Toward the Optimization of a Class of Black Box Optimization Algorithms

Gang Wang Erik D. Goodman William F. Punch
Genetic Algorithms Research and Applications Group (GARAGe)
A.H. Case Center and Department of Computer Science
Michigan State University
East Lansing, MI 48824
wanggan1@cps.msu.edu goodman@egr.msu.edu punch@cps.msu.edu

Abstract

Many black box optimization algorithms have sufficient flexibility to allow them to adapt to the varying circumstances they encounter. These capabilities are of two primary sorts: 1) user-determined choices among alternative parameters, operations, and logic structures, and 2) the algorithm-determined alternative paths chosen during the process of seeking a solution to a particular problem. This paper discusses the process of algorithm design and operation, with the intent of integrating the seemingly distinct aspects described above within a unified framework. We relate this algorithmic optimization process to the field of dynamic process control. An approach is proposed toward the optimization of a process for controlling a specific class of systems, and its application to dynamic adjustment of the algorithm used in the search problem. An instance of this approach in genetic algorithms is demonstrated. The experimental results show the adaptability and robustness of the proposed approach.

1 Introduction

Algorithms called Black Box Optimization algorithms ("BBOs") are used to address problems for which domain knowledge is incomplete or unavailable. Although these BBOs are not guaranteed to find the global optimum, especially for NP-complete problems, they are often a useful tool for discovering approximate solutions. Adaptive sampling search methods, particularly, evolutionary computation, have recently become very popular for such uses. They are loosely modeled on the process of evolution in the real world, but they are only paradigms, which means that when given a specific problem, one must fill in a great many details to design and implement the algorithm. Quantitative analysis to discover these details is difficult simply because of the complexity of the problem space. In the typical process of problem solving, adjustment of the algorithms is often done empirically, with limited guidance from theory, rules of thumb, and prior experience.

An algorithmic approach is proposed toward the optimization of BBOs. When applied to Genetic Algorithms (GAs), they provide a more uniform way to formalize the process of designing, implementing, and running a GA-optimization system. The experimental results show the adaptability and robustness of this approach.

2 Issues in algorithm design for black box optimization

In designing algorithms for solving BBO problems, two aspects of algorithm flexibility are involved:

- **External flexibility** of the user-defined structure of the algorithm and parameters, determining the space of possible algorithms to be used. A specific search paradigm includes choices one can make to address a specific problem. For example, with genetic algorithms, one needs to choose (chromosomal) representation, fitness function, genetic operators, selection method, and many control parameters (*e.g.*, probabilities of crossover and mutation). Most BBO algorithms exhibit this sort of flexibility, giving one both the freedom and the responsibility to "tune" the algorithm to the problem at hand. This is called *external flexibility.*

- **Internal adaptability** of the logical structure of any particular algorithm to learn about the search space of the optimization problem to be solved. For example, in genetic algorithms, the internal adaptability is reflected in the change in the population of individual solutions with time for better explorating the search space and exploiting the promising area.

The "No Free Lunch" theorem [1] in optimization illustrates that the design of a BBO is itself an optimization problem. If this BBO process is complex enough, which is true for most cases, optimization of external flexibility parameters will be another BBO problem. When the external flexibility parameters are fixed directly by the user based on experience or rules of thumb, it is hard to guarantee that the initial algorithm design is good enough to achieve the desired results.

In [4], external flexibility and internal adaptability were considered explicitly in order to find an optimal algorithm design for solving optimization problems.

Adaptation of control parameters during the problem solving process by evolutionary algorithms has been done previously by several authors ([2][3][14][15][16]). This work can be divided into two primary groups: adaptation on the level of individuals and adaptation of population-wide parameters. In a system implemented by Baeck et al. [14], fine-tuning of control parameters is done with each individual – that is, every individual is associated with a set of control parameters, and the performance of these parameters with the corresponding individuals affects the survival of the parameters themselves.

In Breeding Genetic Algorithms [15], adaptation is done at the population level. In that system, multiple subpopulations with different strategies (control parameters) compete with each other, and populations with better performance increase in size, while populations with poor performance decrease in size. However, there is no recombination or mutation of control parameters associated with the larger or smaller subpopulations produced.

