```
End

Function Crossover$^2$( var daGA$_i^1$ : NGA$^1$, var daGA$_j^1$: NGA$^1$ ) : NGA$^1$
var newGA$_c^1$ : NGA$^1$;
Begin
    newGA$_c^1$= daGA$_i^1$;
    PARFOR (i=1...daGA$_i^1$.$\lambda$)
        if (*Random*() < Pci)
            newGA$_c^1$.gapop.nGA$_{ci}^0$ = daGA$_j^1$.gapop.nGA$_{ji}^0$;
    PAREND
    if (*Random*() < Pcc)
        newGA$_c^1$.operators.selection = daGA$_j^1$.operators.selection;
    if (*Random*() < Pcc)
        newGA$_c^1$.operators.crossover = daGA$_j^1$.operators.crossover;
    if (*Random*() < Pcc)
        newGA$_c^1$.operators.Pc = daGA$_j^1$.operators.Pc;
    if (*Random*() < Pcc)
        newGA$_c^1$.operators.Pm = daGA$_j^1$.operators.Pm;
    return newGA$_c^1$;
End

Function Mutation$^2$( var daGA$_j^1$ : NGA$^1$ , var Pm : R) : NGA$^1$
var newGA$_c^1$ : NGA$^1$
Begin
    newGA$_c^1$ = daGA$_j^1$;
    PARFOR  ( i= 1...daGA$_j^1$.$\lambda$)
        if (*Random*() < Pmi)
            NewGA$_c^1$.gapop.nGA$_{ci}^0$ = pInitialize$^0$();
    PAREND
    if (*Random*() < Pmc)
        newGA$_c^1$.operators.selection = *Randomize_S*();  /* Randomize initialization of selector */
    if (*Random*() < Pmc)
        Begin
        newGA$_c^1$.operators.crossover = *Randomize_C*();  /* Randomize initializ. of crossover */
        newGA$_c^1$.operators.Pc = *Random*();              /* operator and probabil. of crossover */
        End
    if (*Random*() < Pmc)
        Begin
        newGA$_c^1$.operators.mutation = *Randomize_M*();  /* Randomize initializ. of mutation */
        newGA$_c^1$.operators.Pm = *Random*();              /* operator and probabil. of mutation */
        End
    return newGA$_c^1$;
End
```

```
        $daGA_j^1$.operators.crossover = *Randomize_C*();
        $daGA_j^1$.operators.Pc = *Random*();
        $daGA_j^1$.operators.mutation = *Randomize_M*();
        $daGA_j^1$.operators.Pm = *Random*();
        $daGA_j^1$.operators.selection = *Randomize_S*();
End

Procedure $nGA^k$(var $daGA_j^k$: $NGA^k$)
var i: N;
Begin
        for (i=1;i<=$daGA_j^k$.tr;i++)
                $G^k$($daGA_j^k$);
End

Procedure $G^2$( var $daGA^2$: $NGA^2$)
var $newGA^2$: $NGA^2$      ;
var i: N;
Begin
        $newGA^2$ = $daGA^2$;
        PARFOR (i=1...$daGA_1^2$.$\lambda$)
                $NewGA_i^1$ = *Crossover*$^2$($daGA_i^1$,*Select*$^2$($daGA_i^1$,$daGA^2$), $daGA^2$.operators.Pc);
                $NewGA_i^1$ = *Mutation*$^2$($NewGA_i^1$, $daGA^2$.operators.Pm);
        PAREND
        $daGA^2$ = $newGA^2$;
End

Function $Env^2$(var $daGA_j^1$: $NGA^1$, var $daGA^2$: $NGA^2$) : R
var oldavg: R;
Begin
        $oldavg = \frac{\sum_{i=1}^{daGA_j^1 \cdot \lambda} daGA_{ji}^0 \cdot fitness}{daGA_j^1 \cdot \lambda}$
        $nGA^1$($daGA_j^1$);
        $daGA_j^1 \cdot fitness = \frac{\sum_{i=1}^{daGA_j^1 \cdot \lambda} daGA_{ji}^0 \cdot fitness}{daGA_j^1 \cdot \lambda} - oldavg$
    return $daGA_j^1$.fitness   ;
End

Function $Env^1$(var $ind_{ji}^0$ : $NGA^0$ , var $daGA_j^1$ : $NGA^1$) : R
Begin
        $ind_{ji}^0$.fitness = *Objectfunction*($ind_{ji}^0$);
        return $ind_{ji}^0$.fitness   ;
End

Function $Select^2$( var $daGA_j^1$ : $NGA^1$ , var $daGA^2$: $NGA^2$) : $NGA^1$
Begin
        for m $\in$ {1,2,...,$daGA^2$.$\lambda$}  s.t.
            $daGA_m^1$.fitness = max{ $daGA^2$.env($daGA_i^1$, $daGA^2$) where $daGA^2$.nbr[j,i]==1)}
        return $daGA_m^1$;
```

Grefenstette, John J. (1992). “*Genetic Algorithms for Changing Environments*,” Parallel Problem Solving from Nature, R.Manner and B. Manderick (eds.). Elsevier Science Publishers, B.V., Amsterdam.

Hart, W.E. , Belew. R.K. (1991) “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.

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

Lin, S-C, Punch, W, and Goodman, E. (1994) *Coarse-grain parallel genetic algorithms: Categorization and new approach*. In Sixth IEEE symposium on Parallel and Distributed Processing, Los Alamitos, CA: IEEE Computer Society Press.

Muhlenbein, H., M. Schomisch, J. Born (1991) *“The Parallel Genetic Algorithm as Function Optimizer,”* Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufman, New York.

Sannier, Adrian V. and Erik D. Goodman (1987). “*Genetic learning procedures in distributed environments*,” Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Assoc., New York.

Schaffer, J. David and Amy Morishima (1987). “*An Adaptive Crossover Distribution Mechanism for Genetic Algorithms*,” Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Assoc., New York.

Spiessens, P. and Manderick, B. (1990). *A genetic algorithm for massively parallel computers*. In R. Eckmiller, G.Hartmann, and G.Hauske, editors, Parallel Processing in Neural Systems and Computers, Dusseldorf, Germany, pages 31-36, Amsterdam, netherlands: North-Holland.

Spiessens, P. and Manderick, B. (1991). *A massively parallel genetic algorithm: Implementation and first analysis*. In Richard K.Belew and Lashon B. Booker, editors, Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers.

Srinivas, M. and Patnaik, L.M. (1994) “*Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*”, IEEE Transactions on Systems, Man and Cybernetics. VOL. 24. NO. 4. April, 1994.

Tanese, Reiko (1989). “*Distributed Genetic Algorithms,”* Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufman, New York.

Wang, Gang (1995). *On the Research and Implementation of A Distributed Adaptive Genetic Algorithm*, Master’s thesis, Beijing University of Aeronautics and Astronautics, Beijing, March, 1995.

Whitley, Darrell and Timothy Starkweather (1990). *GENITOR II: a distributed genetic algorithm*, Expt.Theor. Artif. Intell. 2(1990), 189-214.

Wolpert, D. H. and W. G. Macready (1995). *“No Free Lunch Theorems For Search,*” Technical Report, Santa Fe Institute, Santa Fe, New Mexico.

Appendix A

```
Program DistributedAdaptiveGeneticAlgorithm()
var $DAGA^2$ : $NGA^2$;
Begin
    Read($DAGA^2$.$\lambda$,Pcc,Pci,Pmc,Pmi) ;
    Read($DAGA^2$.nbr);
    PARFOR (i=1...$DAGA^2$.$\lambda$)
        $Initialize^1$($DAGA^2$.gapop.$nGA_i^1$) ;
    PAREND
    $nGA^2$($DAGA^2$);
End

Procedure $Initialize^1$(var $daGA_j^1$: $NGA^1$)
Begin
    Read ($daGA_j^1$.$\lambda$) ;
    PARFOR (i=1...$dagaGA_j^1$.$\lambda$)
        $Initialize^0$($daGA_{ji}^0$.individual) ;
    PAREND
```

lution is driven by lower level evolution, and in turn, reasserts its effect on the lower individual evolution. In their own environments, two kinds of entities adjust themselves to the adaptive “landscape.” In this sense, DAGA2 is an “integrated” multilevel adaptive system.

