N95- 23762

Scheduling With Genetic Algorithms*

Theron R. Fennel, A. J. Underbrink, Jr., and George P. W. Williams, Jr.
The Boeing Company, P.O. Box 240002, M/S JN-55, Huntsville, Alabama USA 35824-6402
1+ 205.461.5133 (voice) 1+ 205.461.3201 (fax)
{*randy, al, george*}@hsvaic.hv.boeing.com

INTRODUCTION

In many domains, scheduling a sequence of jobs is an important function contributing to the overall efficiency of the operation. At Boeing, we develop schedules for many different domains, including assembly of military and commercial aircraft, weapons systems, and space vehicles. Boeing is under contract to develop scheduling systems for the Space Station Payload Planning System (PPS) and Payload Operations and Integration Center (POIC). These applications require that we respect certain sequencing restrictions among the jobs to be scheduled while at the same time assigning resources to the jobs. We call this general problem *scheduling and resource allocation*.

Genetic algorithms (GAs) offer a search method that uses a *population* of solutions and benefits from *intrinsic parallelism* to search the problem space rapidly, producing near-optimal solutions [10, 7]. Good intermediate solutions are probabalistically recombined to produce better offspring (based upon some application specific measure of solution fitness, *e.g.*, minimum flowtime, or schedule completeness). Also, at any point in the search, any intermediate solution can be accepted as a final solution; allowing the search to proceed longer usually produces a better solution while terminating the search at virtually any time may yield an acceptable solution.

Many processess are constrained by restrictions of sequence among the individual jobs. For a specific job, other jobs must be completed beforehand. While there are obviously many other constraints on processes, it is these on which we focussed for this research: how to allocate crews to jobs while satisfying job precedence requirements and Personnel, tooling and fixture (or, more generally, *resource*) requirements.

*Copyright 1994, The Boeing Co., All rights reserved.

WHY A GENETIC ALGORITHM MAKES SENSE

There are a number of reasons why we wanted to explore using genetic algorithms for this scheduling work. While some existing approaches may suffice for basic scheduling, we were also interested in the possibility of global scheduling for complex processes and large assemblies. For example, Space Station experiment payloads that must be scheduled in a 90 day increment may number in the thousands; we cannot truly optimize an increment schedule by restricting our scope to a day or week. Therefore, a solution to our application requires the following characteristics:

- Evidence of scalability: There is considerable evidence that GAs have better scalability characteristics compared to other techniques commonly used for similar problems [14].
- Ease of parallelization: GAs broken into sub-populations with limited communication between them often exhibit super-linear speedup. This effect also has been shown in loosely coupled computers, communicating asynchronously over a network [18].
- Multi-objective optimization: we wanted to combine measures of schedule duration and completeness with resource utilization and task priorities.

OUR APPROACH

We developed a genetic algorithm which satisifies temporal constraints to produce near-optimal schedules with resources assigned to jobs. Our scheduler pre-processes the temporal constraints to eliminate implied or redundant constraints (*e.g.*, transitive constraints that may be specified explicitly) and evolves a population of schedules until termination criteria are met.

Constraint Preprocessing

There are two pre-processing steps before the GA-based scheduler is run:

1. First, we simplify the temporal and precedence constraints by removing redundancy and resolving obvious conflicts
2. Then we derive a partial ordering of the jobs similar to finding a critical path. This partial ordering is used for chromosome repair (see below) and can also establish a lower bound on the duration of the schedule.

Problem Encoding

The chromosomal encoding of schedules is a two-chromosome scheme [12]: one chromosome for the job sequence and one chromosome for the resource allocation. They are described as follows:

chromosome χ_0: An ordered sequence of jobs, coded as J job numbers.

chromosome χ_1: A set of binary coded fields, each of which represents the specific resource which will be used on the job associated with the field.

This encoding scheme effectively allows us to treat the job sequence and resource assignment as two subproblems. Each can be manipulated separately but optimized together.

The Genetic Plan

The term 'genetic plan' identifies the overall approach used for evolving populations of (genetically encoded) schedules. Our basic approach enlists a 'classical' Holland-style generational GA. We employ optional elitism, which is only engaged when the score of the best-ever schedule is not matched in the current generation.