Herdy [16] did very similar work on evolution strategies; in his system, subpopulations with different strategies compete, and higher level strategies combine and mutate with the goal of generating better ones. But there is no exchange of individuals between subpopulations, so they do not take advantage of the individual-level knowledge gained from solving the problem, which is enabled by individual migration.

As we will see in our work, higher level recombination and mutation are used to generate control parameters which are likely better than their predecessors, while lower level migration of individuals makes the performance evaluation on the search processes fair by *sharing* the best knowledge gained so far in solving the problem. This facilitates the emergence of time varying sets of operators, parameters etc, appropriate to the changing phases of problem solution. The work reported here is distinct from other work in these important aspects.

In the following sections, we give a formal description of an optimization process for genetic algorithms and related paradigms; then we return to the proposed approach and discuss the relationship between control of the optimization process and optimal problem solving by an algorithm. With this approach, some of the external flexibilities can be handled in an algorithmic way similar to the internal adaptability in GAs. Therefore, it offers an integrated framework in which to deal with both aspects of flexibility in genetic algorithms.

3 Optimization of an algorithmic problem solving process

In this section, we will cast a GA-based multilevel optimization process into the form of time-invariant state models, and explore the optimization of the performance of those models.

3.1 Algorithmic problem solving process

Following the state model notation in, for example, [5], we shall describe the behavior of a genetic algorithm for a single population as a time-invariant, discrete-time system state model, as follows:

$$X(T+1) = f(X(T), U(T)), T \in 0, 1, \cdots, \infty, X(0) = X_0 \tag{1}$$

- $X = \{x_1, \cdots, x_n\} \in I_n$, the state vector, characterizes the state of the system (which captures its internal adaptability), and for a GA, is the set of all individuals x_i in the population at time (generation) T.

- $U = \{u_1, \cdots, u_r\}$, the input vector which denotes the parameters of the GA (which specify its external flexibility) as a function of time. $U(T)$ represents a complete specification of all previously unspecified parameters – for example, for the single-population simple GA paradigm, specification of crossover operator type and rate, mutation operator type and rate, selection method and fitness scaling (if any), offspring replacement strategy, etc.

Other related subjects concerning the performance of a system in the situation of optimization problem include:

- Function $q(x_i)$, the fitness (or objective function to be optimized) of individual x_i in population $X(T)$.
- Function $Q(X(T)) = \max_{i=1}^{n} q(x_i)$, the maximum fitness of any individual x_i in $X(T)$.
- $X^* \in \{X | X = \{x_1, x_2, ..., x^*, ..., x_n\}\}$, any state which includes an optimal individual x^*.

Note that if $U(T)$ is fixed at $U(0)$, $T \geq 0$, then the system described by f is a finite Markov chain, as described, for example, in [6][7], etc. However, for the systems to be described here, with time-varying operators and parameters, the notation of state models is found more convenient.

Rather than discussing attainment of a global optimum solution x^*, we shall use function $G(U, X_i, X_f, g, \epsilon)$ to denote the probability that system f reaches a state X such that $Q(X)$ is within ϵ of $Q(X_f)$ from initial state X_i, under the control of input $U(T)$, within g generations.

For given constants g, ϵ and optimal solution x^*, we also define a more compact notation, function F, as follows:

$$F(X,U) = \sum_{X_f \ni Q(X_f) = q(x^*)} G_f(U, X, X_f, g, \epsilon) \qquad (2)$$

That is, $F(X,U)$ means the probability that system f moves from state X to any state with performance Q within ϵ of the optimum $q(x^*)$ within g generations under the control of U. F captures the idea of how promising the optimization process f is for generating near-optimal solutions beginning from an arbitrary state X.

Unfortunately, it is practically infeasible to determine the forms of functions G and F for many complex BBO approaches. It is therefore useful to define functions E which capture certain properties of function F, for use in evaluation and comparision of multiple optimization processes. Ideally, we seek an $E(X,U)$ which has the following property:
$(\forall X, U_1, U_2)(F(X,U_1) \geq F(X,U_2)) \Leftrightarrow (E(X,U_1) \geq E(X,U_2))$

In section 4.2, we will see how function E is approximated and tested in the system discussed below.

