Applying a Mutation-Based Genetic Algorithm to Processor Configuration Problems

T L Lau and E P K Tsang
Dept. of Computer Science
University of Essex,
Wivenhoe Park
Colchester CO4 3SQ
United Kingdom
email: {tllau, edward}@essex.ac.uk

Abstract

The Processor Configuration Problem (PCP) is a Constraint Optimization Problem. The task is to link up a finite set of processors into a network, while minimizing the maximum distance between these processors. Since each processor has a limited number of communication channels, a carefully planned layout could minimize the overhead for message switching.

In this paper, we present a Genetic Algorithm (GA) approach to the PCP. Our technique uses a mutation based GA, a function that produces schemata by analyzing previous solutions, and an effective data representation. Our approach has been shown to out-perform other published techniques in this problem.

1. INTRODUCTION

1.1. Constraint Satisfaction Problem.

A Constraint Satisfaction Problem (CSP) can be stated as a problem with a finite set of variables; each associated with a finite domain. Amongst the variables, there exist constraining relationships that limits the possible values they can take at the same time. The challenge is to find solution tuples which map the variables onto values that are within their respective domain, and does not violate any of the constraints placed on them. Constraint Satisfaction Optimization Problems (CSOP) are a sub-class of CSPs and require that the solution found must be optimal [4, 9].

In this paper, we are particularly interested in optimizing one class of CSP where the solution tuples can be expressed as permutation of each other. We chose the *Processor Configuration Problem* for this reason and because of its applicability (see section 1.3).

1.2. Genetic Algorithm and CSP

CSPs are NP-complete in general and although heuristics have been found useful in solving them, complete search algorithms are limited by the combinatorial explosion problem. Tsang argues in [5] to use stochastic methods. The project presented in this paper follows the footstep of those who have used GA to tackle CSOPs [2, 3, 8, 10, 11].

In this paper, we wish to demonstrate that GA can be an effective method to solve the stated class of CSOPs. One novel idea in our approach is to run the GA twice, with the second run based on a schema observed from the best string generated in the first run. This schema shall contain information that fixes crucial data points, ensuring that evolution takes place on other parts of the string, thus compacting the search space upon passing it through the GA again.

1.3. Processor Configuration Problem

By breaking down a complex problem and solving the parts concurrently, one can potentially reach the solution in a much shorter time. A distributed memory multiprocessor system offers such an environment and consists of many processors working in parallel, each with its own memory and set of instructions. These processors work in tandem to solve a complex problem, sharing information where needed by exchanging messages.

The Processor Configuration Problem (PCP) is a real world problem about linking up a finite set of processors into a network, where each processor has a finite number of external communication channels [1]. The PCP was formally defined at the University of Bristol, where members of its research staff regularly construct processor configurations to interconnect networks of transputers for the solution of engineering problems.

PCP plays a significant role in the design of an effective system as the performance of a multiprocessor network depends largely on the efficiency of the message transfer system, which coordinates the shuttling of message packets between processors.

Processors not directly connected to each other will have to go through intermediate processors to talk to its target. When the number of processors in the network increases, the average number of intermediate processors a message will have to pass through to get to its destination would similarly increase. Thus, careful planning on the layout of the processors will reduce the minimum distance between any two processors.

To understand the terms and convention used in this paper, a little introduction is in order:

A *link* is a connection between two processors, and a *path* is made up of one or several such links. The *optimal path* between any two processors is the shortest path between them. A message traveling from one processor to another could have at its option several optimal paths to chose from, each equally short and therefore, equally desirable. Another important term is *diameter*, which is the distance between two processor and is used in conjunction to state the nodes falling within the range of the referenced processor. Note distances between a source and destination are measured in number of processors. To get a better idea of these terms, examine the next diagram.

