

Parallel Genetic Algorithms for Constrained Ordering Problems

Kay Wiese, Sivakumar Nagarajan, and Scott D. Goodwin

Department of Computer Science, University of Regina

Regina, Saskatchewan, S4S 0A2, Canada

Phone: (306)-585-5210, Fax: (306)-585-4745

Email: {wiese,shiv,goodwin}@cs.uregina.ca

Abstract

This paper proposes two different parallel genetic algorithms (PGAs) for constrained ordering problems. Constrained ordering problems are constraint optimization problems (COPs) for which it is possible to represent a candidate solution as a permutation of objects. A decoder is used to decode this permutation into an instantiation of the COP variables. Two examples of such constrained ordering problems are the traveling salesman problem (TSP) and the job shop scheduling problem (JSSP). The first PGA we propose (PGA1) implements a GA using p subpopulations, where p is the number of processors. This is known as the *island model*. What is new is that we use a different selection strategy, called *keep-best reproduction* (KBR) that favours the parent with higher fitness over the child with lower fitness. Keep-best reproduction has shown better results in the sequential case than the standard selection technique (STDS) of replacing both parents by their two children (Wiese & Goodwin 1997; 1998a; 1998b). The second PGA (PGA2) is different from PGA1: while it also works with independent subpopulations, each subpopulation uses a different crossover operator. It is not a priori known which operator performs the best. PGA2 also uses KBR and its subpopulations exchange a percentage q of their fittest individuals every x generations. In addition, whenever this exchange takes place, the subpopulation with the best average fitness broadcasts a percentage q_2 of its fittest individuals to all other subpopulations. This will ensure that for a particular problem instance the operator that works best will have an increasing number of offspring sampled in the global population. This design also takes care of the fact that in the early stages of a GA run different operators can work better than in the later stages. Over time PGA2 will automatically adjust to this new situation.

Topic Areas: Search, Distributed AI, Genetic Algorithms, Evolutionary Computing, Constraint Optimization

Copyright ©1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Introduction

Many problems in artificial intelligence and simulation can be described in a general framework as a *constraint satisfaction problem* (CSP) or a *constraint optimization problem* (COP). Informally a CSP (in its finite domain formulation) is a problem composed of a finite set of *variables*, each of which has a finite *domain*, and a set of *constraints* that restrict the values that the variables can simultaneously take. For many problem domains, however, not all solutions to a CSP are equally good. For example, in the case of *job shop scheduling* different schedules which all satisfy the resource and capacity constraints can have different makespans (the total time to complete all orders), or different inventory requirements. So in addition to the standard CSP, a constraint optimization problem has a so-called *objective function* f which assigns a value to each solution of the underlying CSP. A global solution to a COP is a labeling of all its variables, so that all constraints are satisfied, and the objective function f is optimized.

Since it usually takes a complete search of the search space to find the optimum f value, for many problems global optimization is not feasible in practice. That is why COP research has focused on *local search* methods that take a candidate solution to a COP and search in its local neighborhood for improving neighbors. Such techniques include iterative improvement (hill climbing), threshold algorithms (Dueck & Scheuer 1990), simulated annealing (Cerny 1985; Kirkpatrick, Gelatt Jr., & Vecchi 1983), taboo search (Glover, Taillard, & Werra 1993), and variable depth search. Since these methods are only searching a subset of the search space, they are not *complete*, i.e., are not guaranteed to return the overall optimum. Another optimization technique are *genetic algorithms* (GAs). Genetic algorithms were originally designed to work on bitstrings. These bitstrings encoded a domain value of a real valued function that was supposed to be optimized. They were originally proposed by Holland (Holland 1975). More recently, researchers have focused on applying GAs to combinatorial optimization problems, including constraint optimization problems such as the traveling salesman problem (TSP) and the job shop scheduling problem (JSSP).

Most of the research since 1985 on applying GAs to the traveling salesman problem has focused on designing new representations and genetic operators and comparing their performance (Goldberg & Lingle, Jr. 1985; Grefenstette *et al.* 1985; Oliver, Smith, & Holland 1987; Whitley, Starkweather, & Fuquay 1989; Starkweather *et al.* 1991). Examples of different representations for the TSP are *direct representation*, *ordinal representation*, and *adjacency representation* (Grefenstette *et al.* 1985). We used direct representation for our implementation, where a list 2 4 5 1 3 6 means start at city 2, then go to city 4, then to city 5, and so on. Some of the operators discussed in the literature for direct representation include *partially mapped crossover* (PMX) (Goldberg & Lingle, Jr. 1985), *order crossover* (OX) (Davis 1985), *cycle crossover* (CX) (Oliver, Smith, & Holland 1987), and *edge recombination operator* (ERC) (Whitley, Starkweather, & Fuquay 1989), as well as position based crossover (PBX) and a variation of order crossover (OX2) (Syswerda 1990).

