The Application of a Distributed Genetic Algorithm to a Generic Scheduling System.

CSRP 455

M. McIlhagga and P. Husbands
School of Cognitive and Computing Sciences
University of Sussex
Falmer
Brighton
BN1 9QH

March 7, 1997

Abstract

This CSRP describes a Distributed Genetic Algorithm (DGA) which has been used to solve generic scheduling problems. The system is capable of allowing its user to define and solve any scheduling problem using a Scheduling Description Language (SDL), e.g. job-shop scheduling, time-tabling, resource sequencing etc. We will describe a unique encoding/decoding scheme that allows simple representation, straightforward chromosome recombination and fast schedule building and therefore evaluation times. A comparative study has been made of the DGA, random search and a heuristic method of scheduling using 100 very large scale problems; problems of the order of 500 tasks. This is the first study of its kind to look at problems of this scale. It was found that, although it is possible to reduce the makespan of a schedule by about $\frac{2}{5}$ of a randomly generated solution using dispatching rules, only the DGA produced solutions that had as high as a $\frac{3}{5}$ reduction.

1 Introduction

Scheduling in its various guises has been used by the GA community for a number of years to investigate the application of GAs to a challenging and important class of combinatorial optimisation problems. These studies have concentrated on specific scheduling problems or specific classes of scheduling problems [1]; [9]; [6]. This report describes the application of a Distributed GA to *the generic scheduling* problem (i.e. the entire class of scheduling problems). We have achieved this by formulating and implementing a framework for defining, simulating and solving scheduling problems in a generalised way.

The results reported here are based on preliminary testing of the system using 100 large scale problems in a comparison of three scheduling techniques: random search, dispatching rules (a heuristic technique) and a DGA. The problems were generated to reflect the underlying form of JSS that we have tackled previously [6], however they were scaled up to have approximately 50–100 times more schedulable tasks. We found that the random search and dispatching rules methods were able to reduce the makespan of a schedule (using the mean of 10 random solutions as a base for comparison) by about 40 percent. Whereas, the GA was, on average, able to reduce the makespan by 60 percent.

2 A Generic Scheduling System

MOGS (a Model for Optimisation of Generic Schedules) is a system design [5] developed to find near optimal schedules for any definable scheduling problem. We have implemented a large part of that model as a 'proof of principal' system; this we call SMOGS (Small MOGS). SMOGS implements all of the concepts in MOGS, but is limited in some functionality. Future work will include the full implementation of the system, eradicating certain inefficiencies in the system and implementing further system attributes to allow the solution of larger and more complex problems.

Section 2.1 details how we used a Scheduling Description Language (SDL[1]) to specify the problems that we are interested in. Section 2.2 outlines how SMOGS simulates the scheduling environment that the problem description (formulated in SDL) represents.

[1] It is not possible to fully describe our SDL here.

2.1 Problem Description

MOGS incorporates an SDL that enables the user to describe scheduling problems using a set of general concepts. The user's front-end is currently limited: it reads a problem description file containing a scheduling scenario using SDL. Future developments will investigate possibilities of *visual problem description*. Users may find it an easier and more powerful metaphor to use a visual programming tool to formulate scheduling problems.

SDL is a relatively simple declarative language (we plan to expand it giving the user more flexibility by including a macro language). It enables the user to describe their problem in terms of *tasks* and their *attributes*, *resources* and their attributes, *materials* and their attributes, material *flow rates*, material *release rates*, resource and material *location*, etc.

These terms are largely taken from the language of JSS, however there are some important differences: in MOGS these terms have an exact meaning. In JSS language the term *resource* is badly defined. It usually refers to materials supplied to machines or tools available to machines. Sometimes, however, people or even machines are seen as resources. The term machine is not used at all by MOGS (it uses the term *resource*) and the term *task* largely replaces the terms *job*, *batch* and *operation* (i.e. unit of work).

MOGS regarded each scheduling problem as a set of tasks that need to be completed. Resources are those things that complete tasks (machines, teams of people, individuals etc.). Each task has a varying number of attributes, e.g.: when it must be completed by, the required attributes of a resource capable of completing that task etc. Resources also have attributes: type of task attributes that they process etc. These attributes might be a machine operation such as drilling (JSS) or they might be a skill, such as experience with SQL data-base languages (human resource scheduling) or a rooms seating capacity (room timetabling).

