Appeared in Proceedings, First International Conference on Evolutionary Computation and its Applications, E. Goodman, W. Punch, V. Uskov, Eds., Presidium, Russian Academy of Sciences, June, 1996.

OPTIMIZATION OF A GA AND WITHIN A GA FOR A 2-DIMENSIONAL LAYOUT PROBLEM

Gang Wang, Terrence W. Dexter, William F. Punch, III

({wanggan1, dexterte, punch}@cps.msu.edu)
Department of Computer Science

Erik D. Goodman (goodman@egr.msu.edu)

Case Center for Computer-Aided Engineering and Manufacturing
112 Engineering Building
Michigan State University, East Lansing, MI 48824 USA

ABSTRACT

A GA's performance on a specific problem is related to many factors, such as genetic operators and corresponding parameter settings and the representation of the problem on the chromosome. Optimization of these factors to improve the speed and robustness of search is essential to successful application of a GA. The work reported here uses a two-level GA system (DAGA2) to solve a practical problem: the conceptual layout of machines on a factory floor (not the detailed design), based on a matrix of positive and negative relationships of various strengths between machines. This is a difficult, high-dimensionality problem. Performances of a traditional parallel GA system and our DAGA2 system are compared. The later one is compared with and without the use of several different representations for the problem at various times and in various subpopulations, demonstrating the strong contribution which the use of multiple representations makes to solution of the problem. The authors argue the necessity of optimizing many aspects of the GA in order to obtain useful solutions on such difficult, real problems.

1.0 INTRODUCTION

The performance of a genetic algorithm (GA) on a specific problem is related to many factors, such as the choice of genetic operators and the corresponding parameter settings, and the representation of the problem on the chromosome. Optimization of these factors to improve the speed and robustness of search is essential to successful application of a GA. Section 2 briefly introduces a 2-level GA system which addresses this problem. Section 3 discusses the layout problem of machines on factory floor, based on an abstract entity relationship matrix which provides for each machine its desired quantitative positive or negative relationship with each other machine (i.e., to what extent they should be kept close together or far apart). Section 4 presents results from an ordinary parallel GA, while section 5 presents results from the DAGA2 system. Section 6 describes the introduction of both simultaneous and sequential different representations for the problem, improving the speed and quality of the global search. Section 7 discusses the benefits (or near necessity) for employing problem-specific tuning and multiple representations in order to obtain useful solutions to such difficult, high-dimensionality real-world problems.

2.0 DAGA2: A BRIEF INTRODUCTION

Consider a GA as a complex system: there are at least two levels of entities within it -- upper level structural entities (genetic operators, corresponding parameter settings, population sizes, etc.), and the lower level chromosomes, for which evolution is determined (or controlled) by the upper level entities. Since these upper level factors make a great deal of difference to GA performance, and their optimal choice/selection may vary with the state of the GA, one might expect it to be useful for them also to evolve during the operation of the GA. This idea might be called multi-level evolution. This idea has been formalized in a model called *nGA* (Wang, Goodman and Punch, 1996) in a general framework for expressing such systems.

DAGA2 is an instance of a two-level *nGA*; i.e., it utilizes a GA at two levels: at level 1 are "ordinary GA" processes, in which each *individual* represents a solution to some problem and evolves in the usual fashion; in the higher, level-2 process , the *individuals* are level-1 GA's , and each chromosome represents information about a level-1 (ordinary GA) processes, such as its genetic operators, operator frequencies, population sizes, representation, probability of migration, and other typical GA "parameters." 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" level-1 GA process has a higher probability of survival and mating (via crossover), 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 the level-2 GA. 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. Depending on the settings of its parameters, the DAGA2 system can act as an island parallel GA, a fine-grain parallel GA, and/or a self-adaptive GA, in the sense that the whole GA system, according to "survival of the fittest", can evolve high-performance individuals as well as a high-performance GA itself. A full description of the DAGA2 algorithm can be found in (Wang 1995).

