A Random Generator of Resource-Constrained Multi-Project Network Problems

Tyson R. Browning*
Neeley School of Business
Texas Christian University
TCU Box 298530
Fort Worth, TX 76129
t.browning@tcu.edu

Ali A. Yassine
Department of Industrial & Enterprise
Systems Engineering (IESE)
University of Illinois at Urbana-Champaign
Urbana, IL 61801
yassine@uiuc.edu

This is a near-final preprint of the publication:
Browning, Tyson R. and Ali A. Yassine (2010) "A Random Generator of Resource-Constrained Multi-Project Network Problems," *Journal of Scheduling*, 13(2): 143-161.

*Corresponding author

A Random Generator of Resource-Constrained Multi-Project Network Problems

Abstract

Many scheduling problems in project management, manufacturing, and elsewhere require the generation of activity networks to test proposed solution methods. Single-network generators have been used for the resource-constrained project scheduling problem (RCPSP). Since the first single-network generator was proposed in 1993, several advances have been reported in the literature. However, these generators create only one network or project at a time; they cannot generate multi-project problems to desired specifications. This paper presents the first *multi-network* problem generator. It is especially useful for investigating the resource-constrained multi-project scheduling problem (RCMPSP), where a controlled set of multi-project test problems is crucial for analyzing the performance of solution methods. In addition to the single-project characteristics handled by existing network generators—such as activity duration, resource types and usage, and network size, shape, and complexity—the proposed generator produces multi-project portfolios with controlled resource distributions and amounts of resource contention. To enable the generation of projects with desired levels of network complexity, we also develop several theoretical insights on the effects of network topology on the probability of successful network generation. Finally, we generate 12,320 test problems for a full-factorial experiment and use analysis of means to conclude that the generator produces "near-strongly random" problems. Fully "strongly random" problems require much greater computational expense.

Keywords: Project Scheduling, Multi-project scheduling, Resource constraints, Random network generator, Network complexity

1. Introduction

This paper focuses on the resource-constrained multi-project scheduling problem (RCMPSP).[1] RCMPSPs can represent, for example, a portfolio of product development projects, a set of information technology implementation projects, or a group of buildings to be built by a single construction firm. As projects have become ever-more-common structures for organizing work in contemporary enterprises, issues involving the simultaneous management of multiple projects (or a portfolio of projects) have become more pervasive and acute. For example, studies have shown that managers typically deal with up to four projects at once (Liberatore and Pollack-Johnson 2003; Maroto *et al.* 1999). According to Payne (1995), up to 90% of the value of all projects accrues in the multi-project context, so the impact of even a small improvement in their management could provide an enormous benefit.

There are two main approaches for solving a RCMPSP: (1) the single-project approach and (2) the multi-project approach. The single-project approach aggregates the projects into a single mega-project using dummy activities, thereby reducing the RCMPSP to a RCPSP. The multi-project approach maintains the problem as a portfolio of individual projects (each with its own critical path) rather than artificially combining them into a single project (with a single critical path). *The two approaches can lead to different results* (Kurtulus and Davis 1982). For example, each approach can produce a different schedule with the same priority rule (Lova and Tormos 2001), particularly when the rule is based on the critical path length. Results with the popular "minimum slack" priority rule, for instance, will vary dramatically since combining all of the projects into one meta-project bases all of the slack calculations on a single critical path. While more realistic, the multi-project approach has received less attention in past research.

Most of the previous RCMPSP research explored the performance of heuristic priority rules. Researchers have found conflicting results for which rule performs best (e.g., Davis and Patterson 1975; Lova and Tormos 2001). The literature suggests that performance depends on *project and problem characteristics* (De Reyck and Herroelen 1996; Kurtulus and Davis 1982; Vanhoucke *et al.* 2004). Hence, researchers have sought to improve understanding of the RCMPSP by (1) identifying important problem characteristics and (2) developing *summary measures* for them. Two types of measures are prominent: (a) measures of network size, shape, and connectivity, such as network complexity, and (b) measures of resource loading and contention. By comparing various heuristic and other solution approaches, researchers determined that the problem characteristics have an enormous effect on the

[1] A formal description of the RCMPSP can be found in Appendix A.

ease of solution and its quality. However, the exact nature of these relationships is still not well understood.

Hence, benchmark problems (for rating and comparing the performance of solution procedures—optimal, heuristic, or meta-heuristic) should possess a diversity of problem characteristics with summary measures set at various levels. Until 1993, the quasi-standard benchmark was Patterson's (1984) heterogeneous set of single-project test problems. However, these problems were not generated with a controlled approach, so their parameters did not span the continuum of the important summary measures. Furthermore, all of the problems in Patterson's set have been shown to be "easy" to solve by exact methods (Demeulemeester and Herroelen 1992; Kolisch *et al.* 1995).