3.2 Measuring the quality of optimization

The goal of the problem solving process can be simply defined as minimizing the following expression for M_0, the measure of the final solution, with Q,X and X^* defined as above.

- $|Q(X(\infty)) - Q(X^*)|$

However, with bounded resources – for example, a maximum number of iterations, T – different optimization processes might give very different results. Therefore, the quality, M_1, of such a bounded optimization process, can be stated in various more practical ways:

Given a system f with fixed control inputs $U(T)$, starting from $X(0)$,

- **The average time required for system f to produce optimal solution x^*;**
- **The average time required to produce a near-optimal solution with fitness within ϵ of $q(x^*)$;**
- **The average distance of the limit of the trajectory $Q(X(T))$ from $Q(X^*)$;**
- **The average time to produce a solution which recurs in a large number of runs and is potentially a global optimum solution;**
- **The average time to produce a solution superior to that produced by some other optimization method.**

However, as a problem solver, not only do we need to consider the problem itself, but also the optimal way for solving it. Therefore, given a specific programming paradigm, say, genetic algorithms or genetic programming, the *goal* for optimization of this algorithmic problem solving process at the higher level, is not to optimize $X(T)$ for a fixed $U(T)$, but rather to optimize $U(T)$ so as to minimize the measure M_1. Note that M_1 can have different forms, ideally or practically, as stated above. If M_1 is in the first form listed, we have a typical minimal time problem:

Given an initial population vector $X(0)$ and a target best individual x^*, find an optimal input vector $U(T)$ which changes the state $X(T)$ of the system f from $X(0)$ to state X^*, in minimal time.

A typical procedure in the problem solving process is to manually try a particular function $U(T)$, test its performance by measure M_1, adjust $U(T)$, and repeat until satisfied. Therefore, we have another measure (M_2) to evaluate the cost of this process:

- **Given a system f, the number of tries of input vectors $U(T)$ before measure M_1 is minimized.**

We discuss below an approach to help to minimize measure M_2, thereby increasing the robustness of the optimization process.

3.3 Proposed approach: high level optimizer (HLO)

Given a dynamic system f, with a defined quality measure Q of a state X, input vector $U(T)$, and measure E which characterizes the probability that any state will be transformed to one with $Q(X)$ within ϵ of $q(x^*)$ within time g, define its High Level Optimizer (HLO) as:

1) Initialize m instances of system f with randomly generated input vectors U, $X_i(T+1) = f_i(X_i(T), U_i(T))$, $1 \leq i \leq m$.

2) For each system f_i, run f_i using its control input U_i for a fixed period, then check its performance using measure E.

3) Select among the populations according to their performance measurements $E(X_i, U_i)$, then do recombination and mutation on pairs of U_i.

4) Select some elements of the associated pairs of $X_i(T)$ for exchange;

5) Continue steps 2) and 3) until a stopping criterion is met.

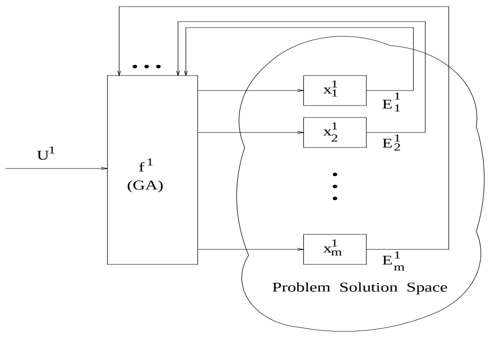


Figure 1: GA configuration

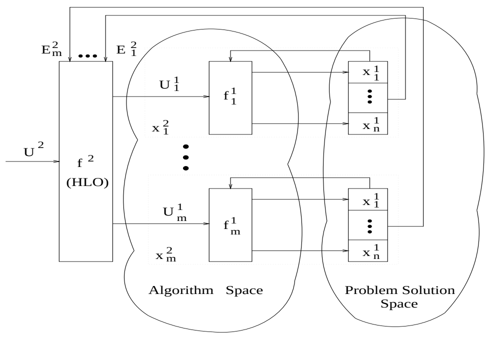


Figure 2: HLO configuration

Our goal in this approach is to improve, in an algorithmic fashion, one or more of the performance measures in $M1$. Of course, the key to actually accomplishing this is the definition of the function E in such a way as to improve F for the particular problem and performance measure at hand. While it is clearly *not* possible to accomplish this in general, it also appears clear that we can find examples of E which are more useful than others for broad classes of significant problems. As we shall see in the following section, another interesting property of this approach is its robustness; that is, it not only tries to minimize measure M_1, but also achieves low values of M_2.