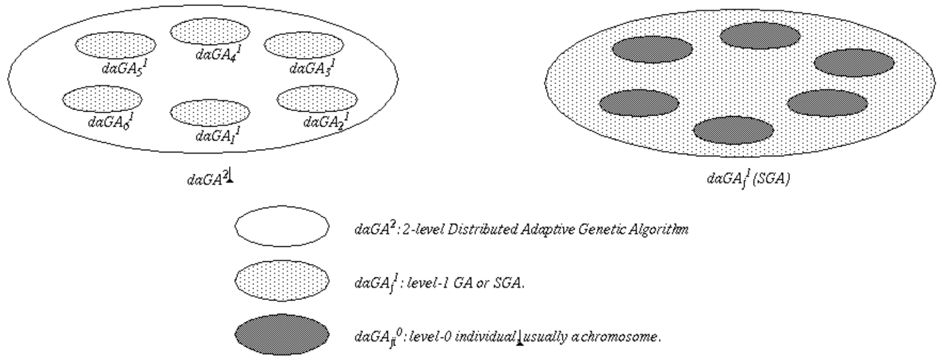

$daGA_5^1$
$daGA_4^1$
$daGA_3^1$
$daGA_6^1$
$daGA_1^1$
$daGA_2^1$
$daGA^2$
$daGA_j^1$ (SGA)
$daGA^2$: 2-level Distributed Adaptive Genetic Algorithm
$daGA_j^1$: level-1 GA or SGA.
$daGA_{ji}^0$: level-0 individual, usually a chromosome.

Figure 1: Conceptual diagram of the DAGA2 system

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

Appeared in Proceedings, First International Conference on Evolutionary Computation and its Applications, E. Goodman, W. Punch, V. Uskov, Eds., Presidium, Russian Academy of Sciences, June, 1996.

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 of 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.

3.0 PROBLEM DEFINITION

Optimizing the layout of machines on a factory floor has emerged as a high-dimensionality problem which is hard to solve. In a factory layout, some machines should be clustered together due, perhaps, to flow of parts from one machine to another. In modern machines, part flow can be dynamic and can result in one machine having a need to be centrally located. Other machines (or stations), however, should be kept separated due to constraints such as chemical interaction, noise pollution or similar such negative relationships. The problem addressed here is not to perform detailed design of the configuration of the machines on the given factory floor. Rather, it is to suggest to the layout planner possible configurations of machines which may be advantageous, based on the input of a matrix of desired relationships between each pair of machines, which may be positive, negative, or neutral, and quantified to any desired degree, and on the geometry of the factory floor and the sizes of individual machines. Using a heuristic method, this problem is hard to solve, in the sense that the suitable location of one machine is often directly related to the positions of others. Thus, these machines constrain each other such that it is difficult to figure out the best overall layout of all machines.

Discussions with factory floor planners indicate that such "association" information typically neglects some significant factors which the designer knows, but does not capture in such matrices. Therefore, automatic layout systems which attempt to specify precisely the location and orientation of the precise footprint of each machine are often rejected by the designer, since they don't let the designer add his/her knowledge to the design. In contrast, the approach used here suggests placement of machines as generalized circles (of various sizes) and leaves room for the designer to use CAD tools to adjust the resulting layouts based on supplemental knowledge. In fact, designers express a desire to examine multiple (near-optimal) solutions, to look for features which are common, and for alternatives with similar "costs."

4.0 A "TRADITIONAL" GA METHOD

When applying GAs to the layout problem, it is natural to think of putting binary (or Gray) codes for discretizations of the x and y coordinates of each machine together on the chromosome. Although very primitive, this idea works because it reduces biasing errors that can occur when solving a permutation problem or using more complex representations. It also keeps the most fundamental building blocks (specifying location of one machine) together.

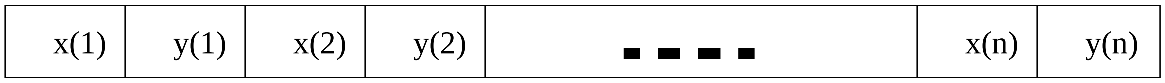


Figure 2. A simple representation of the problem on a chromosome

As Figure 2 shows, each field represents the x or y coordinate of a machine. The alphabet size (discretization) used is typically 512 values/coordinate, and crossover occurs only between the 9-bit fields.

The advantages of using island parallelism are well documented, and it is ubiquitous in the work reported here. A typical island-GA result is shown in Figure 3, the best individual found after 1000 generations (nearly converged) with a total of 25 subpopulations of 100 individuals each, one-point $P_{cross} = 0.7$, $P_{bitmutate} = 0.2$/chromosome..

