

DETC2004/DAC-57159

TASK SCHEDULING OF PARALLEL DEVELOPMENT PROJECTS USING GENETIC ALGORITHMS

Miao Zhuang⁺, Ali A. Yassine

Product Development Research Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801

ABSTRACT

Resources for development projects are often scarce in the real world. Generally, many projects are to be completed that rely on a common pool of resources. Besides resource constraints, there exists data dependency among tasks within each project. A genetic algorithm approach with one-point uniform crossover and a refresh operator is proposed to minimize the overall duration or makespan of multiple projects in a resource constrained multi project scheduling problem (RCMPSP) without violating inter-project resource constraints or intra-project precedence constraints. The proposed GA incorporates stochastic feedback or rework of tasks. It has the capability of capturing the local optimum for each generation and therefore ensuring a global best solution. The proposed Genetic Algorithm, with several variants of GA parameters is tested on sample scheduling problems with and without stochastic feedback. This algorithm demonstrates to provide a quick convergence to a global optimal solution and detect the most likely makespan range for parallel projects of tasks with stochastic feedback.

(Keywords: Design Structure Matrix (DSM), Makespan, Resource Constrained Multi Project Scheduling Problem (RCMPSP), Genetic Algorithm (GA).)

1. INTRODUCTION

The field of product development (PD) considers a wide range of decisions, tasks, and actions that are necessary to ensure that a product idea can be turned into a workable product. Some of

the tasks involved in developing a new product include market analysis, design, prototyping and testing. Real-world product development projects have many complex tasks with several inter-task relationships that affect the ordering and scheduling of these tasks.

A design structure matrix (DSM) is an efficient and commonly used method of showing the relationships between tasks within a project [20]. Given a set of n tasks in a project, the corresponding DSM is a $n \times n$ matrix where the project tasks are the row and column headings listed in the same order. The precedence relationship between tasks corresponds to the off-diagonal elements of the matrix. In order to complete the task, a certain resource needs to be allocated to the task. The resource could be a machine, raw material, person, and the amount of the resource needed to complete the task could be all, or a fraction of the resource.

All cells along the diagonal of the matrix represent a deterministic (i.e. nominal) completion time for the corresponding task. Marks below the diagonal represent a forward flow of information and represents a task's predecessors. A non-zero value above the diagonal represents a feedback of information, or coupled relationship with a certain probability. Figure 1 shows a sample DSM where the resource needed to complete the task is a predetermined person and there is feedback from tasks 3 and 4 to tasks 1 and 3 respectively. That is, the DSM shows a 20% probability that

⁺ Corresponding author: Department of General Engineering,
University of Illinois at Urbana-Champaign,
Tel. (217) 3332731, email: zhuangm@uiuc.edu

task 1 will be repeated after task 3 finishes performing its job. This is similar to the test case that is used.

	Activity Name	1	2	3	4	5	Resource
1	Market need identification	30		0.2			Mary
2	Quantify customer need	1	10				Bob
3	Determine & Prioritize tech. req'ts		1	10	0.5		Mary
4	Review available technology	1	1	1	5		Bill
5	Patent / liter search	1	1	1	1	10	Bob

Figure 1. Example of DSM

A company may often have several concurrent projects. While projects may be unrelated, they are dependent on a common pool of resources to execute. In other words, no precedence relationships exist between projects, but the same set of people is assigned tasks in multiple projects. All of a person's time must be devoted to a task until it is completed before beginning another task (i.e. no preemption is allowed). If multiple projects are to be considered, each project will have a corresponding DSM.

Projects can be rather long and costly, so finding optimal schedules is crucial in product success over competitors. There are many possible objectives when considering a project scheduling problem. These include minimizing project cost, minimizing variation of resource profiles, or minimizing project duration. In particular, minimizing project duration or makespan is of strategic significance in the stage of product planning for product development problems.

The problem described above is known as the resource constrained project scheduling problem (RCPSP)[16]. The scheduling problem can be very complex as the number of tasks increases. Furthermore, the addition of projects may also complicate the scheduling as it may significantly increase the feasible solution space. For instance, if m projects, totaling n tasks are to be performed, then there can be $\frac{n!}{m!}$ potential

feasible schedules when considering intra-project precedence constraints. It is obvious that even a decent increase in total task number may lead to a remarkably large solution space.

This paper introduces a genetic algorithm based approach to solve a resource constrained multi project scheduling problem (RCMPSP) with additional consideration given to concurrent projects and inter-project precedence constraints. The procedure is based on modified genetic encoding of standard GA operators to suit this problem as well as the introduction of another operator that helps consider the concurrent projects. Trials are performed with different variants of the operators and the results are analyzed.

The paper proceeds as follows: Section 2 provides some background of DSM based models, task scheduling and product development, Section 3 presents the real coded genetic algorithm as well as its experimental design, including chromosome representation and initialization, modified existing and new developed genetic operators, Section 4 discusses the results of the computational experiments, Section 5 further investigates the performance of the proposed GA-based approach in comparison to popular heuristics used in RCPSP. In section 6, we demonstrate how the GA-based algorithm works in the presence of stochastic feedback. This paper concludes with a brief summary of the work completed and ramifications of the project and possible extensions of this project for future work in section 6.

