Original Article

International Journal of Educational Research and Technology
P-ISSN 0976-4089; E-ISSN 2277-1557
IJERT: Volume 5 [4] December 2014: 25-34

ISO 9001: 2008 Certified Organization
Website: www.soeagra.com/ijert.html

Application of Genetic Algorithm for Resource-Constrained Scheduling

Mahdi Razi, Iman Razi
Industrial Engineering Department, Amirkabir University of Technology, Tehran, Iran
Industrial Engineering Department, Islamic Azad University, Firouz Kouh Branch, Ghom, Iran
Email: mahdirazi@chmail.ir

ABSTRACT
Goal of Scheduling problem is optimization of a function in deciding a set of activities that must be executed under different constraints. There are two main types of constraints including are precedences between activities, and the availability of finite resources.This research presents a genetic algorithm approach to resource-constrained scheduling using adirect, time-based representation. The new representation encodes schedule information as a dual array of relative delay timesand integer execution modes. This representation includes time-varying resourceavailabilities and requirements. The genetic algorithm adapts todynamic factors such as changes to the project plan or disturbances in the schedule execution.The genetic algorithm was applied to over 1000 small job shop and project scheduling problems(10-300 activities, 3-10 resource types). According to result based on computationally expensive, the algorithm performed fairly well on a wide variety of problems. In addition, the algorithm found solutions within 2% of published best in 60% of the project scheduling problems.The GA performed better than deterministic, bounded enumerative search methods for 10% of the 538 problems tested on project scheduling problems with multiple execution modes.
Keywords: *scheduling, optimization, operations research, procedural search, classification*

Received 02.09.2014 **Revised 12.11. 2014** **Accepted 18.11.2014**

How to cite this article: Mahdi Razi, Iman Razi. Application of Genetic Algorithm for Resource-Constrained Scheduling.Inter. J. Edu. Res. Technol. 5[4] 2014; 25-34.DOI: 10.15515/ijert.0976-4089.5.4.2534

INTRODUCTION

Scheduling consists in deciding when a set of activities must be executed under different constraints, in order to optimize a given objective. The two main types of constraints are precedences between activities, and the availability of finite resources.

Common objectives are to minimize the total duration or to minimize the weighted sum of the tardiness of activities with respect to given due-dates (Blum and Sampels, 2004). Scheduling problems are very varied, both in application domains and in featured constraints. Some typical applications are manufacture scheduling, construction scheduling, code optimization in compilers, and pharmaceutical project planning (Cambazard and Jussien, 2006).

Scheduling problems are optimization problems. Optimization problems may be defined in a constraint-oriented way. In this setting, a problem is defined by decision variables, constraints on the decision variables and an objective function defined on the decision variables. The decision variables are the unknowns of the problem that must be fixed (Birattari, 2009).

This document describes a genetic algorithm for finding optimal solutions to dynamic resourceconstrainedscheduling problems. Rather than requiring a different formulation for eachscheduling problem variation, a single algorithm provides promising performance on manydifferent instances of the general problem. Whereas traditional scheduling methods use search or scheduling rules (heuristics) specific to the project model or constraint formulation, thismethod uses a direct representation of schedules and a search algorithm that operates with noknowledge of the problem space. The representation enforces precedence constraints, and theobjective function measures both resource constraint violations and overall performance.

In its most general form, the resource-constrained scheduling problem asks the following:

Given a set of activities, a set of resources, and a measurement of performance, what is the bestway to assign the resources to the activities such that the performance is maximized? Thegeneral problem

