Syracuse University
SURFACE

Electrical Engineering and Computer Science Technical Reports | College of Engineering and Computer Science

1-1993

Genetic Algorithms for Stochastic Flow Shop No Wait Scheduling

Harpal Maini
Syracuse University, School of Computer and Information Science, hsmaini@top.cis.syr.edu

Ubirajara R. Ferreira

Follow this and additional works at: https://surface.syr.edu/eecs_techreports

Part of the Computer Sciences Commons

Recommended Citation
Maini, Harpal and Ferreira, Ubirajara R., "Genetic Algorithms for Stochastic Flow Shop No Wait Scheduling" (1993). *Electrical Engineering and Computer Science Technical Reports*. 164.
https://surface.syr.edu/eecs_techreports/164

This Report is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in Electrical Engineering and Computer Science Technical Reports by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.

SU-CIS-93-21

Genetic Algorithms for Stochastic Flow Shop No Wait Scheduling

Harpal Maini and Ubirajara R. Ferreira

January 1993

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, NY 13244-4100

Genetic Algorithms for Stochastic Flow Shop No Wait Scheduling

Harpal Maini

Syracuse University, School of Computer and Information Science

Syracuse, NY, USA

&

Ubirajara R. Ferreira

UNESP, State University of Sao Paulo - Engineering School

Guaratingueta', SP, BRAZIL

This work was done at Syracuse University, School of Computer and Information Science, Syracuse, NY, USA with support from CNPq and FUNDUNESP, Brazil.

Contents

1 Abstract 2

2 Problem Definition 2

2.1 Scheduling 2

2.2 Flow Shop No Wait Scheduling 2

2.3 Stochastic Hypothesis 3

2.4 Start Interval Concept 4

3 Genetic Algorithms 6

3.1 A Sequential Genetic Algorithm for Stochastic Flow Shop No Wait Scheduling . 7

3.2 Parallelising the Genetic Algorithm & Design Issues 8

3.2.1 Inherent Parallelism 8

3.2.2 Parallel and Distributed Genetic Algorithms 8

3.2.3 Design Issues 9

3.2.4 A HyperCube Based Parallel Genetic Algorithm for Stochastic Flow Shop No Wait Scheduling 12

3.3 Computation Complexity Comparison of the Parallel and Sequential Genetic Algorithms 14

3.3.1 Speedup 15

4 Conclusions 15

1 Abstract

In this paper we present Genetic Algorithms - evolutionary algorithms based on an analogy with natural selection and survival of the fittest – applied to an NP Complete combinatorial optimization problem: minimizing the makespan of a Stochastic Flow Shop No Wait (FSNW) schedule. This is an important optimization criteria in real-world situations and the problem itself is of practical significance. We restrict our applications to the three machine flow shop no wait problem which is known to be NP complete. The stochastic hypothesis is that the processing times of jobs are described by normally distributed random variables. We discuss how this problem may be translated into a TSP problem by using the start interval concept. Genetic algorithms, both sequential and parallel are then applied to search the solution space and we present the algorithms and empirical results.

2 Problem Definition

2.1 Scheduling

Scheduling models can be described by considering in sequence the resources, tasks, sequencing constraints and performance measures. In a general form, we define machine scheduling as follows:

Data: A set of tasks to be executed and a set of machines that can execute these tasks.

Question: What machine will execute which task and when will each task be executed in order to satisfy an optimization criterion?

A schedule can be defined as:

$$S : T \times M \longrightarrow J$$
$$([t_k, t_l], M_i) \longrightarrow J_j$$

where $J = (J_1, ..., J_n)$ is a set of tasks and $M = (M_1, ..., M_m)$ is a set of machines.

2.2 Flow Shop No Wait Scheduling

The flow shop condition means that:

- The tasks of a job must be processed sequentially and the tasks' execution order is the same for all jobs.
- There is one machine for each task.
- A job is processed using one machine at a time without preemption and a machine must process exactly one job at a time.

The no wait condition means that:

- A job must be processed from start to finish, without any interruption either on or between machines. This means that delays between tasks of the same job are not allowed.

Clearly, in the case of flow shop no wait, the schedule is determined by a sequence of jobs.

If needed, square brackets will be used to denote the position in a sequence in a schedule. Thus $[i]$ denotes the job that is in ith position in the sequence, and $t[i]$ would denote the processing time of that job. When the issue is not the position, w.l.o.g. we will consider $t[i] = t_i$.