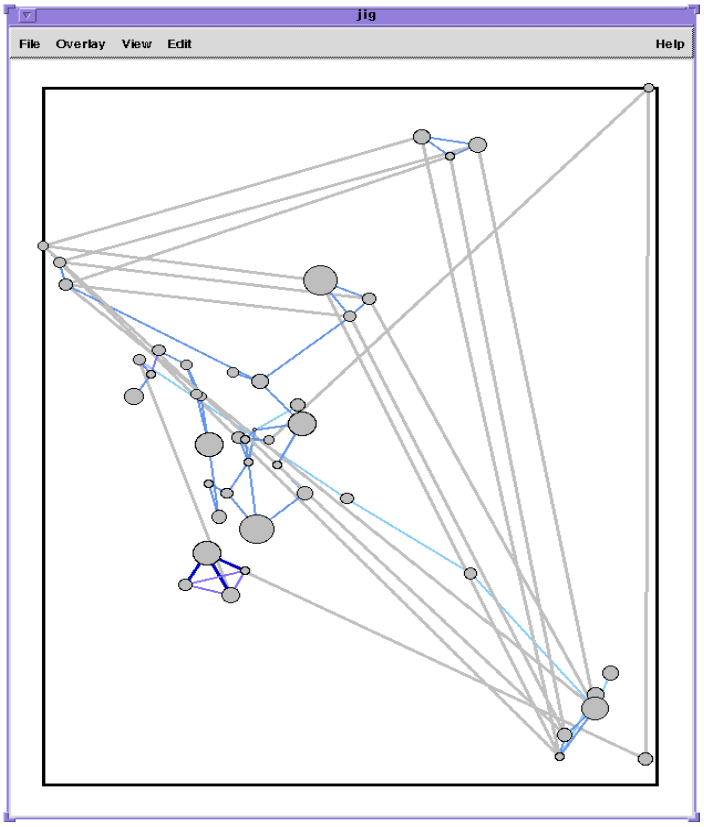


Figure 3: A typical result from a “traditional” GA method

5.0 THE DAGA2 APPROACH

5.1 Dynamic Adjustment of Control Parameters

Setting of control parameters is known to be a problem-specific issue. 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. The same arguments also hold for choice of genetic operators.

Another issue is that 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 the nGA formalism and the DAGA2 two-level GA system.

With the DAGA2 system, this problem can readily be addressed. As discussed in (Wang, Goodman and Punch, 1996), DAGA2 can efficiently adapt itself to the problem at hand. These structure adjustments might include representation, genetic operators and related control parameters. For the example problem considered here -- the 2-dimensional layout problem -- the results are shown in Figures 3-6.

5.2 Dynamic Adjustment of the Representation

It is well know that performance of GA is greatly influenced by the representation employed. Two types of representation changes will be used here: inversions and hierarchical clustering. From the earliest days of GA research, the importance of the order of appearance of entities on the chromosome has been recognized and demonstrated (Holland, 1975, Cavicchio, 1970). Linkage has a dramatic effect on the outcomes of crossover, albeit only a second-order effect in terms of phenotypic expression. For this problem, however, DAGA2 does not use a classical inversion operator on the chromosomes, changing them through time; rather, each subpopulation is provided its own (randomly chosen) inversion map. The loci of migrating individuals are rearranged into the pattern of the receiving subpopulation. This process allows the search to utilize good building blocks found in one representation together with those from another, yielding better results than either representation alone would.

It also allows these representations to be selected for by the second-level competition mechanism present in DAGA2. If some representation is indeed performing better than others for the particular problem being addressed, there is a larger probability that its subpopulation will survive and have a strong influence on the aggregate evolution of the solution.