After two individuals are selected for reproduction in a GA, they undergo crossover and mutation producing two new individuals called children. With the standard selection strategy (STDS), the newly created individuals (children) are then inserted into the next generation. Crossover and mutation, however, can and usually will produce at least one child, whose fitness is lower than the fitness of at least one parent. This observation led us to develop a different selection strategy which evaluates children first, and keeps the best child as well as the best parent to ensure that good previous genetic material is not lost as well as to make sure that the GA makes progress in form of the new child. We have called this selection strategy keep-best reproduction (KBR) and have done several studies on sequential GAs that use KBR (Wiese & Goodwin 1997; 1998a; 1998b; 1998c) with the following conclusions: 1) KBR finds better tours faster than the standard selection/replacement technique used in many GAs. 2) It works well with smaller population sizes than the ones required by STDS especially as the problem size increases. 3) Genetic operator probabilities that yield optimal performance are different from the ones for STDS. 4) GAs using KBR are more robust than GAs using STDS, i.e. a small change in genetic operator probabilities does not lead to a sharp drop or increase in performance. 5) The selection pressure of KBR is much higher than that of STDS, so KBR tends to prematurely converge towards suboptimal solutions due to a loss of diversity in the population. To reintroduce diversity into the population a higher mutation rate is used with KBR than is with STDS.

Background on Parallel Genetic Algorithms

All local search algorithms face the same problem: the tradeoff between the amount of local search that can be performed and the real time available. In the case

that the local search algorithm is a GA, the amount of local search done is the amount of genetic information explored in the form of different schemata. Obviously the size of the population determines to a large extent the amount of genetic search that can be performed. Too small a population restricts the genetic search to a subspace that is improperly constrained due to too few schemata contained in the population. Larger populations tend to contain more schemata and can thus explore more of the search space. But larger populations also lead to an increase in computing time. This problem led to the development of parallel genetic algorithms (PGAs) that can work with larger populations and converge towards solutions in the same time as a serial GA with a much smaller population size needs to converge to a solution of (most likely) lesser quality. Most PGAs fall in one of two categories which were inspired by natural evolution: 1) *Island Model*, a discontinuous model and 2) the continuous *Neighborhood Model*, also known as the *Diffusion Model*, which are discussed and characterised in (Crow 1986) in the context of natural evolution. In the island model, subpopulations are separated. Recombination and selection is local (restricted to the subpopulation). Every so often *migration* between the subpopulations takes place. Early work on island model PGAs can be found in (Cohon *et al.* 1987) and (Petty, Leuze, & Grefenstette 1987).

The neighborhood model is different from the island model in the sense that there are no isolated subpopulations but rather selection and recombination are restricted to an individual's neighborhood. Since neighborhoods are overlapping, genetic information can "diffuse" through the global population as the genetic search goes on. Diffusion model PGAs are typically implemented on a $n \times n$ mesh with one individual per node. Size, shape, and radius of the neighborhood have an impact on the performance of the PGA. For examples see ASPARAGOS (Gorges-Schleuter 1989) and (Manderick & Spiessens 1989), who use a neighborhood model.

We will only consider island model PGAs in this paper. For the remainder of the paper PGA shall refer to island model PGA.

A Standard PGA (Island Model)

We shall refer to the PGA discussed in this section as PGA1. Sequential GAs use a population of n individuals. Let us assume for PGA1 we have p processors. The population is then being divided into p subpopulations of size $\frac{n}{p}$.

Each processor runs a sequential GA using only the $\frac{n}{p}$ population members it was originally assigned. Every x generations, the processors are globally synchronized and exchange a percentage q of their fittest individuals with a neighbor. n , x , and q are program parameters whose choice affects the performance of the GA. See figure 1 for a sketch of PGA1. As shown in the figure, the algorithm starts by partitioning the population space

```

PGA1:
begin
  Initialize a population Pop of n chromosomes
  Select  $Pop_1, \dots, Pop_p \subseteq Pop \mid size(Pop_i) = \frac{n}{p},$ 
     $Pop_1 \cup \dots \cup Pop_p = Pop,$ 
     $\forall i, j, 1 \leq i \leq n \mid i \neq j : Pop_i \cap Pop_j = \emptyset$ 
  x ← number of generations
  while (stopping criteria not reached) do
     $\parallel P_i(x, Pop_i)$  /* execute in parallel */
    Migrate(Pop)
    Select new  $Pop_1, \dots, Pop_p \subseteq Pop$ 
  end while
end PGA1

begin Process  $P_i(x : int, Pop_i : population)$ 
  i ← 0
  for gen ← 1 to x do
    while (i < sizeof(Popi/2)) do
      Select 2 chromosomes for recombination
      apply crossover operator
      apply mutation operator
      evaluate new chromosomes
      insert into next generation
      i ← i + 1
    end while
  end for
  barrier_sync()
end Process Pi