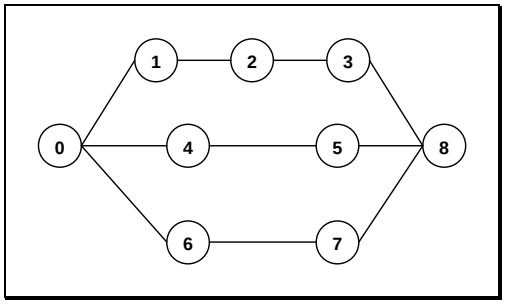

Figure 1. Sample graph showing paths from node 0 to node 8

In the diagram above, 1 - 2, 4 - 5 and 6 - 7 are examples of links. Links are all bi-directional. Assuming that we are passing a message from node 0 to node 8, the paths available are 0 - 1 - 2 - 3 - 8, 0 - 4 - 5 - 8 and 0 - 6 - 7 - 8. However, of the three paths, only 0 - 4 - 5 - 8 and 0 - 6 - 7 - 8 are the shortest and so they are optimal paths from processor 0 to processor 8.

Next the two diagrams below are sample layouts of five processors networks. In both layouts, each processor has a total of four communication channels.

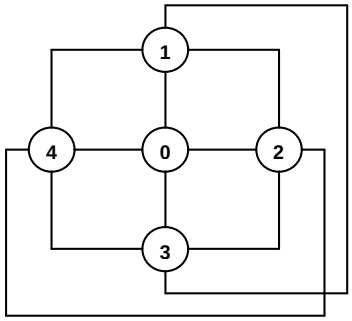

Figure 2. Processor Configuration Without System Controller

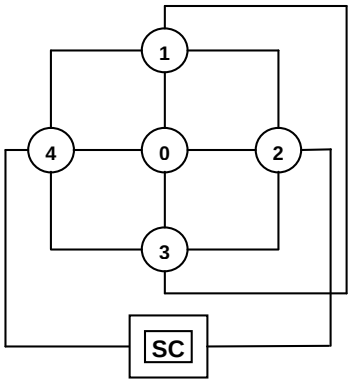

Figure 3. Processor Configuration With System Controller (SC)

In figure 2, the distance between any two processor is one. But in figure 3, we are considering the additional requirement that two communication channels are reserved (each channel belongs to different processors) to interface with the system controller. The network in figure 2 is called a regular network as all processors have the same number of communication channels, and all these channels are connected to other processors. However in figure 3, not all the communication channels are connected to other processors and so the network is called

irregular. Networks that interface with a system controller are the subject of our study in this paper.

The network of processors can be thought of as a graph (from extremal graph theory), with the processors being the nodes and the communication channels assuming the role of arcs [1]. Making use of performance measures from graph theory, one can determine how good a given network configuration is. For the PCP domain, a good measure of quality would be the mean inter-node distance, as defined next.

$$d_{avg} = \frac{\sum_{p=1}^{N} \sum_{d=1}^{d_{\max}} (d \cdot N_{pd})}{N^2} \qquad (1)$$

Where N is the number of nodes and N_{pd} is the number of processors at distance d away from node p (in terms of number of intermediate nodes), and d_{max} represents the maximum range to calculate for (user specified).

Another measurement of quality would be the number of processor pairs whose shortest path exceeds the d_{max}. The equation for measurement N_{over} is defined as follows:

$$N_{over} = \sum_{p=1}^{N} \sum_{d=d_{\max}+1} N_{pd} \qquad (2)$$

2. LAYOUT REPRESENTATION

The general idea is to design the layout for a regular network, and then transform it to the requirement we want (more on that later). To understand the symmetrical property of regular networks, consider a three processors network, each with four communication channels:

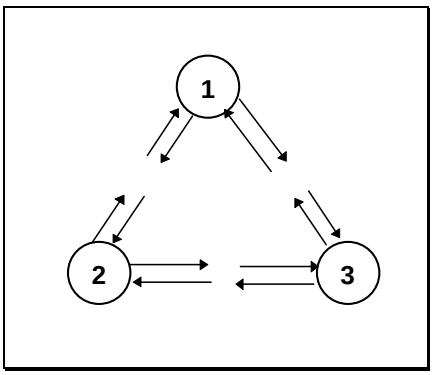

Figure 4. A Sample Network

This could have been easily represented as:

