On the Optimization of a Class of Blackbox Optimization Algorithms

Gang Wang, Erik Goodman, William Punch
Case Center for Computer-Aided Engineering and Manufacturing, and
Department of Computer Science
Michigan State University, East Lansing, MI 48824

GARAGe/Case Center Technical Report 97-02-01

Abstract

Blackbox Optimization(BBO) algorithms are candidate methods when knowledge of the problem is too incomplete to allow development of an efficient heuristic algorithm. Many BBOs have sufficient flexibility to allow them to adapt to the varying circumstances they encounter. These flexibilities include user-determined choices among alternative parameters, operations, and logic structures, and also the algorithm-determined alternative paths chosen during the process of seeking a solution to a particular problem. This paper presents a unifying framework for using this flexibility to tailor a BBO paradigm to the problem at hand, dynamically adjusting the algorithm during the search. It demonstrates this approach applied to a genetic algorithm. The experimental results show the effectiveness and robustness of the proposed approach.

1 Introduction

Algorithms called Black Box Optimization algorithms ("BBOs") are used to solve problems for which domain information is sparse. 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 algorithm space. Therefore, these choices can be a major obstacle to application of these methods to specific problems, which requires "precisely" tuned parameters and algorithm structure. 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. Therefore, the virtue of *adaptability* of a BBO algorithm is to some extent offset by the problem of optimally implementing and applying the algorithm.

This paper considers such issues in the context of the problem solving process. Then an algorithmic approach is proposed toward the optimization of this process. When applied to GAs, it constitutes a more uniform way to formalize the process of designing, implementing, and running an optimization system using genetic algorithms. The experimental results show the effectiveness and robustness of this approach.

2 Algorithm Flexibility

2.1 External and Internal Flexibility

Algorithm flexibility has two aspects:

- **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 representation, genetic operators, selection method, and many control parameters (*e.g.*, probabilities of crossover and mutation). Most Black Box Optimization 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 flexibility** 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, internal flexibility is realized by selection, crossover, and mutation operating on a set of chromosomes. Together, they must maintain a balance between the diversity needed to allow productive search and the focusing of search on promising regions.

Because most BBOs, like a GA, can only make decisions based on the search history they remember, their internal flexibility is often realized in a "short-sighted" manner.

External flexibility is typically controlled directly by user. However, once set, it does not change during the run. One drawback is that once the algorithm is ready to run, it cannot take advantage of its external flexibility. However, in most cases, the user knows very little about the search space, so it is hard to guarantee that an initial algorithm design is good enough to achieve the desired results. Furthermore, the "No Free Lunch" situation [2][3] illustrates that design of a BBO is simply another optimization problem. If this BBO process is complex enough, which is true for most cases, its optimization will be another BBO problem.

Some work has been done on self-adaptation of GAs. For example, many adaptive strategies [4][5][6] which are able to adjust control parameters have been developed. But these strategies were not designed to unify internal and external flexibility. Other approaches (for example, Grefenstette[7]) consider external flexibility explicitly, but view internal and external flexibility separately.

2.2 Unification of External and Internal Flexibility

The illustration below shows that the internal and external flexibilities might not differ radically, from the point of view of an algorithm's application in problem solving. Suppose you are a manager of a company, and there is a job J to be accomplished. To you, any way to accomplish J is generally a solution. To do this, you usually assign the job to people under your supervision. In terms of solving the problem, each specific person to whom you assign the task may be viewed as a particular solution to this problem. Suppose that employees A, B and C are available. Say employee B is assigned this job. Similarly, what comes to his mind is several ways to accomplish J. Again let's say he can accomplish it in ways a, b or c. The larger the number of ways, the more flexibility he has in order to solve this problem. Let's say finally he chooses a to accomplish this job. The specific way he chooses is a specific solution to this problem. It is clear that there are two levels of solution which focus on solving this problem: the manager level and the employee level, and these two solutions, $B : a$, comprise the entire solution to job J.