As mentioned above, in the simple representation method, the (x,y) coordinates of the machines (entities) to be placed are encoded on the chromosome in order of machine number, which is fixed, yielding the single representation used in ordinary GA’s. In DAGA2, the various GA’s encode the (x,y) coordinate pairs in different orders. As Figure 4 shows, these orders are generated randomly, in the beginning of the DAGA2 run. In some representations, some “keep together” machines will probably be close to each other on the chromosome, which facilitates the search done by the crossover operator.

x(12)	y(12)	x(5)	y(5)	x(30)	y(30)	▬ ▬ ▬ ▬	x(8)	y(8)

x(4)	y(4)	x(9)	y(9)	x(20)	y(20)	▬ ▬ ▬ ▬	x(41)	y(41)

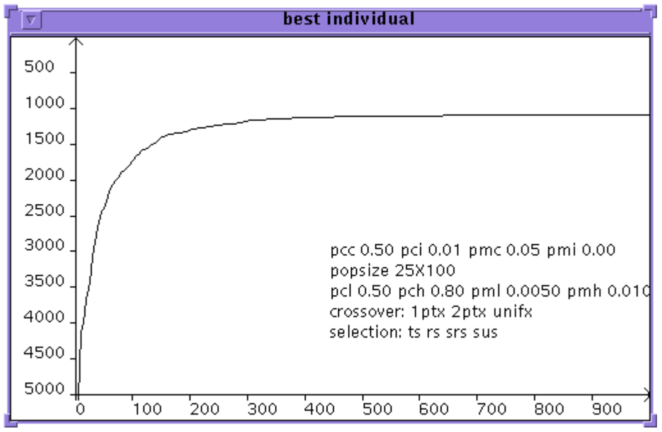
x(33)	y(33)	x(19)	y(19)	x(6)	y(6)	▬ ▬ ▬ ▬	x(24)	y(24)

Figure 4: Three example representations for a GA chromosome in a 2-D layout problem

The competition among multiple representations helps to find a suitable machine order for the representation on the chromosome.

5.3 Experimental Results

Results from DAGA2 running is superior to the traditional method, mainly because DAGA2 uses dynamic control paramenter setting and representation. The following figures shows the trace of all these adjustments. 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. 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 vs. gen means fraction of subpopulations using each crossover method through time, Pc vs. gen and Pm vs. gen mean average crossover and per-bit mutation rates, respectively, across all subpopulations, versus generation.



pcc 0.50 pci 0.01 pmc 0.05 pmi 0.00
popsize 25X100
pcl 0.50 pch 0.80 pml 0.0050 pmh 0.010
crossover: 1ptx 2ptx unifx
selection: ts rs srs sus

Figure 5.a: Best of gen. in DAGA2 run

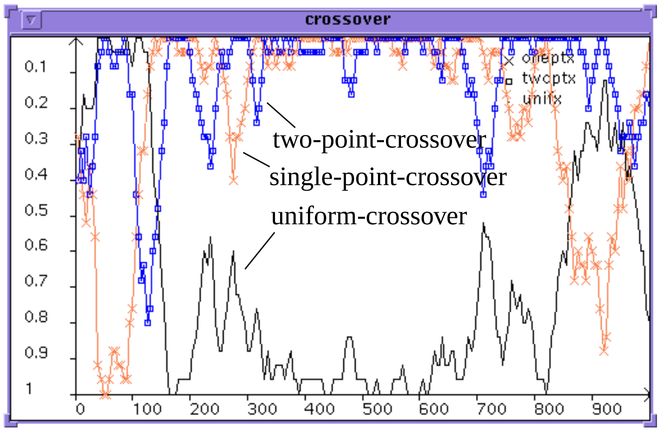


Figure 5.b: Crossover vs. gen. in DAGA2 run

Appeared in Proceedings, First International Conference on Evolutionary Computation and its Applications, E. Goodman, W. Punch, V. Uskov, Eds., Presidium, Russian Academy of Sciences, June, 1996.