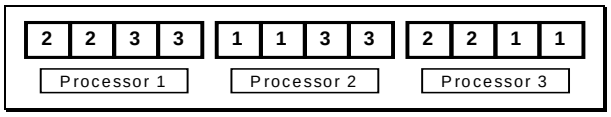

Figure 5. A Direct Representation

Where each box represents a communication channel and each processor has four boxes. The value in each box indicates the processor it is currently connected to. However if we were to make use of the property of symmetry, we could express the network as:

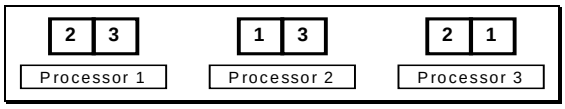

Figure 6. Using Symmetrical Representation

3. LPGA: A MUTATION-BASED GA

3.1. An Overview of the Algorithm

The chief components of this algorithm, called *Lower Power Genetic Algorithm* (LPGA) is its GA engine and the analysis module. Figure 7 shows the organization of this algorithm and how it operates.

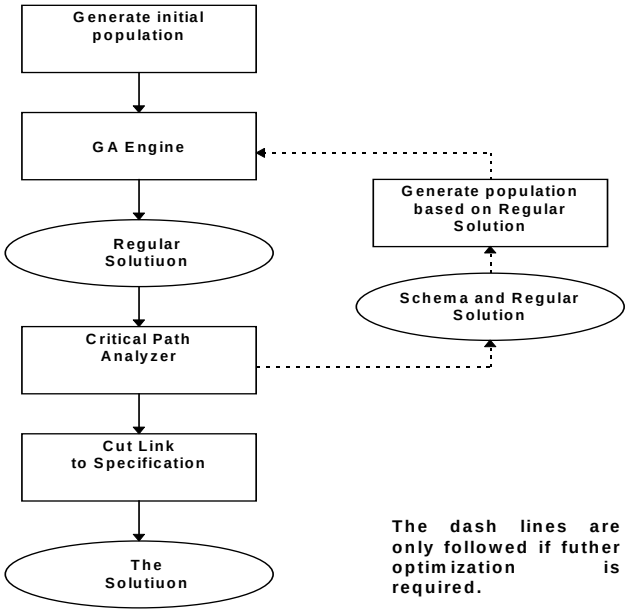

Figure 7. Overview of LPGA

LPGA has two modes of operation: with or without second phase optimization. Without second phase optimization, the program would terminate after stepping through a pre-determined number of generations. But with it, the GA engine is called twice. The first time the GA engine is called (the *first phase*), it produces a solution at the end of a series of evolution, as per normal. This solution string is then analyzed by the Critical Path Analysis module which provides information to guide the production of a schema, which protects links/paths with higher hit rates than the rest. The schema, along with the solution from the first phase, is fed back to the GA engine (the *second phase*) in hopes of reaching better solutions.

The Regular Solution contains the configuration for a network without system controller. The Solution is produced only after removing a link from the regular network configuration. The selection of the link to be removed is also based on the information provided by the Critical Path Analysis module.

3.2. GA Engine

Under the context of our data representation, the standard cross-over operator may not perform efficiently, and so we propose a mutation only approach. For example, with the cross-over operator, unless the positions of the values between the two mating string are aligned correctly, it would otherwise create invalid results (observe the next diagram).