We found ranking selection to be superior to the other techniques we tried with the most fit individual receiving $\sim$1.2 copies in the next generation. This rather low selection pressure was necessary to prevent premature convergence on some of the more difficult problems.

Our approach to genetic operator application treated reproduction, mutation, and recombination each as independent foreground operators, rather than making mutation a background operator which could potentially mutate the product of recombination and reproduction.

Genetic operations at the chromosome level were also kept independent. Once a decision was made to perform recombination or mutation, a second decision was then necessary to determine which chromosome (χ_0 or χ_1) should be manipulated. This decision was biased by the relative sizes of the chromosomes, *i.e.*, the longer chromosome was assigned a proportionally greater probability.

Genetic Operators

Since the genetic representation is distributed between two chromosomes with fundamentally different characteristics, different genetic operators were required for each chromosome. For the job-sequence chromosome (χ_0), the best recombination operator we found was the Partially Mapped Crossover (PMX) [8], though we also tried Random Respectful Recombination (R^3) [16], and Linear Order Crossover (LOX) [4]. For the resource-allocation chromosome (χ_1), the best recombination operator we found was Uniform Crossover (UX) [19], though we also tried conventional one- and two-point crossover. UX is generally considered to be quite disruptive, but since the ordering of fields in the resource chromosome does not attempt to group related fields (assuming this were even feasible), there is little locality to be preserved.

For the job-sequence chromosome, mutation swaps the alleles from two loci in the chromosome, where the first locus is the current locus and the second is either the next (adjacent) locus (50%) or another locus chosen randomly (50%). For the resource-allocation chromosome, mutation selects a random allele value, which effectively halves the mutation rate when compared to bit-flipping mutation.

The Schedule Builder

The schedule builder is responsible for decoding the chromosomes and converting them into a feasible schedule. The basic GA had a very difficult time finding *any* feasible solutions for highly-constrained scheduling problems. We therefore *enforced* feasibility in our schedules by minimally reordering jobs to accommodate precedence constraints.

Chromosome Repair

The basic idea behind chromosome repair is to use heuristic or algorithmic techniques to modify individual solutions and then to probabalistically modify the genetic information to incorporate these changes [4, 15]. In some respects, repair might be viewed as an intelligent mutation operator. Sometimes the repair corrects an illegal chromosome to make it legal (as in our schedule builder), while other times it simply improves a previous legal schedule. Our system implements both kinds of repair with variable degrees of probability, generally 5–10%. This has the effect of enriching the population with good partial solutions which can then be combined via crossover.

Since our implementation has two chromosomes (χ_0 and χ_1), we have at least two opportunities to implement chromosome repair.

Repairing the Resource Allocation Chromosome

This repair strategy ignores the previous genetic information from the resource allocation chromosome and determines a resource allocation from scratch. This is done using a greedy approach to incrementally allocate the best resources for each job, backtracking when there are conflicts preventing all the demands for the job from being satisfied.

Repairing the Job Sequence Chromosome

There are two 'levels' of repair for the job sequence chromosome. The first level repairs the chromosome to reflect the results of the schedule builder. The second level of repair is only invoked some fraction of the times the first level is invoked and causes the job sequence to be modified *before* the schedule builder is invoked. The second level repair is heuristic and simplifies the task of constructing a feasible schedule for the schedule builder (first level repair).

The nature of the second level repair is based on the partial order on the jobs from precedence and temporal constraints. This partial ordering specifies a start time for each job, which would produce a feasible schedule if adequate resources were available to satisfy any resource request. This assumption of (essentially) infinite resources has led us to call this partial ordering an 'infinite resource model' (IRM) of the schedule. When there are many precedence or temporal constraints, this IRM *may* contain a great deal of useful information, especially since highly constrained schedules are the most difficult ones for the GA to solve. Similarly, if there are few (or no) such constraints, the IRM doesn't help very much. But what help it does provide is exactly where the GA needs help, *i.e.*, in repositioning constrained jobs in the job sequence where they can be (feasibly) scheduled.