2. BACKGROUND

Early effort in project scheduling focuses on minimizing the overall project duration (makespan) considering unlimited resource. Well-known techniques include Critical Path Method and the Project Evaluation and Review Technique [19].

The RCPSP is of great practical importance and its general model can be used for applications in product development as well as production planning and a wide variety of scheduling applications. Scheduling problems have been studied extensively for many years by attempting to determine exact solutions using methods from the field of operation's research [16].

It was earlier shown that the scheduling problem subject to precedence and resource constraints is NP-Hard [5], leaving exact methods time consuming and inefficient at solving large problems and real-world applications. Besides the impact of problem size, problem complexity also relies on how highly constrained the problem is. There exist benchmark instances with as few as 60 tasks that have failed to be solved to optimality [7]. Kolisch [16] surveyed a number of techniques developed for resource constrained project scheduling. They include dynamic programming, zero-one programming, and implicit enumeration with branch and bound. Some examples of exact solution methods can be seen in [2,3], and [10]. Among them, branch and bound approach is the most widely applied. It, however, is rather a depth-first or breadth-first search in nature and cannot survive an exhaustive search in a large-scale project scheduling problem. Simulation modeling provides a new angle to view RCPSP. A simulation model is proposed for multi-project resource allocation, interpreted as a multi-channel queuing [12]. The innate drawback of simulation in time and cost, as well as deploying a particular simulation language will hinder its dissemination. In addition, many different heuristic approaches have been developed to solve intractable problems quickly and efficiently. A survey of heuristic algorithms can be found in [8] such as schedule generation scheme, priority rule based schemes, simulation annealing, tabu search and genetic algorithms. The domain of

this comparative study is confined to single projects working on a single resource with certain capacity. Activity list based GA overperforms all other approaches in terms of average derivations from the optimal solution.

Genetic Algorithms were first used by [1] for the RCPSP as well as many other NP-Hard scheduling and sequencing problems. The application of GAs in production scheduling has expanded in recent years. To minimize the penalties caused by early and tardy delivery of components, assemblies and products is an emerging direction for GAs. Ip et al. [14] applied GA to a multi-product scheduling in a multi-process production environment. Pongcharoen et al. [15] enriched the problem used by Ip et al. by incorporating assemblies and components and constructing a tree structure for multi-product manufacturing process. Product structure identifier, product instance identifier and operation identifier are integrated in chromosome representation. A repair process is performed to account for any mismatch caused by genetic operation.

Lam et al. [13] developed a GA to minimize the overall completion time for designing products. The authors adopt a more general problem, which considers precedence relations among activities in an unrelated parallel machine environment with m machines and jobs can be processed on any machine. Meanwhile, resource constraint is relaxed by allowing an individual task to be performed by more than one resource (engineer), and the skill level of resources is weighed for each job. Generally, a large-scale task scheduling problem (TSP) for parallel projects needs to deal with exponentially growing computational complexity. Most of the effort has been made to attack scheduling problems by relaxing resource constraint. Therefore, a task can be handled by different resources at different time and even unlimited resources. In this case, a multi-objective GA can be developed to minimize the makespan while employing the least number of resources.

A variety of operators, as well as implementation methods have been developed for, or are applicable to, scheduling problems. In the RCPSP, each listed task needs to be performed only once. Therefore, real coded GA is ideal for solving order-based problems, e.g. project scheduling problem. Crossover operators such as cycle crossover (CX) [17] and partial-mapping crossover (PMX) [6] can fix character duplication. Union crossover (UX2), another efficient order-based crossover operator, was proposed in [18]. However, These operators do not guarantee the preservation of precedence constraints and would result in the need to repair illegal strings or give the constraint violation a penalty. Union crossover 3 (UX3) was then developed [9] based on UX2, which consider precedence constraints as well. When applied to precedence feasible parent chromosomes, UX3 ensures precedence feasible offspring. Their paper presents a useful multicriterion model to solve the RCPSP, but does not consider parallel projects using resources that cannot be divided. They also do not fully discuss the initialization of a precedence feasible population.

Fang et al. [4] present a GA approach to job shop scheduling problem (JSSP) that does consider the parallel processing of jobs on benchmark problems of varying size. However the tasks from a job must be processed in a serial order and each job has the same number of tasks and must visit all machines. A DSM representation of serial processing would be if 1's were only directly below the diagonal of the matrix. The precedence constraints are of a more complex structure for RCPSP problems, thus creating the opportunity for parallel processing of tasks within a project as well as amongst parallel projects. Also, the structure of the problem as to the number of jobs, tasks, and machines for a real world JSSP would differ from the values of the corresponding variables for a problem that would be more typical of the product development field.