To address these shortcomings, researchers began to develop activity network generators for the (single-project) RCPSP. Demeulemeester *et al.* (1993) developed the first random generator, which can specify only the number of nodes (activities) and arcs (precedence relationships) in a network structure. These networks are called *strongly random* since they are drawn from the full space of all feasible networks with a specified number of nodes and arcs. However, this generator cannot produce networks with summary measures at specified levels. Agrawal *et al.* (1996) extended this work with a generator called *DAGEN*, in which the user can specify the level of a summary measure of network complexity. Kolisch *et al.* (1992; 1995) developed another generator, *ProGen*, which includes the ability to specify some basic network structures—the number of start and finish activities and a desired network complexity—and two resource measures, a resource factor and resource strength for each type of resource. Schwindt's (1995; 1996; 1998) *ProGen/Max* adds maximal time lags between activities and applications for general temporal constraints and cyclical networks. Tavares (1998) used six indicators of network structure to generate networks. Unfortunately, all of these later generators do not generate strongly random networks, because they do not allow selection from the full space of feasible networks. Hence, Demeulemeester *et al.* (2003) developed *RanGen*, which claims to generate strongly random networks that conform to desired values of complexity measures. *RanGen* was later augmented with *RanGen2* (Vanhoucke *et al.* 2004). Finally, Gutiérrez *et al.* (2004) introduced *HierGen* to generate hierarchical project networks. However, all of these generators are aimed at the RCPSP, and thus do not include the important problem characteristics of the RCMPSP. That is, generating a bunch of individual project networks and combining them is an incredibly inefficient (if not impossible) way to seek multi-project problems with desired levels of overall summary measures. Therefore, while the importance and broad applicability of test problem generators is well established, a multi-project problem generator has been missing.

This paper makes three contributions:

- First, it contributes the first random generator of RCMPSPs. It generates a *controlled set* of strongly random, activity-on-node RCMPSPs with specified *problem characteristics*, including network complexity, resource loading, and resource contention.
- In the case of each problem characteristic, we started with a *summary measure* proposed in the former literature. However, we found that these summary measures required improvement, so we developed new ones. In particular, to enable the efficient generation of random networks with desired levels of complexity, we had to dig deeply into the theory of network topology and levels, which we call *tiers*. Thus, this paper makes a second, theoretical contribution by determining the relationship between network topology and the efficiency of random network generation. We also contribute improved measures for resource distribution and contention in RCMPSPs.
- Third, we validated the generator by using it to create 12,320 test problems, controlled to specified levels of the summary measures for a full-factorial experiment. We used analysis of means (ANOM) to explore the characteristics of these problems and any biases in the generator. Through this, we identify an important tradeoff between "strong randomness" and computational expense.

In this paper, we deal with the basic RCMPSP, which we note can be expanded in several possible ways. For example, projects might not begin simultaneously, and new projects might arrive at various rates. Project interdependencies (beyond common resources) might exist. Activities might be performed in various modes, each requiring different types and/or amount of resources. Activity preemption might be allowed, perhaps implying switching or restart costs. Activity durations could be stochastic. Resource transfer times could be non-zero, and resources might be non-renewable. However, to maximize our insights from the basic RCMPSP, we do not address these additional aspects in this paper.

The rest of the paper is organized as follows. In the next section, we discuss important *problem characteristics* and *summary measures* in the RCMPSP, upon which we base the proposed generator, which is detailed in §3. §4 describes our use of the generator to create sets of test problems, and §5 analyzes the results of this application. §6 concludes the paper.

2. Important RCMPSP Characteristics and Summary Measures

A number of *problem characteristics* affect the RCMPSP. We focus on four important ones identified in the literature—network topology and complexity, resource distribution, and resource contention—and their

associated *summary measures*. Past research has established the significance of these measures to the RCMPSP and solution difficulty. In the first three sub-sections, we develop new theory pertaining to the effects of network topology and complexity on the generation of random networks with a desired level of complexity. Then, in §2.4 we briefly review new summary measures for resource loading and contention. A thorough understanding of these problem characteristics and summary measures is an essential prerequisite to generating problems with desired properties.

2.1 Network Complexity

Network complexity generally measures the number of relationships between activities. A greater number of relationships among the activities (higher complexity) leaves less flexibility for doing them at a different time (because more things must happen before them, and more things depend on them before occurring). Therefore, in its most basic form, a measure of network complexity accounts for the number of activities or nodes, *N*, and precedence relationships or arcs, *A*. Pascoe (1966) introduced the first such measure, the coefficient of network complexity, *CNC*, defined simply as A / N. For a network with a fixed *N*, its complexity increases as its number of arcs increases. It has been shown that project scheduling problems become easier with *larger CNC* values, because the degrees of freedom decrease as the activities become more constrained (Alvarez-Valdés and Tamarit 1989; Kolisch *et al.* 1995). Elmaghraby and Herroelen (1980) questioned the ability of the *CNC* to represent the actual complexity of project networks. They showed that it is easy to construct networks with equal numbers of nodes and arcs but with varying degrees of difficulty in analysis. Hence, one should not directly equate network complexity with problem difficulty. Many other network complexity measures have been proposed since the *CNC*. Table 1 reviews and summarizes many of these.

While the simplicity of measures such as the *CNC*, *S*, and T-density is attractive, their inability to distinguish between redundant and non-redundant arcs is a critical inadequacy. An arc $<h,u>$ connecting activities *h* and *u* in a network is redundant if there are arcs $<i_0,i_1>, \ldots, <i_{s-1},i_s>$ with $i_0 = h$, $i_s = u$ and $s \geq 2$ (Kolisch *et al.* 1995)—i.e., a redundant arc is one rendered superfluous by other arcs. Redundant arcs should not increase a network's complexity. Therefore, Kolisch *et al.* (1992; 1995) use *C*, which modifies the *CNC* by accounting only for the non-redundant arcs. The *OS* measure also accounts for arc redundancy.