encapsulates many variations such as the job-shop and flowshopproblems,production scheduling, and the resource-constrained project scheduling problem.
Scheduling requires the integration of many different kinds of data. Constructing a schedule requires models of processes, definition of relationships between tasksand resources, definition of objectives and performance measures, and the underlying datastructures and algorithms that tie them all together. Schedules assign resources to tasks (or tasksto resources) at specific times. Tasks (activities) may be anything from machining operations todevelopment of software modules. Resources include people, machines, and raw materials (Christopher Beck and Philippe Refalo, 2003).
Typical objectives include minimizing the duration of the project, maximizing the net presentvalue of the project, or minimizing the number of products that are delivered late.Planning and scheduling are distinctly different activities. The plan defines what must be doneand restrictions on how to do it, the schedule specifies both how and when it will be done. Theplan refers to the estimates of time and resource for each activity, as well as the precedencerelationships between activities and other constraints. The schedule refers to the temporalassignments of tasks and activities required for actual execution of the plan. In addition, anyproject includes a set of objectives used to measure the performance of the schedule and/or thefeasibility of the plan. The objectives determine the overall performance of the plan andschedule.
Scheduling problems are dynamic and are based on incomplete data. No schedule is static untilthe project is completed, and most plans change almost as soon as they are announced.
Depending on the duration of the project, the same may also be true for the objectives. Thedynamics may be due to poor estimates, incomplete data, or unanticipated disturbances. As aresult, finding an optimal schedule is often confounded not only by meeting existing constraintsbut also adapting to additional constraints and changes to the problem structure.
Genetic algorithms are a stochastic search method introduced in the 1970s in the United Statesby John Holland [Holland 76] and in Germany by Ingo Rechenberg. Based onsimplifications of natural evolutionary processes, genetic algorithms operate on a population ofsolutions rather than a single solution and employ heuristics such as selection, crossover, andmutation to evolve better solutions.

MATERIAL AND METHODS

This section describes the solution method in five parts: (1) the problem model in the form ofassumptions about tasks and resources with their associated constraints, (2) the search method,(3) the schedule representation, (4) the genetic operators specific to the representation, and (5) the implementation of objectives and constraints. The problem model determines the variationsof problems that can be solved, the problem representation determines the bounds of the search space, the genetic operators determine how the space can be traversed, and the objectives and constraints determine the shape of the search space.
The assumptions made when modeling a problem determine the variations of that problem thatthe model will support. The next three sections list assumptions about tasks, resources, andobjectives. The assumptions in the first two sections typically end up being constraints. Thethird section highlights some of the more common objectives that may be defined. Satisfactionof the constraints determines the feasibility of a solution, satisfaction of the objectives determines the optimality of a solution.

Assumptions About Objectives

Any objective measure can be used as long as it can be determined from a complete schedule.Examples of objectives include minimization of makespan, minimization of mean tardiness ofpart delivery times, maximization of net present value, and minimization of work-in-progress.Objectives may include more than one part. For example, minimization of the makespan maybe the primary objective, but only if it does not drive the cost above a certain threshold (Nowicki and Smutnicki, 2005).

Search Method

Genetic algorithms are a stochastic heuristic search method whose mechanisms are based uponsimplifications of evolutionary processes observed in Nature. Since they operate on more thanone solution at once, genetic algorithms are typically good at both the exploration and exploitationof the search space. Goldberg [Goldberg 89] provided a comprehensive description of the basicprinciples at work in genetic algorithms, and Michalewicz described many oftheimplementation details for using genetic algorithms with various data types.
Most genetic algorithms operate on a population of solutions rather than a single solution. Thegenetic search begins by initializing a population of individuals. Individual solutions, orgenomes, are selected from the population, then mate to form new solutions. The mating process,typically implemented by combining, or crossing over, genetic material from two parents to formthe genetic material for one or two

new solutions, confers the data from one generation ofsolutions to the next. Random mutation is applied periodically to promote diversity. If the newsolutions are better than those in the population, the individuals in the population are replacedby the new solutions. This process is illustrated in Figure 1.

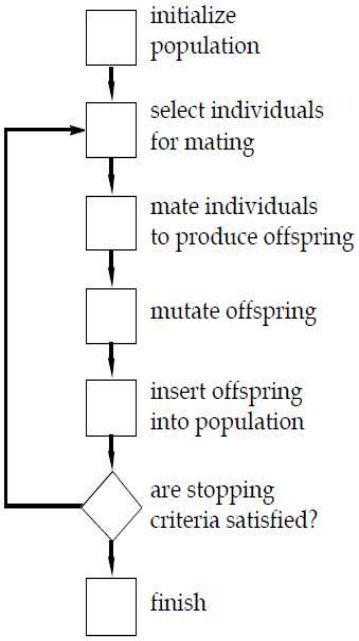

Figure 1 Generic genetic algorithmflowchart. Many variations are possible,from various selection algorithms to awide variety of representation-specificmating methods. Note that there is noobvious criterion for terminating thealgorithm. Number-of-generations orgoodness-of-solution are typically used

In traditional schedule optimization methods, the search algorithm is tightly coupled to theschedule generator. These methods operate in the problem space; they require informationabout the schedule in order to search for better schedules. Genetic algorithms operate in therepresentation space. They care only about the structure of a solution, not about what thatstructure represents. The performance of each solution is the only information the geneticalgorithm needs to guide its search. For example, a typical heuristic scheduler requiresinformation about the resources and constraints in order to decide which task should bescheduled next in order to build the schedule. The genetic algorithm, on the other hand, onlyneeds to know how a schedule is and how to combine two schedules to form anotherschedule.Having said that, many hybrid genetic algorithms exist which combine hill-climbing, repair,and other techniques which link the search to a specific problem space.