3. DESIGN AND IMPLEMENTATION

Genetic Algorithms are robust stochastic search algorithms, introduced first by John Holland and inspired by the process of biological evolution [6]. GAs consider a set of population of solutions as opposed to only one solution throughout local search. Following an initial population through a random generation, new solutions are produced by mating two existing ones (crossover) and/or by altering an existing one (mutation). Selection scheme determines which solutions survive via evaluating fitness or objective function. The fittest solutions take over next generation while the others are deleted. This optimization process finishes with a convergence to an optimal or near-optimal solution.

Figure 2 describes the proposed GA process to solve this parallel project scheduling given DSM as input. A feedback module is built so that feedback information can be integrated in problem encoding. A predefined population of chromosome is then initialized by a random process, considering precedence relationships. Classic genetic operations, crossover and mutation are implemented to continue the random walk in the precedence feasible solution space. Following them, a new developed genetic operation, refresh operator is implemented to parallelize the tasks in the domain of time and resource by accommodating precedence and resource constraints.

Whenever a generation is completed and the population of chromosome is updated, the local makespan optimum is stored. The final optimal solution is selected among them. It ensures to find a global optimal solution, especially when the converging solution is larger than a local optimum that occurs in a certain generation.

Test Case: The sample problem tested by our GA implementation consists of three projects with 33, 17, and 10 tasks respectively, with the tasks assigned amongst 9 different people (see Appendix 1). Task durations range from 1 to 50

time units. Initially, the feedback is not taken into account. The optimal solution to this particular problem is not known, and finding an exact solution may be intractable with respect to the problem size and constraint.

String Representation: Each task is given a unique Identification Number from 1 to 60 and each gene represent a task ID number. Chromosome length is set to 60, the total task number. Each chromosome as a task list must be completed to determine the overall makespan.

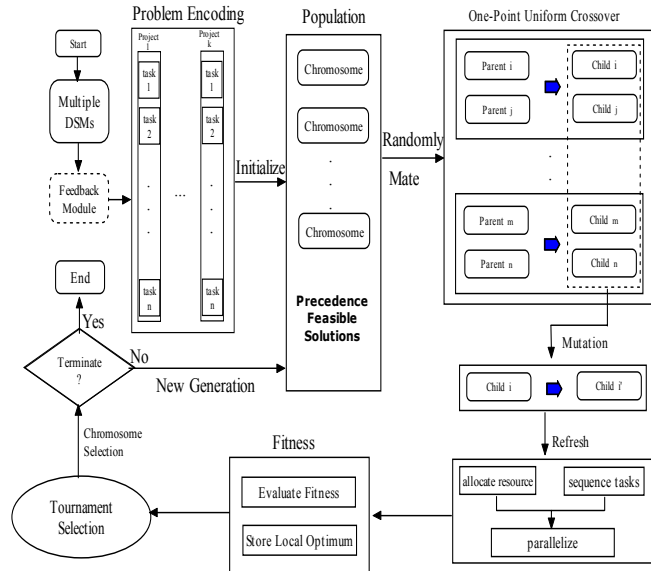


Figure 2. The GA structure

Objective Function:

$$\text{Minimize } T = \max \{t_i + d_i \mid i = 1, 2, \dots, lchrom\}$$

where, t_i is the starting time of task i
 d_i is the duration of task i

Constraint: Optimization of the completion time is subject to the following constraints:

Precedence Constraint: within each project, each task needs to be checked if its immediate predecessor(s) have been done before being performed. Recall that precedence relationships are represented “1” in the DSM. It can be formulated as

$$t_j + d_j \leq t_i, c_{ij} = 1.$$

where c_{ij} is the cell value in the i th row and j th column of DSM

Resource Constraint: Two resource conflicting tasks cannot have overlapping time while performed.

$$t_j + d_j \leq t_i \text{ or } t_j \leq t_i + d_i, \text{ if } r_i = r_j$$

where r_i is the resource value (last column) in the i th row of DSM

Initialization: Highly constrained resource allocation problems have a small feasible search space. Therefore random

generation of strings and incorporating a penalty into the objective function could result in the generation of a large number of infeasible solutions. So a permutation based simulation [7] is used to produce an initial population of precedence feasible individuals. This procedure proceeds as follows:

Step 1. Randomly select a task from all unselected task pool, and check if its immediate predecessor(s) are already selected. If not yet selected, continue this random selection until a satisfying task is found.

Step 2. Repeat step 1 until the set of unselected task is empty, which generate a chromosome that consists of all tasks.

Step 3. Repeat 1 and 2 until all chromosomes of population size are generated.

This chromosome initialization procedure does not however give the makespan, task starting times or consider resource constraints. The initialization procedure simply provides precedence feasible solutions.

Selection: A binary tournament selection is employed in this algorithm. It inexpensively implements a means of obtaining rank-based selection and minimizes the noise of selection through a purely competitive scheme [11]. It picks two individuals from a population of chromosomes and selects the better. Tournament selection has been widely and efficiently used in order-based application.