Since DAGA2 is only a first step in exploring the *nGA* model, much research must be done to further explore the *nGA* model, at the same time, the effectiveness of DAGA2 gives credibility to the usefulness of the *nGA* model as a framework in which to conduct and codify some further GA research. Further exploration should at least include:

(1) A better level-2 fitness function.

(2) A better level-2 mutation function, which might use, for example, two methods to mutate Pm and Pc in the structure. One method would provide only a “small adjustment” near the current value, while the other would initialize randomly within a given range.

(3) A better level-2 selection function. The current level-2 selection function is a “version” of tournament selection, which makes it possible for a better structure to spread among all subpopulations relatively quickly. This may be a disadvantage for time-varying or extremely difficult problems.

(4) More variable population size. According to “survival of the fittest” of a population, it is natural to extend or shrink the subGA’s population size, depending on its performance.

(5) Competition among different representations. Representation, as “second order” information, can make a huge difference in the performance of a GA, so it is also reasonable to include subGA-specific representations (including not only inversion-mediated permutations in representation, but also differing levels of refinement of parameters, spatial resolution, etc.) into the competition among structures, and this is readily done in the *nGA* format.

The *n*G*A* model also holds promise for explorations in artificial life, particularly as concerns group interaction. Non-traditional operators may prove attractive for use at upper levels of an *nGA*-based Alife model.

DeJong and Spears (1993) pointed out that, as evolutionary computation moves forward, there still are many potentionally important dimensions for EC research. We believe that the *nGA* model and the first realization under it, the DAGA2 implementation, have powerful and unique features as adaptive GA’s. They appear to be promising.

References

Baeck, Tomas (1992). “*Self-Adaptation in Genetic Algorithms*”, Proceeding of the First European Conference on Artificial Life, Paris, MIT Press, 1992.

Cohoon, J. P., W.N.Martin and D.S.Richards (1991). “*A Multi-Population Genetic Algorithm for Solving the K-Partition Problem on Hypercubes*” Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufman, New York.

Davis, Lawrence (1989). “*Adapting Operator Probabilities In Genetic Algorithms,*” Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufman, New York.

DeJong, Kenneth and William Spears (1993). “*On The State of Evolutionary Computation*,” Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufman, New York.

DeJong, Kenneth (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. thesis, U. of Michigan.

Fogarty, Terence C. (1989). “*Varying the Probability of Mutation in the Genetic Algorithm,”* Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufman, New York.

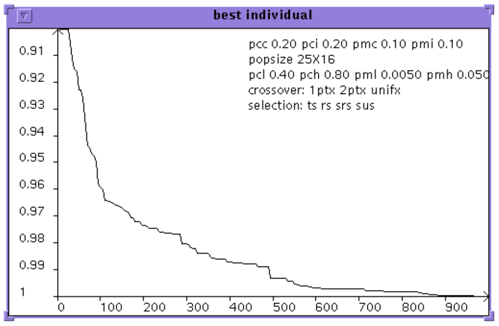
Goldberg, D.E., Deb, K., and Clark, J. (1992). *Genetic Algorithms, noise, and the sizeing of populations.* Complex Systems, 6:333-362.

Goodman, Erik D. (1994). *The Extended SGA / Island Parallel GA System*, Technical Report, Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, East Lansing.

Goodman, Erik D. (1995). *Introduction to GALOPPS -- the Genetic ALgorithm Optimized for Portability and Parallelism,* Technical Report, Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, East Lansing.

Gordon, V. Scott and Darrell Whitley (1993). “*Serial and Parallel Genetic Algorithms as Function Optimizers*,” Proceedings of the Fifth International Conference on Genetic Algorithms.

Grefenstette, John J. (1986). “*Optimization of Control Parameters for Genetic Algorithms*,” IEEE Transactions on System, Man, and Cybernetics, SMC-16(1).



```
pcc 0.20 pci 0.20 pmc 0.10 pmi 0.10
popsize 25X16
pcl 0.40 pch 0.80 pml 0.0050 pmh 0.050
crossover: 1ptx 2ptx unifx
selection: ts rs srs sus
```

Figure 30: best/gen in experiment 8

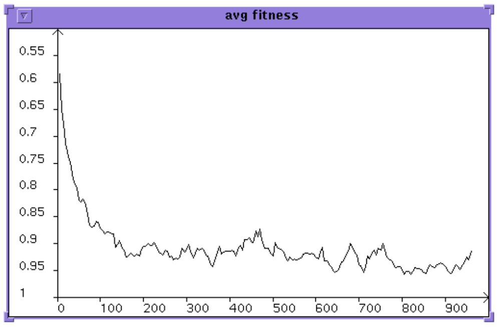


Figure 31: avg. fitness/gen in experiment 8

In 30 runs of DAGA2 on Rastrigin’s problem with this “bad” setting for level-2 parameters, it found the optimal individual within 1000 generations for 24 times, which is not a bad result in the context of Table 2.

This set of experiments illustrates the superior performance of DAGA2 to the other techniques with which it has been compared, at least for some relatively hard benchmark problems. It is relatively insensitive to the user-determined parameters, and clearly exhibits adaptation of subGA’s to the problem being solved. It is expected that its superiority will be more dramatic for many yet more difficult “real-world” problems.

6.0 Comparison with Related Ideas

The idea of using a “GA within a GA” (hereinafter, GAGA) has been considered by GA researchers for many years, since a GA often comes to mind when one faces the problem of optimizing the choice of parameters, operators, etc. for a GA system. However, to date, the instances reported (see, for example, Grefenstette, 1986) have treated the lower-level GA as a static object, evaluated once at the end of its search. This differs from the concept of DAGA2 as presented here in two major ways:

First, the range and method of optimizing control parameters in GAGA was limited because:

(1) It only allowed static parameter setting, so was less flexible in dynamic running of a GA on real problems.

(2) It did not select operators. In our experience, “which operator to use” is often more important than “how often to use it”.

(3) It used only bit-type operators, and did not include non-standard operators such as ox, pmx, etc.

(4) It did not include the power of a spatially distributed population or of island subpopulations, even within a single GA being evaluated.

Second, in GAGA, the result of each GA’s run is the fitness of one instance of control parameters applied *throughout an entire run*; thus, one run of GAGA costs much more than a DAGA2 run, in which the control parameters are optimized in parallel with the optimization of the individuals. In fact, in DAGA2, the optimization on both levels is integrated within a single evolutionary process.

Thomas Baeck (1992) has done some interesting work on “adaptive-genetic algorithms.” In his paper, each chromosome also carries some control parameters for probability of applying various genetic operators. In effect, the probability of genetic operations is adjusted as the individuals evolve. It differs from DAGA2 in that it does not select from among a variety of genetic operators, and does not take advantage of the parallel GA mechanisms as does DAGA2. However, DAGA2 resembles it in many ways, in the sense that the fitness of individuals is “fed back” to the upper-level control parameters which control the genetic operations. Related work can also be found in (Srinivas and Patnaik, 1994), in which the probabilities of crossover and mutation, Pc and Pm, are varied depending on the fitness values of the solutions.

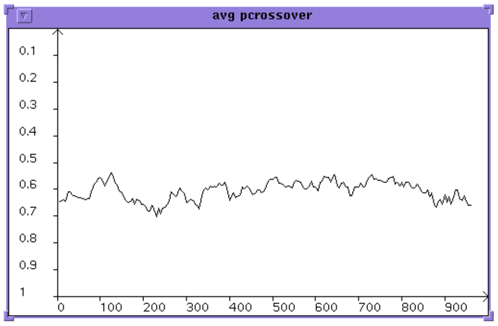
7.0 Conclusions and Future Directions

DAGA2 is a self-adaptive GA. It evolves simultaneously on different levels and scales. Higher level evo-

Figure 19 and 25 indicate that two-point crossover gradually dominates the subpopulations, with one-point crossover comprising the largest minority. It seems that two-point crossover might work better (asymptotically, for this problem) than the others. Thus, we might expect in each run, that the evolution of level-2 structures will converge to some "better" ones (for example, with two-point crossover eventually dominating all subGAs each time we run DAGA2 on the Rastrigin function), much as we expect in an SGA that the whole population will converge to some "optimal point." But as we see in Figure 19, it is even more difficult to say what is optimal in higher level evolution, cause we measure it only indirectly, from the performance of individuals in level-1 evolution. Thus, there is a slight difference between our expectation of lower- and higher-level evolution -- on the higher level, we care most about *whether the structures of the subGAs are appropriate to the current status of individuals within them*. As is true of the general concept of evolution, evolution itself is NOT a goal, but rather, a process which has no given, fixed destination. Many of our experiments exhibit this behavior. Indeed, while demonstrating outstanding performance, DAGA2 has more complexity than ordinary GAs, and there is much research to be done to understand and improve its behavior.