```

Figure 1: Algorithm PGA1

into p disjoint subpopulations, and each subpopulation is set off to evolve independently and in parallel for x generations. After x generations the p subpopulations synchronize using *barrier synchronisation*, a standard synchronisation mechanism in parallel computing which permits multiple processes to synchronize. The synchronisation is complete when all the processes have reached the barrier, which in this case would imply that all of the independently evolving subpopulations have finished evolving for the required x generations. The p subpopulations now exchange individuals with their neighbors. This is being done by the step *Migrate*. The subpopulations are then set off to evolve for another x generations. The steps *Migrate* and *Select new* can be combined into a single step. Note that this algorithm can run on parallel computers with distributed and shared memory. In fact, an existing network of workstations can be utilized for an implementation of algorithm PGA1. The algorithm is totally synchronous and communication is minimal. Without the communication overhead and assuming that the time to perform x generations on each of the p processors is on average the same, the speedup should be linear, i.e. if T_1 is the time required to run a sequential GA with n population

members, then

$$\frac{T_1}{T_p} = p$$

where T_p is the time required by PGA1 to run on p processors. Since GAs have a random component it is possible that PGA1 achieves superlinear speedups in some cases i.e.,

$$\frac{T_1}{T_p} > p$$

In this case, however, the underlying sequential GA would not be the ideal one for this particular problem. A sequential simulation of PGA1 would execute faster than the original sequential GA. If we compare a sequential simulation of PGA1 with PGA1, the best we can expect is a linear speedup.

A Different PGA

The genetic operators PMX, OX, OX2, CX, PBX, and ERC can have varying performance in different problem domains or on different problems in the same domain depending on the structure of the problem (Oliver, Smith, & Holland 1987; Whitley, Starkweather, & Fuquay 1989; Starkweather *et al.* 1991). It is not a priori known which operator performs the best. This is why we propose a new parallel genetic algorithm PGA2 which works as follows:

Like PGA1, PGA2 uses a population of n individuals and p processors. Also the population is divided into p subpopulations of size $\frac{n}{p}$. What is different to other approaches is that instead of executing the same sequential GA on each processor on the local population, PGA2 uses different crossover operators on each processor. PGA2 also uses KBR and its subpopulations exchange a percentage q of their fittest individuals. In addition, whenever this exchange takes place, the subpopulation with the best average fitness broadcasts a percentage q_1 of its fittest individuals to all other subpopulations. In doing so the operator who found the best tours in its subpopulation will have an increasing number of offspring sampled in the global population. It might often be the case that some operators improve their subpopulation more in the early stages of the GA and others more in the later stages. In this case PGA2 automatically switches in "favour" of the new better operator allowing this new operator to broadcast. Figure 2 contains a sketch of algorithm PGA2. The algorithm is similar to PGA1 in appearance, the only differences being the choice of crossover operators, and the broadcast step. Each independently evolving population selects a crossover operator to apply and uses it for its evolution.

PGA2 involves more overhead than PGA1. First, there is the addition of a broadcast, which has significantly higher costs than the nearest neighbor communication used in PGA1 (the actual cost of a broadcast depends on the topology of the inter-processor network). Second, each processor must know the average fitness of all other processors in order to know whether it is broadcasting or receiving. This involves having each

processor send its average fitness to each other. Depending on the topology this could be done in a number of intelligent ways using techniques such as pipelining and packaging.

PGA2:

```

begin
  Initialize a population Pop of n chromosomes
  Select  $Pop_1, \dots, Pop_p \subseteq Pop \mid size(Pop_i) = \frac{n}{p}$ ,
     $Pop_1 \cup \dots \cup Pop_p = Pop$ ,
     $\forall i, j, 1 \leq i \leq n \mid i \neq j : Pop_i \cap Pop_j = \emptyset$ 
  x ← number of generations
  while (stopping criteria not reached) do
    ||  $P_i(x, Pop_i)$  /* execute in parallel */
    Migrate(Pop)
    Broadcast step
    Select new  $Pop_1, \dots, Pop_p \subseteq Pop$ 
  end while
end PGA2

begin Process  $P_i(x : int, Pop_i : population)$ 
  i ← 0
  CO ← select crossover operator to apply
  for gen ← 1 to x do
    while (i <  $sizeof(Pop_i/2)$ ) do
      Select 2 chromosomes for recombination
      apply crossover operator CO
      apply mutation operator
      evaluate new chromosomes
      insert into next generation
      i ← i + 1
    end while
  end for
  barrier_sync()
end Process  $P_i$ 

