals (95%) were computed (see [16]). These confidence intervals have not been represented because it would have messed up the plots since there are multiple curves in each graphic, and also these intervals were too narrow, so they could not be distinguished from the average values.

In this section, a set of four test functions and two classical multimodal functions (Griewank, Rastrigin) have been used to evaluate UEGO. These test functions have different characteristics w.r.t. dimensionality and the number of local optima so, it is possible to illustrate the effect of these characteristics on the performance of UEGO. Comparisons with the ancestor of UEGO, GAS [10] and the simple and multistart version of a hill climber (SHC and MHC) described in [17] will also be given.

3.1 Test Problems

A first experimental stage was carried out on a set of four new defined functions. We chose not to use well-known benchmark functions for testing. The reason for this is that we agree with the ideas discussed in [7], namely that for doing scientific tests it is more convenient to use functions that differ only in controllable features. This allows the analysis of the effect of only one separated

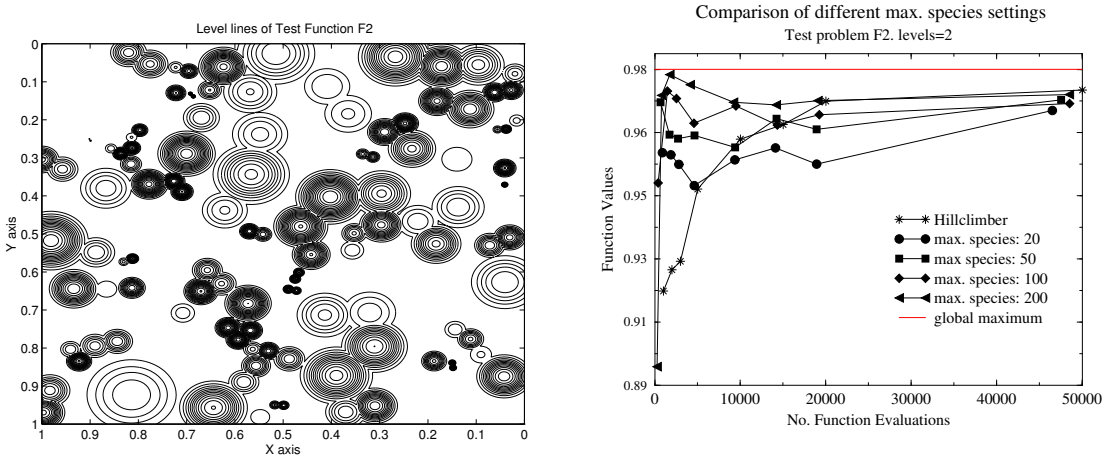

Figure 2: The plot of the test function F2 and a sample from the results of our preliminary experiments.

feature of the test problem, e.g. the number of local optima. There is another reason: using widely accepted benchmark problems prevents developing methods that perform well only on special kind of problems. We believe that it is more important to characterize the ideal problem for optimization methods than trying to show that they outperform other methods on as many benchmark problems as possible. In a second stage, UEGO was evaluated and compared to other algorithms, using Griewank and Rastrigin functions (see Section 4.1).

In the experiments to be discussed here we wanted to examine the effects of dimensionality and the number of local optima. Therefore we used four test functions that are characterized in Table 1. The construction of these functions starts with a user-given list of local optimum sites

Table 1: Characteristics of the four test functions.

	F1	F2	F3	F4
Type	$[0,1]^2 \rightarrow \mathbb{R}$	$[0,1]^2 \rightarrow \mathbb{R}$	$[0,1]^{30} \rightarrow \mathbb{R}$	$[0,1]^{30} \rightarrow \mathbb{R}$
# of maxima	5	125	5	125

(o) and the corresponding function values (f_o). All the function values have to be positive. In the first step, we define bell shapes for every site to create the local optima. The height of a bell is given by the function value f_o of its site o, and its radius r is the distance of o from the closest site. The height of the bell at a distance x from o is $f_o g(x)$, where:

$$g(x) = \begin{cases} 1 - \frac{2x^2}{r^2} \text{ if } x < \frac{r}{2} \\ \frac{2(x-r)^2}{r^2} \text{ if } \frac{r}{2} \leq x < r \\ 0 \text{ otherwise} \end{cases}$$

The objective function is the sum of these bells. In the case of our test functions, the coordinates of the maximum sites and their values were randomly taken from $[0,1]$ using a uniform distribution. The two-dimensional function F2 has been drawn in Figure 2.