Proper choice of representation and tailoring of genetic operators is critical to the performance ofa genetic algorithm. Although the genetic algorithm actually controls selection and mating, therepresentation and genetic operators determine *how* these actions will take place. Many geneticalgorithms appear to be more robust than they actually are only because they are applied torelatively easy problems. When applied to problems whose search space is very large andwhere the ratio of the number of feasible solutions to the number of infeasible solutions is low,care must be taken to properly define the representation, operators, and objective function,otherwise the genetic algorithm will perform no better than a random search.

Some genetic algorithms introduce another operator to measure similarity between solutions inorder to maintain clusters of similar solutions. By maintaining diversity in the population, thealgorithms have a better chance of exploring the search space and avoid a common problem ofgenetic algorithms, *premature convergence*. After a population has evolved, all of the individualstypically end up with the same genetic composition; the individuals have *converged* to the samestructure. If the optimum has not been found, then the convergence is, by definition,premature. In most cases, further improvement is unlikely once the population has converged.

The similarity measure is often referred to as a *distance function*, and these genetic algorithmsare referred to as *speciating*or*niching* genetic algorithms. The similarity measure may be basedupon the data in the genome (genotype-based similarity), it may be based upon the genomeafter it has been

transformed into the problem space (phenotype-based similarity), or it mayintegrate some combination of these.
The steady-state genetic algorithm uses overlapping populations. In each generation, a portionof the population is replaced by the newly generated individuals. This process is illustrated inFigure 2. At one extreme, only one or two individuals may be replaced each generation (closeto 100% overlap). At the other extreme, the steady-state algorithm becomes a simple geneticalgorithm when the entire population is replaced (0% overlap).

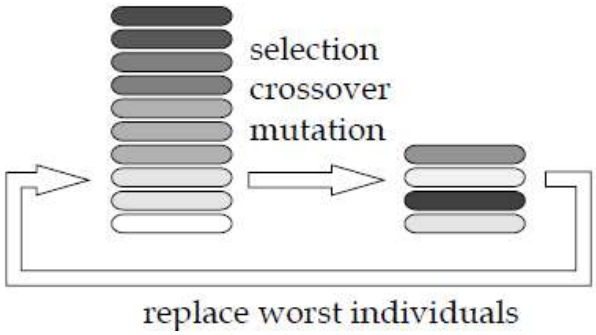

Figure 2TheÒsteady-stateÓ genetic algorithm. This algorithm uses overlappingpopulations; only a portion of the population is replaced each generation. Theamount of overlap (percentage of population that is replaced) may be specifiedwhen tuning the genetic algorithm.

The struggle genetic algorithm is similar to the steady-state genetic algorithm. However, ratherthan replacing the worst individual, a new individual replaces the individual most similar to it,but only if the new individual has a score better than that of the one to which it is most similar.This requires the definition of a measure of similarity (often referred to as a *distance function*).The similarity measure indicates how different two individuals are, either in terms of theiractual structure (the genotype) or of their characteristics in the problem-space (the phenotype).
The struggle genetic algorithm was developed by Gr.ninger in order to adaptively maintaindiversity among solutions. As noted previously, if allowed to evolve longenough, both the simple and the steady-state algorithms converge to a single solution;eventually the population consists of many copies of the same individual. Once the populationconverges in this manner, mutation is the only source of additional change. Conversely, apopulation evolving with a struggle algorithm maintains different solutions (as defined by thesimilarity measure) long after a simple or steady-state algorithm would have converged.Unlike other niching methods such as sharing or crowding (Brucker, 2004), the struggle algorithm requires no niching radius or otherparameters to tune the speciation performance.