MOGS also accounts for (or schedules) the supply and processing of *materials* to resources to compete tasks. In JSS terms this would include the supply of raw materials to machines, for example plastics or metal blanks. It would also include the supply and fitting of tool parts (these are regarded as a special case of material supply, i.e. reusable materials). Materials are supplied from reservoirs; represented as queues or stacks. These are filled according to data suppled by the user which reflects the actual supply of those materials, e.g. as deliveries from third parties or as supplied by the output of some resource. Because the completion of a task is the result of a resource acting on some materials to *produce* another material, it is possible to set up a hierarchical supply and demand situation between a number

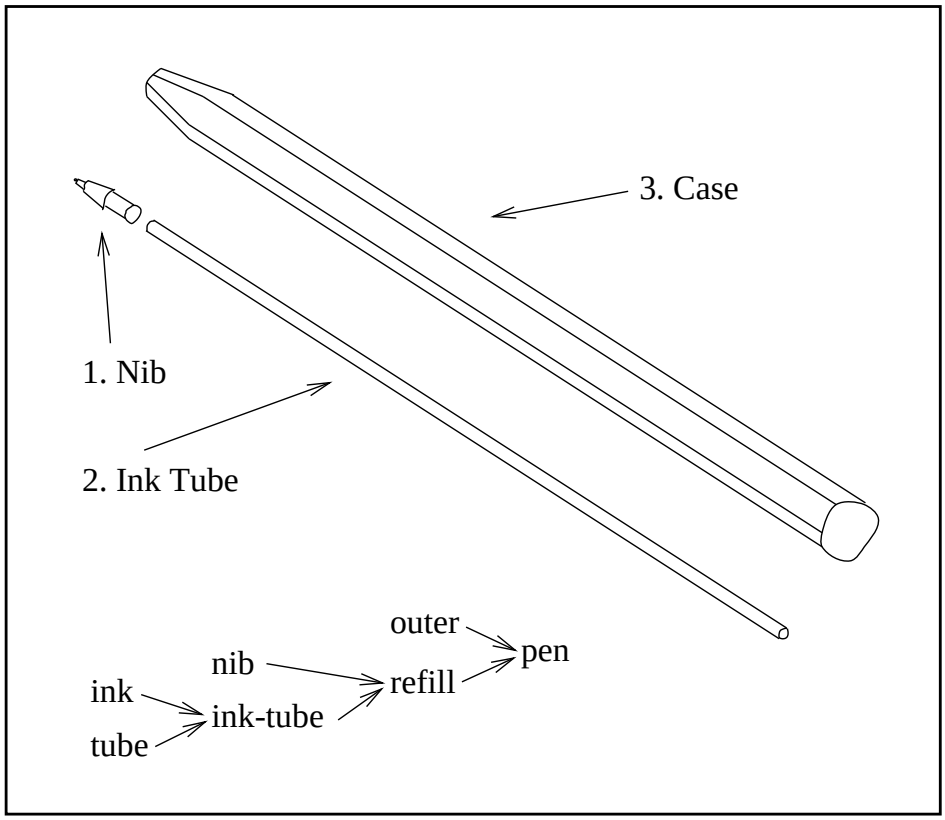

Figure 1: Assemblage Parts of a Ball Point Pen

of resources. This allows the user to implicitly define ordering constraints (they can also do this explicitly if they wish). For instance, an assembly situation can be described by simply stating the input/output relation between various tasks and SMOGS will derived the necessary constraints in the system.

An example of this might be the assembly of a ball-point pen from supplied parts as in figure 1. We can define the task of making an *ink tube* as injecting the ink into an empty tube. We can then define the task of making a refill as assembling a nib and an ink tube, giving the output *refill*. We can also define the task of pen assemblage as combining *refill* and *outer* to give the final output: *pen*. The tree diagram in figure 1 shows that there is a hierarchy of ordering constraints for this assembly process. In MOGS, because we model the use of *materials*, simply defining the tasks also defines

many of the necessary constraints in the problem.

