ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF ELECTRONICS AND COMPUTERS

AUTOMATION AND ROBOTICS LAB.

TECHNICAL REPORT

Review of Parallel Genetic Algorithms Bibliography

Version 1
November 1994

Panagiotis Adamidis
adamidis@eng.auth.gr

Thessaloniki, Greece

TABLE OF CONTENTS

1. INTRODUCTION ***1***

2. THE STANDARD PARALLEL APPROACH ***3***

3. THE DECOMPOSITION APPROACH ***4***

3.1. COARSE-GRAINED PARALLEL GENETIC ALGORITHMS 4

i. Island Model *4*

ii. Stepping-stone model *5*

iii. Other Coarse-Grained models *8*

3.2. FINE-GRAINED PARALLEL GENETIC ALGORITHMS 11

3.3. OTHER PARALLEL GENETIC ALGORITHMS 15

REFERENCES ***18***

Review of Parallel Genetic Algorithms Bibliography

Panagiotis Adamidis
Dept. of Electrical & Computer Engineering,
Aristotle University of Thessaloniki, Greece
Email: adamidis@eng.auth.gr

Version 1 - November 1994

Abstract. The availability of faster and cheaper parallel computers makes it possible to apply genetic algorithms to large populations and very complex applications. This report presents a survey of current implementation techniques for genetic algorithms on parallel hardware.

1. Introduction

Genetic Algorithms (GA) are stochastic search and optimization techniques which were inspired by the analogy of evolution and population genetics. They have been demonstrated to be effective and robust in searching very large, varied, spaces in a wide range of applications.

During recent years the area of Genetic Algorithms in general and the field of Parallel Genetic Algorithms (PGA) in particular has matured up to a point, where the application to a complex real-world problem in some applied science is now potentially feasible and to the benefit of both fields.

The effectiveness of GAs, is limited by their ability to balance the need for a diverse set of sampling points with the desire to quickly focus search upon potential solutions. Due to increasing demands such as searching large search spaces with costly evaluation functions and using large population sizes, there is an ever growing need for fast implementations to allow quick and flexible experimentation. Most GAs work with one large panmictic population. Those GAs suffer from the problem that natural selection relies on the fitness distribution over the whole population. Parallel processing is the natural route to explore. Furthermore, some of the difficulties that face standard GAs (such as premature convergence, and searching multimodal spaces) may be less of a problem for parallel variants.

It is important to distinguish between two approaches to PGAs. The **standard parallel** approach, using the PGA as a means of implementing a (sequential or parallel) GA, and the **decomposition approach** with the PGA as a particular model of a GA [Levi94], [TaBe91].

In the first approach, the sequential GA model is implemented on a parallel computer. A simple way to do this is to parallelize the loop that creates the next generation from the previous one. Most of the steps in this loop (evaluation, crossover, mutation, and, if used, local search) can be executed in parallel. The selection step, depending on the selection algorithm and the problem solved, usually requires a global ranking that can be a parallel bottleneck. Using a distributed memory computer, the communication overhead associated with distributing data structures to processors, and synchronizing and collecting the results, grows as the square of population's size. This can minimize any performance improvements due to multiple processors, unless function evaluation (or local search) is a time-consuming process. This type of parallelism can be efficiently implemented on shared-memory machines.

In a PGA model, the full population exists in distributed form; either multiple independent or interacting subpopulations exist, or there is one population but each population member interacts only with a limited set of neighbors. Generally, the processors send each other their best solutions. These communications take place with respect to a spatial structure of the population. According to their spatial structure, the granularity of the distributed population and the manner in which the GA operators are applied they can be classified in three(3) models.

In a coarse-grained PGA the population is divided into several equal subpopulations, each of which runs a GA independently and in parallel on its own subpopulation. Occasionally, fit individuals migrate randomly from one subpopulation to another (***island model***). In some implementations migrant individuals may move only to geographically nearby subpopulations (islands), rather than to any arbitrary subpopulation (***stepping-stone model***).

In a fine-grained PGA a single population is divided so that a single individual is assigned to each processor. Processors select from, crossover with, and replace only individuals in a bounded region (neighborhood/deme). Since neighborhoods overlap, fit individuals will propagate through the whole population (***Diffusion or isolation-by-distance or neighborhood model***).

These models maintain more diverse subpopulations mitigating the problem of premature convergence. They also naturally fit the model of the way evolution is viewed as occurring, with a large degree of independence in the global population. Parallel GAs based on Subpopulation Modelling can even be considered as creating new paradigms within this area and thus establishing a new and promising field of research.

2. The Standard Parallel Approach

A single master processor supervises the total population and does the selection. Slave-processors receive individuals which are recombined to create offsprings. These offsprings have to be evaluated before they are returned to the master.

Abramson [Abra92], used a 16-processor Encore Multimax shared memory multiprocessor for the concurrent creation of a new population, applied to the school time tabling problem. Instead of one sequential piece of code creating all the children in the new population, a number of agents is responsible for mating. Each agent selects two parents, and produces a fixed number of the children. It is important to minimize interprocess synchronization to maximize speed-up. Speed-up is less than optimal, because a few sections have not been parallelized. The peak speed-up was 9.2 using ten processors with 200 individuals, and 9.3 with fifteen processors and 120 individuals.

The previous technique was also used in [AbMP92] to compute efficient train schedules. They used a sixteen processor Encore Multimax, a 128 processor Fujitsu AP1000 and a network of DEC work stations. The shared memory version (Encore) and the distributed (Fujitsu) performed very well up to sixteen processors (speed-up: 14/13.5). The Fujitsu AP1000 version, continued to show speed-up up to 128 processors, but was well below ideal at 128 processors (48.1). The DEC workstation speed-up (using PVM) were very slow, showing almost no speed-up at two and four processors, due to enormous overheads incurred when ethernet and UNIX TCP/IP are used as the communication protocol.

Beyer [Beye93], implemented the evolution strategy (ES) on the KSR1 for solving large-scale order problems like the optimal arrangement problem for linear accelerators (LINAC) that minimizes the multibunch-beam-breakup (BBU) and the TSP problem. He applied the master-worker paradigm. The master is responsible for the set up of the workers, the user interface, fitness sorting (and selection) and bookkeeping operations. The master reserves one processor and creates (π-1) pthreads running the worker code. Each worker performs, after start up, the selection, mutation and the fitness evaluation. He didn't use any parallel constructs (mutexes or barriers). The ES exhibits a linear speed up as long as $\pi <$ No of offsprings (or to be more precise, if $k(\pi-1)=\lambda$).