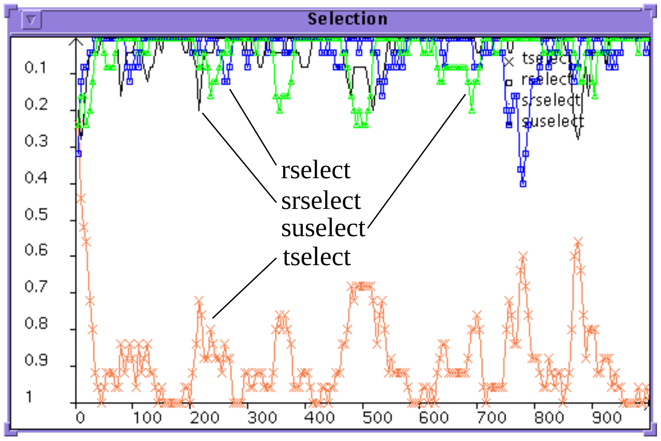


Figure 5.c: Selection vs. gen. in DAGA2 run

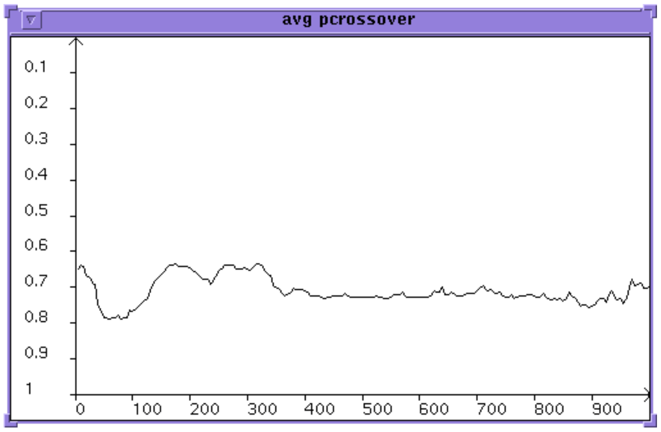


Figure 5.d: Pc vs. gen. in DAGA2 run

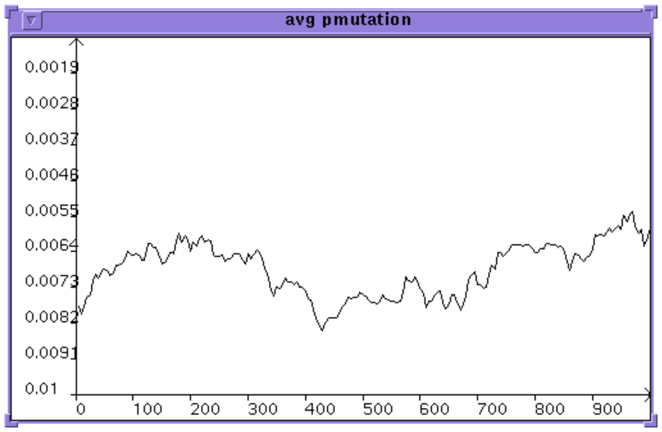


Figure 5.e: Pm vs. gen in DAGA2 run

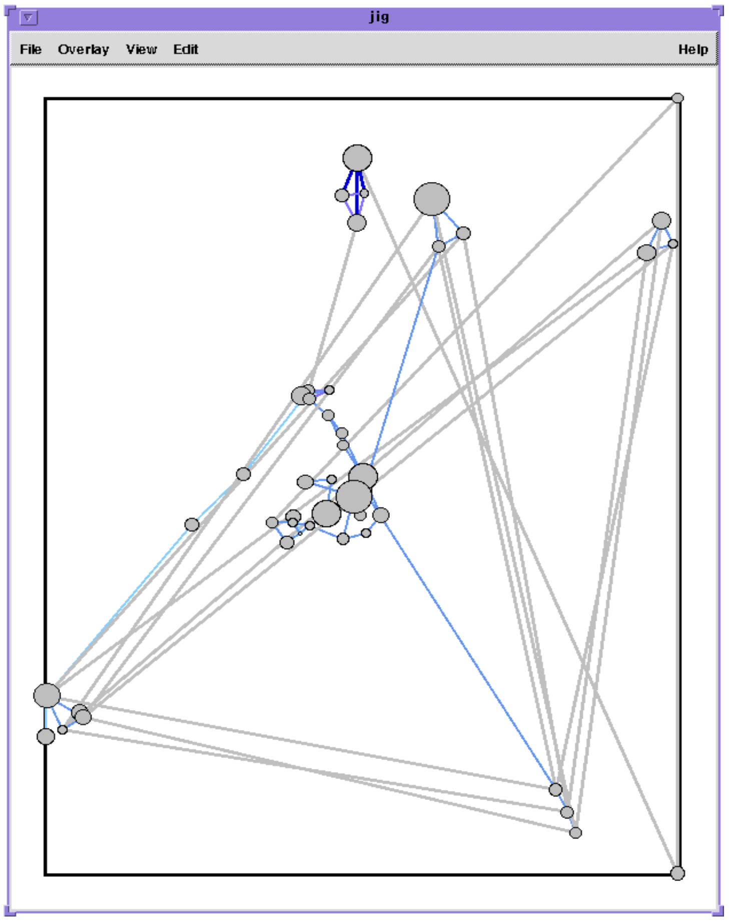


Figure 5.f: Sample final result in DAGA2 run

6.0 CLUSTER OPTIMIZATION IN JIGSAW

6.1 Why Cluster?

However, the method described above has some limitations. For example, in Figure 6.a, machines A, B and C should be located as close as possible, while they should be “pushed away” by some other machines D and E as far as possible. In this case, the “close” relationships combine to form a much stronger attraction than the drive to push them away (in other words, if A, B and C are far away from each other, the penalty for the final fitness value is much greater than the gain they get from enhancing the single “push away” relationship). Any significant change to any ONE of A, B or C is impossible (Figure 6.b). There