3.2 The Optimizer and the Settings

For real function optimization the optimizer used by UEGO was the hill climber suggested in [17]. The parameters of the hill climber algorithm were set as in [17]. The parameter ρ_{ub}, that controls

Table 2: Tested values of UEGO parameters for the real functions.

evals	levels	max_spec_num	min_r	threshold
1000, 2000, 3000, 5000, 10000, 15000, 20000, 50000	fixed to 2	20, 50, 100, 200	.003, .005, .01, .03, .05, .08,	automatically set

the maximal step size, was set to the radius of the species from which the optimizer is called. The accuracy of the search was set to $\min(\rho_{ub}/10^3, 10^{-5})$. No fine tuning of the parameters of the optimizer was done.

3.3 The Experiments

In our preliminary experiments the minimal radius (`min_r`) was calculated automatically, the threshold (ν) was fixed to 1 in every run, and the effects of the remaining parameters were examined. However, it soon became clear that it was not the best choice: in the case of F2 the performance often decreased as the number of evaluations was increased. This effect can be seen in Figure 2, where the average of the most-fit species values of the objective function (for fifty runs of UEGO) as a function of the average value of the number of function evaluations has been represented (notice that the number of function evaluations is always less than `evals`). The reason of this strange behavior is that with the increasing number of evaluations the minimal radius became smaller and smaller according to principles described in Section 2.3, so the search slowed down. Another observation was that the performance seemed to be the best when the number of levels was two.

Due to these results we decided to examine the effect of the minimal radius and the maximal number of species with the number of levels fixed to two. The set of the tested values are shown in Table 2. Experiments were performed for all combinations of these parameter settings.

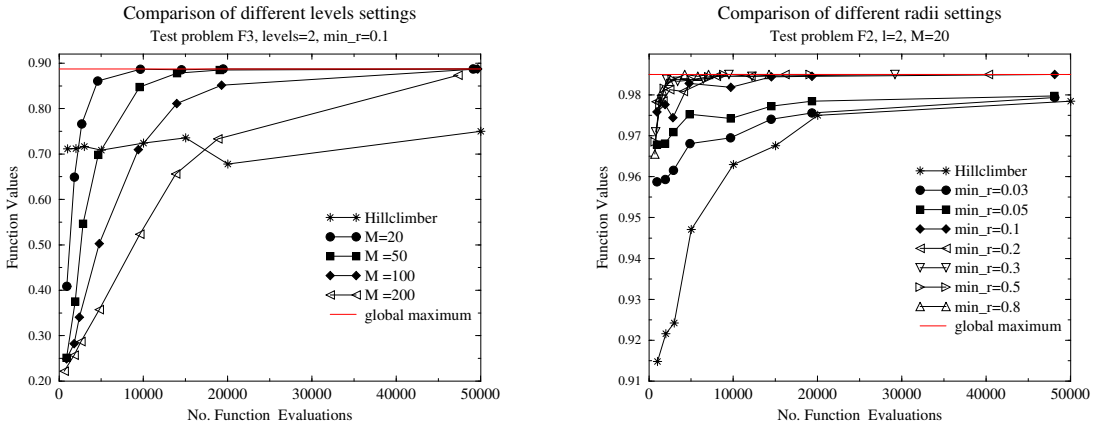

Figure 3: The effect of the different maximal species numbers on F3 and the different radii on F2.

In these experiments, the behavior of UEGO was rather similar for functions F1 and F3, and all the tested values of minimal radii (`min_r`) resulted in almost identical performance, but the results were very sensitive to the values of maximal number of species (`max_spec_num`). The performance decreased with the increasing value of `max_spec_num`. In our experiments the optimal value for the `max_spec_num` parameter was the minimal (20). For F4, changes on the performance of UEGO w.r.t. `min_r` are almost negligible, as happens on F1 and F3.