Among the single-project generators, *ProGen* uses *C* and *RanGen* uses *OS*. *C* continues to be used extensively in the RCPSP literature. Plus, *C* has been used in the few studies of the RCMPSP (Kurtulus and Narula 1985; Lova and Tormos 2001). We propose a measure of network complexity adapted from *C* and *OS*, normalized over

[0,1]:

$$C_l = \frac{A' - A'_{\min}}{A'_{\max^*} - A'_{\min}} \tag{1}$$

where l is the project number, A' is the number of non-redundant arcs, A'_{min} is the lower bound on A' in a network of N_l nodes, and A'_{max^*} is the upper bound.[2] (We will discuss these bounds in §2.2.) As a multi-project complexity measure, we use the vector of constituent project complexities, $\boldsymbol{C} = \{C_1, C_2, \ldots, C_L\}$. While other summary measures of multi-project complexity could be developed, the vector form has the advantage of maintaining the distinctness of the individual networks, which becomes helpful for problem generation and analysis.

Table 1: Summary of network complexity measures

Complexity Measure	Definition / Formula	Comments / Reference
Coefficient of Network Complexity, *CNC*	= A / N, where A is the number of arcs and N is the number of nodes	Based on activity-on-arc representation (Pascoe 1966); but also defined for activity-on-node representation (Davis 1975)
Kaimann's modified *CNC*	= A^2 / N	(Kaimann 1974; Kaimann 1975)
Network Complexity, *C*	= A' / N, where A' is the number of non-redundant arcs; $A' \leq A$	Used by *ProGen* (Kolisch *et al.* 1995)
Total Activity Density, T-density	= $\sum_{i=1}^{N} Max(0, P_i - S_i)$, where P_i is the number of predecessors and S_i is the number of successors for the i^{th} node	(Johnson 1967)
Average Activity Density, *AAD*	= T-density / N	(Patterson 1976)
Cyclomatic Number, *S*	= $A - N + 1$	(Temperley 1976)
Measures of Network Complexity, *MNC*	= $g(A - e_1, A - N + 1)$, where e_1 is the number of arcs out of node 1 and $g(\cdot)$ is a monotonically increasing calibration function, determined empirically	(Elmaghraby and Herroelen 1980)
Reduction Complexity, *RC* Complexity Index, *CI*	The minimum number of node reductions sufficient to reduce a 2-terminal acyclic network to a single edge; the number of precedence relations (including the transitive ones but not including the arcs connecting the dummy start or end activity) divided by the theoretical maximum number of precedence relations $N(N-1)/2$, where N denotes the number of non-dummy activities in the network	Based on activity-on-arc representation (Bein *et al.* 1992); never adapted for activity-on-node representation (Vanhoucke *et al.* 2004); later adopted as the Complexity Index, *CI*, and shown to outperform *CNC* in predicting the computational difficulty of RCPSPs (De Reyck and Herroelen 1996)
(Network) Restrictiveness, *RT* Order Strength, *OS* Network Density, *ND*	= $\frac{2\sum a_{ij} - 6(N-1)}{(N-2)(N-3)}$, where a_{ij} is an element of the reachability matrix (defined as the transitive closure of the adjacency matrix), N does *not* include any dummy start or finish nodes, and a does not include any arcs to dummy nodes.	(Thesen 1977); $RT \in [0,1]$; $RT = 0$ for parallel digraphs, 1 for series digraphs; also referred to as Order Strength, *OS* (Mastor 1970), and Network Density (Kao and Queranne 1982); related to Flexibility Ratio, *F* (Dar-El 1973), in that $RT = OS = 1 - F$; conjectured to outperform *CNC* in predicting the computational difficulty of RCPSPs (Schwindt 1995)
Measure of network parallelism, ω	= $(N - T) / N$, where T is length of the network (i.e., the maximum number of activities in series, analogous to the number of tiers)	(Haberle *et al.* 2000)

2.2 Network Topology and the Effect of Tiers

Generating a network with a desired C_l requires a specific A' relative to N_l. (Hereafter in this section we suppress the project number subscript l.) However, the network topology can constrain A', making certain levels of C unachievable. To explore this issue, we use the concept of morphological or hierarchical levels (Elmaghraby 1977; Tavares *et al.* 1999), which we refer to simply as the *tiers* of a network. A thorough understanding of the

[2] We do *not* include dummy nodes (e.g., start and finish) when measuring N.

relationships between network topology and complexity is essential to the efficient generation of random networks.

Within a network of size N, a tier is defined as a subset of the N activities (1) with no arcs between them, and (2) that depend only on activities from lower tiers. When a project network is parsed into tiers, the number of tiers, T, reflects its degree of serialism or parallelism. For example, a fully serial network will have $T = N$ tiers, and a maximally parallel network has $T = 2$ tiers, wherein all of the activities except one (either the start or finish activity) work concurrently.[3] The number of activities in tier j is denoted by n_j, and the shape of a network, based on its allocation of activities to tiers, is given by the vector $\boldsymbol{n} = \{n_1,\ldots, n_j,\ldots, n_T\}$. The n_1 activities in T_1 are called start activities, and the n_T activities in T_T are called finish activities. Thus, we have four parameters by which to specify a network's topology: N, A', T, and $\boldsymbol{n}$.