is little hope that A, B and C will move “together” as a unit under the current genetic operators, especially when the population has begun to converge.

Thus, when (a) some entities have strong “close up” relationships, and when (b) their physical locations are close together enough, it is natural to treat them as a “cluster” for the subsequent evolution, and thereby to dramatically reduce the search space. Figure 6.d demonstrates this principle.

Appeared in Proceedings, First International Conference on Evolutionary Computation and its Applications, E. Goodman, W. Punch, V. Uskov, Eds., Presidium, Russian Academy of Sciences, June, 1996.

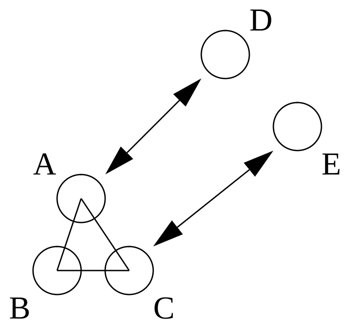


Figure 6.a: Original layout

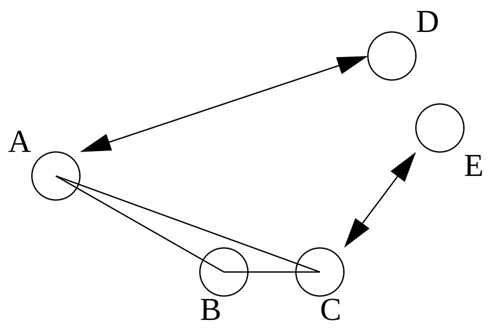


Figure 6.b: Unsuccessful try to change A’s location

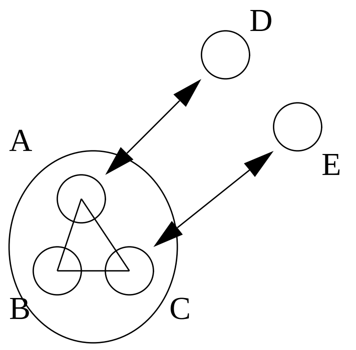


Figure 6.c Treating A,B and C as a cluster

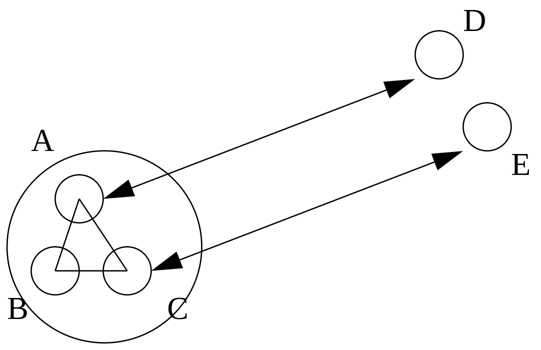


Figure 6.d Moving members of cluster together as a unit.