The *Objective* is to minimize the makespan c_{max}, where the makespan is the maximum flow time for all jobs, that is, the total amount of time required to completely process all the jobs.

For our purpose, we will use as reference, the NP-Complete 3 machine flow shop no wait scheduling problem [5].

2.3 Stochastic Hypothesis

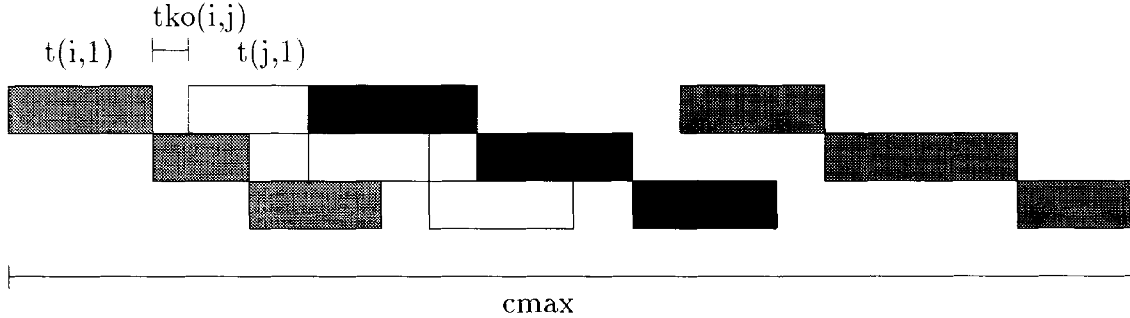
In a general form, a stochastic scheduling model consists of a scheduling model where we consider the duration of tasks, or other characteristics to be randomly distributed according to some probability function.

In our study, the stochastic hypothesis is considering the processing time of job i on machine j to be randomly distributed, according to some continuous and independent distribution functions $T_{ij}, i = 1, ..., n, j = 1, 2, 3$. The objective is to minimize the expected value of the makespan: min $E(c_{max})$.

2.4 Start Interval Concept

In order to give an analytical treatment to the problem, we use the Start Interval Concept, as presented by Ferreira [4].

Let $t_{k0}(i,j)$ be the interval of time that needs to occur between two sequential jobs at the first machine in a Flow Shop No Wait schedule, by processing job i and j in this order.



Hence, the Start Interval is the minimum time necessary to prevent a job from catching up with its immediate predecessor on a downstream machine.

Using this concept, two important matter for a stochastic flow shop no wait schedule are the processing sequence on first machine and the length of the Start Intervals to insert between the jobs in sequence.

Proposition: For a 3-machine stochastic scheduling problem, the constraints flow shop no wait are satisfied

$$\iff \quad \begin{cases} t_{k0} \geq 0 \\ t_{k0} \geq t_{k-1,2} - t_{k,1} & \text{for } k = 2, \ldots, n. \\ t_{k0} \geq t_{k-1,2} + t_{k-1,3} - t_{k,1} - t_{k,2} \end{cases}$$

The proof of this proposition is based on analysing the possible crashes between tasks in machines 1, 2 and 3 [4].

Let $t_{k0}(i,j)$ be the Start Interval required for processing job j after i in sequence, for i number of jobs and j number of machines.

The makespan, c_{max} , is :

$$min c_{max} = min\{\sum_{i=1}^{n-1} t_{k0}([i],[i+1]) + \sum_{i=1}^{n-1} t[i],1 + \sum_{i=1}^{m} t[n],j\}$$

$$= min\{\sum_{i=1}^{n-1} t_{k0}([i],[i+1]) + \sum_{j=2}^{m} t[n],j\} + \sum_{i=1}^{n} t[i],1$$

Since the sum of the processing times at first machine is constant, regardless of sequence, makespan is minimized by minimizing the sum of the
Start Interval Times + Processing Times of the Last Job on all but the first machine in sequence.

The task of minimizing the sum of Start Interval times is analogous to a classic combinatorial problem known as the TSP, Traveling Salesperson Problem.

Given a list of cities $1, 2, ..., n$ and costs c_{ij} for traveling between all pairs of cities, the TSP involves specifying a minimun-cost tour that visits each city once and returns to the starting point.

Clearly each job in the Scheduling problem corresponds to a city; the Start Interval time, $t_{k0}(i,j)$, from one job to another corresponds to the costs of travel between cities, and a schedule of the n jobs corresponds to a "tour" for the Salesperson.