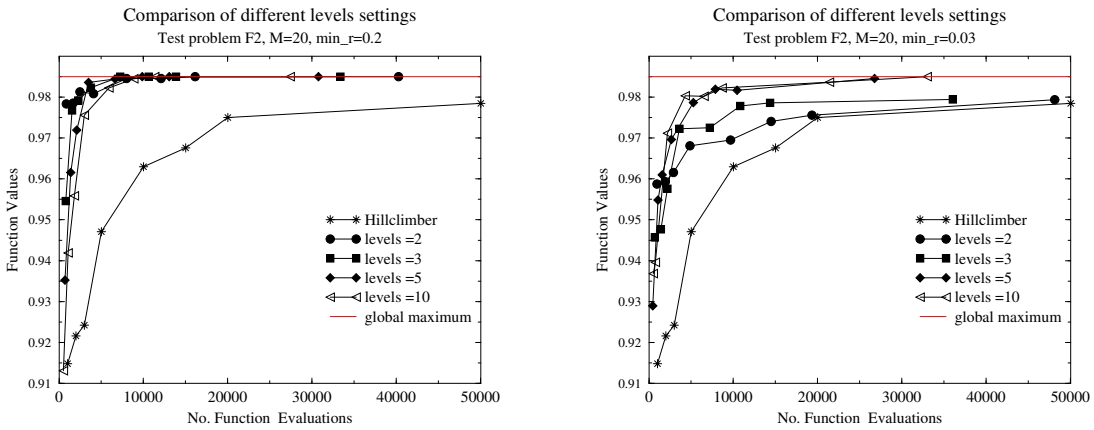

Figure 4: The effect of the different number of levels with the optimal and with a much smaller radius.

On the other hand, for F2 the value of `min_r` did make a difference, mainly when the maximal number of species was relatively small and the performance of UEGO was very robust for the greatest value of `min_r` (0.8), especially for `max_spec_num` equal to 20 and 50. Two characteristic plots illustrating these effects are shown in Figure 3.

The other goal was to find evidence that using more levels than two can be useful. To achieve this goal several values of `level` (2, 3, 5 and 10) were examined with the maximal species number fixed to 20. The experiments were performed for F2 since it was the only function on which the value of `min_r` had significant effect. The result clearly showed that with the optimal minimal radius UEGO was fairly robust w.r.t. the number of levels. However, when the minimal radius was set to a much smaller value than the optimum, higher values of `levels` outperformed the lower settings. This effect is illustrated in Figure 4. The first interesting phenomenon that needs explanation is that in the first set of experiments, results on F1, F3 and F4 were very similar w.r.t. `min_r` while on F2 we got a very different behavior. The first idea that comes to mind is that F1 and F3 have few local optima so it is quite reasonable that setting large maximal number of species would result in a poor performance since most of the evaluations are devoted to search for the non-existent peaks while this problem does not exist in the case of F2. This would also explain that the different values of `min_r` had no effect: most of the search was in fact random search.

For F4 test function UEGO needs more than 50,000 function evaluations for reaching the global optimum. For a small number of species, the algorithm converges relatively fast to a solution, but this solution is a local one. However, when the maximal number of species is great (100, 200), UEGO spends more function evaluations on reaching a solution, but this solution will be a global solution. For these `max_spec_num` values, results in Table 3 will show that the algorithm reaches the real optimum in 40% of executions for `evals`=50000 function evaluations. Remember that F4 is a hard problem (30 dimensions and 125 local optima).

On the other hand, the optimal values of `max_spec_num` for F1 and F3 are 20. So, it seems that for very hard problems is convenient to use a large number of species in order to ensure the convergence to the solution. Therefore we suggest that UEGO should be used for optimizing highly multimodal functions because it ensures easy species creation.

The second phenomenon, namely that it is reasonable to use more levels if the minimal radius is smaller than the optimal minimal radius (for `levels`=2), is much easier to explain. The "cooling" mechanism of UEGO ensures that if the maximal level is high then in some phase of the search the radius is very likely to be near the optimal value while if the maximal level is 2 then the radius immediately becomes the value set by the user. This property is very useful since setting a higher maximal level may ensure a higher degree of robustness.

4 Comparing algorithms

In this section comparisons with a simple hill climber (SHC), a multistart hill climber (MHC) and GAS will be shown. The parameters of SHC, MHC and GAS algorithms were set as follows. The hill climber (SHC) was the optimizer used by UEGO; it means that SHC is UEGO with `levels`= 1. In the multistart case the number of restarts from a new random point was given by the optimal value of the `max_spec_num` for UEGO. The parameters for GAS are very similar to those of UEGO, so they were set to the Best UEGO parameters for every problem using `max_spec_num` as population size.