Genetic Representation

Although much of the early genetic algorithm literature in the United States has focused on bitrepresentations (i.e. solutions were encoded as a series of 1s and 0s), genetic algorithms canoperate on any date type. In fact, most recent scheduling implementations use list-basedrepresentations. But whether the representation is a string of bits or a tree of instructions, anyrepresentation must have appropriate genetic operators defined for it. The representationdetermines the bounds of the search space, but the operators determine how the space can betraversed.
The following representation for scheduling is a minimal representation that can representresource-infeasible solutions. As shown in Figure 3, a genome consists of an array of relativestart times and an array of integer execution modes for each task. Each time represents theduration from the latest finish of all predecessor tasks to the start time of the corresponding task.
Each mode represents which of the possible execution modes will be used for the correspondingtask. As shown in the figure, the modes are typically defined in terms of resourcerequirements. This representation is *not* order-based. The elements in the array correspond tothe tasks in the work order or project plan, but the order of elements relative to each other isinsignificant. Each genome is a complete schedule; the genome directly represents a scheduleby encoding both start times (explicitly) and resource assignments (via the execution mode).

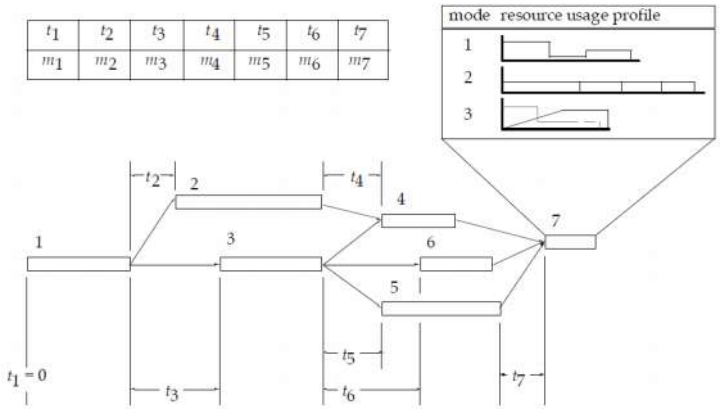

Figure 3The genome and its mapping to the schedule. A single genome is adouble array of floating-point start times and integer execution modes. Eachelement in the arrays corresponds to a task in the project plan or work order. Thetimes represent delay times relative to the estimated finish time of the predecessors.The execution modes vary from task to task and represent one of the possibleexecution modes for the corresponding task.

Genetic Operators

Use of a genetic algorithm requires the definition of initialization, crossover, and mutationoperators specific to the data type in the genome. In addition, a comparison operator must alsobe defined for use with niching/speciating genetic algorithms such as the struggle geneticalgorithm.

Initialization: The real number part of the genome was initialized with random numbers. The range ofpossible values was based upon the average estimated task durations. The magnitude of thenumbers matters because the algorithm finds better solutions faster if the random numbers arethe same order of magnitude as the task durations.

Crossover: The crossover operator included two parts, one for each data type in the genome. Blendcrossover, a real-number-based operator, was used for the array of time values. Uniformcrossover, a type-independent operator, was used for the array of execution modes.

Mutation: Mutation was performed by applying Gaussian noise to each element in the real number arrayand by flipping modes in the mode array. Themean is equal to the previous value. The deviation should be adaptive, but in the testsreported in this thesis, the deviation used to define the Gaussian curve was fixed.

Similarity Measure: The similarity function compares two solutions and returns a value that indicates how much thesolutions differ. Often called a 'distance' function, this operator is typically used by speciatinggenetic algorithms. Many different similarity measures can be defined for any givenrepresentation. This section describes two similarity measures for the scheduling genome: adistance-based measure (Euclidean) and a sequence-based measure (Sequence). Both of thesesimilarity measures neglect the mode components of the genome.

Objective Function: The genome performance measure, often referred to as the *objective function*, consists of twoparts, each based upon the schedule the genome represents. The first part is a measure ofconstraint satisfaction, the second part is based on the schedule performance with respect to theobjectives. Since the genome directly represents a schedule, calculation of both measures isstraightforward. Some typical constraint and objective measures are outlined in this section,followed by an explanation of how the constraint and objective measures were combined toproduce the overall score for each genome.

Constraints

Most measurements of constraint satisfaction were based upon *resource profiles*. Resource profilesdefine resource availability or consumption as a function of time.

Resource Availability: Part of the planning stage is the definition of resource availability. For each resource, a profileof availability can be generated to indicate when and how much of that resource will beavailable. Note that this representation encompasses both resource quantity and temporal restrictions onresource usage.

Temporal Constraints: If a task *must* be started at a specific time, then the corresponding start time in the genome isadjusted by the genetic operators so that the task always starts at that time. If a resource

isavailable only at certain times or for a certain duration, this is reflected in the construction of theavailability profile for that resource.