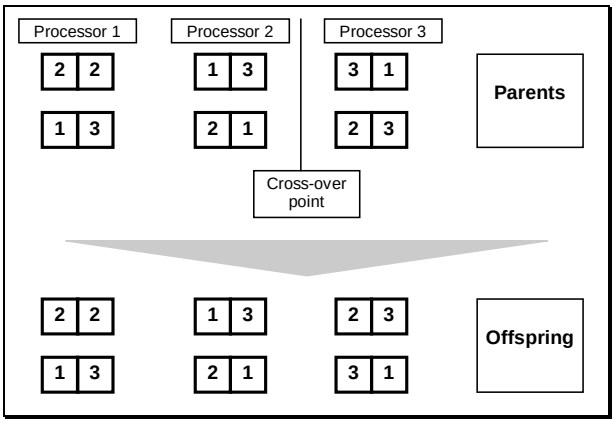

Figure 8. Cross-over Operator

In figure 6, notice that the offspring are unbalanced. For example, the string { 2, 2, 1, 3, 2, 3 } has one reference to processor 1, two references to processor 3 and three references to processor 2. For this representation, a processor must be referenced exactly twice and so some form of repairing mechanism is required. Since the addition of a cross-over operator and a repairing mechanism would require significant computation, so in the interest of performance and simplicity, we designed a mutation based GA engine.

In our implementation, the mutation operator randomly selects two positions in the string and swaps its contents to produce an offspring. The mutation operator will note that the values to be transferred does not end up in variables they are pointing to, e.g. the value 1 must not appear in variable group and so all the permutations produced are valid. Besides this restriction, the positions selected are compared against a schema to see if they are write-protected. If either one of the positions is write-protected, the mutation operator will choose another set of positions to swap and will do this until the chosen set does not violate any constraint.

An effective feature of GAs is that the selection of chromosomes to mate are proportional to their fitness. So fitter chromosomes are given higher chances to mate and in a way, to produce more of its kind. However this is only applicable provided the cross-over operator is intact. Therefore in our case, an alternative that closely mirrors this process was created: the number of offspring a parent can produce is proportional to its fitness. Finally, all the components mentioned in this section follows the control flow as defined in figure 9.

Before each generation begins, the *production table* is constructed. This table contains the maximum number of offspring a parent of a particular fitness may produce. It is built in the following way:

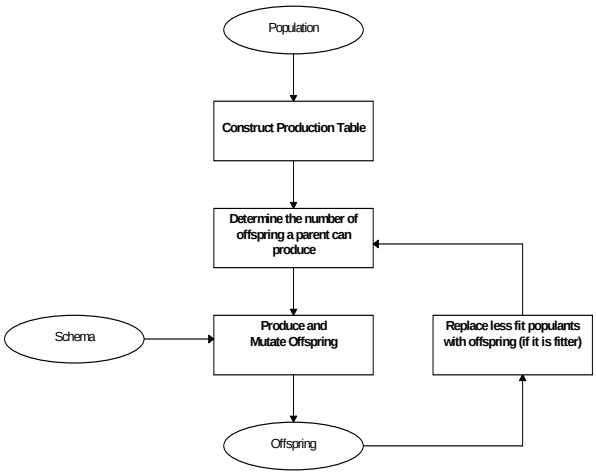

Figure 9. GA Engine

As a parameter to the GA, the user specifies *k* equals to the maximum number of offspring that any parent can produce. The algorithm first orders the strings in descending order of fitness, breaking ties randomly. The ordered population is then divided into *k* equal groups, each allocated a maximum number of offspring from *k* to

1, that it can produce. This information is recorded in the production table.

A string drawn from the population of the current generation is first given the number of offspring it can produce, which is a random number between one and the allocated size as stated in the production table (referenced by the chromosome's fitness value). A copy of the string is mutated upon to produce an offspring and the parts within the string that mutation can occur is described in the schema. During the first phase (see section 3.1 - An Overview of the Algorithm) the entire schema is set to write-enabled; i.e., all positions in the string are susceptible to mutation. But for the second phase, we want to protect parts of the best solution (that was produced in the first phase) that were important to it, and so these parts are indicated on the schema. The offspring created is put into the population and when all the strings from the current population have being processed, the inhabitants of the pool are reassessed and the fitter members are retained for the next generation.