Results for UEGO were obtained with `levels` = 2 in every case. `max_spec_num` = 20, `min_r` = 0.8 for F1, and F3 test functions, and `max_spec_num` = 100, `min_r` = 0.2 for F2 and F4 test functions were the setting parameters for the results named as Best UEGO. Parameters for Worst UEGO were `max_spec_num` = 200, `min_r` = 0.003 for F1, and F3 test functions, and `max_spec_num` = 20, `min_r` = 0.03 for F2 test function, and `max_spec_num` = 200, `min_r` = 0.03 for F4 test function. Numerical results of these comparison experiments are shown in Figure 5 and Table 3.

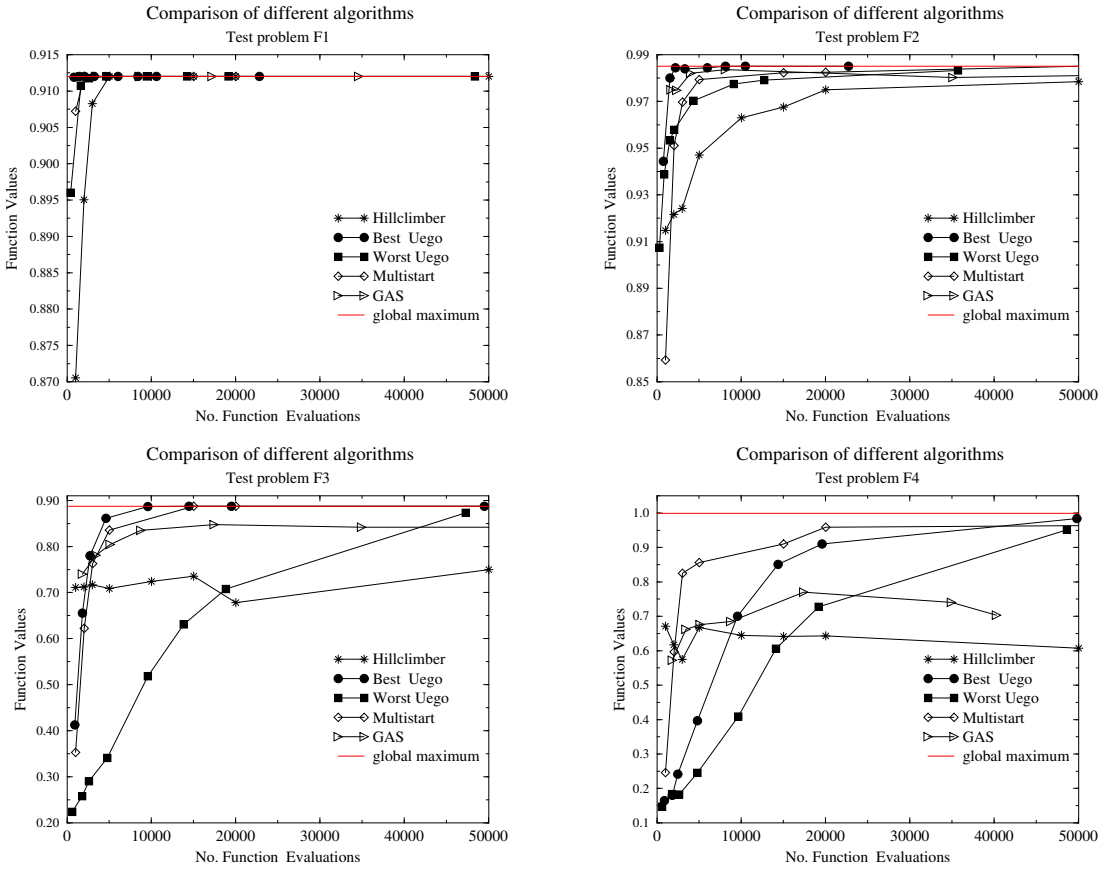

Figure 5: Comparison of the hill climber, the multistart hill climber, GAS and UEGO.

Table 3: Comparison of the UEGO, the multistart hill climber and GAS.