Fogarty and Huang [FoHu90], used a transputer array for the parallel evaluation of a population of 250 individuals applied to the problem of evolving a set of rules to balance a pole on a moving chart. For this problem, evaluating the fitness of a member of the population takes a relatively long time. A host processor ran the main GA program and distributed individuals for evaluation to the other transputers for evaluation. They tested three different topologies: i) Pipeline network, ii) bridged network, and iii) two-dimensional (2-D) array network. Parallel evaluation based on T8 transputers is much faster than the sequential evaluation time on T8. The time on T8 transputers is on average about 15 times faster than

using T4 transputers. The 2-D array network is relatively better than the others when the number of the transputers is increased. A bridged network is slightly better than a pipeline network. The relative increase in performance dropped as the number of transputers grew. This means that communication overheads versus the number of transputers used in the network, increases. The improvement in speedup was slightly sublinear with up to 16-20 processors, but then fell off quickly. Maximum speedup in the range of 25-27 were obtained on 40-72 processors.

3. The Decomposition Approach

3.1. Coarse-Grained Parallel Genetic Algorithms *(Migration model)*

In a coarse-grained PGA, multiple processors run a sequential GA on their population. Processors exchange individuals from their subpopulation with other processors. Some important choices are:

- which other processors a processor exchanges individuals with
- how often processors exchange individuals (migration interval or frequency)
- how many individuals processors exchange with each other (migration rate)
- what strategy is used in selecting individuals

Isolated subpopulations help maintain genetic diversity. Individuals in a subpopulation are relatively isolated from individuals on another subpopulation. Therefore the sub-population of each island is exploring a different part of the input space.

i. Island Model

Levine [Levi94], used the IBM SP1 parallel computer with 128 nodes (IBM RS/6000 Model 370 workstation processor) to solve the Set Partitioning Problem(SPP). Each processor run a steady state GA (SSGA) combined with a heuristic specialized for the SPP. He choose SSGA since it proved more successful, particularly at finding feasible solutions, than generational replacement GA. The migration rate was only one individual. A 2-D toroidal mesh was used. Each processor exchanged individuals with its four neighbors, alternating between them each migration generation.

He tested two methods to choose the individual to migrate: the fittest individual was sent to a neighbor, or the individual was selected via a probabilistic binary tournament with

parameter p_b=0.6. The choice between them is not significant as long as above average fitness individuals are being migrated, although the first method was a little faster at finding an optimal solution. He also tested two strategies for determining the string to delete. The first was to delete the least fit string in the subpopulation, and the other to hold a probabilistic binary tournament with parameter p_b=0.4 and delete the winner. The tournament strategy was significantly slower but it was more successful at finding the optimal solution. On more difficult problems, deleting the worse may be better. The test results from the frequency of migration were ambiguous. He choose the best individual to migrate every 1000 iterations. Each problem was run once using power of two subpopulations with size 100. The model solved real-world SPP problems of up to a few thousand integer variables. For problems with many constraints, a feasible solution was never able to find. Adding subpopulations was beneficial. Optimal solutions were usually found on early generations. Branch-and-cut algorithms had better results, while comparing with branch-and-bound the results were mixed.

Norman [Norm88], applied a PGA to the problem of topology optimization. Individuals can migrate asynchronously between subpopulations. The frequency of migration is variable. Migrations may occur at random between random processors. Fitter individuals have a greater chance of migrating between processors. Individuals are accepted if they are fitter than the fittest individual in the population. If they are as fit as the least fit individual they are accepted with probability p0. If their fitness is between they are accepted with probability p1 (p1: linear interpolation between p0 and 1). Better solutions were found, and the optimization was faster compared with a single GA.

Pettey et al. [PeLG87][PeLe89], implemented a PGA on an Intel iPSC hypercube (eight processors), and test it on De Jong's functions. Each generation, each processor sent its best individuals to each neighbor and received its neighbors' best individuals. The connections between processors, were changing thus creating randomly communicating subpopulations. Received individuals are either accepted and randomly placed in the population, or population is expanded and from the total population a number of individuals equal to the received, are randomly chosen and thrown out. The subpopulation size was fixed at fifty individuals. Their results indicate an increased likelihood of premature convergence. This is because individuals are exchanged every generation and always with the same neighbors. The results also indicate that the use of additional processors did improve convergence speed, but not the solution quality.

ii. Stepping-stone model

Cohoon et al. [CoMR90][CoMR91], used a PGA to study a VLSI application, namely the K-partition problem, on a 16-processor INTEL i860 hypercube. Each processor had its own

subpopulation of 80 individuals. After each ***epoch***, each processor copied randomly selected individuals of its population to neighboring processors. Each processor selected probabilistically (with respect to fitness value) a set of individuals to survive to be its initial subpopulation at the beginning of the next epoch. They run the PGA for 30 epochs with migration rate of nine individuals, and migration interval of 50 generations. They used a penalty function with random choice of scaling coefficient in their fitness function $c(x)+\lambda p(x)$. the scaling factor λ influences how much weight infeasibilities have in evaluating an individual's fitness. Two experiments were done. One used $\lambda=1$ for each sub-population. In the other, each processor chose a value for λ uniformly on the interval (0,1]. The runs with uniformly distributed λ were consistently better than those with λ fixed at 1. Their PGA found the best score overall and did better in its average.

Kroger et al. [KrSV90][KrSV93], used a PGA on a network of 32 transputers to solve the 2-D bin packing problem. At "irregular intervals" a processor received individuals from neighboring processors enlarging its population by Π_{ext}. A parallel elitist strategy was used whereby, whenever a processor improved upon the best individual in its own population, it send a copy of it to all other processors in its neighborhood. The best results were found with a "medium size neighborhood" ($\Pi_{ext}=10$), and a local population of ten individuals. A couple is selected for mating from the set $\Pi=\Pi_{int}+\Pi_{ext}$. An offspring is inserted in Π_{int}, either if it is the local best, or if it is randomly inserted. Solutions found with 64 or 128 processors are only modestly improved.

In [KrSV93] they presented a model called "emigration model", where copies of individuals are transmitted to exactly one neighbor. The received individuals are treated like locally generated offsprings. They used 48 processors. Best results yields the lowest migration interval (five generations) and three individuals as migration rate. The solution is improved increasing the number of processors. For the Diffusion model, the performance is influenced by the cardinality of the neighborhood and the size of each copied population. The emigration algorithm appears to be superior to the other two, and the diffusion approach seems to be superior to the immigration model.

McClurkin et al. [McGD90], compared this model, on different topologies, with the sequential GA and a flood-filled configuration, using Ackley's "porcupine" function. They run the stepping-stone model with the topologies: linear array, ring and mesh connection topologies. The flood-filled configuration is a simple master-worker process where the master processor sends out the parents to be processed and collects the offspring from the worker tasks. Only master holds information about the population. The times for each PGA topology are almost identical. It was found that an extra processor gave at best 30% reduction in execution time. In the flood-filled configuration, the maximum speedup gained was just 20%. The PGA models tend to converge much more quickly than the sequential GA. The more

interconnected the network, the more quicker the algorithm converges. However this may have the drawback, that it may converge too quickly to a point close to but not the optimal.