3.3. Critical Path Analysis

The Critical Path Analysis (CPA) is a heuristic analysis of the utility of each link in a network. The aim is to produce a *hit rate table*, which is a collection of links used in the network, plus the estimated frequency of their use. The hit rate table is created in the following way:

> For each pair of processors *i* and *j*, find the set of all paths between them which have the minimum number of intermediate processors, call this set S_{ij}. For each link between two processors, count the number of paths in all S_{ij} which contains this link. We call these counts the *count table*.
>
> Create a hit rate table in roughly the same way as the count table, except that only one path is selected in each S_{ij}, as described:
>
> First, order the sets S_{ij} in ascending order of their cardinality. Process one S_{ij} at a time according to this ordering. If S_{ij} has one path only, then all links used by this path will have their counts increased by one in the hit rate table. If more than one path exists, then choose the path which contains a link (among all the links in S_{ij}) that has the greatest count in the count table. Update the hit rate table accordingly, using the selected path.

With the hit rate table, a preset percentage (called retention percentage, which a user specified parameter) determines the number of the popular links to keep and these popular links are then encoded as a schema, i.e. the positions of these variables that constitute the links will be write-protected.

3.4. Cut Link

To convert from a regular network to one which meets the specification set out in section 1.3, all one needs to do is cut a link. By cutting this link, we would get two communication channels ready to be linked to the system controller.

Choosing the link is an important task because after a link is removed, the distance between certain processors could be increased. The goal is to cut a link which does not substantially increase both N_{over} and d_{avg}. This module uses the information provided by the CPA module to guide its decision. It looks for the least popular link in the hit rate table and removes it.

4. BENCHMARKING ENVIRONMENT

We call the above algorithm Low Power Genetic Algorithm (LPGA) for it's simplicity and low computational costs. It was implemented in ANSI C for portability. All results reported in this paper ware benchmarked on a Sun 3/50 with SunOS 4.1.

The performance of an algorithm is judged by the time required to reach a solution and the quality of that solution, which is given by the fitness function, d_{avg}. The benchmark suite used here was based on [1, 10, 11], so as to provide a comparison. The environment is as follows: PCP suite with 32 to 40 processors and a d_{max} value of 3.

Each recorded result (for d_{avg} and CPU time) is the average of a trial of five runs. Except for the numbers in section 4.2, which are the least number of paths which exceeded the maximum diameter of three that was *experienced so far*.

For this series of trials, each run starts with a population size of 20 strings, and for each parent string, a maximum of three offspring are produced. Each call to the GA engine will terminate after 200 generations and thus, in the case where the second phase optimization was requested, LPGA will submit its solution after 400 generations. A retention percentage of 25% was used to produce a schema for second phase optimization.

The two competing solvers used in the benchmark are AMP [1] and GAcSP [10, 11]. AMP is an algorithm that uses depth-first search with hill-climbing. Note that for AMP, no runtime results were available and only results for 32 and 40 processor configurations were reported. GAcSP is based on the standard GA with the addition of a repairer and an optional hill-climber.

4.1. Comparing Quality of Results I : d_{avg}

No. of processors	AMP	GAcSP	GAcSP (HC)	LPGA	LPGA (CPA)
32	2.31	2.33	2.29	**2.30**	**2.29**
33	N.A.	2.37	2.32	**2.33**	**2.31**
34	N.A.	2.40	2.34	**2.34**	**2.34**
35	N.A.	2.42	2.37	**2.37**	**2.35**
36	N.A.	2.42	2.38	**2.42**	**2.39**
37	N.A.	2.45	2.41	**2.43**	**2.41**
38	N.A.	2.47	2.43	**2.46**	**2.42**
39	N.A.	2.51	2.45	**2.48**	**2.43**
40	2.53	2.51	2.47	**2.49**	**2.43**

Table 1. *davg* produced by LPGA and other algorithm (dmax = 3)