UEGO			MHC			GAS		
Eval.	N.Spec.	% Succ	Eval.	N.Spec.	% Succ	Eval.	N.Spec.	% Succ
F1, `levels`=2, `max_spec_num`=20, `min_r=0.8`								
811	2.62	88.5	1000	3.56	2	1673	2.30	76.4
1482	2.74	99.1	2000	3.42	98	3410	2.50	98.2
2062	2.86	100	3000	3.14	100	5075	2.54	100
3228	2.88	100	5000	2.84	100	8549	2.50	100
6038	2.94	100	10000	2.72	100	17194	2.76	100
8411	2.98	100	15000	2.46	100	26743	2.78	100
10621	3.24	100	20000	2.28	100	34580	2.82	100
22826	3.32	100	50000	1.02	100	86435	2.78	100
F2, `levels`=2, `max_spec_num`=100, `min_r=0.2`								
751	7.12	0	1000	8.22	0	1631	0.94	0
3020	7.44	0	2000	10.04	0	2371	1.42	0
2049	7.40	3	3000	12.16	0	3343	3.44	0
3019	7.82	48	5000	9.98	4	5997	5.12	0
5151	7.94	96	10000	7.86	56	8120	7.44	0
6228	8.52	100	15000	6.02	84	25276	9.32	0
17152	8.96	100	20000	4.38	100	35122	11.38	20
19246	9.74	100	50000	5.24	100	87930	11.46	28
F3, `levels`=2, `max_spec_num`=20, `min_r=0.8`								
918	2.96	0	1000	20*	0	1818	1.06	0
1807	3.28	0	2000	20*	0	3428	1.080	0
2708	3.54	0	3000	20*	0	5108	1.080	0
4597	3.72	42	5000	20*	0	8748	1.16	0
9596	3.98	68	10000	7*	0	17428	1.16	0
14467	4-64	98	15000	4.88	98	24914	1.46	10
19492	4.94	100	20000	4.86	94	37365	1.28	10
49477	4.98	100	50000	4.98	100	87569	1.00	18
F4, `levels`=2, `max_spec_num`=100, `min_r=0.2`								
899	5.8	0	1000	100*	0	1817	1.28	0
1808	6.5	0	2000	100*	0	3419	2.16	0
2455	8.0	0	3000	100*	0	5102	1.78	0
4812	10.4	0	5000	100*	0	8746	1.32	0
9564	11.8	0	10000	100*	0	17407	2.24	0
14350	12.6	10	15000	100*	0	28922	1.48	0
19587	15.0	16	20000	82*	2	34793	1.16	0
49804	20.4	40	50000	65*	4	40256	0.60	0

Table 3 shows average values of the number of function evaluations, number of local or global optima found (species) and the percentage of success in finding the global optimum. Setting parameter values for functions F1, F2, F3 and F4, were chosen as a result of the preliminary experiments. In this table, the symbol * means that most of maxima found were not local nor global maxima.

It can be seen that for function F1, when `evals` $\geq$ 3000, all the three algorithms reached 100% of success in finding the global optimum, however UEGO was able to find more local optima and it was at least 60% computationally less expensive than MHC and GAS algorithms. For function F3, with only five maxima, GAS was able to find the global optimum in a very few runs (18% success), while UEGO and MHC were fully successful when `evals`=50,000. They found all the local and global optima in most of the runs. Similar results were obtained for function F2, but in this case UEGO outperformed MHC in the number of species (local optima). Finally, results for the hardest function F4, clearly show that though UEGO did obtain only 40% of success in finding the global optimum, it was able to find 20 real local optima while MHC and GAS failed in all the cases.

In short, it can be said that UEGO is slightly better than the multistart hill climber for F1, F2 and F3 and for problem F4, UEGO is the only technique that reaches the global optimum. From Figure 5, it is clear that Best UEGO outperforms SHC and GAS for all the functions.

These results indicate that it is reasonable to use the clustering technique to create starting points since the performance does not decrease but, unlike the multistart hill climber, UEGO provides a great number of reasonably good solutions that are at least as far from each other as the minimal radius (see Table 3). This property is very useful in several applications, for example in decision problems: the expert decision maker has a lot of good solutions to choose from. Also remember that the number of restarts of the hill climber was optimal. GAS is also outperformed which justifies our efforts to eliminate the drawbacks of GAS.

4.1 Results for Griewank and Rastrigin functions

In this section some comparisons of above algorithms for Griewank and Rastrigin functions will be shown. First of all we tried to find the best parameters for UEGO, so we decided to test UEGO using several values of `max_spec_num`, `levels` and `min_r` (in this case, values of `min_r` are normalized to the domain of definition of each function). The ranges of the parameters are shown in Table 4. Experiments were performed for all combinations of these parameter settings, and the threshold was automatically set.