Lemma 1: Adding an arc between two activities within a single tier changes the network topology by increasing T and/or changing $\boldsymbol{n}$.

Proof: See Appendix B for all proofs.

Lemma 1 prohibits the connection of activities within the same tier if the desired network shape (defined by T and $\boldsymbol{n}$) must be maintained throughout network generation.

Lemma 2: Placing arcs only between activities in *adjacent* tiers prevents the generation of redundant arcs—i.e., an activity in tier j may only have successor activities in tier $j + 1$.

If we only allow arc generation between consecutive tiers, then we obviate the need to perform a check of redundancy each time an arc is added to the network.

Therefore, following Lemmas 1 and 2, the maximum possible number of non-redundant arcs, A'_{max}, for a distribution of N activities to T tiers, $\boldsymbol{n}$, is obtained when an arc connects each activity in tier j with each activity in tier $j + 1$:

$$A'_{\max}(N,T,\boldsymbol{n}) = \sum_{j=1}^{T-1} n_j n_{j+1} \text{ , where } \sum_{j=1}^{T} n_j = N \text{ and each } n \text{ is an integer.}^4 \qquad (2)$$

Theorem 1: For a network of size N with T tiers, $A'_{max}(N,T,\boldsymbol{n})$ is maximized by $\boldsymbol{n}^*$, which is obtained by evenly distributing the maximum possible number of activities among any two consecutive tiers, T_j and T_{j+1}, such that T_j

[3] We do not account for the case where $A' = 0$ and all N activities work in parallel. Such a case is not a project, but rather a problem (a portfolio of projects). Therefore, we define a project as a network with $A' > 0$ and $T \geq 2$.

[4] A form of Equation (2) was previously reported by (Tavares *et al.* 1999), albeit without formal proof. They also noted that the maximum number of non-redundant arcs decreases as the *length* of these arcs increases, where the length of an arc $\langle i,h \rangle$ is defined as: $L(i,h) = T^{(h)} - T^{(i)}$, where $T^{(i)}$ and $T^{(h)}$ are the tiers of activities i and h, respectively, and where $T^{(i)} < T^{(h)}$. They designed a complexity measure based on the number of non-redundant arcs with length greater than one. Their analysis concluded that the statistical properties of networks were insensitive to this measure; it had no significant effect on network complexity. Therefore, in line with these findings, we only generate networks with non-redundant arcs of length one.

and T_{j+1} each contain exactly $(N - T + 2) / 2$ activities, when $N - T$ is even, or $(N - T + 2) / 2 \pm 0.5$ activities, respectively, when $N - T$ is odd. Furthermore, when $T = 3$, the largest tier must be T_2; and when $T \geq 4$, T_j and T_{j+1} must not be the start or finish activities (i.e., $j \neq 1$ and $j+1 \neq T$). Thus, when using $\boldsymbol{n}^*$, the maximum A'_{max} is denoted by A'_{max^*} and is given by:

$$A'_{max^*}(N,T,\boldsymbol{n}^*)=\begin{cases} N^2/4 & \text{, if } T\in[2,3] \text{ and } N \text{ is even} \\ (N^2-1)/4 & \text{, if } T\in[2,3] \text{ and } N \text{ is odd} \\ \dfrac{(N-T+2)^2}{4}+(N-2) & \text{, if } T\geq 4 \text{ and } (N-T) \text{ is even} \\ \dfrac{(N-T+1)(N-T+3)}{4}+(N-2) & \text{, if } T\geq 4 \text{ and } (N-T) \text{ is odd} \end{cases} \tag{3}$$

For example, $A'_{max^*}(20, 2,\{10,10\}) = 100$, by equation (2) or (3), because any other $\boldsymbol{n}$ will yield a lesser A'_{max^*}—e.g., $A'_{max}(20, 2,\{11,9\}) = 99$ by equation (2). In other examples, $A'_{max^*}(20, 3,\{1,10,9\}) = 100$, $A'_{max^*}(20, 4, \{1,9,9,1\}) = 99$, and $A'_{max^*}(20, 6, \boldsymbol{n}^*) = 82$, where $\boldsymbol{n}^* = \{1,8,8,1,1,1\}$ or any other arrangement where (a) the two 8s are adjacent and (b) no 8 is in the first or last tier.

It is important to note that Kolisch's *et al.* (1995) equation for A'_{max^*} does not explicitly account for T or $\boldsymbol{n}$:

$$A'_{max^*}=\begin{cases} N-2+\left(\dfrac{N-2}{2}\right)^2 & \text{, if } N \text{ is even} \\ N-2+\left(\dfrac{N-1}{2}\right)\left(\dfrac{N-3}{2}\right) & \text{, if } N \text{ is odd} \end{cases} \tag{4}$$

It assumes that A'_{max} is only a function of N. (It also assumes the inclusion of dummy start and finish nodes and at least two intermediate tiers, such that $T \geq 4$.) Only in the specific case when $T = 4$ does equation (3) reduce to equation (4). When $T > 4$, however, it is not always possible to achieve the A'_{max^*} implied by equation (4).