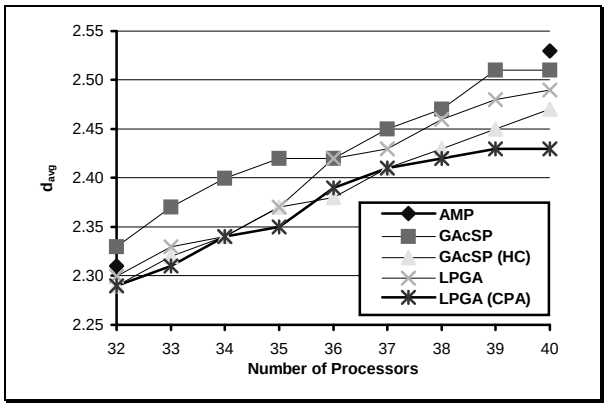

Chart 1. Comparing *davg* produced by LPGA and other algorithm

The d_{avg} Comparison chart shows that LPGA compares favorably with GAcSP with Hill Climbing, which produced the best results reported so far. As LPGA (with Critical Path Analysis) performs the best of the lot with consistently the lowest results.

4.2. Comparing Quality of Results II : N_{over}

No. of processors	AMP	GAcSP	GAcSP (HC)	LPGA	LPGA (CPA)
32	N.A.	12	1	**10**	**1**
33	N.A.	15	4	**10**	**3**
34	N.A.	24	3	**12**	**4**
35	N.A.	33	11	**19**	**9**
36	N.A.	31	8	**25**	**14**
37	N.A.	47	18	**30**	**16**
38	N.A.	49	19	**34**	**22**
39	N.A.	65	25	**44**	**28**
40	N.A.	62	34	**50**	**29**

Table 2. *Nover* produced by LPGA and other algorithm (dmax = 3)

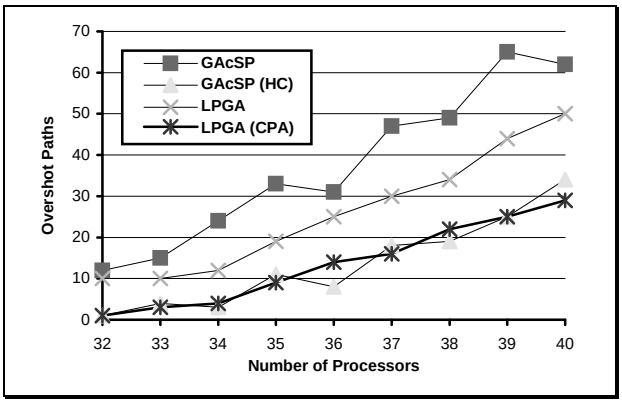

Chart 2. Comparing *Nover* produced by LPGA and other algorithm

Another indication of the quality of a solution is the number of paths which is longer than the user set diameter. In our test cases, the value of three is used and therefore, any processor-to-processor pair with path of more than three nodes is considered an "overshot". Thus, an optimal solution would be one with the least number of "overshots", i.e. the smallest N_{over}. LPGA's performance in this area was better than GAcSP but losses out to GAcSP (with Hill Climbing). However, with the Critical Path Analysis module activated, LPGA returns the best results.

4.3. Comparing Speed

No. of processors	AMP	GAcSP	GAcSP (HC)	LPGA	LPGA (CPA)
32	N.A.	1160	9103	**813**	**1652**
33	N.A.	1101	4784	**852**	**1738**
34	N.A.	1018	5979	**875**	**1903**
35	N.A.	1656	6046	**908**	**2099**
36	N.A.	2095	10135	**942**	**2159**
37	N.A.	2488	7682	**995**	**2214**
38	N.A.	5032	10304	**1033**	**2271**
39	N.A.	3240	11925	**1087**	**2356**
40	N.A.	5432	14908	**1131**	**2420**

Table 3. Execution time (all times in CPU seconds)

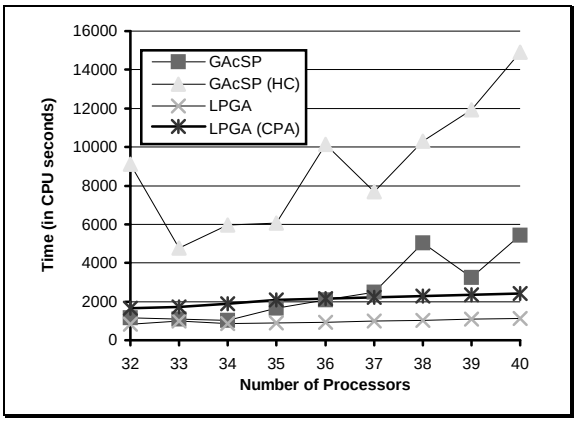

Chart 3. Comparing Execution Time

In terms of timing, LPGA out performs both flavors of GAcSP. With the solution optimization module turned on, the results of LPGA are roughly twice without it. This performance (in time) halving is expected, since LPGA with Critical Path Analysis was engineered to use the same GA engine and that engine is called twice instead.