This aspect of the system adds to the dimensionality of the problem. Although the problems we have used for this study are based on job-shop problems they also include the aspect of material supply: transport rate, stock control etc. This more accurately models the real scheduling problems that industry encounters. These problem are in the class of JSS problems, but are much harder than those usually tackled [8] because they have been reformulated to include material scheduling.

2.2 Objective Functions

SMOGS reads the problem description and creates a model of the scheduling environment. Candidate solutions are created by one of the search techniques available, presently: random search, dispatching rules or a DGA. One (or a combination) of a number of possible objective functions are used to determine the worthiness of that schedule. The objective function is a *discrete event simulation* which builds schedules by decoding chromosomes evolved by the DGA. These are mapped into a 'gantt chart' (cf. [2]) like data structure via a 'resource availability graph' (see section 2.3). The resource availability graph is built only once at the beginning of each run. At the present time the user can set the objective function to any one or user defined combination of: makespan, mean flow time, resource utilisation, proportion tardy, total tardy, total lateness, total earliness, mean earliness, mean 'on-timeness'. Other less JSS like objectives will be made available in later versions.

Total earliness is the sum of the amount of time each task is completed early (lateness not included). Mean earliness is total earliness divided by the number of tasks. Mean on-timeness is the mean (modulus) difference between the expected completion of tasks and their actual completion. All other terms are the more usual metrics found in the literature [2], [7], and are defined in table 1.

Given n tasks, $TA_j (1 \leq j \leq n)$. Where the tasks are to be carried out by r resources, RE_i, $(1 \leq i \leq r)$. For a given schedule TA_j there exists a number of useful features and statistics for that schedule, see table 1.

2.3 Resource Availability Graph

When the problem description is read from a file, a data structure know as the resource (availability) graph is created and modified according to the problem at hand. The resource graph is essentially a decision tree.

release date	RD_j
a due date	DD_j
a completion time	CT_j
flow time	$FT_j = CT_j - RD_j$
lateness	$LA_j = CT_j - DD_j$
tardiness	$TN_j = \max(0,LA_j)$.
Duration of TA_j on RE_i	Du_{ij}
makespan or total schedule time	$MS = CT_{max} - RD_{min}$
mean flow time	$MFT = sum(j=i, j=n, FT_j/n)$
total tardiness	$TTN = sum(j=1, j=n, TN_j)$
total lateness	$TLA = sum(j=1, j=n, LA_j)$
initial resource availability date	AD_i
resource utilisation	$RUi = sum(j=1,j=n, DU_{ij}/(MS-AD_i))$

Table 1: Cost Functions

Possible paths through which are defined by each legal chromosome. The tree limits the space of all possible solutions to the space of legal solutions: matching viable resources to each task and possible supplies of material to each resource. Each branch of the tree has a 'logical' index which is used to map chromosome genes to 'physical' resources, reservoirs and materials etc. The upshot of using this scheme is that all genes in all chromosomes are inevitably legal and the only search space that can be represented is a legal one. Moreover, aspects of the encoding (see section 4) allow random bit mutation and simple crossover to be applied without fear of corrupting the legality of the chromosome. This meant that a GA tool kit could be used to implement the search aspect of the system. Only the crossover function that combines the task orders had to be written specifically for this application.

3 Search Techniques Available

3.1 Random Search

The Random search implemented in this study works by finding 30,000 consecutive random solutions to each scheduling problem and storing the best as they are generated. To generate a random solution it was necessary to call the random number generator some 1500 times per schedule.

3.2 Dispatching Rules

Dispatching or priority rules are a popular heuristic used in constructing schedules in many types of scheduling problems [2]. Often used within simple constructive search algorithms to find the next operation to process, the most common are: SPT, Shortest Processing Time; FCFS, First Come First Served; MWKR, Most Work Remaining; LWKR, Least Work Remaining; MOPNR, Most Operations Remaining; RDM, Random.