Muhlenbein et al, applied a PGA to the optimization of continuous functions [MuSB91a] [MuSB91b]. The tested functions are: the five De Jong's functions, the highly multimodal Rastrigin's function(F6), and two other functions proposed by Schwefel(F7) and Griewangk(F8). Sub-populations are located in a circular ladder-like 2-D structure. Every k generations, the best individual of a population is send to its neighbours. If the subpopulation does not improve for a number of generations one individual attempted to improve itself by applying a local search heuristic(hill-climbing), . Elitism is also used. The non-optimized PGA for F1 and F2 needs less function evaluations than optimized panmictic GAs. The discontinuous F3 is solved better by standard GA, indicating that maybe PGA is tuned more to continuous functions than to discontinuous. Results for F4 are in the same order. F5 is solved better by the PGA. For F6, the PGA found the optimum in all runs performing better than previous experiments, and for F7 the global optimum was not found in 4 out of 50 runs. They conclude that parallel search pays off only if the search space is large and complex, and that it is able to solve complex problems that they have not been previously solved.

Starkweather, et al. [StWM90], described a PGA where each processor sent copies of its best individuals to one of its neighbors, which replaced its worst strings with these. A ring topology was used where, on iteration one, processor p0 sends to p1, p1 to p2 etc., and on iteration two, p0 sends to p2, p1 to p3 etc. All sends were done in parallel. In their tests the total population size was fixed, and they experimented with various-sized partitions of the total population among the processors. Using no mutation, performance improved for two of the four problems as the number of subpopulations was increased, but degraded on the other two. Using adaptive mutation, with the mutation probability increasing as the similarity of the two parents was increased, the runs were more successful and achieved good results relative to the serial GA. The more distributed the GA, the more often adaptive mutation was invoked, since smaller populations converge more rapidly than larger ones. Also migrating individuals too often, or not often enough, degrade performance. In [WhSB90] they used the PGA to optimize connections and architecture of neural networks. In each swap five of the best individuals migrated to neighboring subpopulations. Swapping took place after 5X to 10X recombinations, where X is the size of the subpopulation. Using ten populations of 500 individuals each, and adaptive mutation, they were able to solve the fully connected 2-bit adder problem on 28 out of 30 runs (93%). The success rate of the serial search is 56%, and the convergence rate for Back-Propagation is 90%.

Tanese [Tane87][Tane89], applied a PGA to the optimization of Walsh-like functions using a 64-processor Ncube computer. Periodically, fit strings were selected and send to neighboring processors for possible inclusion in their future generations. Exchanges over neighboring processors varied over time, taking place over a different dimension of the

hypercube each time.. In [Tane87] he made some experiments, where different mutation and crossover rates were used. The PGA was able to determine the global maximum as often as the sequential GA. He reported near-linear speedup for runs of 1000 generations, and eight as the optimal number of subpopulations.

In [Tane89], he experimented with two implementations. The first is the *partitioned* GA in which there is no migration and the second *distributed* GA where migration is used. In the first case a total population size of 256, was partitioned into various power-of-two subpopulation sizes. The partitioned GA performed as well or better with respect to the best fitness value, than the traditional GA, even with small subpopulation sizes such as 8 or 4. However, with respect to the average fitness of the final generation, it was consistently worse than the traditional GA. In the second case, each processor reproduced extra offsprings during a migration generation, and selected migrants uniformly from among the "overfull" population. A higher average fitness could be obtained if many migrants were send infrequently or if only a few migrants were send more frequently. Often the partitioned GA found fitter individuals than the PGA with migration. Best results were achieved with a migration rate such as 20% of each subpopulation migrating every 20 iterations.

Toth and Lorincz [ToLo93], organized individuals into subpopulations trying to optimize problems depending on external variables. First they digitized the problem, taking representative examples (*sites*) in the space of external variables. Then they allowed individuals to migrate to neighboring sites. They used the Kohonen algorithm to locate the subpopulations in the external space. Every individual is present in the mating pool of his own subpopulation with 100% probability and with p_{mig} in the mating pool of a neighboring subpopulation. For breeding they used the relative fitness of an individual defined as $f^R=p_{mig}(af+b)$. They used the PGA to guide a robot arm to catch a ping-pong ball, falling from varying heights, with a bat. The external variable was the height from which the ball was dropped (1-1.5m). Using 50 subpopulations with 200 individuals each, the performance was good regardless of p_{mig}. With $p_{mig}=0.7$ the number of subpopulations could be decreased by a factor of twenty, without any change in final performance, while with $p_{mig}=0$ any substantial decrease led to severe fall of performance. A height variation of 5-10cm did not change the optimal solution seriously.

iii. Other Coarse-Grained models

Bianchini and Brown [BiBr93], tested different PGAs on the 0-1 integer linear programming problem on a transputer network. All the implementations were synchronized. The processors communicated after they had run a certain number of generations. Implementations used:

- Centralized: Interconnection topology is a star with the master in the center. It works like the standard parallel approach. The master collects the population, executes the replication algorithm and sends couples of individuals to the slave processors. The slaves apply the genetic operators, test the validity and compute the fitness of the generated individuals, and return the new individuals to master. In order to reduce communication overhead, groups of individuals are send to each processor which execute the replication algorithm on its own group.

- Semi-Distributed (see also [SWLD]): Clusters of processors work as in the centralized implementation. The cluster's masters are connected in a torus topology. Communication between masters can happen with a certain frequency, in asynchronized fashion, exchanging some of the best individuals between clusters of processors. The slaves behave exactly like the slaves in the centralized implementation.

- Distributed (Stepping-Stone model): Each processor holds a piece of total population and runs a complete sequential GA on it. Processor interconnection topology is again a torus. Processors sent some of their best individuals to their neighbors. The periodic communication between processors is designed to reduce the problem of processors getting stuck on local minima.

- Totally Distributed: Each processor holds a subpopulation and runs a complete GA on it. No communication between processors.