Therefore, for the stochastic flow shop no wait scheduling, we have an equivalent TSP formulation and using our stochastic flow shop no wait scheduling model we research the application of genetic algorithms as they have been applied to TSP and other combinatorial optimization problems [3].

In order to get the $t_{k0}(i,j)$ values, in the particular case where the random variables are independent and normally distributed, $t_{k0}(i,j)$ can be determined by numerical integration in function of the probability of processing all jobs with the flow shop no wait condition satisfied [4]. In this work, consider $t_{u,v}$ as a random variable processing time of job u at a machine v with

. expected value $\mu_{u,v} = E[t_{u,v}]$ and

. standard deviation $\sigma_{u,v} = \sigma[t_{u,v}]$

. $t_{k0}(i,j)$ the delay between jobs i and j in this order.

As a consequence of the last proposition, we generate random values for $\mu_{u,v}$ and $\sigma_{u,v}$ in two real intervals and, for i number of jobs and j number of machines, take the maximum between the inequalities:

$$\begin{cases} t_{k0} \geq 0 \\ t_{k0} \geq (\mu_{i,2} + k\sigma_{i,2}) - (\mu_{j,1} - k\sigma_{j,1}) \\ T_{K0} \geq (\mu_{i,2} + k\sigma_{i,2} + (\mu_{i,3} + k\sigma_{i,3}) - [(\mu_{j,1} - k\sigma_{j,1}) + (\mu_{j,2} - k\sigma_{j,2})] \end{cases}$$

to generate $t_{ko}(i,j)$ values.

3 Genetic Algorithms

Genetic algorithms are emerging as an important tool in a programmers' armoury for searching a large collection of potential answers for the best one. In particular, for combinatorial optimization problems from the class of NP-Complete problems. These problems have a large number of possible solutions but do not have efficient techniques for finding the optimal one.

Genetic algorithms are a class of general purpose search strategies. The power of genetic algorithms is derived from the assumption that the best solution will be found in regions of the search space containing relatively high proportions of good solutions, and that these regions can be identified by a judicious and robust sampling of the search space. John Holland, who is the founder of this field, elaborated on two themes: the ability of simple representations to encode complicated structures and the power of simple transformations to improve on such structures. He suggested that a rapid improvement of say, bit strings, could occur under certain transformations, so that a population of bit strings could evolve. The underlying theme is that of *natural evolution*. Each species is searching for beneficial adaptions to a changing environment. The knowledge that a species has gained is embodied in its structural make up. Borrowing on this, genetic algorithms code members of a species, say possible solutions to a flow shop no wait scheduling problem, as genotypes where an allele represents a job in the schedule. These schedules are then allowed to evolve, using some methods of allele recombination, under a competitive environment to yield fitter (better) schedules. They can thus be viewed as a suboptimal optimization strategy.

In this paper, we present genetic algorithms and their implementation on sequential and parallel machines, for three machine stochastic flow shop no wait scheduling. In this model, the processing times are known only in the shop and the formulation is based on the start interval

which takes place between two consecutives jobs to guarantee the condition FSNW.

Solutions to the problem, schedules, are encoded as genotypes in a population that change over the course of generations. Reproduction operators drive the search towards areas of the search space where there is likely to be a proliferation of good solutions. Operators such as crossover and mutation are used. Crossover involves combining beneficial genetic information from two or more individuals to create another individual. This may, for example, be done by combining some of the jobs on one schedule with those from another schedule. Mutation is a localized change to a genotype. For example, swapping two jobs on a schedule. With crossover, beneficial mutations on two parents can be combined immediately upon reproduction. If the most successful parents reproduce more often than the less successful parents, and crossover occurs, then the probability becomes high that this will happen.

In this section we present sequential and parallel genetic algorithms and experimental results on the solution quality obtained by using them. We also show the performance of a proposed crossover operator in both the sequential and parallel cases.

3.1 A Sequential Genetic Algorithm for Stochastic Flow Shop No Wait Scheduling

A description of the sequential genetic algorithm designed follows:

- randomly generate a set of popsize number of schedules.
- compute their fitness and rank them using a linear ranking scheme.
- repeat the following steps num generations number of times.
  - select individuals for reproduction.
  - repeat the following steps popsize number of times
    - pick two individuals and apply crossover(with some probability) to produce an offspring.
  - apply mutation(with some probability) and hill climbing on the offspring.