As a heuristic technique for building schedules we currently use of a dispatching rule based on slack time (the difference between time remaining to due date and anticipated total process time). Whenever a resource becomes available, the task chosen to be processed next is the one with least slack time (LST). Slack time is a concept that applies to a task. In order to decide between candidate task/resource combinations, we make use of the SI (shortest imminent processing time) rule. Thus the general scheduling problem is tackled by using a LST rule to choose between tasks, and the SI rule to choose the task-resource combinations. This technique is one with which we have made comparisons before. It was first used by Khoshnevis

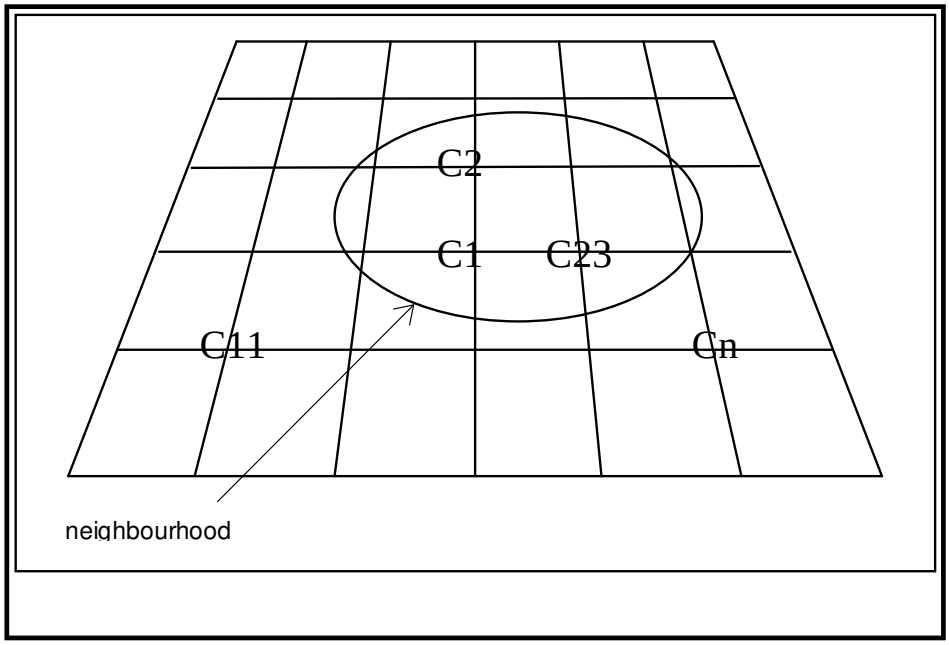

Figure 2: A Distributed GA Uses a 2D Geographical Grid for Breeding

and Chen [3] and later by Palmer [7]. Recently we compared Palmer's comparison of Khoshnevis and Chen's Dispatch Rules and Simulated Annealing with a distributed GA approach [6]. Further heuristics will be experimented with in later versions of the system, but these rules are applicable to a very wide range of problem types.

3.3 Genetic Algorithm

The Genetic algorithm uses a fairly standard distributed model where each chromosome has a 2D location on a toroidal grid (see figure 2).

The GA was parameterized in a similar way to previous work [6]. However, due to the relatively long evaluation time ($\approx$1 second per schedule) the population was reduced, from a more ideal size of 1600 chromosomes, to 300 chromosomes. Other parameters were:

- Parent selection: ranked local section with a neighbourhood of the 12 nearest. The nearest 12 chromosomes to an initial randomly selected parent are placed in a selection pool. The pool is ranked in fitness

order and a standard rank selection is applied to choose the second parent.

- Replacement strategy: replace worst parent. There is only a finite amount of memory allocated to the population. Each allocated slot for a chromosome is always in use. Creating a new chromosome means deleting one already in the population. The memory that the worst parent of the new chromosome occupies is over-written by its new offspring, however its geographical location may not be the same as its now deleted parents.

- Placement strategy: place new chromosomes 'random-local' to their parents. An empty random location is selected within the neighbourhood used to choose the second parent of the new chromosome. This becomes the new location of the child chromosome.

- Operators: crossover (see section 5) and mutation at a rate of 0.056 (this is the probability that **any** one bit in each 'gene' is flipped).

We used GPDGA[2], the Generic Parallel Distributed Genetic Algorithm tool kit to implement the evolutionary search aspect of this study [4].