Similarly, if you have to write a computer program to solve an optimization problem, the solution to this problem is not only the output of the program: it also includes the process of program design, testing, running, and implementation of the solution. Therefore, the problem solving process can be divided into several levels, and each level has its own choices to be made. Adaptability constitutes the availability of alternatives and a decision process to make the choices at any level.

Therefore, external and internal flexibility can be viewed from the same perspective, and treated within a uniform framework.

In the following sections, we give a formal description of an optimization process for a genetic algorithm or similar process; 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, external flexibilities can be processed in a way similar to the algorithmic way the internal flexibilities are processed in GAs. Therefore, it offers an integrated framework for dealing with both aspects of adaptability in genetic algorithms.

3 Toward Optimization of an Algorithmic Problem Solving Process

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

3.1 Algorithmic Problem Solving Process

Following the state model notation in, for example, [8], we shall describe the behavior of a genetic algorithm for a single population as a time-invariant discrete-time system state model, as follows:

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

The state vector $X = \{x_1, \cdots, x_n\} \in I_n$ characterizes the system at time T (which is the source of its internal flexibility). For a GA, X is a vector composed of all individuals x_i in the population at time (generation) T. The control vector $U = \{u_1, \cdots, u_r\}$ denotes the parameters of the GA (which provide its external flexibility), which in this system may be different at different generations T. $U(T)$ represents a

complete specification of all 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.

We shall define the function $q(x_i)$ as the fitness (or objective function to be optimized) of individual x_i in population $X(T)$. Then we define $Q(X(T))$ as the maximum fitness of any individual x_i in $X(T)$. We will denote as X^* 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 [9][10], etc. However, for the systems to be described here, with time-varying operators and parameters, the notation of state models is simpler.

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 $U(T)$, within g generations.

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

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

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 black-box optimization 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:

For any X, U_1, U_2,

$$F(X,U_1) \geq F(X,U_2)) \Leftrightarrow (E(X,U_1) \geq E(X,U_2) \quad (3)$$

We can see how function E is approximated and tested in the system discussed below.

3.2 Measuring the Quality of Optimization

The usual goal of a GA is to optimize $Q(X(T))$ within a practical time limit. Given a system f and fixed control $U(T)$, the quality of this optimization process ($M1$) can be evaluated in many ways. Of the examples below, the first three are idealized, while the last two are more practical.

- the average time required to produce an optimal solution x^*;
- the average time required to produce a near-optimal solution 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.

The *goal* for optimization of an 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 $M1$.

For example, the minimal time problem is given in the following form:

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

In practice, we manually try a particular function $U(T)$, test its performance by measure $M1$, adjust $U(T)$, and repeat until satisfied. Therefore, we have another measure ($M2$) to evaluate the cost of this process:

- Given a system f, the minimal number of tries before measure $M1$ is satisfied.

Of course, different approaches to optimization of this process have different qualities. We discuss below one which may help to minimize measure $M2$.

3.3 Parallel Problem Solving From Society

Suppose that NASA has a very important and complicated research project to be accomplished within its research centers. NASA might decide to assign it to several selected centers, giving each the freedom to make its own project plans. Then, at specified intervals, NASA checks the progress of each center.

When it believes that the approaches at some centers are more promising, it suggests that the centers modify the poorer ones with lessons learned from more promising ones to date, while putting more resources into the promising ones.

The example above provides some insights into the design of a meta-algorithm which manipulates the external flexibility of search algorithms, while simultaneously solving the problem.

3.4 Proposed Approach HLO

Given a dynamic system f, with defined measure Q of the quality 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 Higher-Level Optimizer (HLO) as:

1) Initialize m instances of system f with randomly generated input vectors U_i, governed by :

   $X_i(T+1) = f_i(X_i(T), U_i(T))$, $1 \leq i \leq m$.

2) For each system f_i run under its control U_i for a fixed period, calculate 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.

Our goal in this approach is to improve, in an algorithmic fashion, one 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 $M1$, but also has a low value of $M2$.