Precedence Feasibility: Precedence feasibility is enforced by the representation and genetic operators, so precedenceinfeasiblesolutions are not possible.

Objectives: Many different measures of schedule performance exist. The representation described in Section4.3 permits modification of objective measures with little or no effect on the search algorithm orgenetic representation. The next three sections highlight some of the more commonperformance measures.

Due Dates and Tardiness: The performance of many projects is measured in terms of due dates or deviation from projectedfinish times. These measures are calculated directly from the schedule. For example, if a workorder specifies that 80% of the jobs must be completed by their specified finish times, theperformance measure can be calculated directly

Cost: The total cost of a schedule can be found by adding the individual costs of each activity giventhe execution mode and resources applied to it. Since the schedule is explicitly defined, anygenome can be used to calculate a net-present value or virtually any other cost measurement ofperformance. If each task has a cost, c_i, determined from the scheduled modes, then the totalcost is simply the sum of the costs of each task.

Makespan: The length of time required to complete a schedule is calculated directly from the information inthe genome. The makespan is simply the finish time of the last task. Note that a schedule mayindicate a makespan when, in fact, that schedule is infeasible due to violations of resourceconstraints.

Composite Scoring: The score for any genome consists of two parts: a constraint satisfaction part and an objectiveperformance part. Since the objective measures are, in practice, meaningless if the schedule isinfeasible, none of the objectives are considered until all of the constraints have been satisfied.

The degree to which constraints are violated determines how feasible the schedule is, and if theschedule is feasible the objective performance is then considered.

Constraint Satisfaction Part: Each schedule contains multiple constraints, each of which measures some aspect of thefeasibility of the schedule. For each constraint, i, a measure of constraint violation, xi, wasdefined. For resource availability, the constraint violation measure was equal to the differencebetween the resources available and the resources required. Temporal constraints were typically measured based on the variance between actual times and desired times.

Objective Performance Part:A project may have a single objective or multiple, possibly conflicting, objectives. Eachobjective is normalized then the lot is averaged to form the overall objective performance.Each objective is normalized to a scale from 0 to 1, inclusive, where 1 indicates perfectsatisfaction of the objective measure. The normalization is done using the specifications-basedtransformations described in the previous section.

RESULT AND DISCUSSION

The Test Problems: The genetic algorithm was run on the following sets of test problems: Patterson's project scheduling problems (PAT)single mode project scheduling set by Kolisch et al (SMCP)single-mode full-factorial set by Kolisch et al (SMFF)multi-mode full-factorial set by Kolisch et al (MMFF)job-shop problems from the operations research warehouse (JS)the benchmarx problems by Fox and Ringer (BMRX).

First introduced by James Patterson in his comparison of exact solution methods for resourceconstrainedproject scheduling, the Patterson set (PAT) consists of 110 project schedulingproblems whose tasks require multiple resources but are defined with only one execution mode.

The problems in the Patterson set are considered easy. First of all, with only 7-48 tasks perproblem, the problems are not very big. Perhaps more importantly, the resource constraintsare not very tight; in many cases the optimal resource-constrained solution is the same as theresource-unconstrained solution.

Kolisch described a method for generating project scheduling problems based on variousparameters for controlling number of tasks, complexity of precedence relations, resourceavailability, and other measures (Rossi et al., 2006). The SMCP, SMFF, and MMFF problem sets weregenerated using ProGen, Kolischs implementation of the algorithm he described.

The single mode set (SMCP) are similar to the Patterson set, but they range in size from 10 to 40tasks and include more resource restrictions. The set includes 200 problems with 1 to 4renewable resource types. Each task has only one execution mode.

The single mode full factorial set (SMFF) consists of 480 problems. Each problem has 30 tasksand 1 to 4 resource types, all renewable. Each task has only one execution mode. The set wasgenerated by varying