They used eight 25MHz T805 transputers. They run these implementations for 500 generations with total population size of 320 individuals. Other parameters are:

Subpopulations on master: 33 (centralized), 31 (semi-distributed)

Subpopulations on slaves: 41 (centralized), 43 (semi-distributed)

Migration interval: 50 generations (semi-distributed, distributed)

Migration rate: 48 individuals (semi-distributed) and four (distributed)

Their results indicate that some diversity of population improves the quality of genetic search, while some centralization of the population produce better solutions, when running for a fixed number of generations

Husbands and Mill [HuMi91], proposed a model based on simulated co-evolution and applied it, to a highly generalized version of the manufacturing scheduling problem, trying to simultaneously optimize the individual plans and the overall schedule. They used parallel GA search in a form of distributed problem solving, that is solving co-operative and interacting subproblems. The model involves a number of separate subpopulations each evolving using a GA, under the pressure of selection to find near-optimal process plans for each of the components. The genotype for each population is different and represents a solution to one of the subproblems. First the cost of the members of each subpopulation according to local criteria, is computed. Each population is ranked according to this cost. Equally ranked members of all populations interact with each other and noone else. In this way the separate species co-evolve in a shared world. Possible conflicts between species (e.g. disputes over

shared resources) are decided by a further co-evolving species, the Arbitrators. Arbitrators evolve under a pressure to make decisions that benefit the whole ecosystem. A final cost for each organism (including Arbitrators) is calculated, and it is used to breed back the organisms in their population.

They implemented the system on a 500 transputer based parallel machine. Machining costs (costs for the process plan organisms) of the best individual in each population reduced with time, and also the Arbitrator costs and the total elapsed time were reduced. There is some tension between the various objectives, one cost may momentarily rise while others drop, but the overall trend is down. A model of real job-shop is used and the components planned for are of medium to high complexity needing 25-60 operations to manufacture. Typically each operation has eight candidate machines and each of these machines has six possible setups. Very good solutions were found within a few minutes.

Liepens and Baluja [LiBa91], used a PGA with a central processor phase. In parallel, fifteen subpopulations of ten individuals each, run a GA on their own subpopulations. Next, during the central processor phase, the most fit individual from each subpopulation is gathered along with an additional fifteen randomly generated individuals. Under the control of the central processor a recombination phase of these thirty individuals occurs. The best individual is then injected into the populations of one-third of the processors. They claim that small subpopulations remain more heterogeneous.

Potts et al. [PoGY94], introduced three new operators to evolution of multiple parallel populations- migration, artificial selection and recycling, in a system called "GAMAS". The problems tested were the file allocation problem, a simple XOR neural network, and a sine envelope sine wave function. GAMAS creates multiple populations (*species*), and derive solutions from the combined evolutionary effects of the species. It uses one point crossover, roulette selection, fixed mutation and crossover rates, fitness scaling, migration of individuals between the species, artificial selection of highly fit individuals from the species and reintroduction back into the evolutionary process and "species recycling".

GAMAS uses four species. Each species is unique in its purpose. Initially generates species II, III, and IV. Species II (exploration species) is assigned a high mutation rate and the initial population is biased to more zeros in each individual. Species III (exploration species) is assigned a low mutation rate and is biased to more ones. Species IV is assigned a mutation rate that falls between the other two, with an equal chance of generating zeros and ones for any given loci. It uses Species I as a dominant incipient species. It artificially selects the most highly fit individuals produced in the initial generation of the other three species and places them in Species I. Artificial selection can be compared to elitism. Artificial selection continues with each successive generation by replacing individuals in Species I with individuals produced in the other three species if they are found to be fitter. Species I is held in isolation and its individuals are not allowed to reproduce or mutate. At predetermined generations (3), the

individuals in Species I are reintroduced into the evolutionary process by replacing all of the individuals contained in Species IV. GAMAS forces the migration of randomly selected individuals between Species II, III, and IV. Migration occurs at a higher frequency during early generations and slows at subsequent generations. After sixty generations, Species II, III, and IV are reinitialized ("Recycling").

For the file allocation problem, GAMAS found the optimal solution in 99.8% of all runs, while the simple GA found it only 3% of the time. Fitness scaling improves selection process. The performance of the algorithm is slightly improved with biased initial populations. Artificial selection and reintroduction has a significant improvement on the performance. Without reintroduction the optimal was not found. When the mutation rate is set equal in all species, the performance is diminished. Migration improves performance as well.

With XOR, GAMAS found numerous optimal solutions by generation 20 in all runs, while serial GA improved only slightly over 60 generations. Its performance for the third problem tested is comparable to the best found in the literature.

Sepehri et al. [SWLD94], used a hybrid PGA for estimation of a hydraulic system parameters in heavy-duty hydraulically-actuated manipulators, particularly, identification of hydraulic compliance (identification of joint flexibility). They combined "synchronous master-slave GA" and "cooperating sequential GA". The synchronous master slave GA, the master is responsible for distributing individuals from a global population to slaves, and reproduction, crossover and mutation on the entire population. The slaves calculate fitness values for the individuals assigned to them and return the results to the master.

The cooperating sequential GA is a collection of many independent GAs running and communicating with one another simultaneously. They used a parallel system with 16 T800 transputers. The proposed hybrid architecture was composed of a serially interconnected network for the local GAs and high speed local buses for master-slave connection. They used four nodal GAs, each one connected to three other transputers which act as slave processors. Each nodal GA (NGA) exchanged the best individual from its population with another NGA at certain frequency (0.3 individuals per generation). They achieved a speed factor of 12 with 120 global population size and 30 individuals in each NGA.

3.2. Fine-Grained Parallel Genetic Algorithms

Diffusion or isolation-by-distance or neighborhood model

In a fine-grained PGA usually one individual is assigned to each processor. The individuals have only local interactions and neighborhoods/demes, as opposed to global ones. Demes are overlapping. Some important choices in this case are:

- neighborhood/Deme size
- processor connection topology
- individual replacement scheme

Baluja [Balu92], used the MasPar MP-1 with 4096 processing elements to implement and test the diffusion model on three different population topologies: i) Circularly linked linear ordering of processors. Each processor evolves two individuals per generation, chosen from a population of ten individuals. The population is comprised of one individual from each of the four immediate left and immediate right processors, and the two individuals in the local processor evolved from the previous generation; ii) 2-D toroidal array of processors. In this case, the population of ten is comprised from the eight immediately neighboring processors, and the two individuals from the local processor; iii) Linear ordering of processors. One individual from each of the three immediate left processors and one from the 4,5,6 processors from the right are used in the candidate population. and the two individuals from the local processor. The four remaining are filled with two copies of each of the local individuals evolved in the previous generation.

The third model allows a faster spread of good individuals than the first, and a slower spread than the second. Six classes of problems were tested. De Jong's five function test suite, three subset-sum problems, two orderings of a partially deceptive order 4 problem, two orderings of fully deceptive order 4 problems, two sizes of the gap problem and three versions of all ones problem.

Preliminary results have shown that this PGA model is able to solve problems more efficiently than a simple GA. The 2-D array topology seemed to work best.