4 DAGA2: Testing the HLO approach on genetic algorithms

DAGA2 (Distributed Adaptive Level-two Genetic Algorithm) [8] is an instance of the HLO approach. In DAGA2, multiple GAs with different parameter settings are supervised and evaluated dynamically. Using higher-level selection and genetic operators, each search process (GA) has a chance to learn better controls from others, thus making it possible for each individual optimization process to improve its M_1.

In DAGA2, control of the optimization process involves the following factors:

- selection operator type and corresponding parameters.
- crossover operator type and crossover probability.
- mutation operator type and mutation probability.

Briefly, DAGA2 first initializes multiple GAs with randomly generated controls, then operates its parallel subpopulations for a fixed number of generations, evaluating the performance of each subpopulation using a "level-two" fitness function E. Each level-two control structure (chromosome) is then, with some probability, subjected to crossover and mutation. Crossover involves fitness-weighted selection of a neighboring level-two chromosome. Migration of level-one chromosomes among neighboring subpopulations is performed at the same interval as the higher-level operations, assuring the benefits of a parallel GA and the distribution of the fitness gains made by any level-one subpopulation among its neighbors. Then the subpopulations, with possibly different individuals and typically better (more fit according to the level-two fitness function) operators/rates, continues to run, and the process is repeated.

The function E for a GA population process might be defined in several ways. Some elements which might be included are:

- best fitness of individuals in the GA population,
- average fitness of individuals in the GA population,
- number of fitness function evaluations,
- diversity measure of the GA population.

Of course, considering some or all of these measures does not allow one to estimate exactly the probability that the algorithm will find the global optimum solution within a particular timeframe. However, for many practical problems, combination of certain of these measures seems to be useful for guiding the selection operation of an HLO-based GA scheme, as seen below.

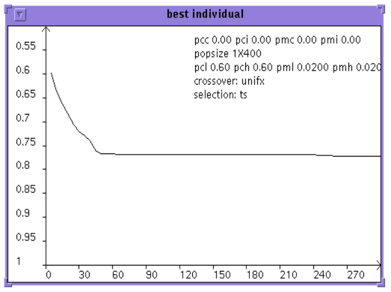
Excellent results from running DAGA2 on a variety of common benchmark functions have been reported elsewhere ([9]). Here we discuss its performance on some simple functions selected to illustrate two aspects of DAGA2 performance: adaptability and robustness.

4.1 DAGA2 adaptability

Adaptability is a very general term, which usually means the capability for a system to adjust its internal structure in order to perform a given task which is newly presented or altered since its previous presentation. If the input vector $(U(T))$ is fixed, then the adaptability of the GA refers to its capability to evolve individual chromosomes with better fitness values. In the case of DAGA2, adaptability is found on two levels – that within the individual GAs on the lower level, and the capability to evolve higher level algorithms (each GAs' external flexibility) for better performance.

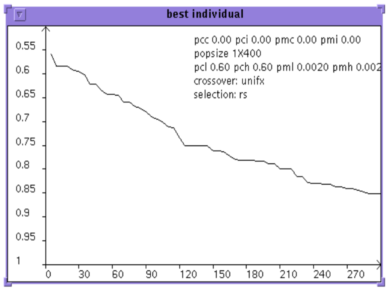
To demonstrate simply the capability of DAGA2 to evolve a suitable GA design or a sequence of designs during the course of solving an optimization problem, we run it first on a "Counting Ones" problem, which assigns fitness as the proportion of 1's on the chromosome. It is a very easy problem for a GA with appropriate parameter settings; However, some crucial parameters, probability of mutation(Pm) and probability of crossover(Pc), should be set appropriately based on the chromosome length.