Since we will use networks where $N = 20$ later in the paper, we note in this case equation (3) reduces to:

$$A'_{max^*}(20,T,\boldsymbol{n}^*)=\begin{cases} 100 & \text{, if } T\in[2,3] \\ \dfrac{(22-T)^2}{4}+18 & \text{, if } T\geq 4 \text{ and } T \text{ is even} \\ \dfrac{(23-T)(21-T)}{4}+18 & \text{, if } T>4 \text{ and } T \text{ is odd} \end{cases} \tag{5}$$

Corollary 1: $A'_{max^*}(N, T, \boldsymbol{n}^*)$ is a decreasing function of T.

Thus, A'_{max^*} is constrained by the choice of T. A direct implication of Corollary 1 is that networks with the greatest numbers of non-redundant arcs have only two or three tiers.

Now, returning to the complexity measure defined in equation (1), we let $A'_{min} = N - 1$ (which can occur

in the fully serial case, where $T = N$, *or* in the maximally parallel case, where $T = 2$ and $\boldsymbol{n} = \{N - 1, 1\} \neq \boldsymbol{n}^*$)[5] and $A'_{max^*} = N^2 / 4$ (from equation (3)), such that:

$$C(A', N) = \frac{A' - (N-1)}{\left(\frac{N^2}{4}\right) - (N-1)} = \frac{4A' - 4N + 4}{(N-2)^2} \tag{6}$$

Solving for A', we get:

$$A'(C, N) = C + N + \frac{CN^2}{4} - CN - 1 \tag{7}$$

where A' is rounded to the nearest integer. For a given N, this simplifies to a linear function of C—e.g., $A'(C, 20) = 81C + 19$, such that a network containing the maximum possible number of non-redundant arcs would have $C = 1$ and $A' = 100$. For instance, a project network of size $N = 20$ and $C = 0.69$ would require the presence of $A'(0.69, 20) = 81(0.69) + 19 = 74.89 \rightarrow 75$ non-redundant arcs.

2.3 Determining Feasible *T* and *n*

We seek to generate networks of a specified size and complexity, *but with the other aspects of their topologies being strongly random*. Thus, N and C are given, and, according to equation (7), these imply a requisite A', which increases linearly with C and quadratically with N. However, while T and $\boldsymbol{n}$ will be chosen randomly, they must be chosen from a feasible region where the number of *possible* non-redundant arcs is greater than the amount required by N and C—i.e., such that $A'_{max}(N,T,\boldsymbol{n}) \geq A'$. Theorem 1 tells us that this possibility is maximized when $\boldsymbol{n} = \boldsymbol{n}^*$ and $A'_{max} = A'_{max^*}$. However, we are only forced into using this maximum as $C \rightarrow 1$. Otherwise, jumping to this solution prevents the consideration of other networks that *might* work, and thus the generation of strongly random networks. Therefore, before generating networks, we need to understand the bounds on T and $\boldsymbol{n}$. According to Corollary 1, our choice of T is constrained by A'_{max}.

Theorem 2: To generate a random network with a desired N and C, T must be chosen from an interval $[2, T_{max}]$, where T_{max} is the upper bound on T that allows $A'_{max} \geq A'$. The upper bound is defined by:

$$T_{\max}(C, N) \leq \begin{cases} N + 2 - \sqrt{C(N-2)^2 + 4} & \text{, if } (N - T) \text{ is even} \\ N + 2 - \sqrt{C(N-2)^2 + 5} & \text{, if } (N - T) \text{ is odd} \end{cases} \tag{8}$$

Theorem 2 implies that generating a random network with a desired N and C requires that T be sampled from a distribution over $[2, T_{max}]$. For example, $T_{max}(0.69, 20) = 6$ by equation (8). By equation (7), $A'(0.69, 20) = 75$. By equation (5), $A'_{max^*}(20, 6, \boldsymbol{n}^*) = 82$. Since the maximum possible number of non-redundant arcs within a six-tier network is 82, and the network with $C = 0.69$ requires 75 non-redundant arcs, a six-tier network is feasible. A

[5] Both the fully serial and maximally parallel cases have $C = 0$, since $A' = N - 1$, but these cases are distinguished by T.

seven-tier network would not be. Figure 1 shows the required number of non-redundant arcs for a desired C and the T_{max} for which $A'_{max*}(20, T, \boldsymbol{n}^*) \geq A'(C, 20)$.

High C constrains not only T_{max} but also the allocation of activities among the tiers, $\boldsymbol{n}$. For example, consider a project where $N = 10$ and $C = 0.75$. By equation (7), $A'(0.75, 10) = 21$. Even if $T = 2 \leq T_{max}$, some of the combinations of $\{n_1,n_2\}$ still do not allow for $A' \geq 21$ (q.v. equation (2)). If $T = 3$, $A'_{max*}(10, 3, \boldsymbol{n}^*) = 25$ with $\boldsymbol{n}^* = \{4,5,1\}$ or $\{1,5,4\}$. If $T = 4$, $A'_{max*}(10, 4, \{1,4,4,1\}) = 24$. If $T = 5$, $A'_{max*}(10, 5, \boldsymbol{n}^*) = 20$ with $\boldsymbol{n}^* = \{1,3,4,1,1\}$ or $\{1,1,4,3,1\}$.[6] Thus, when attempting to generate a network, sampling T near T_{max} *drastically limits the possible allocations of activities to tiers.*