Collins and Jefferson [CoJe91], applied a PGA on the graph partitioning problem although their target application involve the evolution of artificial organisms. Individuals were placed on a one or 2-D grid, with one organism per grid location. The two parents of each offspring are the highest scoring individual encountered during two random walks that begin at the offspring location, one parent per walk. Deme size is a function of the length of random walks. Their results indicate that local selection/mating appears to be superior to panmictic mating. Local mating finds optimal solutions faster, and multiple optimal solutions in the same run. It maintains much greater diversity of genotypes, allows a high degree of inbreeding (a measure of how similar the mates are on average) and is much more robust. Longer random walks result in faster evolution. For the same random walk length, 2-D local mating is faster than 1-D local mating. Increasing he population size causes only slight speed improvements.

Davidor [Davi91], proposed the ECO GA model where the population is placed on a 2-D grid, having its opposite edges connected together so that each grid element has eight adjacent elements. At initialization, individuals are placed on the grid at random but no more than one individual per grid element. For reproduction a neighborhood of size nine is used. Parents

selection is done probabilistically according to the individuals relative fitness. An offspring is put back on a grid element in the vicinity of its parents. If the grid element is already occupied by an 'adult' individual, then a conflict is initiated, which is resolved probabilistically according to their fitness. The model was applied to a demonstrative multi-modal function optimization problem. "Initial results indicate that this model improves on the problems associated with controlling the convergence."

Gorges-Schleuter [Gorg90], applied a PGA on the TSP problem, using a 64-processor Parsytec transputer system with a sparse graph population topology (ring ladder and torus). She used a population size of 64 and a neighborhood size of eight(8). An individual's fitness was defined relative to other individuals in its neighborhood.

The selected individual is replaced by its offspring using the following survival strategies:

- A offspring is accepted
- B offspring is accepted only if it is fitter than the local weakest +1%
- C offspring is accepted only if it is fitter than the local weakest
- D offspring is accepted only if it is fitter than the local average
- E elitism - offspring is accepted only if it is fitter than the local parent

Elitism (E) outperforms the other in both, convergence speed and final quality, and works better with proportional mate selection. Rank selection yields a bit slower convergence speed. Population size and deme size interact proportionally. A relatively small deme size is always a good choice. With a small deme size, communication costs were negligible and linear speedup was achieved.

The gene pool of the ladder population develops continuously, whereas the torus populations has a phase of fast loss of variability ending up early at almost no variability, because of the fast propagation of genes. Panmictic population and survival strategy B, converges extremely slow, while strategy E looses variability very fast, and with strategy C, fixation is very slow and is reflected in the convergence speed.

Muhlenbein et al. [MuGK87][MuGK88], applied a PGA with selection pressure to the TSP, and graph partitioning problem. They used an Encore System with 16 processors. Each individual, located upon a different processor, attempted to improve itself by applying a local search heuristic. (hill climbing). Individuals are placed on a 2-D grid. The algorithm ran loosely synchronously and at the end of each optimization step each individual selected a mate from within a small neighborhood of its own processor. Each individual had four neighbors. The quality of the solution obtained, depended on the number of individuals on the population and on the number of processors. The minimum number of processors depends on the problem size. In [Muhl89], Muhlenbein applied the PGA to the quadratic assignment problem using a 64-processor system. The individuals are placed on two rings. Each individual had four neighbors, two on each ring. The global best is included in the neighborhood with a weight of two, having

seven individuals used for recombination. A new optimum for the largest published problem was found. In [Muhl92], he also applied elitism to the TSP, and graph partitioning problem. He tried to avoid premature convergence by allowing only slow propagation of highly fit individuals across the full population. This depends on the topology of the processor's neighborhood (*population structure).* Using a 2-D circular ladder with two individuals per 'step', he claimed that he avoided premature convergence. A neighborhood size of eight was used by each individual.

von Laszewsski and Muhlenbein [vLaM90], tackled the graph partitioning problem with individuals mapped to a 64-processor transputer system. The parent individual was replaced if the offspring was at least as good as the worst string in the neighborhood. A small neighborhood size in conjunction with a large population size gave the best results.

Spiesens and Manderick [SpMa91], implemented and analysed a model called *FG-algorithm,* on the DAP massively parallel computer, with 32x32 processors. Individuals are placed on a planar grid. Selection and crossover are restricted to an individual's immediate neighborhood, forming small clusters of identical genotypes. The diffusion of clusters through the recombination of individuals at the edges, makes it possible to explore new regions of the search space. For selection/mating it is necessary to get the individuals of the neighboring processors to the processor itself. Experiments were carried out on an "ugly" deceptive function, which consists of a sum of 10 disjoint order four deceptive problems. Using proportionate selection, a cluster with the same or marginally better fitness value as its surroundings will eventually disappear. Using fitness scaling, the performance was improved. It was also improved using local tournament selection and higher mutation rates. Using proportionate selection (with or without scaling), the performance is slightly improved with the neighborhood size. Improvement is less profound if scaling is used. Using tournament selection, performance decrease with the neighborhood size (4,8,12). Uniform crossover had a better performance than two-point and one-point crossover.

Talbi and Bessiere [TaBe91], used a PGA to solve the graph partitioning problem and specially the mapping of parallel programs on parallel architectures. They used a supernode based on transputers, connected on a torus. The population is mapped on the torus, one individual per processor. Even though, the size of the neighborhood is an adjustable parameter, they restrict neighborhood to only directly connected individuals, in order to avoid overhead and complexity of routing. Each individual receives the individuals from their neighbors, and reproduces with them. Because of the communication overhead, they do not consider the best solution found globally, but the best solution placed on the root processor through a "spy" process.

Varying the number of processors they found a linear speedup. The quality of the solution improves with an increase of population's size. For a too small population a premature convergence occurs and the optimal solution will not be ever reached. For a given solution's

quality they observed a superlinear speedup when varying the number of processors. They compared their results with hill-climbing and simulated annealing. PGA outperformed the other two, both in the quality of the solution and the time used.

Talbi and Muntean [MuTa91] [TaMu91] applied this algorithm to the mapping problem (placement of communicating processes on a distributed memory architecture). A pipeline of 32 processes was to be mapped on a ring of eight processors. For a given number of generations the solution quality improves with an increase of population's size. The algorithm has a near-linear speedup. The PGA outperforms hill-climbing, both in quality of the solution and the time used in search. A comparable solution can be obtained with simulated annealing, but PGA is less time consuming (~1/23). In [TaMu93] they used a supernode with 128 transputers. They also proposed a hybrid GA, where the initial population is generated by a hill-climbing algorithm. The solution quality obtained by the hybrid algorithm is better, but the price to pay is a greater search time.