4 Genetic Encoding

The chromosome is coded as an array of integers, as in table 2. It is split into three sections. First, the ordering of tasks is represented. Second, the resources to be allocated to each task are encoded. Finally, the reservoirs that will supply each of the materials required by the resource for each task are encoded. The chromosome is of a fixed length for any one problem, but varies between problems. The chromosomes is of a fixed length because there are a defined number of tasks in a problem and only one resource will be allocated to each tasks (i.e $2 \times T$ genes). However, the number of materials need for each task can vary, so it is not possible to determine the chromosome length from just a cursory look at the problem description. The third section maps an appropriate material supply reservoir for *each* material needed to complete each task.

If we say that T is the number of tasks in the problem and that each integer (probably 32 bits) is a gene then (refer to table 2):

[2]GPDGA was developed under the EPSRC funded project no GR/J 40812.

CHROMOSOME STRUCTURE																								
format	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	GG	GG	G	G	G	GGG	G
type	O	O	O	O	O	O	O	O	T	T	T	T	T	T	T	T	T	TT	TT	T	T	T	TTT	T
map	N	N	N	N	N	N	N	N																
label	task order								resources								reservoirs							
number	8								8								12							
bytes	32								32								48							

Table 2: Example Encoding of a Chromosome.

1. The first T genes contain integers in the range $0 < Tn < T$ denoting the ordering of tasks, i.e. how they are place on the gantt chart. The ordering not only defines the precedence of one task over another on a given resource, but also assigns precedence of material supply to tasks earlier in the order, (even though their time slot might be later than a task later in the order which is placed on a less utilised resource).

2. The next T genes in the chromosome denote the resources that tasks will use to complete. The format is such that the first resource is allocated to task 1 in the problem data structure, not the first task in the ordering defined in the first part of the chromosome.

3. The rest of the chromosome is dedicated to mapping which reservoir will supply what material to which task–reservoir pair. A reservoir can be thought of as a buffer that releases materials to (in the case of a job-shop, machines) the resources which process tasks. It is mapped out in the following way:

   for each task in the problem data structure
   and for each material required by that task
   there is a gene that denotes the reservoir
   that will supply the material.

Example 1: This very simple problem has 8 tasks, 1 of which requires 3 materials, 2 of which requires 2 materials, the rest requiring only 1. The chromosome structure would look like table 2.

4.1 Encoding Types

As table 2 suggests, the encoding for the first part of the chromosome is different from that of the second and third part. The O (table 2) or Order Type of encoding contains a number in the range 1 to the number of tasks (T) and the whole section contains 1 (and no more) of each number in that range. This encoding requires special crossover and mutation routines (see section 5). This section of the chromosomes represents the order and therefore the priority with which tasks are placed into the schedule. This process of constructing the schedule should not be confused with the actual scheduled times of each task. It is quite possible for the chromosome to place the task that is scheduled to start last onto the schedule data structure first; it simply monopolises its resource and, as a result, will not have to compromise the start and completion dates that it wanted.

The numbers represented in the second and third section of the chromosome do not map to resource and material indecies described by the user in SDL. Rather, they map to legal resources and reservoirs for each task in the *resource graph*, making each gene by definition legal. All bits are set in these genes. They are decoded by finding the remainder of an integer division of the gene value with the maximum number of legal resources or reservoirs for that task.

The T or Torrodal type was especially formulated to allow the representation of varying ranges of numbers using a blind encoding. That is, all T types can have any bit in the gene set and still maintain a legal encoding of any range of integer values (with the proviso that numbers are within the combinatorial range of the integer on that specific machine). This is achieved by making all binary values that the gene can hold legal and by taking the integer remainder of a division of the chromosome by the maximum value of its range (T) as the decoded value. This causes a minor problem for mutation, but one that is simply solved (see section 5). In order to implement this model a chromosome map or template must be maintained that describes the range of numbers in each gene (see Table 2). In this case the range of e.g. resources, denotes the list of viable resources for that task detailed in the resource availability graph data structure and **not** the list of all resources in the problem. Using this encoding with a graph of this type in the evaluation module forces any gene value to map to a resource that has the right attributes to enable it to carry out that task; illegal encodings are therefore not possible.