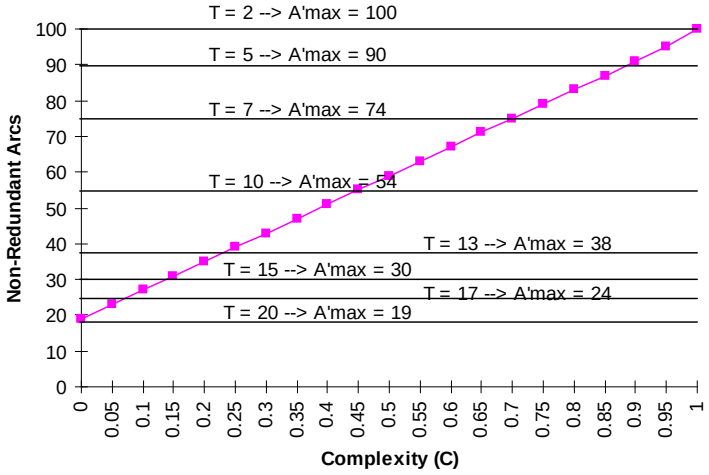


Figure 1: The function $A'(C, 20)$

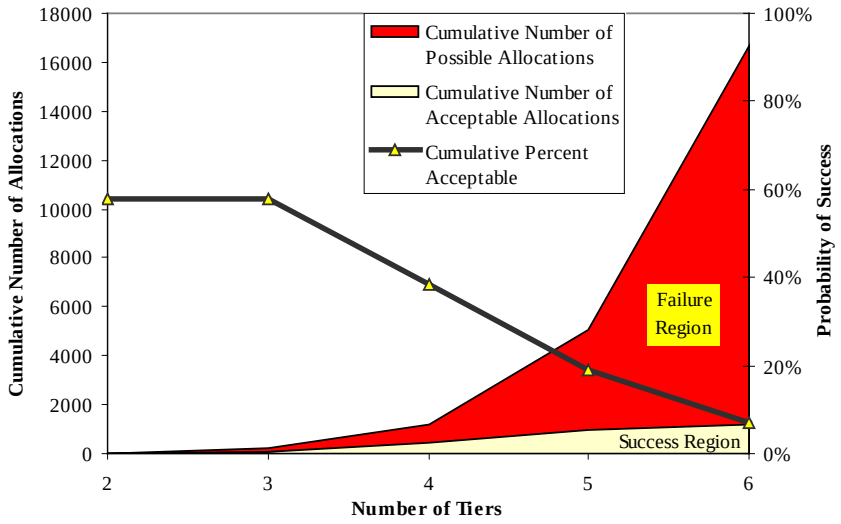
The number of possible allocations of N activities to any possible number of tiers is $2^{N-1} - 1$. The number of possible $\boldsymbol{n}$s for a given T_{max} is found by summing the first T_{max} binomial coefficients, except for the first (which is always one):

$$\boldsymbol{n}(N, T_{\max}) = \sum_{i=1}^{T_{\max}-1} \frac{(N-1)!}{(N-i-1)!\,i!} \tag{9}$$

For example, for $N = 20$ and $T \in [2,20]$, there are 524,287 possible $\boldsymbol{n}$s. However, $C = 0.14$ (a fairly low C) implies that $T_{max} = 14$, which reduces this number to 507,623. $C = 0.69$ ($T_{max} = 6$), a fairly high C, reduces this number to 16,663. Figure 2 illustrates the relationship, showing the number of possible allocations of N activities to T tiers and the acceptable subset of these $\boldsymbol{n}$s which meet the criterion of $A'_{max} \geq A'(0.69, 20) = 75$. The main point here is that as $T \rightarrow T_{max}$, the likelihood of a random allocation of activities to tiers being able to yield a network with the required C diminishes rapidly. Thus, limiting the upper bound on T to something even slightly less than T_{max} will dramatically decrease the sampling of infeasible networks and thus the time to generate feasible networks.

[6] In each of these cases, the reverse order of each vector provides an equivalent A'_{max}—e.g., {4,5,1} is equivalent to {1,5,4}.

However, limiting T to anything less than T_{max} implies that the generator is no longer strongly random. Generating strongly random networks (in the purest sense) requires searching a landscape largely devoid of successful allocations.



Number of Tiers, T	2	3	4	5	6	7
Number of Possible Allocations, $\boldsymbol{n}(20, T)$	19	171	969	3876	11628	27132
Number of Acceptable Allocations	11	99	337	518	237	0
Probability of Success	58%	58%	35%	13%	2%	0%
Cumulative Number of Possible Allocations	19	190	1159	5035	16663	43795
Cumulative Number of Acceptable Allocations	11	110	447	965	1202	1202
Cumulative Percent Acceptable	58%	58%	39%	19%	7%	3%

Figure 2: The probability of successful generation decreases as $T \rightarrow T_{max}$ (shown for the case of N = 20 and C = 0.69)

2.4 Resource Measures

We now turn to two other important characteristics of RCMPSPs, the longitudinal distribution or loading of resource requirements across the problem duration and the degree of contention for various resource types. Kurtulus and Davis (1982) defined a summary measure for each in the multi-project context: the average resource loading factor, *ARLF* (to identify whether the bulk of a problem's total resource requirements fall in the front or back half of its critical path (CP) duration, and the relative size of the disparity), and the *average* (resource) *utilization factor*, *AUF*. However, these measures have several shortcomings which prompted our development of improved measures.