This algorithm was also successfully used by Ahuactzin et al [ATBM92], to the problem of planning a collision free path for an holonomic mobile robot. Using 128 transputers, the planning time was under one second. Since it does not make use of a precomputed representation of the configuration space, it may be suitable to plan paths in a dynamical environment. In [BATM93], they generalized the solution, proposing a path planner able to drive a robot in a dynamic environment where the obstacles are moving.

The method, called "Ariadne's Clew Algorithm", is based on the combination of two local planning algorithms, an '*explore*' and a '*search*' algorithm. The purpose of 'explore' is to collect information about the environment with an increasingly fine resolution by placing landmarks in the searched space. The goal of 'search' is to opportunistically check if the target can be reached from any given placed landmark. 'Explore' and 'search' may run in parallel. 'Explore produces a landmark, while 'search' exploits the previous one. Both of them run the PGA described earlier. The supernode used (128 transputers), produces a plan in about two seconds, given the precise geometrical description of the arms of another robot used as a dynamic obstacle, and some static obstacles.

3.3. Other Parallel Genetic Algorithms

Gruau [Grua94], used a mixed model which combines the stepping-stone model and the isolating-by-distance model. Individuals are distributed on islands which form a two-dimensional (2-D) torus. Each island, mapped on one processor, can send individuals only to its four neighboring islands. Inside one island the individuals are arranged on a 2-D grid. The whole spatial structure is a 2-D torus of 2-D grids. Some sites of the 2-D grid are not occupied.

The density of population is kept around 0.5. During mating, a site s is randomly chosen on the grid. From this site, two successive random walks are performed. The two best individuals found, are mated. The offspring is placed on site s. If site s was occupied by another individual i, i is placed on a randomly chosen neighboring site. During migration, a site s is selected on a border of the 2-D grid, a random walk starts from this site, the best individual found is send to the processor next to the border, and placed in a site with the same coordinates as s. This model was used to find a boolean neural net (with weights ±1) that computes the parity of 51 inputs. He encodes neural networks with the cellular encoding to develop a family of neural nets. He uses a learning procedure for the neural network as a hill-climber (*switch learning*)

He used an iPSC860 MIMD machine with up to 64 processors. Optimal population size depends on the number of processors. The computation of the speedup must try to optimize the population size for a given number of processors. He observed a super-linear speed-up until sixteen processors. After that performance drops because roughly 128 individuals are enough to solve the target problem, and the mixed PGA works well for subpopulations of at least eight individuals.

Kitano et al. [KiSH91], proposed a system called GA-1, which offers opportunities for research into large-scale rule learning with GAs, providing a flexible software environment in which specific GA-based learning systems composed of various generation models, migration models, and mating and reproduction strategies are easily configurable. It is based on IXM2 massively parallel associative processor, which consists of 64 T800 transputers. Each processor (associative processor) operates with eight chips of associative memory (512 words by 40 bits per chip). Network processors are used to handle communication between associative processors. One top-level network processor deals with communication among the lower-level network processors, and eight lower-level network processors each of which is connected to eight associative processors. IXM2 employs a full connection, so that communication between any two processors can be attained by going through only two network processors. Due to the associative memory chips, GA-1 is capable of carrying out 256K parallel bit-vector matching when all 64 PEs are used (4K parallel bit-vector matching per PE). One can assign one individual or one population per processor. Depending on the mapping of population GA-1 provides three layers of mating: (1) intra-processor, (2) intra-cluster, (3) inter-cluster. It is also possible to consider a multiple population mapping as a model of multiple species, or even a co-evolution model. Although, GA-1 supports use of global synchronous mating strategy, the principal strategy advocated in GA-1 is the continuous generation model, where each individual performs mating with no synchronization using either a global or a local mating scheme. This minimizes idle processor time.

Performance data demonstrate that GA-1 provides orders of magnitude (1/45) of speed-up in training and executing GA-based rule learning system, comparing with other machines

such as CM-2. Regardless of implementation and mapping, some sort of central or local control at coarse-grain level processor is necessary to carry out GA operations.

Macfarlane and East [MaEa89][EaMa93], compared a standard GA, a migration model (sub-populations in a ring, and migration of fixed percentage individuals (10%) in one direction around the ring), and two diffusion models with neighborhood of four individuals and population distributed over a torus (A-replace self with offspring, B-replace self only if offspring is fitter). They used five functions: De Jong's F3 and F5, Ackley's "porcupine"(ACK), Tanese function [Tane87](TAN), and a five equal peaks function used by Goldberg(GOL). All models found global optimum for F3, ACK and TAN. The performance ranking of the models was: Diffusion B, Migration, Standard, Diffusion A. For F5 the algorithms were trapped in local optima and only a percentage of the runs reached the global optimum. Performance order of the percentage is the same as previously, but standard GA did better than migration GA in average best solution found after 100 generations. Goldberg's function was used to find how well niches are formed and maintained in these models. The order of their performance was: Diffusion B, Migration, Diffusion A, Standard. Final test was done for the time it was needed to run 100 generations on function TAN using 1,2,...,8 processors. Total population size was kept fixed. They ranked as: Migration, Diffusion, Standard.

Surry and Radcliffe [SuRa94], created a system called RPL2(Reproductive Plan Language), that acts as an extensive interpreted language for writting and using evolutionary computing programs. It supports arbitrary genetic representations, all structured population models (diffusion, island, stepping-stone) with further hybrids and runs on parallel or serial hardware, while hiding parallelism from the user.

It has been used in a number of applications.

- Optimizing a retail network for Ford cars.
- Optimizing stock market tracking.
- Airline crew scheduling formulated as the standard set partitioning problem.
- Neural Network topology optimization.

Dekker, Ribeiro et al. [DeRi93] [KiRT93] [RiAT93] described a GA programming environment (GAME), developed for ESPRIT III Project PAPAGENA. This environment provide a general-purpose toolkit for the programming and simulation of a wide range of GA applications. It comprises a library of common GAs and a powerful graphic interface. In addition, it exploits the intrinsic parallelism of GAs and develops a compiling and mapping system targeted at transputer based machines, and future architectures. The application areas at a demonstrator level, span important areas such as protein folding and financial and economic modelling.