The upshot of using these representations and encoding is that the process of breeding and evaluation is very simple to implement and fast to ex-

ecute. Much of the schedule building work has been 'taken out of the loop' by building the resource availability graph and the chromosome 'range' map at the start of the run. Thus enabling the solution of larger problems. The real significance of this rather specific encoding is that it is not specific to any sub-class of *the generic scheduling problem*, but encodes for the whole class of problems.

5 Operators

Crossover in the first (type O) section of the chromosome works in the following way: a sub-string of one parent is found and inserted into the other parent once the items in the sub-string have been removed from the receiving parent. Sub-string length, position and insertion point are chosen at random.

Crossover for type T is more straight forward. It is implemented in the following way. N crossover points are chosen in the parent chromosomes. Often this will be one crossover point per chromosome, however this can be defined by the user. The section before the crossover point in parent A is concatenated with the section after the crossover point in parent B to form the new child chromosome.

Mutating the first section of the chromosome, which represent a unique ordering of tasks, is more problematic than bit mutation of the binary strings used for the second and third section of the chromosome. Mutation of an ordered set can take a number of forms. In all cases, the restriction that one of each of the numbers in the range $\{1, T\}$, where T is the number of genes in the ordered section of the chromosome, must hold. This can be achieved by implementing 1 or all of:

1. Swapping the order of two juxtaposed tasks in the chromosome.
2. Swapping the order of two task allocated the same resource.
3. Moving a higher priority task on Resource N to just after the juxtapose (lower priority) task on Resource N.

Mutation of type T genes can be done in two ways:

1. Mutate one or more bits. This can cause big or small changes in what any give gene represents and there is no way of knowing what that might be by. having the bit position in the gene. This is not a problem as this method is functionally equivalent to random mutation in the working bit range of a chromosome that is directly encoded.

2. When using a direct representation, random walk or Lower order mutation (near the LSB of gene) is equivalent to (e.g. for lower three bits) the addition of one of the following set of numbers chosen at random : $\{1, -1, 2, -2, -4, -4\}$. This sort of mutation can be implemented by decoding the chromosome and making the random addition and re-encoding the new value, taking into account the result of the integer division of the chromosome by the range (see section 4).

6 Results of Study

6.1 Description of Study

A comparison of random search, dispatch rules and a distributed GA as techniques for the solution of large scale general scheduling problems was made. The set of problems described below are modelled on standard JSS problems. However, they are much more than that. Because of the way MOGS models material flow a more natural representation of ordering constraints is present. Another difference between these and many JSS problems is that resources are not already allocated to tasks. These factors combined with the scale of the problems in terms of the number of tasks means that we are dealing with much harder problems than 'tradition' JSS.

The problem set that we used comprised 100 scheduling problems, i.e. 100 files containing a lengthly description of a problem using our SDL. Each problem had between 400 and 500 tasks (randomly distributed throughout the 100 problems).

The significant parameters that combine to define the search space of each problem are:

- The number of tasks, the number of resources capable of completing each task (typically 4-10 in this study).
- The number of reservoirs capable of supplying each of those resources (1,2 or 3 in this study).
- The number of attributes that each tasks has (upto 10).
- The number of input materials and material flow as defined by resource location, release rates and travel rate.

A typical search space is 1×10^{1500} solutions. These problems are only limited by the parameters of this study and are by no means the limit of

MAKESPAN PERCENTAGE DECREASE				
	Mean	Std. Dev.	Min	Max
Random	39.2580	14.0824	22.8734	62.4510
Heuristic	42.8347	12.0045	18.9035	64.9084
Evolutionary	60.0127	13.7439	19.2982	75.9534

Table 3: Statistic for Percentage differences in Makespan (all Techniques)

the system. Memory and processor speed allowing, it is possible to define a scheduling problem of any size — there is no practical upper limit on the number of tasks, resources, or materials etc., other than the type length of the variables used to store these values and the physical limits of the available computing resources.

6.2 Results

Table 3 shows all of the figures for percentage **decrease** of makespan (from the mean of ten random solutions) for all techniques: the comparative decrease made by the DGA is 17.178 percent over heuristic search and 20.7547 percent over random search.