three parameters: network complexity, resource factor, and resourcestrength. These factors correspond roughly to the interconnectedness of the task dependencies,the number of resource types that are available, and resource quantity availability.
The multi-mode, full factorial set (MMFF) consists of 538 problems that are known to havefeasible solutions from an original set of 640. The possibility of generating problems with nosolution arises with the addition of non-renewable resources. The problems include fourresource types, two renewable and two non-renewable. The number of activities per project is
10, and each activity has more than one execution mode. The set was generated by varyingthree parameters: network complexity, resource factor, and resource strength. Complete detailsof the problem generation are given in Kolisch description.
The jobshop problems (JS) are from the Ôjobshop1Õ compilation of problems from the operationsresearch library (Parr, 2009). The set consists of 82 problems commonly cited inthe literature. The problems are the standard nxmjobshop formulation in which n jobs with msteps (tasks) are assigned to m machines (resources). They range in size from 6x6 to 15x20. Inother words, they range from 36 tasks and 6 resources to 300 tasks and 20 resources. Each taskhas its own estimated duration, and each task must be performed by one (and only one)resource in a specific order. The objective of each problem is to minimize the makespan.
Descriptions of the problems may be found in (Caseau and Laburthe, 1994). The benchmarx problem was proposed by Barry Fox and Mark Ringer in early 1995. It is asingle problem with 12 parts. Each part adds additional constraints or problem modificationsthat test various aspects of a solution method. The first four parts are fairly standardformulations. It gets harder from there. The problem is large: 575 tasks, 3 types of laborresources and 14 location-based resources. In addition to resource/location constraints, itincludes many temporal restrictions such as three shifts per day with resources limited to certainshifts and task start/finish required within a shift or allowed to cross shifts. The last of thetwelve parts includes multiple objectives. By varying resource availability and work ordersafter a schedule has been determined, the problem also tests the ability of solution methods toadapt to dynamic changes.
The characteristics of the problem sets are summarized in Table 1. With the exception of the lasteight parts of the benchmarx problem, optimal solutions and best-known solutions arecommonly available.

Table 1 Characteristics of the test suites. Tasks in the project scheduling problems typically required more than one resource per task, whereas those in the job-shop problems required only one resource per task. All of the problems have feasible solutions. Optimal solutions are known for many of the problems, best-known solutions are used for comparison when no optimal solution is known.

	number of problems	number of activities per problem	number of renewable resources per problem	number of non-renewable resources per problem	characteristics
PAT	110	7-48	1-3		project scheduling, single mode, multiple resources per task
SMCP	200	10-40	1-6		project scheduling, single mode, multiple resources per task
SMFF	480	30	4		project scheduling, single mode, multiple resources per task
MMFF	538	10	2	2	project scheduling, multi-mode, multiple resources per task
JS	82	36-300	6-20		job-shop scheduling, single mode, one resource per task
BMRX	1 (12)	575	17		general scheduling, one resource per task

Although the representation supports multiple objectives, with theexception of the benchmarx problem, the objective for all of these problem sets was only tominimize the makespan. In addition, only the benchmarx problem specifies temporalconstraints. All of the problems with renewable resources specify uniform resource availability,so a feasible solution is guaranteed for those problems. The multi-mode full factorial setincludes non-renewable resources, so a feasible solution is not guaranteed for problems in thisset. However, the published results of Sprecher and Drexl show optimal solutions for the 538problems in the MMFF set.

Most of the results were achieved using a steady-state genetic algorithm. However, some runswere made using the struggle genetic algorithm in order to evaluate the effects of speciation onthe genetic algorithm performance on these problems.

No specific attempt was made to tune the genetic algorithm; it was run for a fixed number ofgenerations with roulette wheel selection, a reasonable mutation rate, population size, andreplacement rate.

The genetic algorithm required no modifications to switch between any of these problem sets.Thebenchmarx problem required additional data structures to include shift constraints andother modeling parameters, but no change to the algorithm or genome was required.

Genetic Algorithm Performance

Figures 4 summarize the performance of the genetic algorithm on the PAT, SMCP, SMFF,MMFF, and JS problem sets. In each figure, the results of the genetic algorithm are compared tothe optimal score if it is known, or the published best if an optimal score is not known. In theseproblem sets, the performance measure is simply the makespan. The figures show the geneticalgorithm performance relative to the best solution, so a value of 0% means that the geneticalgorithm found the optimal makespan, a value of 100% means that the genetic algorithm foundamakespan twice as long as the published best.

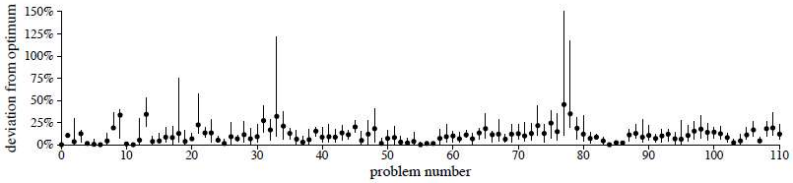

Figure 4 Summary of best, mean, and worst genetic algorithm performance on thePatterson problem set using a steady-state genetic algorithm for 500 generations withpopulation size of 50 individuals (PAT-SS-500-50).