```

Figure 2: Algorithm PGA2

Conclusion

We expect PGA2 to run minimally slower than PGA1 in terms of total computing time per evaluated individual because of the higher communication overhead. However, PGA2 should on average perform much better than PGA1 because it can use the best crossover operator for a specific problem without a priori knowing which one that is. Both parallel algorithms are expected to have near linear speedups, making it possible to solve larger COPs than in the sequential case. The versions of PGA1 and PGA2 that use keep-best reproduction are expected to outperform the versions that use standard selection.

References

Cerny, V. 1985. Thermodynamical Approach to the Traveling Salesman Problem. *Journal of Optimization Theory and Applications* 45:41-51.

Cohon, J.; Hedge, S.; Martin, W.; and Richards, D. 1987. Punctuated Equilibria: A Parallel Genetic Algorithm. In *Proceedings of the 2nd International Conference on Genetic Algorithms ICGA-87*, 148-154.

Crow, J. 1986. *Basic Concepts in Population, Quantitative, and Evolutionary Genetics*. New York: Freeman.

Davis, L. 1985. Job Shop Scheduling with Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms ICGA-85*, 136-140.

Dueck, G., and Scheuer, T. 1990. Threshold Accepting; a General Purpose Optimization Algorithm. *Journal of Computational Physics* 90:161-175.

Glover, F.; Taillard, E.; and Werra, D. D. 1993. A User's Guide to Tabu Search. *Annals of Operations Research* 41:3-28.

Goldberg, D., and Lingle, Jr., R. 1985. Alleles, Loci, and the Traveling Salesman Problem. In *Proceedings of the 1st International Conference on Genetic Algorithms ICGA-85*, 154-159.

Gorges-Schleuter, M. 1989. ASPARAGOS, An Asynchronous Parallel Genetic Optimization Strategy. In *Proceedings of the 3rd International conference on Genetic Algorithms, ICGA-89*, 422-427.

Grefenstette, J.; Gopal, R.; Rosmaita, B.; and van Gucht, D. 1985. Genetic Algorithms for the Traveling Salesman Problem. In *Proceedings of the 1st International Conference on Genetic Algorithms ICGA-85*, 160-168.

Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

Kirkpatrick, S.; Gelatt Jr., C.; and Vecchi, M. 1983. Optimization by Simulated Annealing. *Science* 220:671-680.

Manderick, B., and Spiessens, P. 1989. Fine-Grained Parallel Genetic Algorithms. In *Proceedings of the 3rd International conference on Genetic Algorithms, ICGA-89*, 428-433.

Oliver, I.; Smith, D.; and Holland, J. 1987. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In *Proceedings of the 2nd International Conference on Genetic Algorithms ICGA-87*, 224-230.

Petty, C.; Leuze, M.; and Grefenstette, J. 1987. A Parallel Genetic Algorithm. In *Proceedings of the 2nd International Conference on Genetic Algorithms ICGA-87*, 155-161.

Starkweather, T.; McDaniel, S.; Mathias, K.; Whitley, D.; and Whitley, C. 1991. A Comparison of Genetic Sequencing Operators. In *Proceedings of the 4th International Conference on Genetic Algorithms ICGA-91*, 69-76.

Syswerda, G. 1990. Schedule Optimization Using Genetic Algorithms. In Davis, L., ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.

Whitley, D.; Starkweather, T.; and Fuquay, D. 1989. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms ICGA-89*, 133–140.

Wiese, K., and Goodwin, S. D. 1997. A New Selection Strategy for Genetic Algorithms That Yields better Results. In *Workshop on Constraints and Bioinformatics/Biocomputing, Constraint Programming CP97*. <http://www.soi.city.ac.uk/~drg/cp97-workshop.html>.

Wiese, K., and Goodwin, S. D. 1998a. Keep-Best Reproduction: A Selection Strategy for Genetic Algorithms. In *Proceedings of the ACM Symposium on Applied Computing SAC'98*.

Wiese, K., and Goodwin, S. D. 1998b. The Effect of Genetic Operator Probabilities and Selection Strategies on the Performance of a Genetic Algorithm. In *Proceedings of Canadian AI'98*.

Wiese, K., and Goodwin, S. D. 1998c. A Study of Keep-Best Reproduction under Different Genetic Sequencing Operators. In preparation.