– replace the old population with the newly generated one. Copy fittest(old population, new population) to the new population.

Important design criterion are: *population representation, population size, reproduction operators, probabilities, fitness evaluation and replacement criterion.* We address some of these issues in this presentation and others are slated for future research.

3.2 Parallelising the Genetic Algorithm & Design Issues

3.2.1 Inherent Parallelism

The sequence of populations generated by the algorithm constitutes a search trajectory through the space of all possible answers - the search being steered towards individuals or regions of the search space with better solutions. Consider a particular set of regions in the search space called Hyperplanes or Schema. A schema is a set of all strings having certain defining values at designated positions in the string. By formally treating each schema as a random variable whose mean is estimated by the average fitness of its instances in the population, genetic algorithms can be mathematically analysed. The key idea of the Schema Theorem is that manipulating a few strings results in sampling several schema. This is known as *Inherent or Intrinsic Parallelism.* These ideas were first formalized by Holland, [1].

3.2.2 Parallel and Distributed Genetic Algorithms

As discussed above genetic algorithms are intrinsically parallel in the way they search. In addition, they are also amenable to an implementation on a parallel or distributed system. This combines the hardware speed of parallel processors and the software speed of intelligent parallel search. The main limitation of a search strategy is its computational complexity. Parallel processing can significantly increase the number of search tree nodes evaluated in a given amount of time. There are two different approaches to parallelising search - one is to parallelize the processing of individual nodes, the speedup being limited to the degree of parallelism available in the evaluation and generation of new nodes. The other approach is *search space decomposition.* Here different processors are assigned different parts of the search space(subpopulations).

Parallel and distributed implementations follow largely the latter strategy with some differences between different implementations in the processor synchronisation strategy and the local search mechanism. The population can be fragmented and assigned to different processors, each of which runs something like a sequential genetic algorithm. *This is the approach we followed in parallelising the genetic algorithm.* We describe it in more detail later. Massively parallel implementations on machines such as the CM2 and CM5 also permit parallelising the processing of individual search tree nodes, to some extent.

Issues that justify parallelising a genetic algorithm

- Speedup.
- Possible to apply different genetic operators and parameters at different processors.
- Maintains diversity as per evolutionary theories.
- Possible to perform a multi-modal search, i.e. locate more than one solution to an optimization problem.

3.2.3 Design Issues

The most common technique of *population representation* is as strings of binary or real numbers. John Koza has researched representing individuals as LISP S-expressions, a method which is coming to be known as Genetic Programming.

We represent indivuals as a sequences of integers, where each allele indicates the job to be processed.

The *initial population* can be generated randomly or as the output of a greedy algorithm or some other heuristic. In practice it has been observed that seeding the initial population should be done with care as allele loss and hence premature convergence can result from this. We assign the initial population randomly.

Fitness Evaluation is critical is genetic algorithm design. It is commonplace to use the objective function of a combinatorial optimization problem as the fitness function in a genetic algorithm. In function optimization problems the function itself may be used as the fitness measure. The fitness function may be dynamically varying, i.e. a function of time. It is also

important to consider the computational cost of a fitness function. In our case we use the makespan, c_{max} as the fitness function.

The *crossover operator* is the primary tool in sampling schemata in proportion to their relative fitness. Problems where genotypes are represented as bit strings can be tackled using the traditional 1pt and 2pt crossover operators used by Holland. Obviously, these cannot be used for TSP like problems, for which Grefenstette and others have suggested several crossover operators that produce offspring that represent legal schedules.

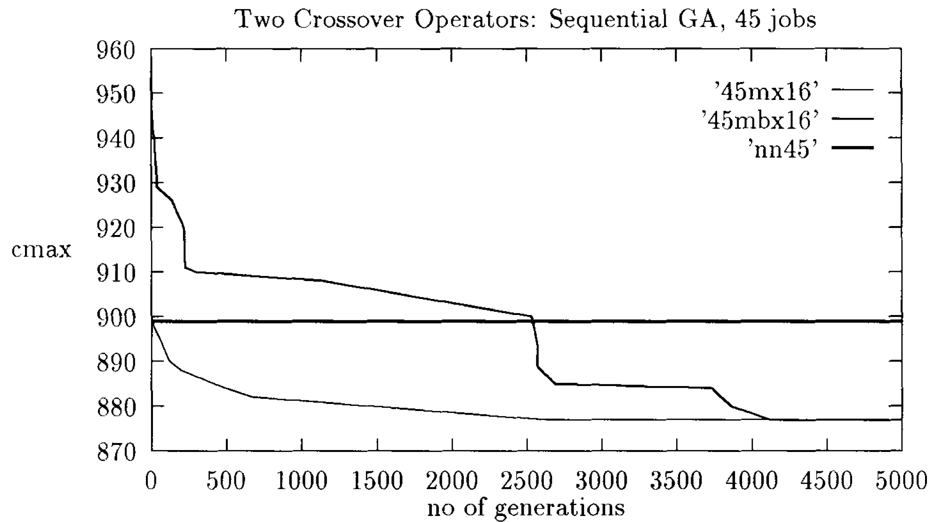
We have experimented with the following two crossover operators:

Greffenstette's XOVER operator (x)

- randomly choose a job as the current job for the offspring schedule.
- consider the four edges incident to the current job based on edge costs such that the probability associated with an edge incident to a previously visited job is 0.
- select an edge based on this distribution - if none of the parental edges leads to an unvisited job, create an edge to a randomly chosen unvisited job.
- repeat until all cities have been visited.

Branch and Bound XOVER (bx)

- pick a starting job at random.
- find the cities following this one in each of its parents.
- find the cost of choosing either of these cities to follow the current job on the schedule.
- find the cost of not choosing either of these cities to follow the current job on the schedule. i.e. find the least cost of choosing some other job following this one.
- assign a probability of selection based on these costs such that a previously visited job has a zero probability of selection.
- choose a job to follow the current one based on this distribution.
- repeat until all cities have been visited.



The *mutation operator* in use is adapted from Kernighan and Lin's 2-Opt heuristic, and involves swapping a pair of adjacent cities on a schedule. We intend to research other k-Opt heuristics as mutation operators.

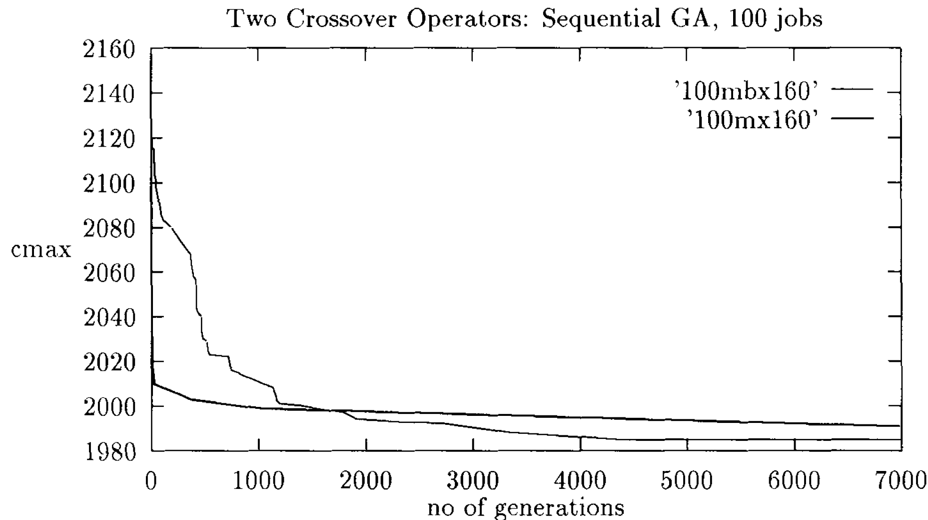
The *hill climbing* strategy in use is the following:

- repeat for all edges in the schedule;
- reverse the edge if this yields a fitter schedule.

This yields a locally optimal schedule.

The replacement strategy in use is *elitist replacement* where the fittest individual from the preceding generation survives into the next one.

Results by running the sequential implementation for 45-jobs and 100-jobs problems are shown in the following graphs, representing 'fitness' X 'number of generations'. In all instances, we use 0.01 and 0.7 as probabilities for mutation and crossover, also the mean and deviation were randomly generated in the intervals [10 , 20] and [1 , 3] respectively. We can see the behavior of each operator and compare the results with a nearest neighbour heuristic solution. We see that the Branch and Bound crossover operator seems to yield a better solution quality.