5.5 Experiment 8 -- Robustness Demonstration on Rastrigin Function

It is true that DAGA2 evolves better level-1 control parameters; however, to enable it, the user must set level-2 control parameters -- i.e., those which control the evolution of structures. These control parameters include Pcc, Pci, Pmi and Pmc (please see Sec. 3.3.4 for definition). However, although it may seem that the user faces the same situation as with an SGA -- having to set GA parameters -- the DAGA2 system has shown itself to be relatively insensitive to the level-2 parameter settings. Even if DAGA2 is given a"bad" set, experiment 8 shows that the effect on performance is not devastating.



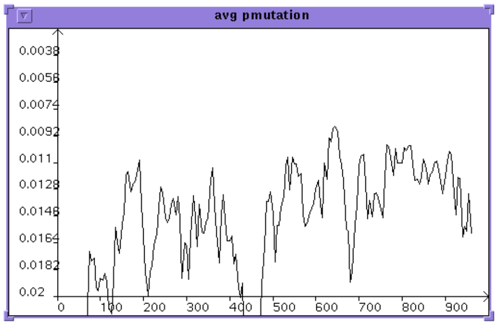
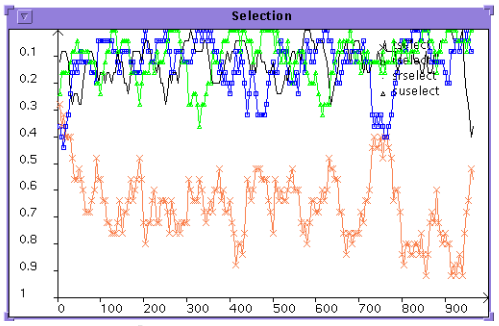


Figure 26: Pc/g in experiment 8

Figure 27: Pm/g in experiment 8

As a "bad" example, in experiment 8, we set Pmi and Pmc to 0.2, which means that new (randomly initialized) individuals are introduced for 20% of the individuals at each level-2 time step *tr*, and the crossover and mutation rates and all operators of the level-1 subpopulations are changed at random with 20% probability at each level-2 time interval *tr*. This results in erratic variations in parameters at level 1 during the entire run. As shown in Figure 30, even this does not prevent DAGA2 from finding the optimal individuals, although not as quickly.



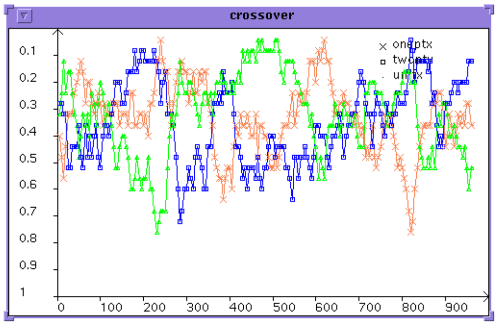


Figure 28: selection/gen in experiment 8

Figure 29: crossover/gen in experiment 8

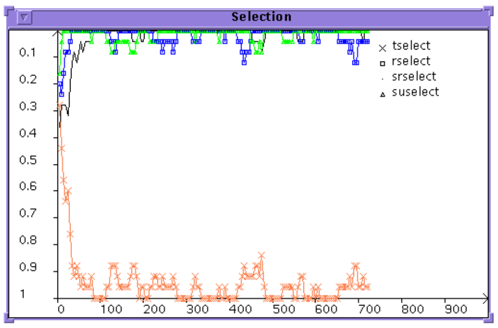


Figure 18: selection/gen in experiment 6.1

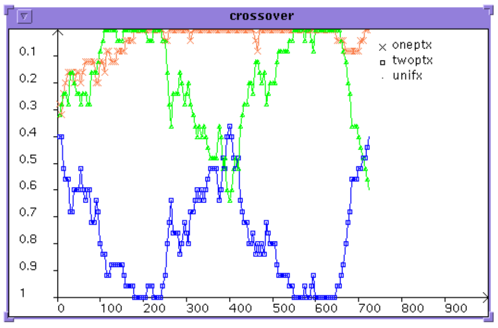
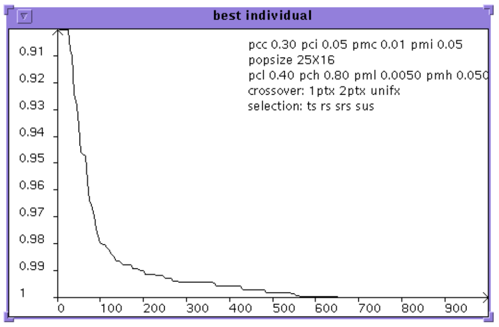


Figure 19: crossover/gen in experiment 6.1

The following figures are from a second run with a different random seed.



pcc 0.30 pci 0.05 pmc 0.01 pmi 0.05
popsize 25X16
pcl 0.40 pch 0.80 pml 0.0050 pmh 0.050
crossover: 1ptx 2ptx unifx
selection: ts rs srs sus

Figure 20: best/gen in experiment 6.2

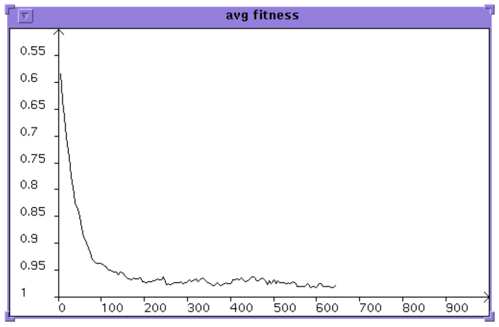


Figure 21: avg. fitness/gen in experiment 6.2

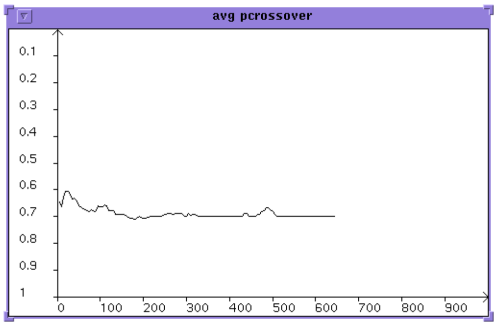


Figure 22: Pc/gen in experiment 6.2

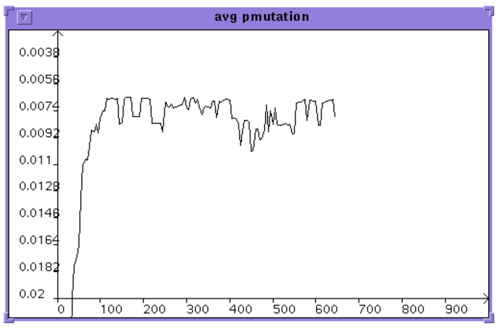


Figure 23: Pm/gen in experiment 6.2

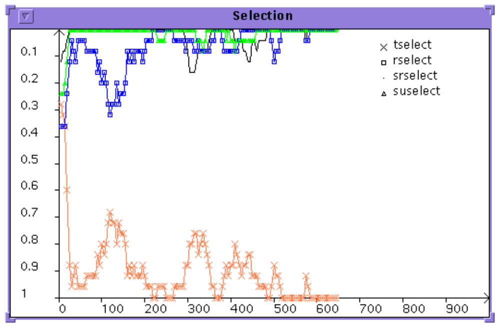


Figure 24: selection/gen in experiment 6.2

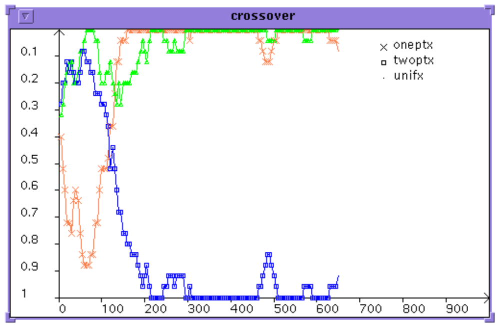


Figure 25: crossover/gen in experiment 6.2

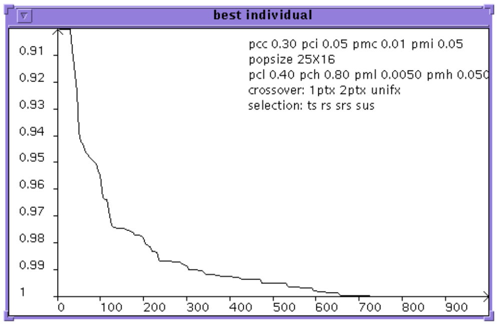
TABLE 2. Performance Comparison on Rastrigin's and Grewangk's Functions

I-ESGA	13	0.6	3	.066
I-pCHC	10	0.9	3	.047
I-Genitor	23	0.2	6	.035
Cellular	24	0.2	1	.106
DAGA2	30	0.0	11	.045

Note: Table 2 shows the performance of a number of GA systems on Rastrigin's Function and Grewangk's Function. In each case, the total population size was 400. The second and fourth columns list the number of times the optimal solutions is reached in 30 runs of each GA system. Each run of each system is for 1000 generations. All statistics except those for DAGA2 are from (Gordon and Whitley, 1993).