Refresh: Chromosome fitness, task starting times, and resource constraints are considered by another operator, named Refresh. This operator acts to set the starting time of each task and to obtain the makespan of the completion of all parallel projects for each individual real coded string of the population. The makespan is then used for the fitness measurement for evaluating members of the population. Based on the ordered lists of tasks given as the chromosome representation, it allocates the limited resources to tasks in turn and keeps track of when resources become available. The steps to this process are as follows:

Step 1: Initialize the available time of all resources to 0

Step 2. Start with the first task index listed in the chromosome, compute

$$\max \{t_l + d_l, \max \{s_i + d_i\}\}$$

where, l is the index of the task last performed by the resource to use

i is the index immediate preceding task of this task

t_i, s_i, d_i are defined as above

Step 3: Repeat the step 1 and 2 until all the tasks have been scanned and computed and the makespan of string is then obtained.

It is important to note that based on the chromosome representation of task indexes, different chromosomes may potentially have the same fitness value and essentially represent the same schedule after the refresh operator has assigned task-starting times and makespan. For example, interchanging two tasks in the sequence that have the same starting time would result in a different chromosome, but the same schedule. Expressed in chromosome representation, such tasks can have as large as several tasks in between as long as these possess the same starting time. The swap of them poses no impact on the optimal solution. A question is therefore raised concerning how to capture the underlying schema. Similar issues were discussed in [4] concerning a JSSP, although the authors believe this number is small compared to the number of distinct schedules as problems become large.

Mutation: the exchange of two neighboring alleles is considered without violating precedence relationship. However this also may be ineffective since there may be redundancy in representation. The alleles in neighboring string positions could be switched while still representing the same schedule, as discussed above.

Crossover: Another point of interest is the GA processing of schema. The order of the first several tasks is the key of what the whole string will be made up of. It provides the basis for the remaining tasks and to a certain degree decides how good the order of the remaining tasks will be. Therefore, Good task combinations, good schemas, are most likely to be among the first several tasks. The starting times for schemata with fixed positions early in the genotype are less variable. There are only a limited number of tasks that can be scheduled to start at time $t=0$ based on precedence relationships whereas there are many more potential tasks that could start at a much later point in time. For example, schema of the form 1, 2, *, *, *, *, *, * will always set tasks 1 and 2 to the same starting time if they do share different resources. A schema defined later in the genotype such as *, *, *, *, *, *, 8, 5 is likely to represent many different starting times for tasks 8 and 5 based on the ordering of the unfixed positions. The average fitness of such a schema will roughly correlate to the mean fitness of the entire population [4].

The issues of preserving good building blocks and maintaining randomness and population diversity to find non-redundant solutions are both important in obtaining good results. To consider both of these issues as well as find good solutions to the test problem, experiments have been performed with two different crossover operators.

The crossover operators used are a Union crossover operator 3 (UX3) [9] and a one-point uniform crossover [6]. Union crossover 3 (UX3) is essentially a random shuffle of two exclusive substrings that maintain precedence feasibility. UX3 is performed as follows:

- Step 1:** Given two parent strings 1 and 2, select two positions along both parents randomly through the whole string.
- Step 2:** Select the middle substring defined between the two crossing sites from parent 1, named substring 1.
- Step 3:** Find the exclusive substring from parent 2 after deleting tasks that appear in substring 1, named substring 2.
- Step 4:** Randomly select tasks from the two exclusive substrings; reselect tasks if the immediate predecessors are not selected.
- Step 5:** Fill the task index into the offspring chromosome, positioned from left to right.
- Step 6:** Repeat Steps 3 and 4 until the offspring chromosome is completed.

Another crossover procedure used is based on a one-point crossover method that will generate precedence feasible offspring if applied to precedence feasible parents. This procedure is performed as follows:

- Step 1:** Given two parent strings, select the same position along both parents at random.
- Step 2:** Select the first half substrings from parent 1 and the parent 2, named substring 1 and 2.
- Step 3:** Find the exclusive substrings from parent 2 for substring 1 and from parent 1 for substring 2, named substring 3 and 4 respectively.
- Step 4:** Place substring 1 into unfixed positions at the beginning of offspring 1. Place substring 2 into unfixed positions at the beginning of offspring 2.
- Step 6:** Position substring 3 into unfixed positions following substring 1 in offspring1. Position substring 4 into unfixed positions following substring 2 in offspring 2.

One-point crossover keeps the first half of both parents intact and is less random than UX3. It is capable of preserving schemata in a more effective manner. Although crossing in the beginning of the string could disrupt schema, the probability of disrupting short defining length, low order schemata is rather low [6]. UX3 attempts to preserve ordering but is expected to change allele positions more frequently than the one-point version.

4. COMPUTATIONAL RESULTS

The trials performed using UX3 performed rather poorly. The permutation-based simulation to initialize the population generated some good solutions. Since UX3 is essentially a random shuffle, it caused significant change to the string representations of the parent populations, disrupting potential buildings blocks and high fitness schema at each generation. If better solutions were found with either of these crossover operators, it would be due to the randomization and not the recombinative theories fundamental to Genetic Algorithms.