3.2.4 A HyperCube Based Parallel Genetic Algorithm for Stochastic Flow Shop No Wait Scheduling

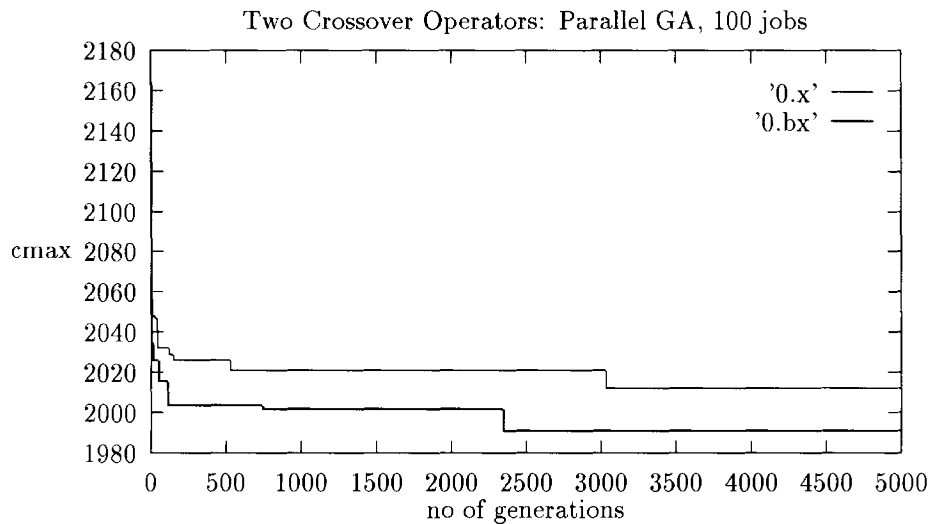
The parallel genetic algorithm we designed incorporates features of both the highly parallel and the distributed approaches. It incorporates the local neighbourhood structure of a massively parallel genetic algorithm and decomposes the population by assiging a subpopulation to each processor as in a distributed genetic algorithm. This exploits the "embarassingly parallel" nature of the problem(a convenient search space decomposition) and allows for mating to be restricted over a neighbourhood, thereby permitting allele diffusion over the population without too much genotype migration. It is important to note that the algorithm was designed to keep interprocessor communication costs low.

- Read Genetic Algorithm parameters and data from a Data file.
- Allocate a HyperCube of Dimension d, i.e. 2^d processors.
- Generate an initial population at each node, i.e. a total of d SubPopulations.
- repeat the following steps until a termination criterion is satisfied.
  - Compute the fitness of each Genotype and order Subpopulations according to fitness.
  - Each Subpopulation communicates its fittest Genotype to immediate neighbours. The Neighbourhood fittest is found.

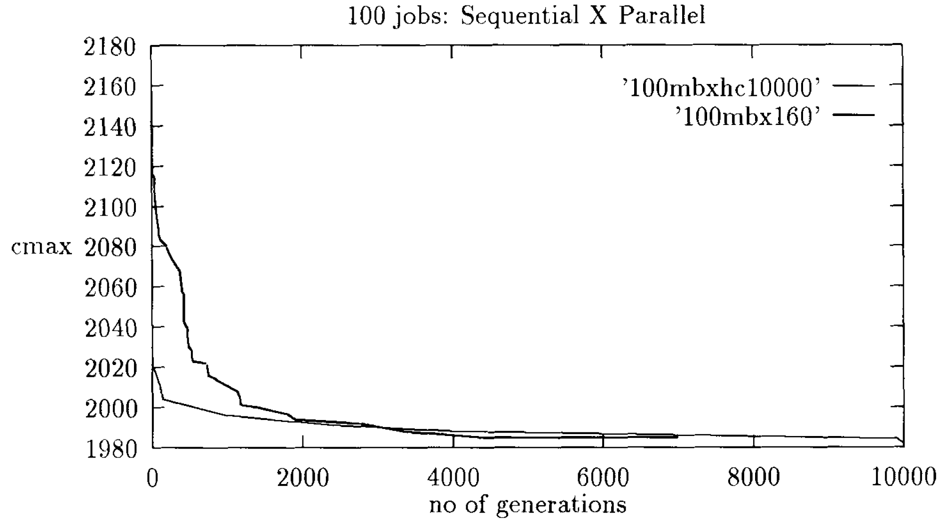
- Each Genotype in the Subpopulation mates with the neighbourhood fittest, or a locally selected mate with some probability. This generates a new generation.
- Individuals in the new generation perform Hill Climbing to improve their fitness and are mutated with some probability.
- Display individuals and their fitnesses.