To measure resource distribution or loading, we use a *normalized ARLF* (*NARLF*) defined as:

$$NARLF = \frac{1}{L \cdot CP_{max}} \sum_{l=1}^{L} \sum_{t=1}^{CP_l} \sum_{k=1}^{K_{il}} \sum_{i=1}^{N_l} Z_{ilt} X_{ilt} \left(\frac{r_{ilk}}{K_{il}} \right) \tag{10}$$

where $Z_{ilt} = \begin{cases} -1 & t \le CP_l/2 \\ 1 & t > CP_l/2 \end{cases}$, $X_{ilt} = \begin{cases} 1 & \text{if activity } i \text{ of project } l \text{ is active at time } t \\ 0 & \text{otherwise} \end{cases}$, $Z_{ilt}X_{ilt} \in \{-1, 0, 1\}$, CP_{max} = Max(CP_1, …, CP_L), K_{il} is the number of types of resources required by an activity i in project l, and r_{ilk} is the

amount of resource type *k* required by task *i* in project *l*. This formulation normalizes the *ARLF* over a problem's critical path duration rather than over each individual project's CP duration.

We can also consider the variance of a problem's constituent *ARLF*s from its *NARLF*:

$$\sigma^2_{ARLF} = \frac{1}{L}\sum_{l=1}^{L}\left(ARLF_l - NARLF\right)^2 \tag{11}$$

To measure resource contention, we propose a *modified average utilization factor* (*MAUF*), which is calculated for each resource type as an averaged ratio of the total amount required to the amount available in each time interval over the problem's critical path duration:

$$MAUF_k = \frac{1}{S}\sum_{s=1}^{S}\frac{W_{sk}}{R_k D_s} \tag{12}$$

where R_k is the (renewable) amount of resource type *k* available at each interval, *D* is the size of the interval, *S* is the number of intervals (indexed in *s*), and the total amount of resource *k* required over any interval is given by:

$$W_{sk} = \sum_{t=1}^{D}\sum_{l=1}^{L}\sum_{i=1}^{N_l} r_{ilk} X_{ilt} \tag{13}$$

where r_{ilk} is the amount of resource *k* required by activity *i* in project *l*, and *X* is defined in equation (10). The *MAUF* for a problem involving *K* types of resources is given by:

$$MAUF = \text{Max}(MAUF_1, MAUF_2, \ldots, MAUF_K) \tag{14}$$

Since equation (14) takes a maximum, it fails to distinguish between cases where all *MAUF*s are similar from cases where they vary widely. Therefore, we augment the *MAUF* with another summary measure:

$$\sigma^2_{MAUF} = \frac{\sum_{k=1}^{K}\left(MAUF - MAUF_k\right)^2}{K} \tag{15}$$

Further motivation and analytical support for these new summary measures—*NARLF*, *MAUF*, and σ^2_{MAUF}—are available in (Browning and Yassine 2009). Both the original measures (*ARLF* and *AUF*) and the newer ones are implemented in the proposed generator to facilitate their comparison.

3. Description of the Multi-Project Problem Generator

In the previous section, we investigated *problem characteristics* affecting the RCMPSP—network complexity and topology, resource loading, and resource contention—and their associated *summary measures*. These investigations are essential to the development of rules and guidelines governing the random generation of problems and understanding the generator's performance. We now describe the generation procedures.

3.1 Setting Input Values

The user specifies the inputs shown in Table 2. *L* must be at least one and can be chosen randomly or set by

the user. N_l is typically two at a minimum, although single-activity projects are not prohibited. The generator can choose each N_l randomly, or the user can specify a desired value. L can influence the degree of resource constraints since, all else being equal, a portfolio of 5 projects would expect greater resource contention and delay than a 2-project problem. However, this effect is already accounted for in the *MAUF* measure. Moreover, no specific relationship has been reported between project or problem size and the solution quality obtained by priority rules (Hartmann and Kolisch 2000; Kurtulus and Davis 1982; Lova and Tormos 2001). For these reasons, we would typically hold L and N_l constant, since allowing these to vary could obscure the insights from varying the other factors. The number of resource types, K, can vary by project and be set by the user or determined at random. However, since K affects the *NARLF*, it is advantageous to hold it constant as well. $K = 4$ allows enough variety and leaves room for resource usage manipulations performed by the algorithm..

Table 2: Summary of user input variables, with recommended setting or sampling range

Input Variable	Name	Setting or Range*
L	Number of projects in the problem	3
N_l	Number of activities in project l	20
K_l	Number of resource types used by project l	4
d_{il}	Duration of activity il	Integer in [1,9]
r_{ilk}	Amount of resource type k required by activity il	Integer in [1,9]
$\{C_{des,1}, \ldots, C_{des,l}\}$	Desired complexity of each project	[0,1]
$NARLF_{des}$	Desired *NARLF*	[-3,3]
α	*NARLF* sensitivity threshold	~0.025
$MAUF_{des}$	Desired *MAUF*	[0.6,1.6]
β	*MAUF* sensitivity threshold	~0.025
$\sigma^2_{MAUF,des}$	Desired *MAUF* variance	[0,0.25]

*These parameters and ranges may be adjusted by the user.