These experiments used a chromosome length of 900, population size at 400. Before running DAGA2 on the problem, we simulated a Simple GA (SGA) within DAGA2, for comparison, by fixing both probability parameters like Pm and Pc and selecting specific operators from a set to use in one single population.



pcc 0.00 pci 0.00 pmc 0.00 pmi 0.00
popsize 1X400
pcl 0.60 pch 0.60 pml 0.0200 pmh 0.020
crossover: unifx
selection: ts

Figure 3: best/gen in experiment 1

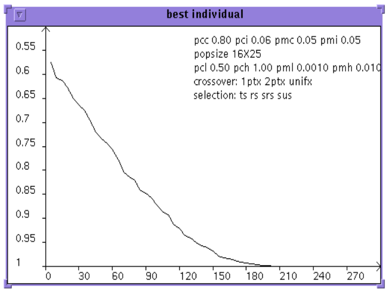


pcc 0.00 pci 0.00 pmc 0.00 pmi 0.00
popsize 1X400
pcl 0.60 pch 0.60 pml 0.0020 pmh 0.002
crossover: unifx
selection: rs

Figure 4: best/gen in experiment 2.

Experiment 1 used tournament selection, uniform crossover and bit-wise mutation, but the mutation rate, Pm, is set too high, at 0.02/bit, for a 900-bit chromosome. It is not surprising that SGA progress ceased at generation 60.

Experiment 2 used roulette wheel selection, uniform crossover and bit-wise mutation. This time, Pm was set appropriately, but with roulette wheel selection, selection pressure was too light; thus, although the SGA continued to make progress, it still had not found the optimal value within 300 generations.



pcc 0.80 pci 0.06 pmc 0.05 pmi 0.05
popsize 16X25
pcl 0.50 pch 1.00 pml 0.0010 pmh 0.010
crossover: 1ptx 2ptx unifx
selection: ts rs srs sus

Figure 5: best/gen in experiment 3

Experiment 3 illustrated the use of DAGA2. For this case, available selection methods were tournament selection, roulette wheel selection, stochastic remainder sampling and stochastic universal sampling; available crossover methods were single-point, two-point and uniform crossover; mutation was bitwise. The ranges for Pc and Pm were [0.5, 1.0] and [0.001, 0.01] (per bit), respectively. In this case, the level-two fitness function was defined very simply: the increase in average fitness value in the subpopulation over the previous level-two evaluation plus the average (raw) fitness value in the current subpopulation. The result is that DAGA2 typically found the optimal individual at about generation 220. Other data from that experiment are given below.

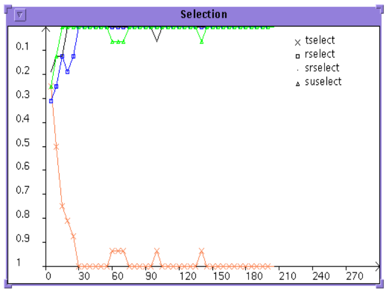


Figure 6: Selection/gen in experiment 3

In all of the figures following, selection/gen means the fraction of subpopulations (i.e., level-2 individuals) using each method of selection, versus generation; crossover/gen means fraction of subpopulations using each crossover method, versus generation. They show that DAGA2 can find "suitable" parameter settings and operators by itself.

In Figure 6, tournament selection soon dominated other selection methods; while in Figure 7, several crossover operators competed; eventually, one-point crossover and uniform crossover were favored.

In Figure 8, the average value of Pc among all sub-

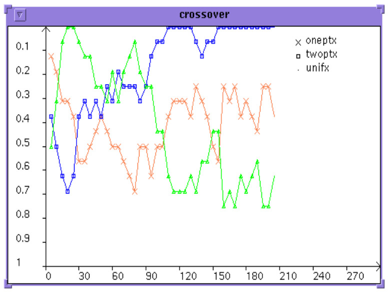


Figure 7: Crossover/gen in experiment 3

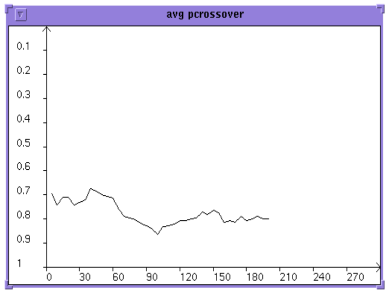


Figure 8: Pc/gen in experiment 3

GAs clustered around 0.8; Figure 9 shows the obvious trace of Pm evolution, by eventually allowing only one or two bit mutation per chromosome. The average of Pm among all subGAs decreased from above 0.005 to around 0.0015 (note that from the schema theorem [10], it is reasonable to assign Pm around 0.0015 if the chromosome length is 900). It should be emphasized that this evolution of upper-level control parameters occurs without user intervention, except to set the search ranges initially. In our experience, DAGA2 was very insensitive to those ranges, even working reasonably well, for example, when they are set to their extremes ([0, 1]).