6.2 Implementation

Figure 7 shows a simple representation encodeing the clustering information in the JIGSAW problem. For each cluster, the GA keeps the coordinates of the machines in this cluster relative to a reference point, allowing the calculation of the absolute coordinates of each machine and the use of the “standard” fitness function to evaluate the fitness of this clustered chromosome. This hierarchy of representations (at different levels of refinement) is thus similar to those used in the authors’ “injection architecture” for improving performance of coarse-grain parallel GA’s.

gx(1)	gy(1)	gx(2)	gy(2)	▬ ▬ ▬ ▬	gx(k)	gy(k)

Figure 7. A simple representation of a JIGSAW “clustered” chromosome

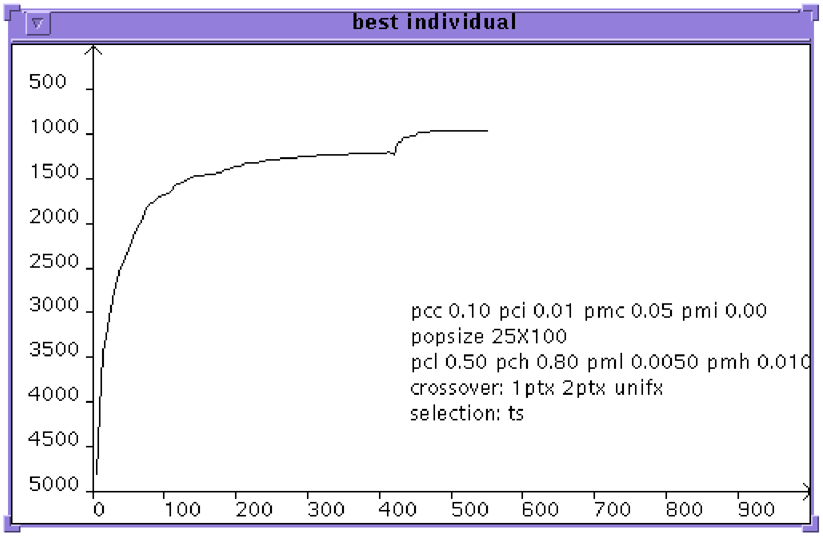
Although the clustering idea is very attractive, it is hard to say *a priori* which entities should be clustered with each other, given a relationship matrix describing all of the machines. The approach taken here is to use a combination of the *a priori* matrix information and proximities determined by early progress in the evolution of a solution. This is accomplished using an “injection-like” method [Lin, Punch and Goodman, 1994]. In the approach taken here, the evolution process is divided into two phases. In the first phase, DAGA2 runs GA’s in parallel, as described above, until they begin to converge. Then, within the current best layout in each GA population, entities close to each other and strongly positively related in the matrix are clustered to form a single entity in the new representation of the layout for phase two. This latter phase then seeks an optimal arrangement for these clusters.

This method resembles in some ways the "injection architecture" used in island parallel GA's in [Lin, Punch and Goodman, 1994]. In the "injection architecture," individuals from subpopulations using coarse representations for the problem migrate to other subpopulations using more refined representations. In this problem, the first phase of evolution offers the later phase suitable information to construct the appropriate clusters. A natural extension would be to have third and subsequent phases which alternate between the clustered and the initial representations, "fine tuning" the cluster-derived result, and in that case, the similarity is even closer. However, that process has not yet been tested.

6.3 Experimental Results

In this experiment, we use DAGA2 to run multiple GA's simultaneously. After several hundred generations, when each GA has converged to some extent, the analysis of the best individual is done for each GA separately, dividing its entities into clusters, and then randomly generating new locations for each cluster to initiate the second phase of the search process.

Figure 8 shows the best-of-generation fitness throughout the entire run. Figures 9.a and 9.b show the best individual before and after clustered optimization, respectively. The cluster optimization take place at about generation 400, and is followed by a new period of dramatic increase in fitness, with progress greatly improved over the largely stagnated search preceding the clustering operation.



pcc 0.10 pci 0.01 pmc 0.05 pmi 0.00
popsize 25X100
pcl 0.50 pch 0.80 pml 0.0050 pmh 0.010
crossover: 1ptx 2ptx unifx
selection: ts