5.4 Experiment 7 -- Behavior on Rastrigin Function

In experiment 7, we apply DAGA2 to Rastrigin's problem, which is a relatively difficult function for optimization by a GA. To enable comparison of results to what is published, DAGA2 used the same conditions (total population size, number of generations) as were used in (Gordon and Whitley, 1993). The following data are from two replications of the same experiment (with different random seeds).



pcc 0.30 pci 0.05 pmc 0.01 pmi 0.05
popsize 25X16
pcl 0.40 pch 0.80 pml 0.0050 pmh 0.050
crossover: 1ptx 2ptx unifx
selection: ts rs srs sus

Figure 14: best/gen in experiment 6.1

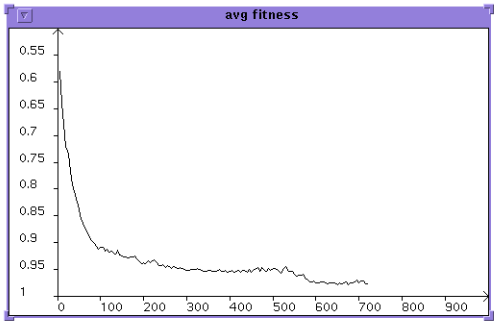


Figure 15: avg. fitness/gen in experiment 6.1

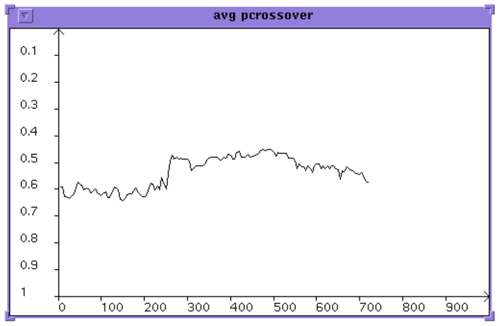


Figure 16: Pc/gen in experiment 6.1

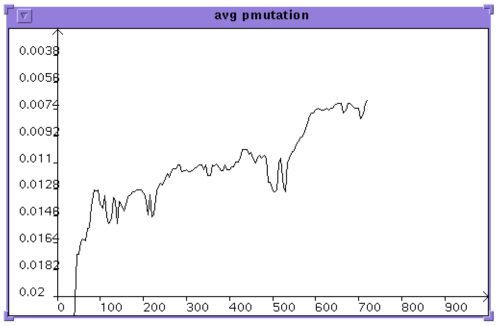


Figure 17: Pm/gen in experiment 6.1

F8: Grewangk Function: $f(x_i|_{i=1,10}) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad x_i \in [-512,512]$

On the very easy functions F1-F3, DAGA2 does not compare well, of course, with other methods. DAGA2 takes many generations to "adjust" itself to a whole new environment each time it is applied to a problem (Table 1), while most of the other systemts used good, static parameter values found *a priori* or through systematic exploration. Of course, DAGA2 searches for these parameter settings as it solves the problem, so it pays a "startup" penalty which is relatively large on very easy problems.

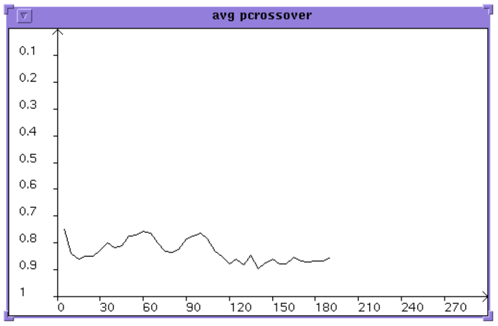
On more difficult functions for which "good" static parameters are not known *a priori*, or for which it may help to use different settings or different operators during different phases of the search, DAGA2 performs quite well in comparison to other systems, including use of a similar number of total function evaluations. For example, DAGA2 finds the best individual (antifitness of 0.0) in 30/30 runs (the best result from (Gordon and Whitley, 1993) is 24/30) and in 11/30 runs on the Grewangk function (vs. a best of 7/30 in Whitley), where Whitley's results were better than those of any of the other GA systems to which he compared them. DAGA2 does well in "directing" an SGA in setting its control parameters. The result is not astonishing, but rather relates comprehensibly to the characteristics of DAGA2, including its parallel and adaptive character. It is in part because DAGA2 adjusts and evolves its structures in the given problem environment that individuals are able to evolve higher fitnesses.

TABLE 1. Performance on DeJong's Test Suite -- avg. # generations to optimum, 30 runs

	F1		**F2**		**F3**	
System	avg	std. dev.	avg	std. dev.	avg	std. dev.
SGA	30.7	7.4	284	198	16.7	4.2
ESGA	28.9	6.8	83	55	15.3	4.1
pCHC	28.4	6.5	153	139	16.9	3.7
Genitor	17.0	4.1	190	160	8.2	2.1
I-SGA	41.3	11.2	417	253	22.0	5.3
I-ESGA	32.3	7.6	81	40	18.3	5.0
I-pCHC	33.2	7.4	78	57	18.8	4.4
I-Genitor	23.2	5.3	112	94	12.3	3.6
Cellular	32.5	8.0	105	94	17.9	4.6
DAGA2	42.83	11.38	235.00	145.67	30.87	7.86

TABLE 2. Performance Comparison on Rastrigin's and Grewangk's Functions

Function	**Rastrigin Function**		**Grewangk's Function**	
System	Number of times (in 30 runs) the optimal solution is found	Average best value found	Number of times (in 30 runs) the optimal solution is found	Average best value found
SGA	0	6.8	0	.161
ESGA	2	1.5	1	.107
pCHC	23	0.3	0	.072
Genitor	0	7.9	3	.053
I-SGA	0	3.8	7	.050



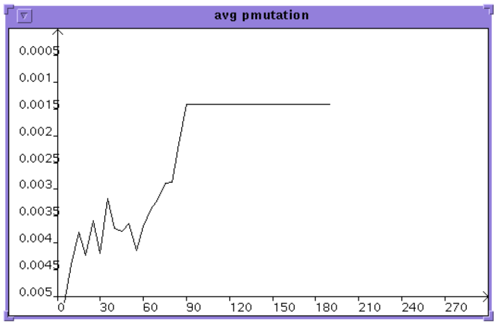
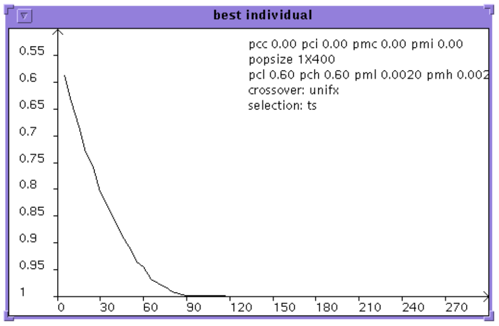


Figure 11: Pc/gen in experiment 4

Figure 12: Pm/gen in experiment 4

We have seen in experiments 3 and 4 that DAGA2 seems to exhibit some stability in its selection of control parameters. Furthermore, the quality of the values found as static settings for an SGA is confirmed in experiment 5. Note that this time tournament selection, uniform crossover, Pm at 0.002, Pc at 0.6 are used, as were found to be effective in experiments 3 and 4; the result is much better than those in experiments 1 and 2, even better than those in experiment 3 and 4, 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, which the system must then use 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, at least for a sufficiently simple task that static values for these settings are appropriate throughout the entire run. When the problem becames difficult to the extent that it is hard to find good parameter settings (see experiment 6), or that static parameter settings and genetic operator choices are not good enough to cope with the dynamic character of evolution within a run, DAGA2 performs rather well. This is demonstrated in experiments 7 and 8.