4 DAGA2: Testing the HLO Approach On Genetic Algorithms

DAGA2 (Distributed Adaptive Level-two Genetic Algorithm) [11] is an instance of the HLO approach. In DAGA2, the search processes of multiple GAs with different controls are supervised and evaluated dynamically. With higher-level selection and genetic operators employed, each search process has a chance to learn better controls from others, thus making it possible for every optimization process improve its quality $M1$.

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 initializes multiple GAs with randomly generated controls, then operates its parallel subpopulations for a fixed number of generations and evaluates the performance of each subpopulation using a "level-two" performance measure related to E. Then each level-two chromosome is, 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 gains made by any level-one subpopulation among its neighbors. Then the subpopulations, with possibly different individuals and better (more fit according to the level-two fitness function) operators/rates, continue to run, and the process is repeated.

According to how we choose to evaluate the performance of the GA through time, the function E for a GA process might be defined in several ways. Some elements which might be considered include:

- best fitness of individuals in a GA
- average fitness of individuals in a GA
- number of fitness function evaluations
- a diversity measure of a population

Of course, in general, 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 seem to be useful

in 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 ([12]). Here we discuss its performance on some simple functions selected to explicate particular questions.

4.1 Adaptability of DAGA2

To demonstrate simply the capability of DAGA2 to evolve to a suitable GA design during the course of solving an optimization problem, we run it 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, such as Pm and Pc, must be set appropriately by the user to correspond to the chromosome length.

These experiments used chromosome length 900, population size 400. Before running DAGA2 on the problem, we executed a Simple GA (SGA) within DAGA2, for comparison, by fixing Pm, Pc, selection, crossover and mutation operators, and using only a single population. Experiments 1 and 2 illustrate the results of poor choices for these parameters and operators in an SGA.

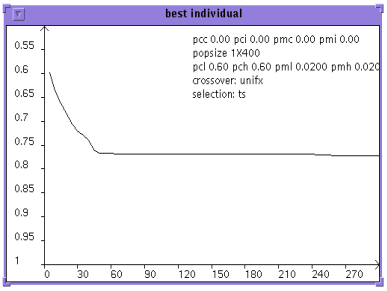

Figure 1: best/gen in experiment 1

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

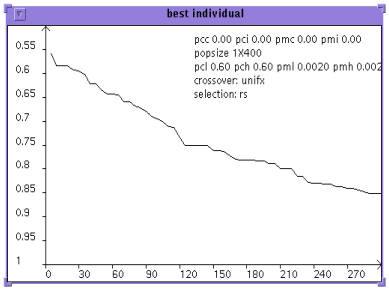

Figure 2: best/gen in experiment 2

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

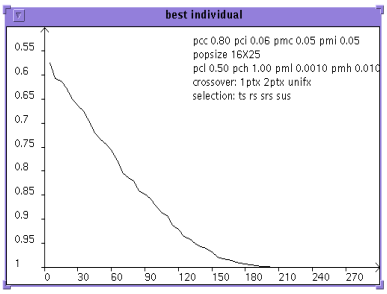

Figure 3: best/gen in experiment 3

Experiment 3 illustrates 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], respectively. In this case, the level-two fitness function was defined very simply: the increase in average fitness value in the subpopulation since the previous level-two evaluation plus the average (raw) fitness value in the current subpopulation.

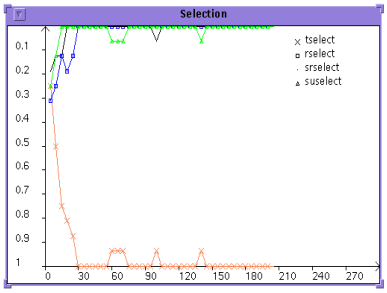

Figure 4: selection/gen in experiment 3

In this experiment, DAGA2 typically found the optimal individual at about generation 220. Other data from that experiment are given below. 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 4, tournament selection soon dominated other selection methods; while in Figure 5, several crossover operators competed; eventually, one-point