These operators in a way provide too much variation and diversity rather than discovering new areas of the feasible space to search and build upon. This simply shows that this level of randomization is too disruptive for our problem. Numerical results are not given for either of the UX3 variants due to poor performance.

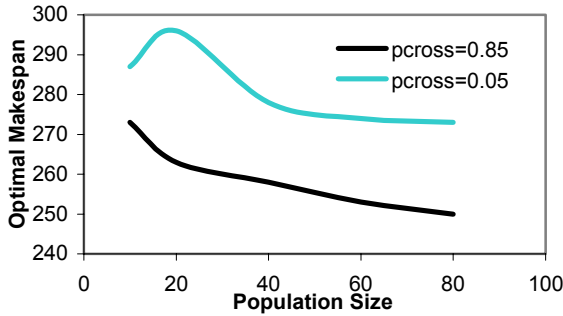


Figure 3 Optimal Solutions versus Population Size

As the population size increases, the local optimal makespan for each population improves. Figure 3 provides us an insight into the change of the optimal solution over population size and how crossover rate impact the performance of crossover operator. When population size reaches the task string size, the optimal makespans become stable and bring the percentage error down to be less than 1%. For such kind of scheduling problem, a decent solution is expected when population size is $O(l)$, l is the string size. With large enough population, the initialization ensures that good schemas appear.

When crossover rate, $pcross = 0.85$, the GA generated much better makespans than those at $pcross = 0.05$. It indicates that one-point uniform crossover dictates the evolution. When $pcross = 0.05$, our GA is almost nothing but a tournament selection, which miss some good shots. It is shown that one-point crossover preserve the good schemas by protecting the first half of tasks intact.

It is shown in Figure 4 that when $pcross=0.05$, the makespans converge very quickly for each population size, taking less than 10 generations. It indicates a premature convergence. Tournament selection takes a jump at each generation by discarding half seemingly bad strings. Apparently, the risk is higher of losing good schemas with half strings abandoned. When population size reaches $O(l)$, the generation required for convergence becomes lower. It further indicates that diversity within the initialization population is the critical for a convergence to a globally optimal makespan.

It is demonstrated in Figure 5 that average makespan decreases as generation increases. A fast convergence rate is exhibited for one-point crossover.

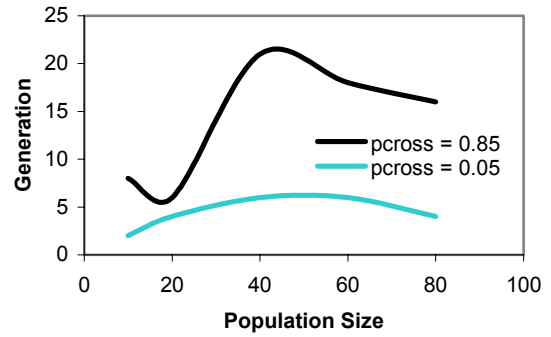


Figure 4 Convergence Speed

It is observed from Figure 6 that the global optimal makespan does not appear until several generations. One-point guided crossover operator distinguishes the good schema and makes them overshadow the whole population. This also displays a fast convergence of local optimal solutions.

Figure 7 and 8 provide the average makespan and makespan variation for UX3. In contrast to one-point crossover, UX3 does not lead to any convergence in either average makespan or variation even after 26 generation when population size is 60 and crossover rate is 0.85. The average makespan is fluctuating around 300, far away from the average makespan shown in Figure 5 for one-point crossover. Furthermore, the variation is not stable and a large gap between worst and best solutions exists in each generation throughout the run. There is no evidence of makespan convergence. Even though we have a global optimal makespan in the middle, it is lost with GA running.