The evolutionary technique produces decreases in makespan of 20.7547 percent over random search. This figure, as can be seen from Table 3, is a highly significant improvement on the heuristic (dispatch rule) approach. We also made a comparison of relative resource slack time between schedules generated by the different techniques. Slack time is simply the time that the machine is not utilised by a task. This varies slightly from resource utilisation, as a task may hold a resource unutilised as it waits for materials.

We found that the evolutionary approach could make better use of the resource efficiency attribute of each resource, and thereby leave less resources standing idle during the period that the schedule covers. Table 4 shows the differences between each of the techniques.

The reduction of resource slack time and therefore related (but not the same) resource (machine) utilisation is a significant indication of why the DGA was able to find better schedules than the other techniques.

MEAN PERCENTAGE RESOURCE SLACK TIME	
	Mean
Random	20.8234
Heuristic	18.7123
Evolutionary	12.5137

Table 4: Mean Percentage Slack Time for Each Resource

7 Conclusions

Because the mean improvement in makespan using random search (from a mean of 10 randomly generated solutions) is 39 percent, it would be foolish to entirely dismiss this technique.

The mean improvement in makespan using a heuristic scheduler is 42 percent. Although the figure may not appear to be that much better than random search, if the improvement were to be divided into computational time taken to find solutions we find that the heuristic method is exceptionally efficient.

It is the case that where the schedule is required before the more computationally intensive technique (the genetic algorithm) is capable of exceeding the quality of the solution found by heuristic means, that heuristic search will out perform all other techniques. However this time window is very small and in most cases we can take advantage of an evolutionary approach.

Of the techniques investigated, only the Genetic Algorithm can provide useful solutions with little resource slack time. Current literature does not contain any study where the problems are of this order of magnitude. Nor do any other systems use a completely generic approach to scheduling problems.

These problems are extremely difficult to tackle. The search spaces are of the order of 1×10^{1500} solutions where each solution takes up to a second to evaluate. We can only hope to search a very tiny percentage of the search space, so appropriate direction as to where to search is paramount. It is this sort of problem that distributed GAs seem to thrive on. Although the solutions found may be sub-optimal, there is now no doubt that powerful evolutionary search techniques, such as those developed for this system, are capable of producing solutions of a quality that far exceeds that of those

produced by the traditional techniques in place in industry today. This problem set has really stretched the limit of what is possible with current GA technology.

Future work will include a full implementation of the MOGS system design. Although this report has not covered the difference between the MOGS specification and the SMOGS implementation, various classes of scheduling problems can be easily specified using MOGS (which supports a hierarchical task structure) that are difficult to represent and solve using SMOGS. A full implementation of MOGS will also provide a faster and more effective problem modeller.

References

[1] L. Davis. Job-shop scheduling with genetic algorithms. In J. Grefenstette, editor, *Proc. Int. Conf. on GAs*, pages 136–140. Lawrence Erlbaum, 1985.

[2] S. French. *Sequencing and scheduling: an introduction to the mathematics of the job-shop*. Ellis Horwood, 1982.

[3] B. Khoshnevis and Q. Chen. Integration of process planning and scheduling functions. *Journal of Intelligent Manufacturing*, 1:165–176, 1990.

[4] M. McIlhagga. A generic encoding for scheduling problems. Technical report, University of Sussex, 1995.

[5] M. McIlhagga. Gpdga user documentation. Technical report, University of Sussex, 1995.

[6] M. McIlhagga, P. Husbands, and R. Ives. A comparison of simulated annealing, dispatching rules and a coevolutionary distributed genetic algorithm as optimization techniques for various integrated manufacturing planning problems. problem. In *Proceedings of PPSN IV*, volume LNCS, 1141, pages 604–613. Springer Verlag, 1996.

[7] G. Palmer. *An Integrated Approach to Manufacturing Planning*. PhD thesis, University of Huddersfield, 1994.

[8] K. Sycara, S. Roth, and M. Fox. Resource allocation in disributed factory scheduling. *IEEE Expert*, pages 29–40, Feb. 1991.

[9] H. Tamaki and Y. Nishikawa. Parallelled genetic algorithm based on a neighborhood model and its application to job-shop scheduling. In *Proceedings of PPSN II*. Springer Verlag, 1992.