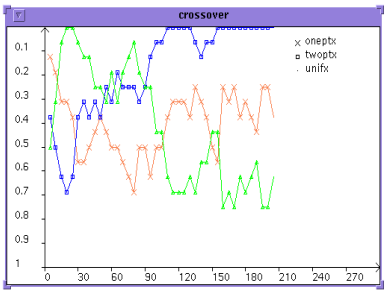

Figure 5: crossover/gen in experiment 3

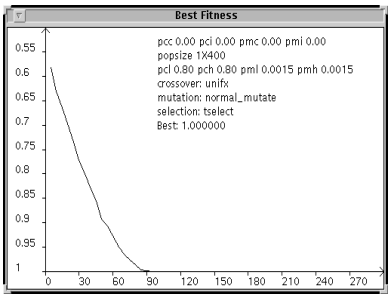

Figure 8: Best/gen in experiment 4

crossover and uniform crossover were favored.

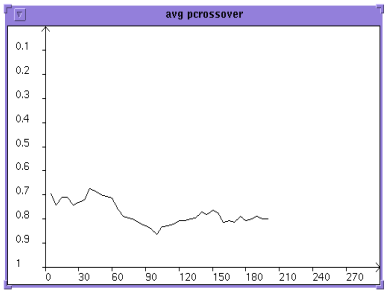

Figure 6: Avg. Pc/gen in experiment 3

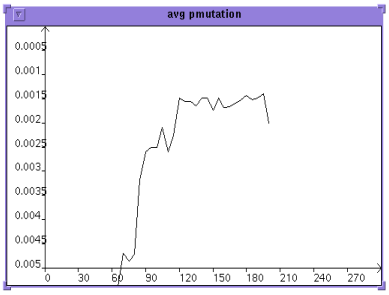

Figure 7: Avg. Pm/gen in experiment 3

In Figure 6, the average value of Pc among all subGAs wandered around 0.8; Figure 7 shows the obvious trace of Pm evolution. The average of Pm among all subGAs decreased from above 0.005 to around 0.0015 (note that from the schema theorem [13], 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 were set to their extremes ([0., 1.]).

While Figure 3 - Figure 7 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 seems to exhibit some capability to evolve better control parameters. Furthermore, the quality of the values found by DAGA2 was 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 evolved by DAGA2 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 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, especially for a 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

Holland described a "Royal Road with potholes" problem at ICGA-93 ([14]), 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" ([15]) to attain level 3 with high probability in 10,000 function evaluations. He has also explained why it is difficult for a GA to attain the final level of a Royal Road problem. 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], tourna-

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.

ment selection (tourneysize 2), one-point, two-point and uniform crossover, bitwise mutation or multi-bit mutation ([16]), 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 migrating (by copying) to each of 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 is less protective of diversity, 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 decribed 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 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. The difference between 19 and 20 successes may be due to chance, but the function producing 20/20 was used for other runs until further study shows that the superior number of function evaluations of function (2) does not sacrifice robustness. 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 really constitute optimization at level 3, which is not automated in DAGA2 (but 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 was 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 reducing 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.

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 flexibility 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.

References

[1] R.S. Michalski, Theory and methodology of inductive learning. In Michalski, R.S., Carbonell, J.G., & Mitchell, T.M. (Eds), *Machine learning: An artificial intelligence approach* (pp. 323-348). Tioga Publishing, 1983.

[2] 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.

[3] W. E. Hart, R. K. Belew. Optimizing an arbitrary function is hard for Genetic Algorithms, In R.K.Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190-195, Los Altos, CA, 1990. Morgan-Kaufman.

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

[5] J. David Schaffer, and Amy Morishima. An Adaptive Crossover Distribution Mechanism for Genetic Algorithms, *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Assoc., New York.

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

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

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

[9] Kenneth A. De Jong, William M. Spears and Diana F. Gordon. Using Markov Chains to Analyze GAFOs. *Foundations of Genetic Algorithms*, Morgan Kaufman, New York.

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

[11] Gang Wang, T. Dexter, Erik D. Goodman and William 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.

[12] Gang Wang, Erik D. Goodman and William F. Punch, Simultaneously Multi-Level Evolutions. GARAGe Technique Report 96-03-01. 1996.

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

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

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

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