Figure 8: Best fitness/generation in clustering optimization

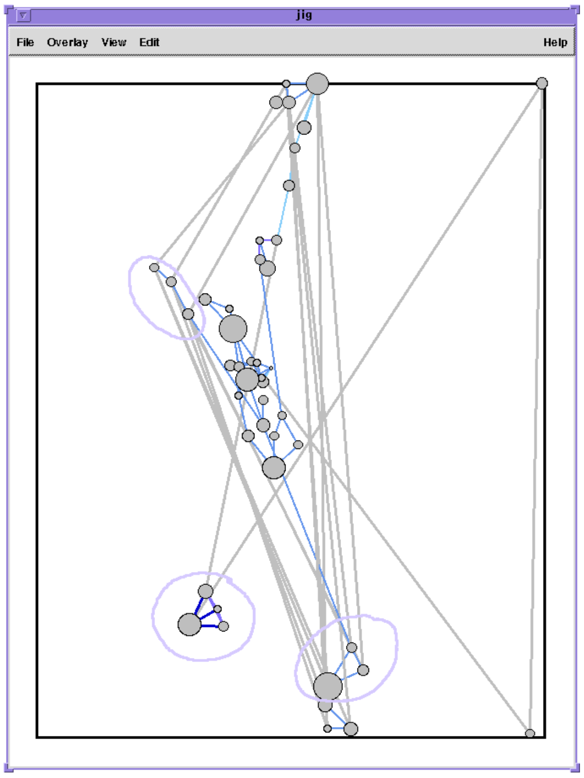


Figure 9.a Before cluster optimization

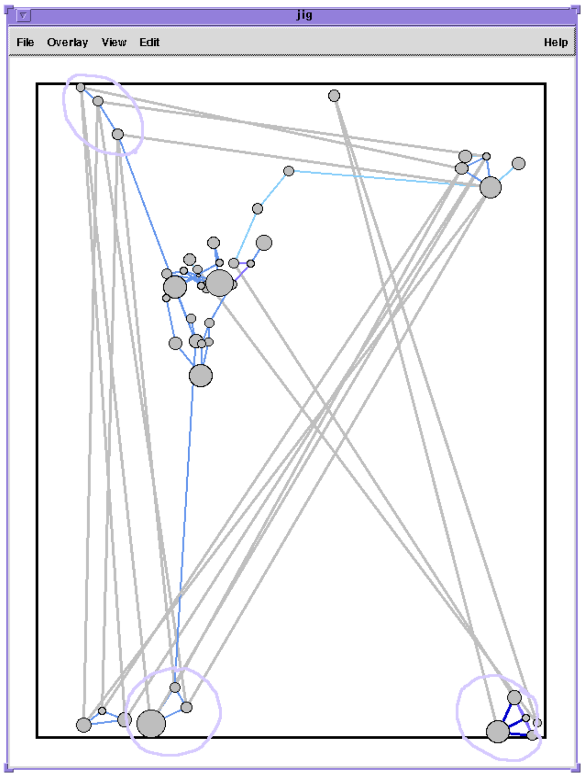


Figure 9.b After cluster optimization

7.0 CONCLUSION

The goal of this paper is to show step by step the importance of optimizing multiple aspects of a GA when applying it to a specific problem. The experiments performed demonstrate that dynamic adjustment of control parameters as well as genetic operators make difference to the quality of final result. In the meanwhile, the change in representations to a “clustered” form of lower dimensionality can have a significant positive effect on this difficult problem. Combined with the power of a multi-level GA (DAGA2), it appears that excellent, albeit not globally optimal, solutions can be obtained in a reasonable time to problems with search spaces on the order of 10^{240}. This result helps in understanding more about the role of dynamic representations together with dynamic GA operators/parameters in parallel subpopulations in attacking high-dimensionality problems.

8.0 REFERENCES

Cavicchio, Daniel J. (1970). *Adaptive Search Using Simulated Evolution*. Ph.D. Dissertation, Ann Arbor: University of Michigan.

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

Dexter, Terrence W., Goodman, Erik D. and Punch, W.F., III (1996) “*Local Optimization in Two-Dimensional Layout Problems*”, Technical Report, Case Center, Michigan State University.

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.

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.

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

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, *Proc. Fourth International Conference on Genetic Algorithms*, Morgan-Kaufman, New York, pp. 190-195.

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

Lin, S-C, Punch W.F. and Goodman, E.D. (1994). "*Coarse-Grain Genetic Algorithms: Categorization and New Approaches*", Sixth IEEE Parallel and Distributed Process, Oct 94, pg.28-37.

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.

Wang, Gang, Goodman, Erik D. and Punch, William F. (1996). "*Simultaneous Multi-Level Evolution*", GARAGe Technical Report, Department of Computer Science, Michigan State University.

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.