While Figure 5 - Figure 9 represent a single run, the results of repeated runs with different random seeds closely resembled them, nearly duplicating the operator usage percentage and probability histories.

We have seen in experiment 3 that DAGA2 has the ability to evolve suitable control parameters. Furthermore, the values found by DAGA2 were tested as static settings for an SGA in experiment 4, using tournament selection, uniform crossover, Pm at 0.0015, and Pc at 0.8, which were found to be effective in experiment 3. The result is much better than those in experiments 1 and 2, and even better than that of experiment 3, which is also not surprising. As we see in the definition of DAGA2, control parameters in the subGAs are initialized randomly in a given

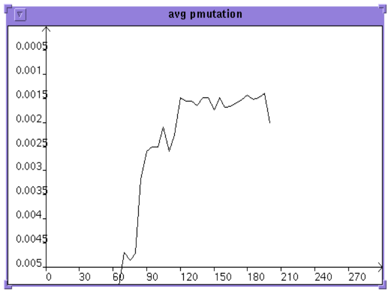
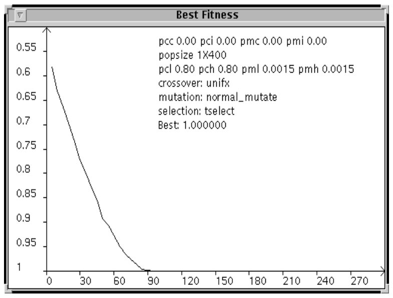


Figure 9: Pm/gen in experiment 3



```
pcc 0.00 pci 0.00 pmc 0.00 pmi 0.00
popsize 1X400
pcl 0.80 pch 0.80 pml 0.0015 pmh 0.0015
crossover: unifx
mutation: normal_mutate
selection: tselect
Best: 1.000000
```

Figure 10: Best/gen in experiment 4

range, so the system always needs some time (generations) to "adjust" to optimal values; thus, it is natural that DAGA2 doesn't behave better than an SGA with suitable parameter settings on this task, which is sufficiently simple task that static values for these settings are appropriate throughout the entire run. As we shall see in other experiments on more difficult problems, for which it is harder to find good parameter settings, or on which static parameter settings and genetic operator choices represent a limitation, DAGA2 performs rather well.

4.2 Robustness of DAGA2 performance

Given a system and a task, one could consider the robustness of the system to be its capacity to achieve a desired value of M_1 given randomly selected values for system inputs ($U(0)$, fixed for all $T \geq 0$), from within a given range of possible values.

Holland described a "Royal Road with potholes" problem at ICGA-93 ([11]), in which he challenged GA methods to complete the assembly of building blocks to level 3 (the penultimate level) of a level-4 problem – i.e., to generate a chromosome with either the first eight or last eight blocks set appropriately. He used a "Cohort GA" ([12]) to attain level 3 with high probability in 10,000 function evaluations. However, we use here the more difficult, level-4 Royal Road simply because we wished to study the robustness of DAGA2 on a problem which was difficult, but in which we could easily visualize the nature of the difficulty.

- **Control of level two DAGA2 process**

  For this experiment, DAGA2 was configured so that each subpopulation had 8 (adjacent and diagonal) neighbors. Among level-one subpopulations, DAGA2 used Pm in the range [0.0, 0.002], tournament selection (tourneysize 2), one-point, two-point and uniform crossover, bitwise mutation or multi-bit mutation ([13]), and offspring replacing parents. The range of level-1 crossover values, Pc, is shown in Table 1, as are tr, the number of generations (level 1) between operations at level 2, and Pci, the fraction of a level-1 subpopulation migrated to its neighbors at each interval tr. Probability of mutation of level-2 (control) parameters, Pmc, was 0.3, and crossover probability among parameter strings, Pcc, was 0.8. All runs were terminated when Royal Road level 4 was attained or 60,000 function evaluations were performed.