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: ts

Figure 13: best/gen in experiment 5

5.3 Experiment 6 -- DeJong, Rastrigin and Grewangk Test Functions

Gordon and Whitley (1993) tabulate the results of several well-known GA’s applied to three functions from DeJong’s test suite (F1-F3), as well as to Rastrigin’s and Grewangk’s functions, which are two commonly used multimodal benchmarking functions.

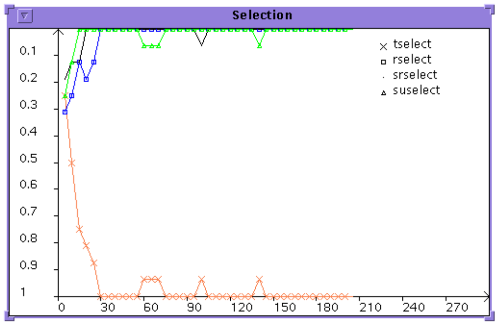
F1: $f_1(x) = \sum_{j=1}^{3} x_j^2, x \in [-5.12, 5.12]$

F2: $f_2(x) = 100\left(x_1^2 - x_2^2\right) + (1 - x_1)^2, x_j \in [-2.048, 2.048]$

F3: $f_3(x) = \sum_{j=1}^{5} integer\left(x_j^2\right), x_j \in [-5.12, 5.12]$

F6: Rastrigin Function: $f(x_i|_{i=1,20}) = 200 + \sum_{i=1}^{20} x_i^2 - 10 \bullet \cos 2\pi x_i, \quad x_i \in [-5.12, 5.12]$

gen mean average crossover and per-bit mutation rates, respectively, across all subpopulations, versus generation.



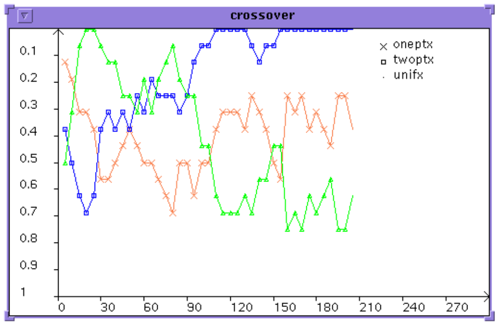
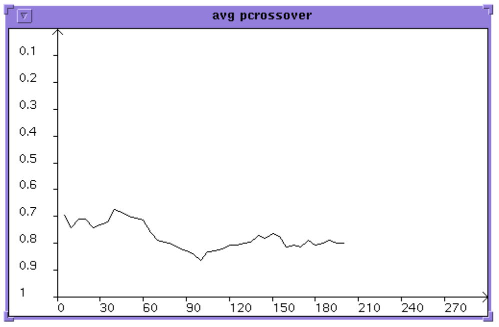


Figure 5: selection/gen in experiment 3 Figure 6: crossover/gen in experiment 3

In Figure 5, tournament selection soon dominates other selection methods; while in Figure 6, several crossover operators compete; but, one-point crossover and uniform crossover eventually dominate.



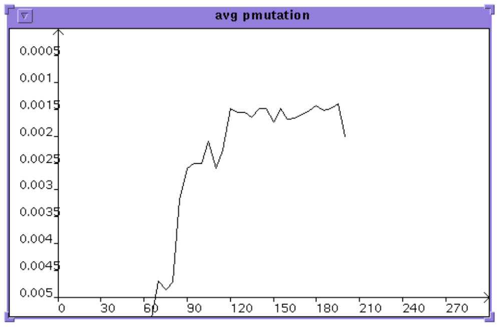
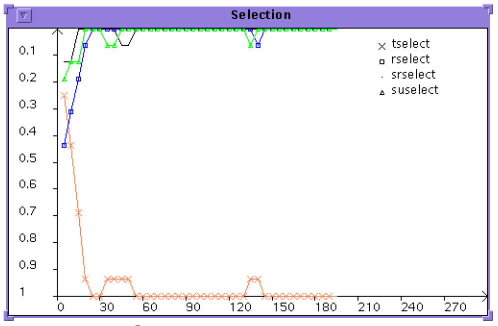


Figure 7: Pc/gen in experiment 3 Figure 8: Pm/gen in experiment 3

In Figure 7, the average value of Pc among all subGAs wanders around 0.8; Figure 8 shows the obvious trace of Pm evolution. The average of Pm among all subGAs decreases from above 0.005 to around 0.0015 (note that from the schema theorem (Holland, 1975), 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. Experiment 4 is a repetition of experiment 3, to illustrate that DAGA2 exhibits evolutionary behavior similar to that of experiment 3.



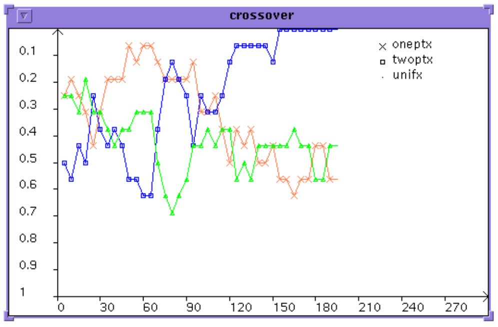
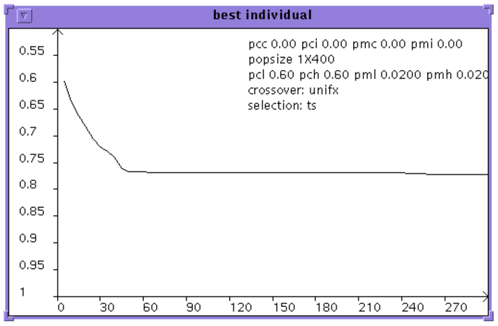


Figure 9: selection/gen in experiment 4 Figure 10: crossover/gen in experiment 4

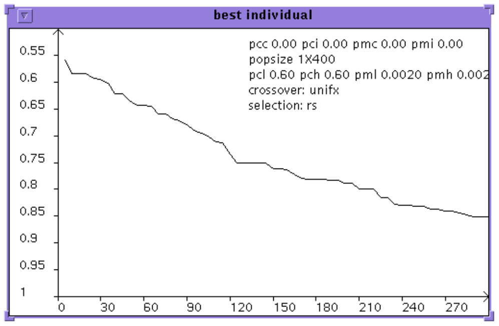
priately by the user, to correspond to the chromosome length. It is obvious that an SGA performs poorly if these parameters are not set appropriately, as illustrated in experiments 1 and 2; however, in experiment 3, DAGA2 demonstrates its level-2 adaptive character: we don't have to specify values for Pm and Pc, or choose selection or crossover methods well; DAGA2 derives them automatically.

Note that these experiments used chromosome length 900, population size 400. To simulate a Simple GA (SGA) within DAGA2, for comparison, Pm, Pc, selection, crossover and mutation operators were all fixed, and only a single population was used. Examples illustrate the results of poor choices for these parameters and operators in an SGA.



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 2 best/gen in experiment 1



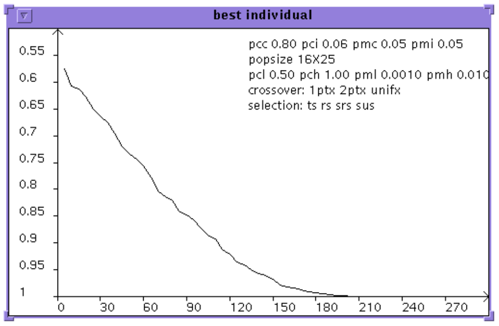
pcc 0.00 pci 0.00 pmc 0.00 pmi 0.00
popsize 1X400
pcl 0.60 pch 0.60 pml 0.0020 pmh 0.002
crossover: unifx
selection: rs

Figure 3: best/gen in experiment 2

Experiment 1 uses 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 the SGA progress ceases at generation 60.

Experiment 2 uses roulette wheel selection, uniform crossover and bit-wise mutation. This time, Pm is set well, but selection pressure is too light; thus, although the SGA continues to make progress, it still has not found the optimal value within 300 generations.