Table 4: The values of the UEGO parameters for the test functions.

evals	levels	max_spec_num	min_r
10000, 15000, 20000,	2, 3,	5, 10,	0.1, 0.2,
30000, 50000, 200000	5, 10	50, 200	0.5, 0.8,

Both functions are ten-dimensional and they have several optima, so these test functions are actually very hard multimodal problems. Best performance over these test functions were reached when the number of `levels` was 10, because in this case, the convergence is rather slow, and the probability for finding best solutions increase. With respect to the optimal `max_spec_num`, it tends to be high. Particularly, for comparisons we chose the values 50 and 200 for Griewank and Rastrigin test functions respectively. The optimal `min_rad` was 0.1 and 0.2, respectively.

Table 5 shows some comparisons among UEGO, MHC and GAS. For each algorithm the number of function evaluation (Eval), the most-fit species function value (Val) and the number of species found (N. Spec) are shown. It is interesting to say that the real optimum (whose value is zero) was not found for any of the algorithms, for any of the parameter combinations tested. However, UEGO found the global optimum when more function evaluations were allowed, i.e. for Rastrigin test function, UEGO needs about 25,000,000 function evaluations to find the global optimum. We

have tried also to find the optimum using MHC and GAS algorithms, but they did not reached it in those experiments, even when the number of function evaluations was higher than 100,000,000. For Griewank function, the three algorithms found the global optimum in 1,000,000 function evaluations. The number of global and local optima found by UEGO and MHC were quite similar.

To summarize the results: it has been demonstrated that both the clustering and the level-based "cooling" techniques show some advantages over its predecessor. On the other hand, UEGO can be parallelized almost as effectively as the multistart hill climber. Without going into the details we mention that a parallel version of UEGO using a simple asynchronous Master Slave model has been implemented and executed on a Cray T3E using up to 33 processors [14, 15] and for a set of eleven test functions an almost linear speed up was obtained. Using 33 processors the values of the speed up range between 27 and 32.

Table 5: Comparisons for Griewank and Rastrigin Test Functions

UEGO			MHC			GAS		
Eval.	Val.	N.Spec.	Eval.	Val.	N.Spec.	Eval.	Val.	N.Spec.
Griewank Test Function								
2153	-0.1891	42.34	10000	-0.4805	50	15003	-0.4738	6.70
3434	-0.0874	42.96	15000	-0.3260	50	23876	-0.3658	10.54
4400	-0.0591	46.86	20000	-0.1797	50	33129	-0.2545	11.86
6267	-0.0315	48.84	30000	-0.0630	50	54823	-0.1258	11.78
17717	-0.0236	49.10	50000	-0.0200	50	85640	-0.0914	11.70
33057	-0.0103	49.18	200000	-0.0180	50	231462	-0.0243	11.80
Rastrigin Test Function								
5043	-49.49	200	10000	-55.61	200	15311	-27.59	20.92
5686	-45.01	200	15000	-42.46	200	24214	-22.25	49.76
6757	-45.63	200	20000	-35.71	200	33289	-18.98	49.78
8467	-47.29	200	30000	-34.69	200	55122	-16.76	49.70
11918	-41.77	200	50000	-25.87	200	85746	-14.40	49.42
36827	-19.23	200	200000	-20.92	200	235875	-14.23	49.68

4.2 A Note on Parameter Setting

As shown clearly by the previous sections, there is no unique best way to set the parameters of UEGO. The optimal setting depends on the problem structure. This fact is not surprising since the heuristics used in the algorithm make explicit assumptions about the structure of the domain. It is not special to UEGO either since general theoretical results about the relationship between search domains and optimizers [18] predict this effect.

If information is available on the number and distribution of local optima of the problem to be solved then it is possible to set the parameters based on the design principles of UEGO and on the empirical results discussed in Section 3.3. If there are many local optima then the maximal number of species should be high and if they are close to each other then the minimal radius should be small. Of course, if the structure of the problem is unknown then preliminary experimentation is necessary. One possible strategy suggested by our experiments is to use a small minimal radius, more levels, e.g. 10, and to use a high maximal number of species, e.g. 200. The number of species created during the optimization gives a hint about the number of local optima, though this is only a heuristic since e.g. having many species is not sufficient nor necessary for having many local optima especially if the number of function evaluations is small.

Table 6: The values of the UEGO parameters for the subset sum problem.

evals	levels	max_spec_num	threshold	min_r
3000, 10000, 30000, 100000, 300000	2, 3, 5, 10	5, 10, 20, 40, 100, 200	automatically set	fixed to 1

5 Experiments with the Subset Sum Problem

In this section we will discuss the performance of UEGO on an NP-complete combinatorial optimization problem: the subset sum problem. A comparison with GENESIS [6] and a hill climber will be presented. As another result of the experiment the behavior of the parameters of UEGO will be illustrated.