Schedule Evaluation

We explored a fairly large variety of composite evaluation functions. We defined several different evaluation criteria and finally settled on a particular combination which seems to work reasonably well for the problems we have tried. The individual criteria are separate, independently computable functions and their resulting values are combined by a higher level function which supports adjusting the weights of the individual criteria. The set of criteria in our final evaluation function are:

- Schedule Duration: The number of time units (*e.g.*, hours or minutes) scheduled to complete the jobs.
- Resource Utilization: The ratio of resource time scheduled to the schedule duration.
- Schedule Completeness: The ratio of jobs scheduled to the total number of jobs (*i.e.*, a legal schedule may not include all jobs).
- Priority: A weight score accumulating higher values for higher priority jobs.

FUTURE WORK

Considerable work remains before we can determine the true value of this approach to scheduling. A primary requirement for a better understanding would have to be more detailed comparisons against other algorithms, including a more elaborate set of benchmark tests. We would also like to implement this approach on a parallel architecture and test this implementation on some very large problems.

We would also like to explore the use of Pareto optimal selection strategies to better support multi-objective optimization. These are based on non-dominance of solutions and appear to better support multi-objective optimization. [5, 13]. Finally, we would like to compare our multiple-chromosome approach to a single chromosome implementation and determine the value (if any) of multiple chromosomes *per se*.

CONCLUSIONS

We developed a genetic algorithm for scheduling and resource allocation. We employed several interesting GA features, including a multiple-chromosome schedule encoding, multiple repair strategies, and several order-preserving operators.

A significant consequence of chromosome repair was that we found post-GA hill-climbing unnecessary. Since any improvements made via chromosome repair are then available to the GA, which can potentially improve upon them further, we opted to include these heuristic techniques in the chromosome repair strategies. Use of repair at higher probablilities leads to premature convergence of the population to relatively poor solutions, providing evidence that good solutions are not solely the result of repair.

In our tests, the scheduling algorithm creates schedules which are as good as or better than the results from a critical-path scheduler currently in use within the company. Additionally, the scheduler is able to schedule general resources more efficiently than the critical path scheduler.

Our limited test results encourage us to continue developing the genetic algorithm scheduler to include more schedule evaluation criteria. We also hope to explore the possibility of large-scale scheduling for manufacturing processes.

REFERENCES

[1] Richard K. Belew and Lashon B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, California, 1991. Morgan Kaufmann.

[2] D. E. Brown and C. C. White, editors. *Operations Research/Artifical Intelligence: The Integration of Problem Solving Strategies.* Kluwer Academic Publishers, Boston, 1990.

[3] Lawrence Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[4] E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation* [11], pages 824-829. Sacramento, California.

[5] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Forrest [6], pages 416-423.

[6] Stephanie Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, 1993. Morgan Kaufmann.

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, New York, 1989.

[8] David E. Goldberg and Robert Lingle, Jr. Alleles, loci, and the traveling salesman problem. In Grefenstette [9], pages 154-159.

[9] John J. Grefenstette, editor. *Proceedings of the First International Conference on Genetic Algorithms and their Applications.* Lawrence Erlbaum Associates, 1985.

[10] John H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, 1991. Revised from the 1975 edition.

[11] IEEE. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, April 1991. Sacramento, California.

[12] Kate Juliff. A multi-chromosome genetic algorithm for pallet loading. In Forrest [6], pages 467-473.

[13] Sushil J. Louis and Gregory J. E. Rawlins. Pareto optimality, ga-easiness and deception. In Forrest [6], pages 118-123.

[14] H. Mülenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In Belew and Booker [1], pages 271-278.

[15] David Orvosh and Lawrence Davis. Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints. In Forrest [6], page 650.

[16] Nicholas J. Radcliffe. Forma analysis and random respectful recombination. In Belew and Booker [1], pages 222-229.

[17] J. David Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California, 1989. Morgan Kaufmann.

[18] R. Shonkwiler. Parallel genetic algorithms. In Forrest [6], pages 199-205.

[19] Gilbert Syswerda. Uniform crossover in genetic algorithms. In Schaffer [17], pages 2-9.