Experiment 3 illustrates the use of DAGA2. For this case, available selection methods were tournament selection, roulette wheel selection, stochastic remainder sampling and stochastic unversal sampling; available crossover methods were single point, two point and uniform crossover. The ranges for Pc and Pm were [0.5, 1.0] and [0.001, 0.01], respectively.



pcc 0.80 pci 0.06 pmc 0.05 pmi 0.05
popsize 16X25
pcl 0.50 pch 1.00 pml 0.0010 pmh 0.01
crossover: 1ptx 2ptx unifx
selection: ts rs srs sus

Figure 4: best/gen in experiment 3

In this experiment, DAGA2 found the optimal individual at about generation 220. Although this is not a very good result, it at least shows that DAGA2 can find "suitable" parameter settings and operators itself. 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 through time, Pc/gen and Pm/

3.3.5 Running DAGA2

When running DAGA2, the user furnishes the level-2 control parameter list above (*Pcc, Pci, Pmc, Pmi, nbr, tr*). As we have seen in the definition of the *nGA* model, definition of the information flow direction and frequency of information exchange between subGA's is also necessary when defining an instance of an *nGA*. These are defined here using matrix *daGA*^{2}*.nbr* (a neighbor list incidence matrix) and integer variable *daGA*^{2}*.tr*.

These parameters control the evolution of the structures. While this user-specified list might seem to be merely a "pushoff" of the parameter selection problem to a higher and even more abstract level, it is believed, although not yet conclusively demonstrated, that the quality of search occurring in DAGA2 is less sensitive to the settings of the level-2 (user-specified) parameters than is the normal search process of an ordinary GA to its choice of operators, probabilities, selection pressures, etc. This expectation has been met to date in the initial experimental testing of DAGA2.

4.0 Implementation

DAGA2 was first implemented around the "Extended SGA" precursor of GALOPPS (Goodman, 1994), using PVM to parallelize the evolution of different GA's. It is now being transplanted to GALOPPS 3.0 (Goodman, 1995). Each subGA in DAGA2 is a task in PVM, and all the genetic operations on level 2 are realized by a communication module implemented through PVM. With the non-blocking and blocking communication mechanisms available in PVM, DAGA2 has two versions -- *synchronous* mode and *asynchronous* mode. In synchronous DAGA2, all of the subGA's have the same evolution "speed", that is, selection and genetic operations happen at the same freqeuncy (interval of *tr* generations). In asynchronous mode, the speed of evolution of each subGA depends on the workload on its host processor; thus, there will be cases in which some subGA's have run many generations while others have run few. It is worth noting that asynchronous DAGA2 behavior has some interesting differences from synchronous DAGA2. Our experimental results to date are primarily from synchronous DAGA2.

5.0 Experimental Results

5.1 Description of Initial Testing

As described above, DAGA2 is expected to exhibit adaptation of its subGA's to the problems to which it is applied. Hence, a sequence of experiments is planned to examine this phenomenon.

Experiments 1 to 5 are done on a "counting 1's" problem (denoted F0) , which assigns fitness by adding the number of 1's appearing on the chromosome. DAGA2 is expected to find "good" control parameters itself.

Experiment 6 compares DAGA2 results to those in (Gordon and Whitley, 1993). in which performance of many GA systems are compared with DeJong's F1-F3, Rastrigin's function and Grewangk's function. DAGA2 performs very well on the difficult test cases (Rastrigin's function and Grewangk's function) while showing only mediocre performance on the relatively easy problems (F1-F3). This is not suprising, since DAGA2 randomly initializes the control parameters at the beginning, necessitating some adaptation time before it finds structures and settings appropriate to the specific problem. Therefore, DAGA2 performance on very simple problems will not match that of simpler methods. This is not a real disadvantage, since DAGA2 is intended for use only on more challenging problems.

Experiments 7 and 8 examine in more detail the behavior of DAGA2 on Rastrigin's function, and help to demonstrate its insensitivity to the user-supplied level-2 parameters.

A full analysis of DAGA2 as a complex dynamic system is still in progress.

5.2 Experiments 1-5 -- Counting Ones

The first several experiments were done on a "counting ones" problem, a very easy problem for a GA with appropriate parameter settings. However, some crucial parameters, such as Pm and Pc, should be set appro-

3.3.2 Level-2 Fitness Function

In DAGA2, each subGA ($daGA_j^1$) is initialized “randomly”. That means the individuals, genetic operators, and corresponding probabilities are initialized randomly (of course, within some limits, or according to some template). Thus, each subGA (i.e., level-1 GA, $daGA_j^1$) begins its run differently from the other subGAs, since it begins with a different *structure*, which defines the operators, the operator probabilities ($daGA_j^1.operators$), individuals ($daGA_{ji}^0$), etc., that it will initially use. After a period of running, each subGA computes its own “evolution quality” measure according to some fitness function ($daGA^2.env^2$). It is this “higher level” fitness function that determines how well each structure competes in the level-2 evolution process.

Two candidate level-2 fitness functions are listed below:

(1) $$env^2(t+1)(daGA_j^1, daGA^2) = g_j(t+1) - g_j(t)$$

(2) $$env^2(t+1)(daGA_j^1, daGA^2) = \frac{3 \cdot g_j(t+1)}{2} - g_j(t)$$

where: $$g_j(t) = \frac{\sum_{i=1}^{daGA_j^1, \lambda} env^{1\left(daGA_j^1 \cdot tr\right)}(daGA_{ji}^0, daGA_j^1)}{daGA_j^1 \cdot \lambda}$$, i.e., $g_j(t)$ evaluates the average fitness of subpopulation j at time t.

The first function simply rewards the gain in average fitness of a subGA with time. The second rewards a combination of that gain with the absolute level of fitness. In several experiments, the second function appears superior, which also appears reasonable, but exploration is not yet complete.

3.3.3 Evolution of Level-2 Objects

By applying “survival of the fittest” to subGAs, we seek to evolve subGAs (the structures, including the population) which are fit in the current evolutionary environment. The following are important elements of these structure:

(1) level-1 selection operator type and corresponding parameters.
(2) level-1 crossover operator type and crossover probability.
(3) level-1 mutation operator type and mutation probability.
(4) individuals within the subGA.

3.3.4 Level-2 Genetic Operators

Level-2 genetic operators such as crossover ($Crossover^2$) and mutation ($Mutation^2$) are also defined in DAGA2. They are similar to “uniform crossover” and fieldwise mutation, but they treat the chromosome as structured, and use different probabilities for different parts of the structure. Thus, we derive *Pcc* and *Pci* from $nGA^2.operators.Pc$; *Pmc* and *Pmi* from $nGA^2.operators.Pm$. For the various objects in the structure described above, the following level-2 parameters are used:

(1,2,3) *Pcc*: Probability of “crossover controls” -- i.e., parameter for uniform crossover within a subGA’s control parameters and genetic operators portion of its representation. Note that interaction between two subGA’s occurs at time interval *tr* (as defined in the *nGA*). Once a subGA selects a neighbor as its mate, it exchanges each control parameter and genetic operator with probability of Pcc (a form of uniform crossover).

(4) *Pci*: Probability of exchange of individuals between two subGA mates. Denotes the proportion of individuals swapped between the two subGA’s.

(1,2,3) *Pmc*: Probability of change (mutation) for each of this subGA’s control parameters and genetic operators.

(4) *Pmi*: Proportion of this subGA’s population that it reinitializes at random.

migration of individuals, which occurs at a regular interval: a fixed proportion of each subpopulation is selected and sent to its neighbor(s), and individuals are received from neighboring subpopulations. Migration of individuals can occur synchronously or asynchronously.

In this sense, DAGA2 can be viewed as a parallel GA. The key parallel processing in DAGA2 happens at level 1, where we assign each $daGA_j^1$ to a task, and the tasks are distributed among computational hosts.

If the whole population consists of NxN subpopulations, the number of tasks which are distributed among different hosts is NxN. There is one master task and NxN slave tasks. Each slave task runs an (extended) SGA-like program (taken from GALOPPS, a freeware parallel GA package (Goodman, 1995)) with its own control parameters (in procedure $pnGA_j^1$), and reports its status to the master periodically, selects (function $pSelect^2$) an appropriate "neighbor" with which to complete the level-2 genetic operations, and interacts with it (in procedure pG^2) . The master task synchronizes (if run in synchronous mode) or just listens (if in asynchronous mode). The detailed program flow in syn/asynchronous modes can be found in (Wang 1995).