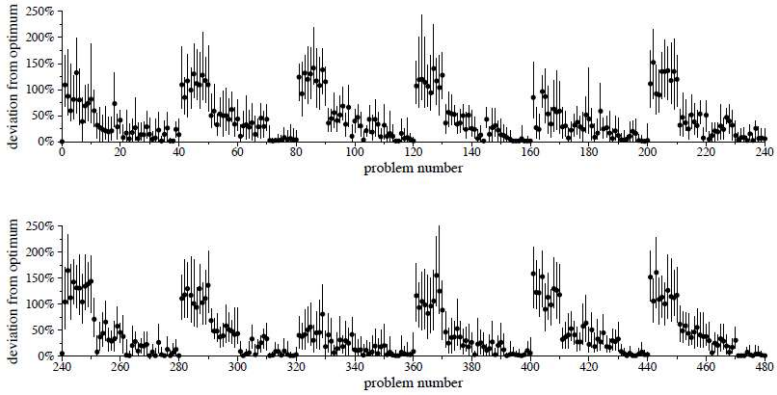

Figure 5 Summary of best, mean, and worst genetic algorithm performance on thesingle-mode full factorial problem set using a steady-state genetic algorithm for 500generations with a population size of 50 individuals (SMFF-SS-500-50).

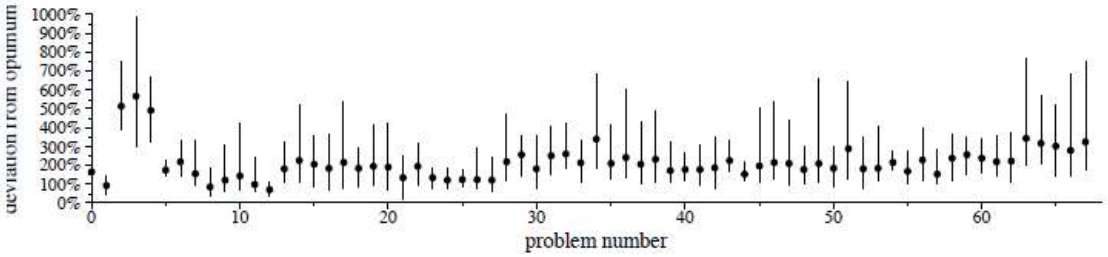

Figure 6 Summary of best, mean, and worst genetic algorithm performance on thejobshop problems using a steady-state genetic algorithm for at most 2000generations with a population size of 50 individuals (JS-SS-2000-50).

In general, the genetic algorithm took more time than would the equivalent enumerative searchor heuristic scheduler. However, it is important to note that no attempt was made to tune thegenetic algorithm parameters. This set of tests focused entirely on creating a representation andset of operators for a baseline comparison; these results represent the worst-case for thisalgorithm and representation.One important area in which the genetic algorithm out-performed the exact solution method of Sprecher *et al* was the multi-modal problems. The genetic algorithm performed well on someproblems that were very difficult for the branch and bound techniques (i.e. the branch andbound method took a long time to find the optimal solution).Typical run times for a single evolution ranged from a few seconds for 100 generations on asmall Patterson problem to over one hour for 5,000 generations on a large jobshop problem.

Implementation details

All of the tests were run using a single implementation of the genetic algorithm; althoughminor changes were made to read various data formats and to accommodate different sets ofobjectives and types constraints, no changes to the genome or genetic algorithm were required.

The implementation was written in C++ using GAlib, a C++ library of genetic algorithmcomponents developed by the author. Tests were run on a variety of Silicon Graphicsworkstations with MIPS R4x00 CPUs running at 100 to 150 MHz.

CONCLUSION

There is a distinct need for more realistic problem sets. In particular, no problem sets exist withmultiple objectives, and the few that include multiple execution modes are far too easy. Only the Benchmarx set includes temporal constraints. Creating such problem sets is no trivialmatter; these problems are difficult to formulate even when many simplifying assumptions aremade. The Benchmarx set is a step in the right direction.