5. CONCLUSION

In this paper, we have presented a novel GA algorithm called LPGA for tackling PCPs. We have demonstrated that LPGA has produced better results than those published so far, in both speed and quality. The success of this work is mainly due to the use of an effective data representation and the CPA module that produces good schemata to guide the GA engine onto better solutions.

Performance in terms of turnaround time is heavily tied to the elegance and simplicity of the data representation. The data structure used in this case made use of the symmetrical property of the Processor Configuration Problem and thus, all candidate solutions produced by the solver obey the constraints of the problem. As such, there is no need for an elaborate checking/repairing mechanism to service the solutions and therefore, the resulting algorithm had a quicker response time. Further, the employment of this representation made the cross-over operator inefficient, so the GA engine implemented in this project relies purely on the mutation operator to vary the parent's offspring.

If the user can afford to spend time in seek of better solutions, LPGA can be executed with second phase optimization. Where this additional optimization is required, the GA engine is guided by the schema constructed based on information produced by the CPA module.

When compared with the other methods stated in this paper, LPGA outshines both AMP and GAcSP in terms of speed and quality. AMP is mainly a complete search method with generating a valid configuration as its primary goal, whereas LPGA is a stochastic method with quality of answer as its motivating factor. LPGA's advantage over GAcSP lies in the more specific and efficient data structure used, and in LPGA's use of a graph analyzing technique to guide the construction of better solutions.

REFERENCES

[1] Chalmers A. and Gregory S. (1992) ***Constructing Minimum Path Configurations for Multiprocessor Systems.*** Technical Report CSTR-92-12, April 1992, University of Bristol, Computer Science Department.

[2] Eiben A., Raué P-E. and Ruttkay Zs. (1993) *Heuristic* ***Genetric Algorithms for Constrainted Problems.*** Proceedings of Dutch National AI Conference, NAIC '93, pages 241 - 252.

[3] Eiben A., Raué P-E. and Ruttkay Zs. (1994) ***GA-easy and GA-hard Constraint Satisfaction Problems.*** Vrije Universiteit Amsterdam, Artificial Intelligence Group, Department of Mathematics and Computer Science.

[4] Freuder E. and Mackworth A. (Editor) (1994). ***Constraints-based Reasoning***. MIT Press.

[5] Freuder E. (Chair), Dechter R., Selman B., Ginsberg M. and Tsang E. (1995). ***Systematic Versus Stochastic Constraint Satisfaction (Panel Paper).*** 14th International Joint Conference on Artificial Intelligence.

[6] Goldberg D. (1989) ***Genetic Algorithm in Search, Optimization and Machine Learning.*** Addison-Wesley Pub. Co., Inc.

[7] Holland J. (1975) ***Adaptation in Natural and Artificial Systems.*** The University of Michigan Press.

[8] Tsang E. and Warwick T. (1989) ***Applying Genetic Algorithms to Constraint Satisfaction Optimisation Problems.*** Proceedings of the European Conference on

Artificial Intelligence. Stockholm, Sweden.

[9] Tsang E. (1993) ***Foundations of Constraint Satisfaction.*** Academic Press Limited.

[10] Warwick T. and Tsang E. (1993) ***Using a Genetic Algorithm to tackle the Processor Configuration Problem.*** Symposium on Applied Computing, 1994, pages 217-221

[11] Warwick T. (1995) ***A GA approach to Constraint Satisfaction Problems.*** Ph.D. thesis, to appear. University of Essex, Department of Computer Science.