3.3 DAGA2 Viewed As A Self-Adaptive GA

3.3.1 Background

The word "self-adaptive" we use does not refer to the adaptive character demonstrated by individuals in a GA population during evolution, but rather to the higher-level adaptation demonstrated by the algorithm while running on a specific problem. That is, it involves the *automatic*, *dynamic choice* of appropriate control parameters, operators and/or encoding strategies while applying the GA to a practical problem for which the form of an "ideal" GA is not known, or for which a sophisticated, time-varying sequence of operators, operator frequencies, mate selection rules, etc., may be useful. Since the GA can exhibit "poor" performance if applied with poor representations, operators, and/or parameter settings, and in that sense is not as "robust" as some methods, it is often important to choose the particular "instantiation" of a GA in a careful, problem-specific manner, to avoid premature convergence or essentially random search.

Grefenstette (1986) explored a hierarchical GA, which is discussed in Section 6.0 below. DeJong (1975), Grefenstette (1992), and Goldberg (1992) *et al.* are among those who have studied the issues of parameter setting for a variety of problems. They have sought generally appropriate values for population size, crossover and mutation rates, etc., in relation to some fairly simple problem characterizations (e.g., chromosome length, deceptiveness). However, they recognize the obstacles which inhibit such an approach.

First, the goal itself is difficult to define precisely. "Good" parameter settings are needed, but it is not easy to evaluate how good a value is, independent of the problem type. In theory, whenever a suite of parameter settings fits one type of problem, we can devise another type of problem on which this suite works poorly. In fact, the "no free lunch" theorem for search (Wolpert and Macready, 1994) guarantees that no such optimal settings exist for all possible problems. Hart and Belew (1991) similarly dispatch the possibility that there exists any one set of parameters which are optimal for all GA search. Many users adhere to DeJong's early suggestion of some representation-dependent "default" parameter values; although these values seems to work well for a large class of "real-world" problems, GA researchers (including DeJong) would not likely assert the global usefulness or optimality of such values.

Second, the methods mentioned above are for *static* parameter setting in a fixed environment, which represents a serious limitation (Grefenstette, 1992). Parameter settings (or even operators) effective early in a search might not be as effective in later phases of evolution. Some strategies (Fogarty, 1989)(Schaffer, 1987)(Davis, 1989), for example, have been proposed to address this problem. Genitor II (Whitley 1990) is another example of such work. As Whitley has found, discovering a good, fixed parameter adjustment strategy seems to be very "subtle", and experimental results turn out to be some what surprising. This fact indicates that the behavior of complex systems (such as GA's) is not always in accord with the intuition of the investigator. It is this fact that partially motivated the design and implementation of nGA as well as DAGA2.

In DAGA2, evolution operates simultaneously on two different scales (levels): the lower level evolution (nGA_j^1) occurs within each subGA (subpopulation), and is exactly the ordinary GA process; the higher level evolution (nGA^2) occurs among subGA's. At this level, structures compete to survive and to breed, so that subGAs will eventually adapt themselves to the specific problem they are solving ("self-adaptation").

GA" parameters and other aspects of the system. This means that the system is evolving at a higher level, as well as at the level of the "classical" individuals. The following sections explore this concept in more depth both by defining an instance of an *nGA* and by discussing the results of using an *nGA*.

3.0 DAGA2: An Instance of a Two-Level *nGA*

3.1 Introduction

DAGA2 is an instance of a two-level *nGA* (*$daGA^2$*); i.e., it utilizes a GA at two levels: at level 1 are "ordinary GA" processes, $daGA_j^1$ (where j indexes these entities), in which each *individual*, $daGA_{ji}^0$ (where *i* indexes the individuals within $daGA_j^1$) represents a solution to some problem and evolves in the usual fashion; in the higher, level-2 process ($daGA^2$), the *individuals* are level-1 GA's ($daGA_j^1$), which represent and manipulate information about level-1 (ordinary GA) processes, such as genetic operators, operator frequencies, and other typical GA "parameters." Thus, *individuals* in the level-2 process are (or characterize) level-1 *processes*. These level-2 individuals interact much as do individuals in traditional level-1 GA's. Because the level-2 GA individuals represent control parameters (including operators, etc.), their structures differ accordingly, as does the measure of "evolution quality." This measure of "evolution quality" acts as a fitness metric for each level-1 GA, determining its fitness to survive in the competitive environment of level 2. A "better" GA has a higher probability of survival, while a "poorer" GA gradually disappears from the level-2 population. In this manner, structures (parameters, operators, etc.) which allow individuals to evolve more effectively (according to the level-2 fitness measure) spread within the population of level-2 GAs. At the same time, exchanges of level-1 individuals and minor (mutational) changes in structures at both levels make it possible for better structures to be generated. The concurrent evolutionary processing occurring at both the level of individuals and the level of structures creates an integrated "landscape" in which DAGA2 evolves.

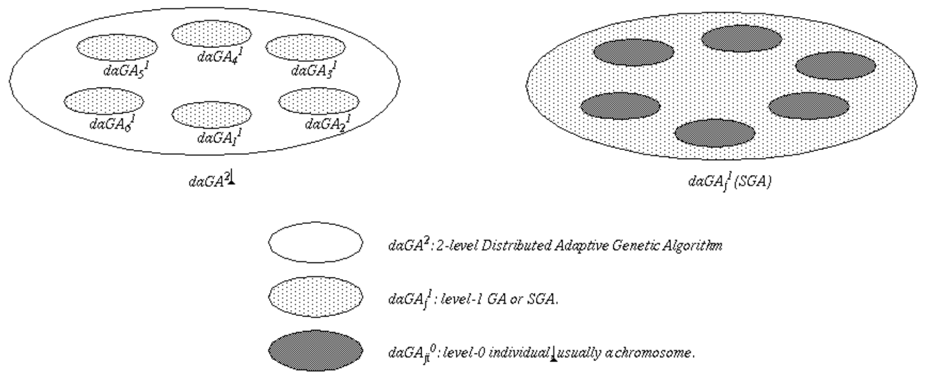

$daGA_5^1$
$daGA_4^1$
$daGA_3^1$
$daGA_6^1$
$daGA_1^1$
$daGA_2^1$
$daGA^2$
$daGA_j^1$ (SGA)
$daGA^2$: 2-level Distributed Adaptive Genetic Algorithm
$daGA_j^1$: level-1 GA or SGA.
$daGA_{ji}^0$: level-0 individual, usually a chromosome.

Figure 1: Demonstration of DAGA2 system.

A full description of the DAGA2 algorithm can be found in (Wang 1995). In Appendix A, we describe elements of the structure of DAGA2 used in its development, which was done within the *nGA* formalism.

3.2 DAGA2 Viewed As a Parallel GA

Many different architectures have been proposed for improving the effectiveness of GA search while dividing the workload among multiple processors, including localization of individuals to cells of a cellular space with localized breeding (Sannier and Goodman, 1987), division into "island" subpopulations (Cohoon, Martin and Richards, 1991), etc. For example, Reiko Tanese (1989) proposed the "distributed genetic algorithm" (DGA) as a way of efficiently parallelizing the classical genetic algorithm on a parallel computer. In their DGA, the population is divided into several subpopulations, each governed by its own SGA (Simple Genetic Algorithm) process. Inter-subpopulation communication takes place by means of

$$crossover^k \in I^{k-1} \times I^{k-1} \times R \rightarrow I^{k-1} = F_C{}^k$$

$$mutation^k \in I^{k-1} \times R \rightarrow I^{k-1} = F_M{}^k$$

$$G^k \in OPERATORS^k \times ENV^k \times GAPOP^k \rightarrow GAPOP^k$$
$$tr^k \in N$$

Notes:

(1) λ denotes the number of subsystems (nGA_i^{k-1}) within system nGA^k.

(2) We use concatenation of subscripts to identify the detailed location of a subsystem. For example, if system nGA_j^k has 3 sub-systems, then they are denoted by nGA_{j1}^{k-1}, nGA_{j2}^{k-1}, nGA_{j3}^{k-1}.

(3) $gapop^k$ denotes the set of sub-systems nGA_1^{k-1}, nGA_2^{k-1}, . . . , nGA^{k-1}.

(4) nbr^k denotes the information flow among sub-systems nGA_i^{k-1}. If $nbr^k[i,j] = 1$, it means that sub-system nGA_j^{k-1} can get structure information from nGA_i^{k-1}.