References

[Abra92] **Abramson D.**, A Parallel Genetic Algorithm for Solving the School Timetabling Problem, in *Proceedings of the 15th Australian Computer Science Conf.*, Hobart, Feb 1992

[AbMP92] **Abramson D., Mills G., Perkins S.**, Parallelisation of a Genetic Algorithm for the Computation of Efficient Train Schedules, in *Proceedings of 1993 Parallel Computing and Transputers Conf.*, Brisbane, Nov. 1993

[ATBM92] **Ahuactzin J.M., Talbi A-G., Bessiere P., Mazer E.**, Using Genetic Algorithms for Robot Motion Planning, in *Proceedings of European Conf on Artificial Intelligence 1992*

[Balu92] **Baluja Shumeet**, A Massively Distributed Parallel Genetic Algorithm (mdpGA), Tech. Report CMU-CS-92-196, Carnegie Mellon University, School of Computer Science, Oct.13, 1992

[BATM93] **Bessiere P., Ahuactzin J-M., Talbi E-G., Mazer E.**, The Ariadne's Clew Algorithm: Global Planning with Local Methods, in *Proceedings of the IEEE-IROS'93 Conference on Intelligent Robots and Systems*, Yokohama, Japan, 1993

[Beye93] **Beyer H-G.** Optimization of large-scale order problems by the Evolution Strategy, in R. Schumacher (Ed.), *One Year KSR1 at the University of Mannheim - Results & Experiences*, Tech. Report RUM 35/93, Dec. 1993, Univ of Mannheim

[BiBr93] **Bianchini R., Brown C.**, Parallel genetic algorithms on distributed-memory architectures, Tech. Report 436, University of Rochester, Computer Science Dept., May 1993

[BrHS89] **Brown D.E., Huntley C.L., Spillane A.R.**, A Parallel Genetic Heuristic for Solving the Quadratic Assignment Problem, in J.D. Schaffer (Ed.), *ICGA-89: Proceedings of the Third International Conf. on Genetic Algorithms*, p.406-415, George Mason Univ., June 1889, Morgan Kaufmann Publishers

[CoMR90] **Cohoon J.P., Martin W.N., Richards D.S.**, Genetic Algorithms and Punctuated Equilibria in VLSI, in H.-P.Schwefel, R.Manner (Eds.), *PPSN 1: Proceedings of Parallel Problem Solving from Nature*, 1st Workshop, 1-3 Oct. 1990, p.134-144, Springer-Verlag, Berlin, Germany, 1990

[CoMR91] **Cohoon J.P., Martin W.N., Richards D.S.**, A Multi-population Genetic Algorithm for Solving the K-Partition Problem on Hyper-cubes, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.244-248, San Diego, CA., July 1991

[CoJe91] **Collins R.J., Jefferson D.R.**, Selection in massively parallel genetic algorithms, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.249-256, San Diego, CA., July 1991

[Davi91] **Davidor Y.**, A Naturally Occuring Niche & Species Phenomenon: The Model and First Results, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.257-263, San Diego, CA., July 1991

[DeRi93] Dekker L., Ribeiro Filho J.L., The GAME Virtual Machine Architecture, in J. Stender (Ed.), *Parallel Genetic Algorithms: Theory and Applications*, p.93-110, Brainware GmbH, Berlin, 1993, IOS Press

[Dori92] Dorigo M., Using Transputers to Increase Speed and Flexibility of Genetics-Based Machine Learning Systems, *Microprocessing and Microprogramming* 34, 1-5, p.147-152, Feb. 1992

[DoMa93] Dorigo M., Maniezzo V., Parallel Genetic Algorithms: Introduction and Overview of Current Research, in J. Stender (Ed.), *Parallel Genetic Algorithms: Theory and Applications*, p.5-42, Brainware GmbH, Berlin, 1993, IOS Press

[EaMa93] East I., Macfarlane D., Implementation in Occam of Parallel Genetic Algorithms on Transputer Networks, in J. Stender (Ed.), *Parallel Genetic Algorithms: Theory and Applications*, p.43-64, Brainware GmbH, Berlin, 1993, IOS Press

[FoHu90] Fogarty T.C., Huang R., Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems, in H.-P.Schwefel, R.Manner (Eds.), *PPSN 1: Proceedings of Parallel Problem Solving from Nature*, 1st Workshop, 1-3 Oct. 1990, p.145-149, Springer-Verlag, Berlin, Germany, 1990

[Gold89] Goldberg D.E, *Genetic Algorithms in Search Optimization & Machine Learning*, Addison-Wesley, Reading, MA, 1989

[Gorg89] Gorges-Schleuter M., ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy, in J.D. Schaffer (Ed.), *ICGA-89: Proceedings of the Third International Conf. on Genetic Algorithms*, p.422-427, George Mason Univ., June 1889, Morgan Kaufmann Publishers

[Gorg90] Gorges-Schleuter M., Explicit Parallelism of Genetic Algorithms Through Population Structures, in H.-P.Schwefel, R.Manner (Eds.), *PPSN 1: Proceedings of Parallel Problem Solving from Nature*, 1st Workshop, p.150-159, Springer-Verlag, Berlin, Germany, 1990

[Grua94] Gruau F., Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm, PhD. Thesis, Ecole Normale Superieure de Lyon, Laboratoire de l'Informatique du Parallelisme (LIP-IMAG), Jan. 1994

[HuMi91] Husbands P., Mill F., Simulated co-evolution as the mechanism for emergent planning and scheduling, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.264-270, San Diego, CA., July 1991

[KiRT93] Kingdon J., Ribeiro Filho J.L., Treleaven P., The GAME Programming Environment Architecture, in J. Stender (Ed.), *Parallel Genetic Algorithms: Theory and Applications*, p.85-92, Brainware GmbH, Berlin, 1993, IOS Press

[KiSH91] Kitano H., Smith S.F., Higuchi T., GA-1: A Parallel Associative Memory Processor for Rule Learning with Genetic Algorithms, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.311-317, San Diego, CA., July 1991

[KrSV90] Kroger B., Schwenderling P., Vornberger O., Parallel Genetic Packing of Rectangles, in H.-P.Schwefel, R.Manner (Eds.), *PPSN 1: Proceedings of Parallel Problem Solving from Nature*, 1st Workshop, 1-3 Oct. 1990, p.160-164, Springer-Verlag, Berlin, Germany, 1990

[KrSV93] Kroger B., Schwenderling P., Vornberger O., Parallel Genetic Packing on Transputers, in J. Stender (Ed.), *Parallel Genetic Algorithms: Theory and Applications*, p.151-186, Brainware GmbH, Berlin, 1993

[Levi94] Levine D., A Parallel Genetic Algorithm for the Set Partitioning Problem, Ph.D. Thesis, Argonne National Laboratory, Mathematics and Computer Science Division, May 1994

[LiBa91] Liepens G., Baluja S., apGA: An adaptive Parallel Cenetic Algorithm, Tech. Report, Oak Ridge National Laboratoty, 1991

[MaEa89] Macfarlane D., East I., An Investigation of Several Parallel Genetic Algorithms, in Turner S.J. (Ed), *OUG-12: Tools and Techniques for Transputer Applications, Proceedings of the 12th Occam User Group Technical Meeting*, p.60-67, Exeter, 2-4 April 1989, IOS Press, Amsterdam, Netherlands