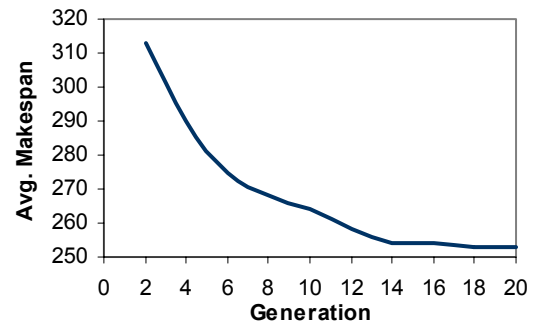


Figure 5. Average Makespan for one-point crossover at population = 60

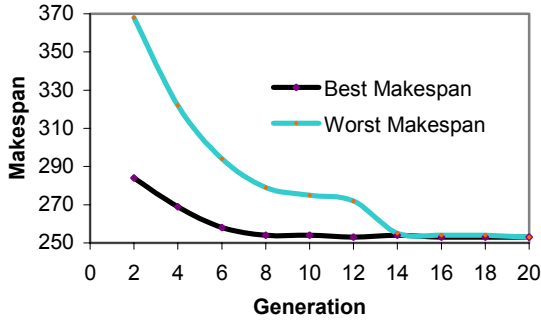


Figure 6. Makespan Variation for one-point crossover at population = 60

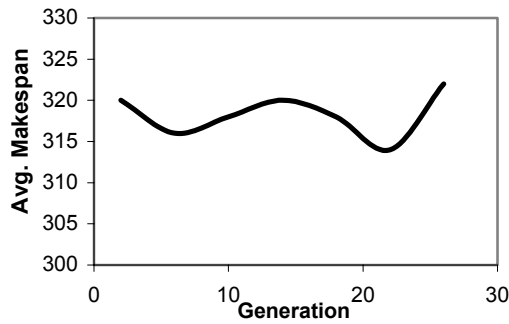


Figure 7. Average Makespan for UX3

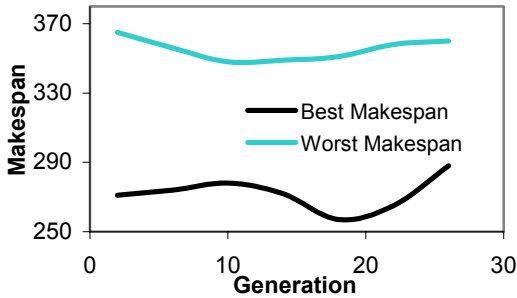


Figure 8. Makespan Variation for UX3

5. COMPARISON TO HEURISTIC RULES

In this section, we relate the performance of the proposed GA to other popular heuristic scheduling rules. A comprehensive categorization of heuristic rule performance in multi-project scheduling is found in Kurtulus et.al [21]. Nine scheduling rules were introduced and their performance was ranked based on different sets of multiple projects. These scheduling rules are summarized as follows:

SOF (Shortest Operation First):

$SOF = \min d_{ij}$, where d_{ij} is the duration of the j th activity from the i th project.

Tiebreaker: FCFS (First Come First Served).

MINSLK (Minimum Slack First):

$$MINSLK = \min SLK_{ij},$$

where $SLK_{ij} = LST_{ij} - \max(EST_{ij}, TIME)$ and terms are as defined in standard CPM. Tiebreaker: FCFS.

SASP (Shortest Activity from Shortest Project):

$$SASP = \min f_{ij} \text{ and } f_{ij} = CP_i + d_{ij},$$

where CP_i is the duration of critical path of the i th project. Tiebreaker: FCFS.

LALP (Longest Activity from Longest Project):

$$LALP = \max f_{ij} \text{ and } f_{ij} = CP_i + d_{ij} \text{ as defined in SASP.}$$

Tiebreaker: GRES = $\max \sum_{k=1}^K r_{ijk}$. where r_{ijk} is the amount of

resource type k required by j th activity of the i th project. Since only one type of resource, personnel, is involved in this study and each task can only be performed by one person, it would be set that $K=1$ and $r_{ijk}=1$. Therefore, the tiebreaker needs to be changed into FCFS.

MOF (Maximum Operation First):

$MOF = \max d_{ij}$, where d_{ij} is the duration of the j th activity from the i th project.

Tiebreaker: GRES initially and converted into FCFS as LALP.

MAXSLK (Maximum Slack First):

$$MAXSLK = \max SLK_{ij}.$$

Tiebreaker: GRES initially and converted into FCFS as LALP.

MINTWK (Minimum Total Work Content):

$$MAXTWK = \min F_{ij}, \text{ where}$$

$$F_{ij} = TWK_i + d_{ij} \sum_{k=1}^K r_{ijk} = TWK_i + d_{ij}$$

and

$$TWK_i = \sum_{k=1}^K \sum_{j=AS_i} d_{ij} r_{ijk} = \sum_{j=AS_i} d_{ij}$$

defines the total work content of the activities already scheduled (i.e., AS_i) from i th project.

Tiebreaker: FCFS.

MAXTWK (Maximum Total Work Content):

$$MAXTWK = \max F_{ij}, F_{ij} \text{ is as defined in MINTWK.}$$

Tiebreaker: FCFS

FCFS (First Come First Served):

First eligible activity is assigned the highest priority.
Tiebreaker: Random.

Specifically, FCFS is expected to behave similar to the initialization step of our GA due to random resource assignment when several precedent-eligible activities are lined up.

A best solution is given by our GA as 252 days. Table 1 provides a summary of the heuristic solutions and also the percentage of improvement made by GA over other heuristic solutions. It is observed that our GA is capable of significantly reducing the makespan, compared to the prevailing heuristic rules.