(5) env^k denotes the method to evaluate the *fitness* of sub-systems. We use nGA_j^k *.fitness* to denote the fitness of individual nGA_j^{k-1} in its environment env^k.

(6) $operators^k$ is a tuple of genetic operators within level *k*.

(7) G^k denotes the transition function for advancing one generation within system nGA^k.
Notation nGA_j^k above can be augmented with argument t to make time explicit (as in $nGA_j^k(t)$), which means the status of nGA_j^k at time *t*. With this notation for time, we can more precisely specify G^k as follows:

$$nGA_j^k(t+1).gapop = nGA_j^k(t).G(nGA_j^k(t).operators,\ nGA_j^k(t).env,\ nGA_j^k(t).gapop)$$

(8) tr^k denotes the number of generations of system nGA^k within one generation of system nGA^{k+1}. Here, *tr* means "*time resolution*".

For the sake of generality, the notion of *selection*, which is an important element in the transition function G^k, has not been segregated explicitly. This is because selection is commonly practiced in many different ways, so any particular formalization of the process would necessarily exclude some common methods for implementing it. Some approaches do selection as a binary operation, using relative fitnesses to influence selection of pairs for crossover. Some augment that by using other properties of potential mates (Hamming distance, etc.) to influence that process. At the other extreme, some systems do fitness-biased selection only as a unary operation either before or after new offspring are generated, using only relative fitnesses, fitness rankings, etc. Differences as to which new offspring survive, and which pre-existing individuals they replace, add yet more variability. While each of these "versions" of selection operations is easily formalized within this framework, we have not chosen to "dignify" any particular one with the title *selection*. It is sufficient to indicate that, in some form, selection is a part of the transition function G^k which maps the set of genetic operators and their probabilities of application, the environment, and the population at time *t*, to the next population at time *t+1*, as defined in (7) above.

The recursive nature of an *nGA* model enables a system to be composed of several subsystems, and these subsystems respectively to be composed of even smaller subsubsystems, and so on. Hence an nGA_j^k system is hierarchically organized.

The *nGA* model opens a door for further development of parallel GA's based on the scalability of the *nGA* into different scales of parallelism. For example, when we parallelize at the level of env_j^1, we get micro-grained GA's (also called global parallelization); when we parallelize at the G_j^2 level we get coarse-grained GA's (Lin, Punch and Goodman, 1994); fine-grained GA's (Spiessens and Manderick, 1990, 1991) could be implemented by parallelizing at the G_j^1 level. Furthermore, we can explore multiple levels of parallel computation by parallelizing at the env_j^1, G_j^1 or G_j^i levels simultaneously, which could enable us to do more complicated problem solving.

However, the most significant character of the *nGA* is its structural flexibility. With such a system, one observes evolution not only of the *individuals*, but also of the system -- including genetic operators, "typical of the problem), selection method, crossover and mutation operators and frequencies, population size, and many others. All of these are typically preset by the user before the actual operation of the GA begins.

Generally, a system with a multi-level structure performs best when each level of the system is harmonized in some fashion with the performance of the other levels. In this sense, a GA can be made to operate as a function optimizer at two levels, evolving not only individuals (or candidate solutions), but also optimizating the *system* in which individuals evolve. Such system-level evolution is called in this paper s*tructure evolution*, and is a feature of the *nGA* model presented here.

We define a model (*nGA*) to formalize a type of complex GA-like system, G, with the following characteristics:

(1) The system is composed of a finite number of items at each of several levels.

(2) The structures of items of the same level are similar, and can be modified within some limitations.

(3) This within-level similarity permits items on the same level to exchange some structural information in defined ways, while maintaining the validity or admissability of their structures at that level.

(4) Items on the same level can replace one another while maintaining the integrity of any higher-level structures to which they belong.

One can observe many *G* systems in the real world, such as various characterizations of human society, corporations, and so on. All these systems evolve in some similar ways, resembling in many aspects our *nGA* model.

Definition 1: An nGA_j^k is an 8-tuple which denotes the *j-th* node in a level-*k G* system,

$$nGA_j^k = \langle \lambda_j^k, gapop_j^k, nbr_j^k, env_j^k, operators_j^k, tr_j^k, G_j^k, fitness_j^k \rangle$$

We call this 8-tuple the *structure* of system nGA_j^k. Without loss of precision, we denote it as:

$$nGA_j^k = \langle \lambda^k, gapop^k, nbr^k, env^k, operators^k, tr^k, G^k, fitness^k \rangle$$

if we use , $GAPOP^k$, NBR^k, ENV^k, $OPERATORS^k$, N, G^k,to stand for the sets to which λ, $gapop$, nbr, env, $operators$, tr, G, $fitness$ belong, respectively, then:

$$nGA_j^k \in N \times GAPOP^k \times NBR^k \times ENV^k \times OPERATORS^k \times N \times G^k \times R = I^k$$

The items in the tuple are defined recursively over *k*, as follows:

for *k* = 0:

nGA_j^0 = a string (typically), but intuitively, any “chromosome,” representing a candidate solution to a problem,

I^0 = { admissable chromosomes }.

for *k* > 0:

$$\lambda \in N$$

$$gapop^k = \{nGA_1^{k-1}, nGA_2^{k-1}, nGA_3^{k-1}, \ldots, nGA_\lambda^{k-1}\}$$

$$nbr^k = \left[nb_{ij}^k\right]_{i,j=1\ldots\lambda}, nb_{ij}^k = 0{,}1$$

$$env^k = I^{k-1} \times I^k \rightarrow R$$

$$operators^k = \langle crossover^k, mutation^k, Pm^k, Pc^k \rangle$$

Simultaneous Multi-Level Evolution

Gang Wang, Erik D. Goodman and William F. Punch, III

Genetic Algorithm Research and Applications Group (GARAGe)

Department of Computer Science and Case Center for Computer-Aided Engineering & Manufacturing

Michigan State University

wanggan1@cps.msu.edu, goodman@egr.msu.edu, punch@cps.msu.edu

Abstract

A Genetic Algorithm (GA) is a form of complex system in which various structures interact via sufficiently complicated operators and rules that it is difficult to characterize the behavior of the system exactly. However, competition at this level is in the control of upper-level structure(such as genetic operators and corresponding parameter setting, etc.). Thus, how to optimize this structure to make individuals(chromosomes) evolve better is essential to GA application. In this paper, we first present a model or formalism for a multi-level GA (denoted *nGA*) which describes evolution occurring on several levels and scales simultaneously. Then we present an instance of an *nGA* (DAGA2) with a particular desired set of distributed and adaptive characteristics. One of the main features of DAGA2 is "on-line" competition among multiple GAs during problem solution. Performance of this system, realized using the PVM (Parallel Virtual Machine) tools, was determined for several well-known test functions. The results demonstrate that this model, in contrast to some earlier "GA within GA" approaches, shows promise not only as a theoretical framework, but also as a practical tool for GA search.

Keywords

Genetic algorithm, adaptive genetic algorithm, multi-level evolution, multi-level GA

1.0 Introduction

A Genetic Algorithm (GA) is a form of complex system in which various chromosome interact via sufficiently complicated operators and rules that it is difficult to characterize the behavior of the system exactly. However, competition at this level is controled by many upper-level factors such as genetic operator it uses, corresponding parameters, etc. Thus, optimization of these upper level entities to make individuals evolve better is essential to GA application. In section 2, we first present a model or formalism for a multi-level GA (denoted *nGA*) which describes evolution occurring on several levels and scales simultaneously. Section 3 defines an instance of an *nGA* (DAGA2) with a particular desired set of distributed and adaptive characteristics. One of the main features of DAGA2 is "on-line" competition among multiple GAs during problem solution, in this way, better GA is supposed to "pop up" during this high level competition, while chromosomes within each GA are evolved simultaneously. Performance of this system, realized using the PVM (Parallel Virtual Machine) tools, was determined for several well-known test functions, which is in section 4 and section 5. In section 6 and section 7 we compare our works with related works and discuss future research in this direction. We hope this model as well as DAGA2 system, give more insight into the dynamic behaivors of GA as a complex system.

2.0 The *nGA* model

A GA is an adaptive system (Holland, 1975), and many of its elements are modeled on adaptation in nature, at several levels. A GA abstracts some of the mechanisms found in evolution for use in searching for optimal solutions within complex "fitness landscapes". Like the natural world, a GA itself is a complex system in which many elements interact. These elements include the encoding mechanism (or "representation"