[MaSp89] Manderick B., Spiessens P., Fine-Grained Parallel Genetic Algorithms, in J.D. Schaffer (Ed.), *ICGA-89: Proceedings of the Third International Conf. on Genetic Algorithms*, p.428-433, George Mason Univ., June 1889, Morgan Kaufmann Publishers

[McGD90] McClurkin G.D., Geary R.A., Durrani T.S., An Investigation into the Parallelising of Genetic Algorithms (Extended Abstract), in Prichard D.J., Scott C.J. (Eds), *Applications of Transputers 2, Proceedings of the Second International Conf. on Applications of Transputers*, p.581-7, July 1990, IOS, Amsterdam

[Muhl89] Muhlenbein H., Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, in J.D. Schaffer (Ed.), *ICGA-89: Proceedings of the Third International Conf. on Genetic Algorithms*, p.416-421, George Mason Univ., June 1889, Morgan Kaufmann Publishers

[Muhl92] Muhlenbein H., Parallel Genetic Algorithms and Combinatorial Optimization, in O. Balchi, R. Sharda, S. Zenios (Eds.), *Computer Science and Operations Research*, p.441-456, 1992, Pergamon Press

[MuGK87] Muhlenbein H., Gorges-Schleuter M., Kramer O., New Solutions to the Mapping Problem of Parallel Systems: The Evolution Approach, *Parallel Computing* 4, p.269-279, 1987

[MuGK88] Muhlenbein H., Gorges-Schleuter M., Kramer O., Evolution Algorithms in Combinatorial Optimization, *Parallel Computing*, 7, p.65-85, 1988

[MuSB91a] Muhlenbein H., Schomisch M., Born J., The parallel genetic algorithm as function optimizer, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.271-278, San Diego, CA., July 1991

[MuSB91b] Muhlenbein H., Schomisch M., Born J., The parallel genetic algorithm as function optimizer, *Parallel Computing*, 17, p.619-632, Sept. 1991

[MuTa91] Muntean T., Talbi E.G., A parallel genetic algorithm for process-processors maping, in M.Durand, F. EL Dabaghi (Eds.), *High Performance Computing II, Proceedings of the Second Symposium*, p.71-82, North-holland, Amsterdam, Netherlands, 1991

[Norm88] Norman M.G, A genetic approach to topology optimization for multiprocessor architectures, Tech. Report ECSP-TR-7, Univ. of Edinburgh, Dept. of Physics, 1988

[PeLe89] Pettey C.C., Leuze M.R, A Theoretical Investigation of a Parallel Genetic Algorithm, in J.D. Schaffer (Ed.), *ICGA-89: Proceedings of the Third International Conf. on Genetic Algorithms*, p.398-405, George Mason Univ., June 1889, Morgan Kaufmann Publishers

[PeLG87] Pettey C.C., Leuze M.R., Grefenstette J., A Parallel Genetic Algorithm, in J. Grefenstette (Ed.), *ICGA-87: Proceedings of the Second International Conf. on Genetic Algorithms*, p.155-161, MIT, Cambridge, MA, July 1987, Lawrence Erlbaum Associates

[PoGY94] Potts J.C., Giddens T.D., Yadav S.B., The Development and Evaluation of an Improved Genetic Algorithm Based on Migration and Artificial Selection, *IEEE Trans. on Systems, Man, and Cubernetics*, 24 (1), January 1994, p.73-86

[RiAT93] Ribeiro Filho J.L., Alippi C., Treleaven P., Genetic Algorithm Programming Environments, in J. Stender (Ed.), *Parallel Genetic Algorithms: Theory and Applications*, p.65-84, Brainware GmbH, Berlin, 1993, IOS Press

[SWLD94] Sepehri N., Wan F.L.K., Lawrence P.D., Dumont G.A., Hydraulic Compliance Identification using a Parallel Genetic Algorithm, *Mechatronics*, 4(6), p.617-633, 1994

[SpMa91] Spiesens P., Manderick B., A Massively Parallel Genetic Algorithm. Implementation and First Analysis, in R-K. Belew, L.B. Booker (Eds.), *ICGA-91: Proceedings of the Fourth International Conf. on Genetic Algorithms*, p.264-270, San Diego, CA., July 1991

[StWM90] Starkweather T., Whitley D., Mathias K., Optimization Using Distributed Genetic Algorithms, in H.-P.Schwefel, R.Manner (Eds.), *PPSN 1: Proceedings of Parallel Problem Solving from Nature*, lst Workshop, 1-3 Oct. 1990, p.176-185, Springer-Verlag, Berlin, Germany, 1990

[SuRa94] Surry P.D., Radcliffe N.J., RPL2: A Language and Parallel Framework for Evolutionary Computing, Tech Report EPCC-TR94-10, University of Edinburgh, Edinburgh Parallel Computing Centre, 1994

[TaBe91] Talbi E-G., Bessiere P., A Parallel Genetic Algorithm for the Graph Partitioning Problem, in *Procs of the 1991 International Conf. on Supercomputing*, p.312-320, Cologne, Germany, 17-21 June 1991, ACM Press

[TaMu91] Talbi E-G., Muntean T., A New Approach for the Mapping Problem: A Parallel Genetic Algorithm, 1991

[TaMu93] Talbi E-G., Muntean T., General Heuristics for the Mapping Problem, in *Proceedings of the World Transputer Conf.*, 1993

[Tane87] Tanese R., Parallel Genetic Algorithms for a hypercube, in J.J. Grefenstette (Ed.), *ICGA-87: Proceedings of the Third International Conf. on Genetic Algorithms and Their Applications*, p.177-183, MIT, Cambridge, MA, July 1987, Lawrence Erlbaum Associates

[Tane89] Tanese R., Distributed Genetic Algorithms, in J.D. Schaffer (Ed.), *ICGA-89: Proceedings of the Third International Conf. on Genetic Algorithms*, p.434-439, George Mason Univ., June 1889, Morgan Kaufmann Publishers

[ToLo93] **Toth G.J., Lorincz A.** Genetic Algorithm with Migration on Topology Conserving Maps, in *ICANN'93: Proceedings of the International Conference on Artificial Neural Networks*, p.605-8, Sept. 1993, Amsterdam

[vLaM90] **von Laszewsski G., Muhlenbein H.,** Partitioning a graph with a Parallel Genetic Algorithm, in H.-P.Schwefel, R.Manner (Eds.), *PPSN 1: Proceedings of Parallel Problem Solving from Nature*, 1st Workshop, 1-3 Oct. 1990, p.165-169, Springer-Verlag, Berlin, Germany, 1990

[WhSB90] **Whitley D., Starkweather T., Bogart C.,** Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity, *Parallel Computing*, 14 (3), p.347, Aug. '90