Scheduling Rules	Makespan (day)	Percentage of Improvement by GA
SOP	329	23.40%
MOP	356	29.21%
SASP	380	33.68%
LALP	346	27.17%
MINSLK	392	35.71%
MAXSLK	425	40.71%
MINTWK	340	25.88%
MAXTWK	383	34.20%

Table 1. Heuristic solutions of scheduling rules used

The proposed GA-based scheduling method has demonstrated its advantage over other prevailing heuristic rules in terms of accuracy and efficiency in this particular test case. As a result, a question arises whether this trend can still hold for other project networks with different characteristics (i.e. sizes and complexities [21]). With the underlying schemata incorporated, the GA should be able to replicate an evolution towards better individuals, for any network structure, in the long run. In addition, given a large enough generation, the optimal solution is ensured. Thus, we conjecture that the performance of the proposed GA-based scheduling method will outperform other heuristic rules even for other network structures. However, our future work will focus on the verification of this hypothesis by testing other project networks with varying characteristics.

6. PROJECTS WITH STOCHASTIC FEEDBACK

A schedule without any feedback, i.e. no non-zero values above the diagonal of the DSM representation, is merely a baseline schedule. In reality, some downstream activities may be forced to be repeated due to changes in upstream information. In order to accommodate this problem, this GA-based approach randomly generates a value based on the feedback probability and thus decides whether the feedback will be part of schedule or not. If a task has more than one feedback activity, this task

will be repeated only after all feedback activities are completed¹.

31 random trials with possible different feedback activities are generated and an optimal schedule is obtained for each. A logistic function is found to best fit the resulting makespan distribution as shown in figure 9. It is helpful to identify the most likely makespan range and provide a better understanding of how long the project may last. As such, we can conduct a sensitivity test on different set of projects and evaluate how the optimal schedule is sensitive to changes in feedback structure.

7. CONCLUDING REMARKS

This paper has proposed an implementation of Genetic Algorithms to solve a DSM representation of the resource constrained project scheduling problem for minimal makespan. A population was initialized such that all members are precedence feasible, and the refresh operator was introduced to maintain precedence and resource feasibility while obtaining the over completion time (makespan) for fitness evaluation. This operator is necessary to assign starting times since tasks could be performed in parallel; resources were shared amongst projects and to determine the overall makespan of all projects.

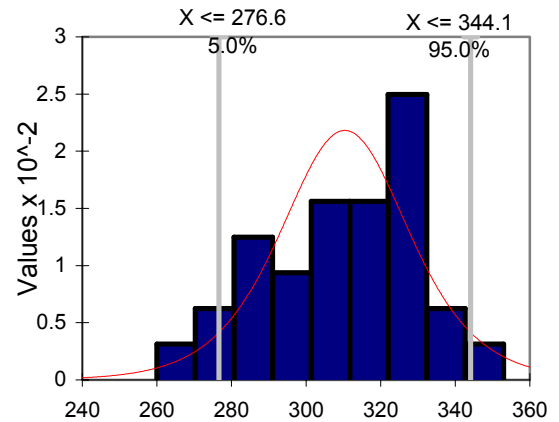


Figure 9. Makespan Distribution for projects with stochastic feedback

Two different crossover operators were also experimented that maintained the precedence feasibility. It is observed that crossover methods that are more randomized such as UX2 or UX3 could not discover solutions better than those generated by conservation operators such as one-point uniform crossover. The performance of UX3 is even worse than a simple one-point crossover even though UX3 is merited in a multicriteria resource constrained project scheduling problem given in [9]. This is most likely due to the disruption of fitness schemata while exploring more feasible solutions.

¹ This is a modeling assumption, but can be easily relaxed.

Good solutions were found however by using a simple one-point uniform crossover. This method performs well via the preservation of potentially high fitness schemata to allow for the combination of good building blocks. It also accommodates feedback which is of paramount importance in a product development project management.

The comparison with the makespans given by heuristic rules, further prove the quality of this GA method. Its improvement over heuristic rules is very apparent. It is anticipated that this method will provide a more accurate and efficient way to plan structure and predict PD timeline.

This GA-based methodology can be readily extended to parallel projects that include tasks with stochastic duration and execution priority. In the latter case, preemption will be allowed and resources are ensured available whenever tasks of higher priority are ready to be performed. Finally, extensions to a multi-objective GA can be proposed to deal with project scheduling considering both completion time and resource cost.

ACKNOWLEDGEMENT

The authors would like to thank Professor David E. Goldberg in the Department of General Engineering at University of Illinois at Urbana-Champaign. He has provided very helpful comments and suggestions. Also, we wish to acknowledge Luke Schenk for some useful discussions.

REFERENCES

[1] Davis, L. (1985). Job shop scheduling with genetic algorithms. Proceedings of the 1st international conference on genetic algorithms. Lawrence Erlbaum.

[2] Demeulemeester, E., & Herroelen, W. (1992). A branch-and-bound procedure for the resource constrained project scheduling problem. *Management Science*, 38, 1803-1818.

[3] Demeulemeester, E., & Herroelen, W. (1996). An efficient optimal solution procedure for resource constrained project scheduling problem. *European Journal of Operational Research*, 90, 334-348.

[4] Fang, H. L., Ross, P., & Corne, D. (1993). A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. Proceedings on the 5th international conference on genetic algorithms. Morgan Kaufman.

[5] Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. (New York: W. H. Freeman & Co.)

[6] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

[7] Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics* 45:733-750.