5.1 Problem and Coding

In the case of the subset sum problem we are given a set $W = \{w_1, w_2, \ldots, w_n\}$ of n integers and a large integer M. We would like to find a $V \subseteq W$ such that the sum of the elements in V is closest to, without exceeding, M. This problem is NP-complete. Let us denote the sum of the elements in W by SW.

We created our problem instances in a similar way to the method used in [12]. The size of W was set to 50 and the elements of W were chosen randomly with a uniform distribution from the interval $[0, 10^{12}]$ instead of $[0, 10^3]$ (as was done in [12]) to obtain larger variance. According to the preliminary experiments, the larger variance of W results in harder problem instances which is important since comparing methods on almost trivial problems makes little sense. The problem instance used here turned out to be so hard that none of the methods employed could find an optimal solution. Based on the results of [9], M was set to $SW/2$. As was shown in [9], this is the most GA-friendly setting, so there is no bias against GENESIS introduced by the problem instance.

We used the same coding and objective function as suggested in [12]. For a solution $(x = (x_1, x_2, \ldots, x_{50}))$,

$$f(x) = -(a(M - P(x)) + (1 - a)P(x))$$

where $P(x) = \sum_{i=1}^{50} x_i w_i$, and $a = 1$ when x is feasible (i.e. $M - P(x) \geq 0$) and $a = 0$ otherwise. Note that the problem is defined as a *maximization* problem.

5.2 The Optimizer and GA Settings

In UEGO, the optimizer was chosen to be a simple SHC as was discussed in the Introduction. In our implementation the SHC works as follows: mutate every bit of the solution with a given probability (but mutating one bit at least), evaluate the new solution and if it is better than or equal to the actual solution, it becomes the new actual solution. This type of SHC worked best in [13], as well. The mutation probability was set at $4/n$ where n is the chromosome length. This value was the same in all the experiments carried out including those with GENESIS. The other GA parameters were a population size of 50, 1-point crossover with probability 1, and elitist selection.

5.3 The Experiments

One of the two main goals of these experiments was to analyze the effects of the user-given UEGO parameters described in Section 2.3. To perform this analysis, several values were chosen for each parameter (see Table 6) then UEGO was run 50 times for every possible combination of these values. This meant that $5 \cdot 4 \cdot 6 \cdot 50 = 6000$ experiments were performed for one problem instance. Three problem instances were examined but since the results were similar in each case, only one problem instance is discussed below. Figure 6 shows the effects of the different parameter settings.

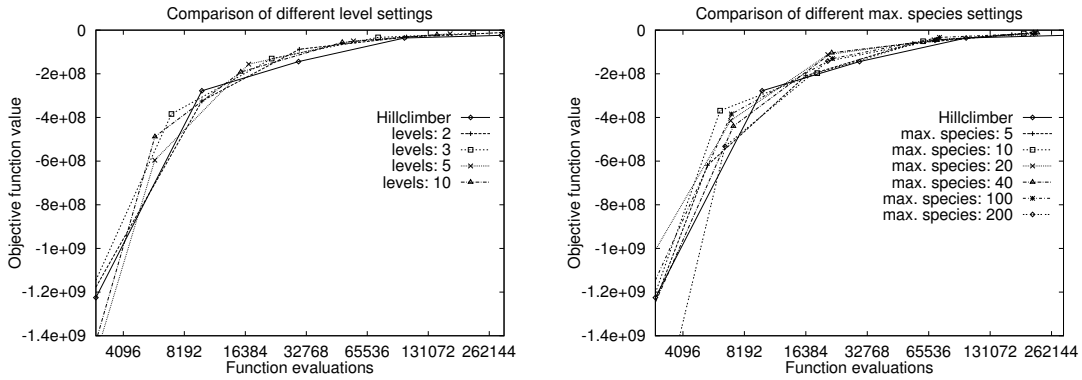

Figure 6: With the various level settings, `max_spec_num` is 100 and for the different max. species settings `levels` is 3.