- **Number of level-two subpopulations**

  Most runs were done with 36 subpopulations of 70, because earlier experiments with 5x5 and 4x4 arrays with the same total population size (approximately 2,500 individuals) were much less robust in solving the Royal Road level 4 problem. This is not surprising, both because the smaller number of subpopulations cannot preserve diversity as well even in an ordinary parallel GA, and because doing more evaluations in fewer subpopulations during each interval tr inhibits the rate at which DAGA2 can learn about good parameter settings. That is, with 36 subpopulations and tr=5, DAGA2 performed about 400 level-2 function evaluations during a typical run; with fewer subpopulations, it had much less opportunity to learn about good parameter settings.

- **Level two fitness functions**

  The one genuinely difficult decision to make for DAGA2 and any other multi-level evolutionary scheme is the choice of the level-2 fitness function, which expresses what the level two processes are striving to optimize. The goal is a function E which varies monotonically with F, as described above, so it can be used to judge which level-1 GA processes have greater "promise" of attaining a near-optimal solution. It is also among the most important issues to study using a tool like DAGA2. Therefore, in the following experiment, we used DAGA2 with a variety of level-2 fitness functions (these investigations are ongoing).

1) Number of offspring generated with higher

Expt.	*Level2 Params.*	*Succ. Times*	*Avg. Func. Evaluations*
5	6x6x70 Pc=[0.2,0.7] tr=5 Pci=0.07	20/20	47297
6	as in exp. 5 exc. Pc=[0.1,0.6]	14/20	36473
7	as in exp. 5 exc. Pc=[0.0,1.0]	14/20	34414
8	as in exp. 5 exc. Pc=[0.1,0.8]	18/20	43850
9	as in exp. 5 exc. tr=8	15/20	52476
10	as in exp. 5 exc. Pci=0.05	19/20	46871

Table 1: Results of experiments 5-10, showing attainment of RR level 4 (successes) and average number of function evaluations required, under various DAGA2 level-2 control settings. 6x6x70 means a 6x6 square of subpopulations, each 70 individuals.

fitness than both parents in last tr generations

2) (Fitness of best individual) / (number of function evaluations in last tr generations)

3) (Number of offspring generated with higher fitness than both parents in last tr generations) / (number of function evaluations in last tr generations)

Experiment 5 and all others in Table 1 show results with fitness function (3). Experiment 5 was also run with fitness function (1), succeeding 4 of 20 times, with an average of 57,674 function evaluations on successful runs. Fitness function (2) succeeded in 19 of 20 runs, with an average of 36,459 evaluations on successful runs. Measures which reward progress per function evaluation seem likely to produce superior results for many problems when the number of allowable function evaluations is capped, as it was at 60,000 for these runs. We do not yet know whether the greater "exploitation" produced by this per-evaluation pressure increases or decreases the probability of success for this and similar problems given much more generous caps on evaluation numbers.

Experiments on this fitness function constitutes an attempt at optimization at level 3, which is not automated in DAGA2 (that would be in DAGA3). We believe it is important to try to characterize how to define an appropriate function for solving any particular class of problems in the HLO framework.

• **Sensitivity to level 2 parameters**

Experiment 5 showed that DAGA2 could attain Royal Road level four in 20 of 20 runs, in an average of 47,297 function evaluations. DAGA2 is clearly not solving the level 3 problem as efficiently as Holland's Cohort GA, but that is not the goal of this effort; we also do not know how efficiently Holland's Cohort GA would solve the level 4 problem. The other experiments explored the sensitivity of DAGA2's performance to various of its level-2 parameter range settings.

Experiments 6, 7 and 8 vs. 5 showed that altering upper and lower limits (and therefore initial random values) of Pc had a negative effect. However, even with no limits on crossover percentage (experiment 7), DAGA2 reached level 4 within 60,000 evaluations in 14/20 runs, demonstrating its insensitivity to this input parameter, and its capacity to evolve a successful optimization algorithm.