[8] Kolisch R., S. Hartmann (1998). Heuristic algorithms for solving the resource constrained project scheduling problem: classification and computational analysis. *Handbook on Recent Advances in Project Scheduling*, ed. by Weglarz, 197-- 212. Kluwer.

[9] Leu, S., & Yang, C. (1999). A GA-based multicriteria optimal model for construction scheduling. *Journal of Construction Engineering and Management*, ASCE, 125(6), 420-427.

[10] Sprecher, A. (1996). Solving the RCPSP efficiently at modest memory requirements. *Manuskripte aus den Insituten fur Betriebswirtschaftslehre*. No. 425. Kiel, Germany: University of Kiel.

[11] Goldberg, D. E., Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, 69-93.

[12] S.M.T. Fatemi Ghomi, B. Ashjari (2002). A simulation model for multi-project resource allocation. *International Journal of Project Management*, 29, 127-130

[13] F. S. C. Lam, B. C. Lin, C. Sriskandarajah, H. Yan (1999). Scheduling to minimize product design time using a genetic algorithm. *International Journal of Production Research*. 37(6), 1369-1386

[14] W. H. Ip, Y. Li, K. F. Man, K. S. Tang (2000). Multi-product planning and scheduling using Genetic Algorithm approach. *Computer & Industrial Engineering*. 38, 283-296

[15] P. Pongcharoen, C. Hicks, P. M. Braiden (2004). The development of genetic algorithms for the capacity scheduling of complex products, with multiple levels of product structure. *European Journal of Operational Research*. 152, 215-225.

[16] Kolisch R., and R. Padman 1997. An integrated survey of project scheduling. Technical Report 463, *Manuskripte aus den Instituten f`ur Betriebswirtschaftslehre der Universit`at Kiel*.

[17] Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators on the traveling salesman problems. *Genetic algorithms and their application* (pp. 227-230). Proc., 2nd Int. Conf. Hillsdale, NJ: Lawrence Erlbaum Associate.

[18] Poon, P. W., and Carter, J. N. (1995). "Genetic algorithms crossover operators for ordering applications." *Comp. Ops. Res.*, 22(1), 135-147.

[19] Spinner, M., 1989, *Improving Project Management Skills and techniques* (Englewood Cliffs, NJ: Prentice-Hall).

[20] Yassine, A. and Braha, D. (2003). Four Complex Problems in Concurrent Engineering and the Design Structure Matrix Method. *Concurrent Engineering Research & Applications*, 11(3).

[21] Kurtulus, I. and Davis, E.W. (1982). Multi-Project Scheduling: Categorization of Heuristic Rules Performance. *Management Science*, 28(2), 161-172.

APPENDIX 1 THREE-PROJECT TEST CASE

P1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	Resource	
1	30																																		Sheridan
2	1	10																																	Jean
3		1	10																																Jean
4	1	1	1	5																														Jean	
5	1	1	1	1	10																													Sheridan	
6		1	1			2																												Mickey	
7	1	1	1	1	1	1	10																											Sheridan	
8		1	1		1		1	2																										Sheridan	
9		1	1		1			1	10																									Tom	
10	1									2																								Tom	
11						1				1	1																							Mickey	
12							1	1				2																						John	
13												1	15																					Al	
14	1				1				1	1	1	1	1	10																				John	
15									1	1					1	15																		Mickey	
16														1	1	10																		Jean	
17														1		1	1																	Sheridan	
18		1	1		1	1				1	1			1			1	5																John	
19					1					1		1						1	15															Joy	
20	1				1			1	1									1	1	1														Fritz	
21		1	1										1					1	1	5														Lanny	
22												1	1							1	20													Sheridan	
23		1	1		1					1	1							1	1			5												Mickey	
24	1																						5											John	
25	1									1												1	5											Al	
26	1	1	1							1												1		5										John	
27		1	1		1					1																20								Mickey	
28		1			1																	1	1	1	1	1	1	15						Jean	
29		1											1								1	1							1	10				Al	
30																						1								1	20				Fritz
31																													1	1	5				John
32	1																														1		10		Tom
33	1										1												1											20	Tom

P2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Resource
1	44																	Sheridan
2	1	5																Jean
3			11															Jean
4		1		5														Jean
5	1				2													Sheridan
6				1	5													Mickey
7					1	2												Sheridan
8	1					1	10											Sheridan
9		1						5										Tom
10		1	1							20								Tom
11				1						1	50							Mickey
12	1						1			1	10							Sheridan
13	1									1		20						John
14	1									1			20					Joy
15	1													20				Fritz
16	1	1														30		Lanny
17	1							1	1	1	1	1	1	1	1	1	1	Sheridan

P3	1	2	3	4	5	6	7	8	9	10	Resource
1	4										Sheridan
2	1	10									Jean
3		1	4								Jean
4			1	2							Jean
5					4						Sheridan
6					1	10					Al
7						1	4				John
8							1	2			John
9							1	1	2		John
10				1				1	1	1	John