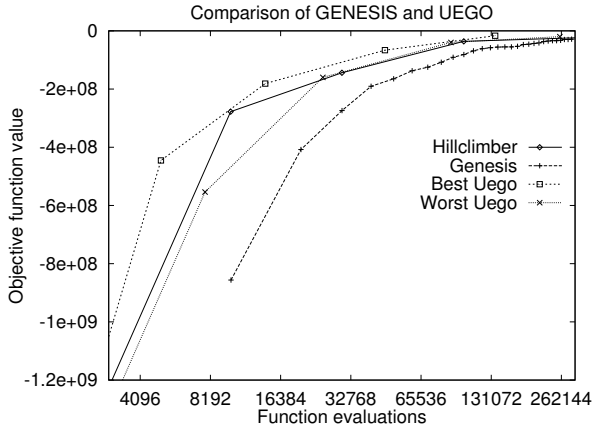

Figure 7: The parameters for the best UEGO were `max_spec_num`=20 & `levels`=10, and for the worst `max_spec_num`=5 & `levels`=2.

As the plots are typical it was inferred that the parameters of UEGO must be fairly robust for this particular problem class. It has to be noted that this does not automatically imply that similar robustness would be observed on other domains as well (see Section 4.2).

The other goal of the experiments was to make a comparison. Figure 7 shows the relevant results. Note that it was difficult to select the best and the worst performance because the curves cross, but the plots give a good approximation. Here SHC is simply UEGO with the setting of `levels`=1.

6 Summary

In this paper, UEGO, a general technique for accelerating and/or parallelizing existing search methods was discussed. As was shown, most of the parameters of the system are hidden from the user due to an algorithm for calculating those parameters from a couple of simple parameters. This algorithm is based on *principles* stated in Section 2.3 and the *speed* of the applied optimizer.

Experimental results were given for real and combinatorial problems. It was shown that the user-given parameters are robust in the case of the subset sum problem and the advantages of the UEGO clustering technique and the level-based "cooling" technique were demonstrated on the real domain.

Other experimental results were also given, such as the comparison of the technique with several methods. In the case of the subset sum problem it was shown that UEGO is slightly better than a GA and an SHC. On the real domain UEGO outperformed the multistart hill climber in the sense that either the quality of the global optimum was better or the number of local optima found by UEGO was larger. GAS was outperformed w.r.t. both aspects. The later indicates that the modifications in order to eliminate the drawbacks of GAS has been successful at least in the case of the problems we considered.

References

[1] T. Bäck, editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*, San Francisco, California, 1997. Morgan Kaufmann.

[2] D. Beasley, D. R. Bull, and R. R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[3] K. Deb. Genetic algorithms in multimodal function optimization. TCGA report no. 89002, The University of Alabama, Dept. of Engineering mechanics, 1989.

[4] K. Deb and D. E. Goldberg. An investegation of niche and species formation in genetic function optimization. In J. D. Schaffer, editor, *The Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[5] A. E. Eiben, J. K. van der Hauw, and J. I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[6] J. J. Grefenstette. Genesis: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165, 1984.

[7] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995.

[8] H. Ishibuchi, T. Murata, and S. Tomioka. Effectiveness of genetic local search algorithms. In Bäck [1], pages 505–512.

[9] M. Jelasity. A wave analysis of the subset sum problem. In Bäck [1], pages 89–96.

[10] M. Jelasity and J. Dombi. GAS, a concept on modeling species in genetic algorithms. *Artificial Intelligence*, 99(1):1–19, 1998.

[11] A. Juels and M. Wattenberg. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 430–436. The MIT Press, 1996.

[12] S. Khuri, T. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In *The Proceedings of CSC'94*, 1993.

[13] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill-climbing? In J. D. Cowan et al., editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994.

[14] P. M. Ortigosa. *Métodos estocásticos de Optimización Global. Procesamiento paralelo.* PhD thesis, Department of Computer Architecture and Electronics, University of Almería, Almería, Spain, 1999. Available as http://dali.ualm.es/papers/99/tesispilar.ps.gz.

[15] P. M. Ortigosa, I. García, and M. Jelasity. Two approaches for parallelizing the UEGO algorithm. To appear in Optimization Theory: Recent Developments from Mátraháza, 2001.

[16] R. R. Sokal and F. J. Rohlf. *Biometry*. W. H. Freeman and company, New York, 1981.

[17] F. J. Solis and R. J.-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[18] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

[19] M. Yagiura and T. Ibaraki. Genetic and local search algorithms as robust and simple optimization tools. In I. H. Osman and J. P. Kelly, editors, *Meta-Heuristics: Theory and Application*, pages 63–82. Kluwer Academic Publishers, 1996.