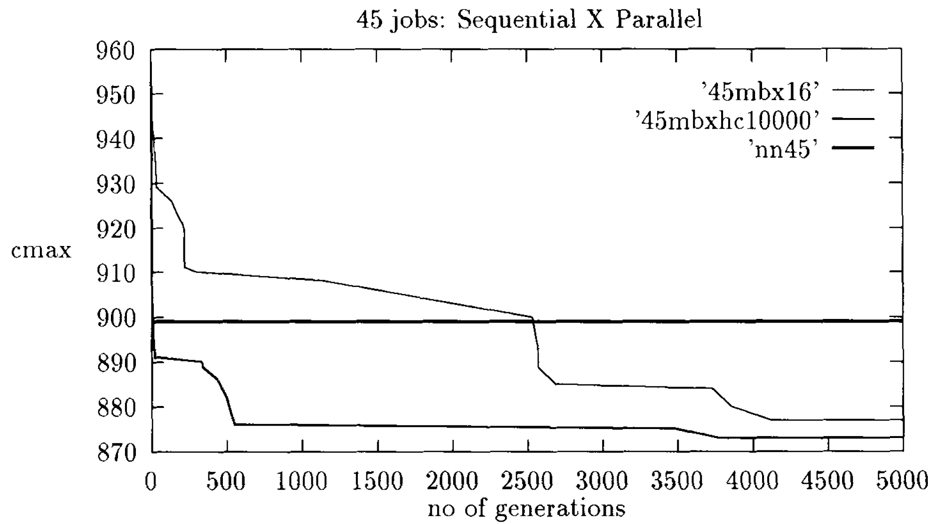
The following graphs some of the experimental results obtained by implementing this algorithm on a nCUBE2 parallel computer.

Comparing the two crossover operators for the 100-jobs problem, the Branch and Bround crossover operator seems to yield a better solution quality.



Using the same population size for both the sequential and parallel genetic algorithms we found that the parallel genetic algorithm yielded better solutions.





3.3 Computation Complexity Comparison of the Parallel and Sequential Genetic Algorithms

Let n = population size, s = subpopulation size, l = genotype length, p = number of subpopulations.

- Selection takes $O(n \log n)$ for the sequential and $O(s \log s)$ for the parallel algorithm.
- Crossover takes $O(nl^2)$ for the sequential and $O(sl^2)$ for the parallel algorithm.

- Mutation takes $O(nl)$ for the sequential and $O(sl)$ for the parallel algorithm.
- Thus the amount of work done is $O(n \log n + nl^2)$ for the sequential and $O(p(s \log s + sl^2))$ for the parallel algorithm.

Thus the amount of work, per generation, done by the parallel genetic algorithm is less than that done by the sequential genetic algorithm.

3.3.1 Speedup

We have found that the computation complexity is $O(s \log s + sl^2)$ for the parallel algorithm. If there are n individuals divided into p subpopulations this implies that $n = sp$ and that the total amount of work done is $O(p(s \log s + sl^2) = O(n \log s + nl^2)$ for the parallel genetic algorithm and $O(n \log n + nl^2)$ for the sequential genetic algorithm. This indicates that less computational work is done in a parallel genetic algorithm.

Also, speedup being measured as the ratio of time taken for a sequential algorithm to that of a parallel algorithm yields:

speedup$(p) = C_1(n \log n + nl^2)/(C_2(sp \log s + spl^2))+$ communication_cost(p)),

where C_1 and C_2 are constants and p is the number of processors used.

This algorithm was implemented on the nCube2 which has a message *latency* of about $150\mu secs$ and a *communication time* of under $1\mu sec$ per byte. In each iteration of the algorithm at worst about:

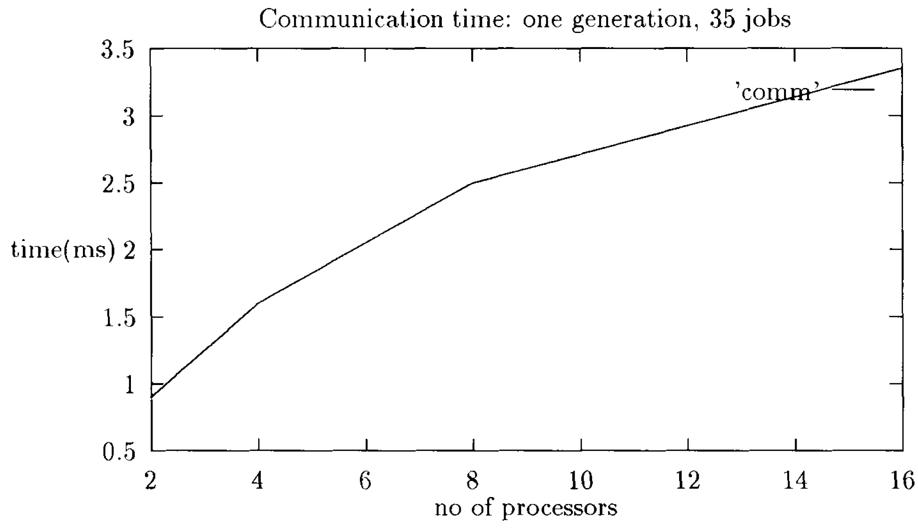
number_of_jobs $\times$ no_of_bytes_per_float $\times$ no_of_processors $\times$ degree_of_hypercube $\times$ μseconds_per_byte number of microseconds are expended in communication.

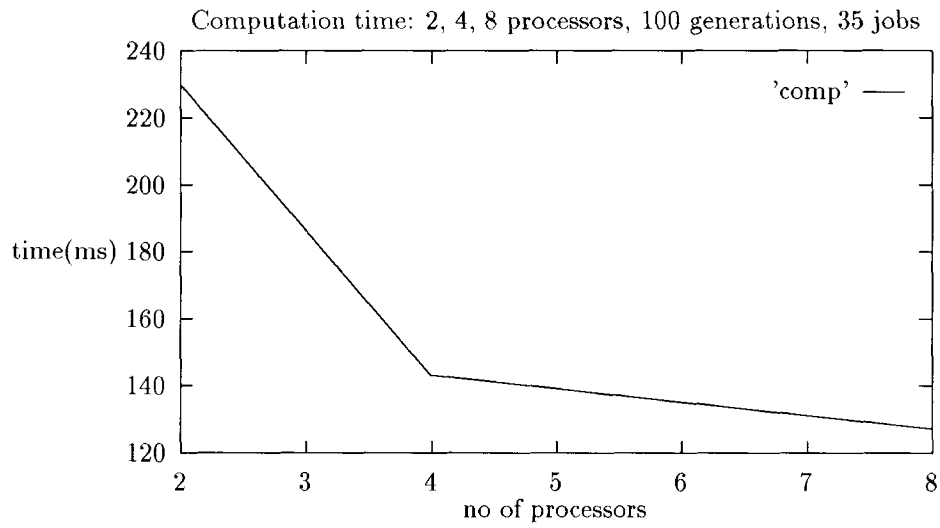
For a typical case of 35 job schedules on a hypercube of degree 3 this is about $3 milliseconds$ which compared with a computation time of about $1.3 secs$ is negligible:

4 Conclusions

This research has lead us to conclude the following:

- Genetic algorithms are an effective method of approximating solutions to stochastic flow shop scheduling.


Communication time: one generation, 35 jobs
'comm'
time(ms)
3.5
3
2.5
2
1.5
1
0.5
2
4
6
8
10
12
14
16
no of processors


Computation time: 2, 4, 8 processors, 100 generations, 35 jobs
'comp'
time(ms)
240
220
200
180
160
140
120
2
3
4
5
6
7
8
no of processors

- They can be effectively parallelized to yield a speedup in processing time and an improved solution quality.
- It is important to incorporate problem specific heuristics as part of the genetic search strategy. Either by way of a localized improvement such as hill climbing or through reproduction operators, or both. We found it very siginificant to solution quality in our experiments.

We plan to continue this research by developing better operators for this problem and devising a diversity measurement scheme. We also intend to compare the perormance of this algorithm on TSP problems with other "good" TSP heuristics, in particular: solution quality versus search time.

References

[1] J.H Holland *Adaption in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.

[2] Gregory J.E. Rawlins *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1991.

[3] J.J. Grefenstette *Incorporating problem specific knowledge into genetic algorithms* in Genetic Algorithms and Simulated Annealing, L. Davis, Morgan Kaufmann, 1987.

[4] U.R. Ferreira *Stochastic scheduling flow shop no wait problems* ITA, Sao Jose dos Campos, Tese de Doutorado. Sao Jose dos Campos, SP, Brazil, 1990.

[5] H. Rock *The three machine no wait flow shop problem is NP-complete* Journal of the Association for Computing Machinery 31, pp336-345, 1984.