Experiment 9 changed tr to 8 generations, which had some impact on both success rate and number of evaluations when successful, but success rate remained at 75%.

Experiment 10 decreased Pci from 0.07 to 0.05, which reduced the number of migrants to each subpopulation from each of its 8 neighbors at intervals tr from 4 to 3. As expected, the influence on the results was very small.

4.3 Discussion

Adaptability and robustness are two closely related issues describing an optimization algorithm's performance on a given range of problems. It is obvious that robustness depends on adaptability. From our experience of applying GAs to various problems, the adapability offered by a classical GA is often not strong enough to achieve adequate performance. At the same time, it shows obvious sensitivity to many aspects of the problem solving process and is not as robust as might be expected.

DAGA2 is intended to address this shortcoming. From the two sets of experiments above, we can see that DAGA2 is able to adjust its internal GAs to appropriate settings during the problem solving process, thus offering a higher level of adaptability to the system. This consequently improves its robustness, which is a very important issue in real-world problem solving.

5 Conclusions

This paper presents a view of function optimization as a two-level optimization process: a lower level representing a black box optimization paradigm, with its internal adaptability for search of the problem space, plus a higher level algorithm which searches the space of external flexibility of the black box paradigm. This search process occurs “on-line” – that is, within a reasonable number of total function evaluations for solution of the low-level optimization problem, rather than as an “off-line” study in preparation for lower-level problem solution.

While many questions remain to be explored, particularly regarding choice of the level-two fitness function appropriate for a given class of problems, DAGA2 appears to be successful in optimizing a particular class of black box optimization algorithms, GAs, in an integrated algorithmic fashion, to solve various interesting problems. This demonstrates the power of the *HLO* framework presented in this paper. One of our current tasks is to quantify its thereotical and practical ability to boost performance for a given black box optimization algorithm.

References

[1] D.H. Wolpert and W.G. Macready. *No free lunch theorems for search.* Tech. Rep. No. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, 1995.

[2] T.C. Fogarty, Varying the Probability of Mutation in the Genetic Algorithm, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman, New York.

[3] L. Davis, Adapting Operator Probabilities In Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman, New York.

[4] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics,* SMC-16(1), pages 122-128, 1986.

[5] R. Gabasov and F. Kirillova *The Qualitative Theory of Optimal Processes.* Translated by John L. Casti, Marcel Dekker, Inc. New York, 1976.

[6] K.A. De Jong, W.M. Spears and D.F. Gordon. Using Markov Chains to Analyze GAFOs. *Foundations of Genetic Algorithms*, Morgan Kaufman, New York.

[7] A.E. Nix and M.D. Vose. Modelling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence #5*, pages 79-88, 1992.

[8] G. Wang, T. Dexter, E.D. Goodman and W.F. Punch, Optimization Of a GA and Within the GA for a 2-Dimensional Layout Problem. *Proceedings on the First International Conference on Evolutionary Computation and Its Applications*, Russian Academy of Sciences, 1996.

[9] G. Wang, E.D. Goodman and W.F. Punch, Simultaneously Multi-Level Evolutions. GARAGe Technique Report 96-03-01. 1996.

[10] J.H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.

[11] M. Mitchell and J.H. Holland, When Will a Genetic Algorithm Outperform Hill Climbing? *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993.

[12] John H. Holland, Personal Communications, 1996.

[13] E.D. Goodman, An Introduction to GALOPPS – the Genetic ALgorithm Optimized for Portability and Parallelism System, Release 3.2, *GARAGe Tehnical Report 96-07-01.* 1996.

[14] T. Baeck, F. Hoffmeister, H. Schwefel, A Survey of Evolution Strategies, *Proceedings of the Fourth International Conference of Genetic Algorithms*, pages 2-9, San Diego, 1991.

[15] D. Schlierkamp-Voosen, H. Muhlenbein, Adaptation of Population Sizes by Competing Subpopulations, *Proceedings of International Conference on Evolutionary Computation (ICEC'96)*, pages 330-335, Nagoya, Japan, 1996.

[16] M. Herdy, Reproductive Isolation as Strategy Parameter in Hierarchical Organized Evolution Strategies, *Parallel Problem Solving from Nature (PPSN II)*, Bruxelles, pages 207-217, September 1992.