The genetic algorithm performed best (compared to exact solution methods) on the problemswith multi-modal activities. The extra combinations introduced by the multiple executionmodes did not hurt the genetic algorithm performance. In fact, in some cases it made theproblem easier for the genetic algorithm whereas it made the search more difficult for thebranch and bound methods. This suggests that the genetic algorithm (or a hybrid whichincludes some kind of genetic algorithm variant) is well-suited to more-complicated problemswith a mix of continuous and discrete components.

As illustrated in Figures 5, the genetic algorithm did not perform well on problems inwhich the resources were tightly constrained. This comes as little surprise since therepresentation forces the genetic algorithm to search for resource-feasibility, and tightlyconstrained resources mean fewer resource-feasible solutions. As is the case with mostoptimization methods, adding more constraints correlates to increased difficulty in solving theproblem.

As illustrated in Figure 6, the genetic algorithm did not perform well on the job shop problems.

This is due to the structure of the jobshop problems. As illustrated in Figure 3, the job shop problems are typically parallel in nature. Since the representation uses relative times,modification of a single value affects all successive activities if they depend strictly upon thepredecessor tree of the activity being modified. As a result, one small change has a great affecton a large part of the schedule. A typical project plan, on the other hand, has moreinterconnections, so a change to a single activity may not affect directly as many successors.

The struggle genetic algorithm consistently found better solutions than the steady-statealgorithm at some cost in execution time. Since it must make comparisons and often discardsnewly created individuals, the struggle genetic algorithm performs more evaluations than thesteady-state genetic algorithm, but it *always* found feasible solutions, whereas in some runs thesteady-state algorithm did not. The struggle algorithm deserves more study, in particular withrespect to comparison methods of genomes and parallelization of the algorithm.

The representation described in this work is minimal (or nearly so) for this class of problems. If,as Davis notes [Davis 85], there is an inverse relationship between knowledge in arepresentation and its performance, then the methods described in this work can be improvedupon a great deal.

What can be done to improve the genetic algorithm performance? Hybridize the representationand/or algorithm and improve the operators. Combining the genetic algorithm with anothersearch algorithm should provide immediate improvement. A hybrid representation thatexplicitly contains both the resource-constraints as well as the precedence constraints wouldpermit the algorithm to attack the problem from both the resource-constraint perspective as wellas the precedence/temporal constraint perspective. Alternatively, a hybrid that maintains bothabsolute and relative times but operates on one or the other depending on the problemcomplexity and/or structure might improve the poor performance on

problems with parallelstructure such as the job shop problems. Finally, the crossover and mutation operators can betuned to adapt to specific problem structures. For example, one might use a mutator that looksat the parallel/serial nature of the precedence relations as it makes its modifications.

REFERENCES

1. J. Christopher Beck and Philippe Refalo. (2003). A hybrid approach to scheduling with earliness and tardiness costs. Annals OR, 118(1-4):49–71.
2. Peter Brucker. Scheduling Algorithms, 4th Edition.Springer, 2004. 14.
3. C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. Journal of Mathematical Modelling and Algorithms, 3(3):285–308, 2004.
4. Mauro Birattari, Zhi Yuan, PrasannaBalaprakash, and Thomas Stützle. Automated algorithm tuning using f-races: Recent developments. In M. Caserta and S. Voß, editors, Proceedings of MIC 2009, the 8thMetaheuristics International Conference, page 10 pages, 2009.
5. HadrienCambazard and NarendraJussien. Identifying and exploiting problem structures using explanation-based constraint programming. Constraints, 11(4):295–313, 2006.
6. Yves Caseau and François Laburthe. (1994). Improvedclp scheduling with task intervals.In Proc. 11th Intl. Conf. on Logic Programming, 1994.
7. A. Cesta, A. Oddi, and Stephen F. Smith. (2000). AIterative flattening: A scalablemethod for solving multi-capacity scheduling problems. In Proceedings of AAAI .
8. J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. Annals of Operation Research,26(1-4):269–287, 1990.
9. Eugeniusz Nowicki and Czeslaw Smutnicki. (2005). An advanced tabu search algorithm for the job shop problem. J. of Scheduling, 8(2):145–159, 2005.
10. Terence Parr. Language Implementation Patterns: (2009). Create Your OwnDomain-Specific and General Programming Languages. Pragmatic Bookshelf.
11. Francesca Rossi, Peter van Beek, and Toby Walsh. (2006). Handbook of ConstraintProgramming (Foundations of Artificial Intelligence).ElsevierScience Inc., New York, NY, USA, 2006.