The desired complexity, C_{des}, for each project in the problem can either be specified or chosen randomly from [0,1]. While $|NARLF_{des}| \geq 3$ is feasible, most problems do not require such extreme values, and the generator will have a greater difficulty creating a problem as $|NARLF_{des}|$ increases, especially when $NARLF_{des} > 0$.[7] Similarly, solutions with $MAUF_{des} > 1.6$ become difficult to generate (especially if $\sigma^2_{MAUF,des} = 0$, as discussed below), while problems with $MAUF_{des} < 0.6$ usually cease to be interesting (because the resource constraints mostly disappear). Therefore, we follow Kurtulus and Davis (1982) and propose *NARLF* settings over [-3,3] and *MAUF* settings over [0.6,1.6], although values outside these ranges are feasible. If $\sigma^2_{MAUF,des} = 0$, then all resource types in the problem will be similarly constrained, whereas $\sigma^2_{MAUF,des} > 0$ implies that one resource type will be more constrained than

[7] In equation (10), *NARLF* depends on the duration of the longest project in a problem. Since all projects in a problem begin at the same time, it is easier to front-load resources (negative *NARLF*) than to back-load them (positive *NARLF*), because there will tend to be fewer remaining projects near the end of a problem. This effect is ameliorated somewhat by using a constant N for all projects in a problem.

the others. The other variables in Table 2 are discussed below.

3.2 Generating the Basic Project Networks

Following Figure 3, after the inputs are read, the main loop is executed at least once per project. For project l, its number of activities, N_l, is either given or determined randomly. Each activity i in project l is assigned a random duration, d_{il}. Following Kurtulus and Davis (1982), we let d_{il} equal a random integer in [1,9]. Next, T_l is chosen randomly from [2, T_{max}], where T_{max} is based on $C_{des,l}$, as described in §2.3.

Once T_l is selected, the N_l activities are allocated to the tiers. After attempting to weight the number of activities assigned to the front (or back) half of a project based on $NARLF_{des}$, we found that the subsequent $NARLF$ calibration to values in [-3,3] works better when roughly half of the activities are assigned to each half of the project. Therefore, the allocation of activities to tiers is accomplished by the algorithm in Table 3, which uses N and T instead of N_l and T_l for simplicity.

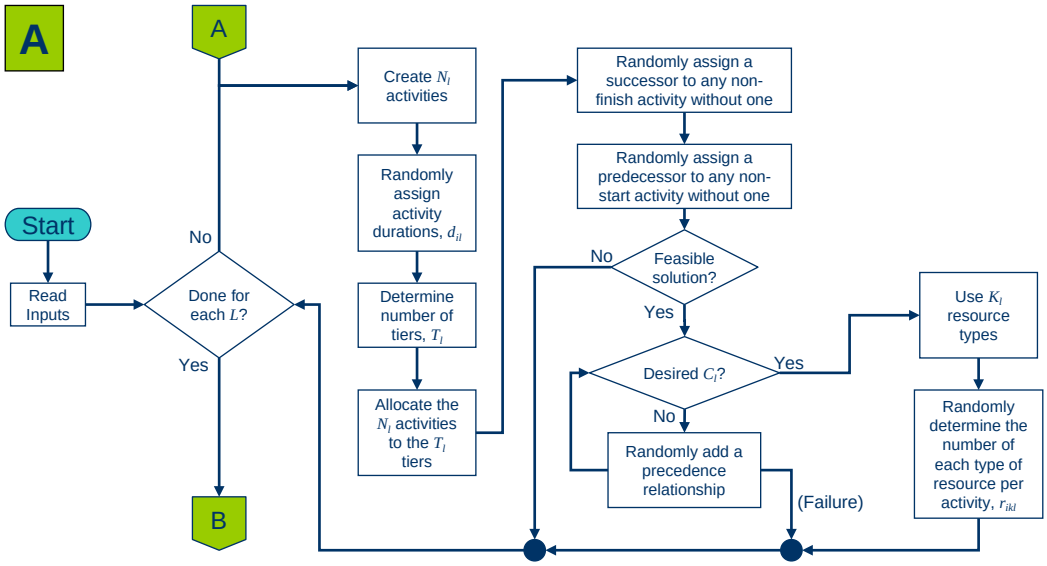


Figure 3: Overview of algorithm (part A) for generating the basic project networks

Once the activities have been allocated to the tiers, non-redundant arcs are created. Each non-finish activity is assigned a random successor in the next tier. Then, each non-start activity is checked to ensure that it has a predecessor. If it does not, then one is randomly assigned from the preceding tier. Note that this approach, also used by Kolisch *et al.* (1992; 1995), may yield infeasible solutions when C_{des} is low, because A' may be too large. In such a case, the algorithm must regenerate the project. In most cases, however, the A' established by this procedure will be too small. In this event, additional successors (always in the adjacent tier, to ensure non-

redundancy) are added at random until the desired A' and C_{des} are achieved.[8] The algorithm could also fail here if there are no remaining positions for non-redundant arcs, given the randomly selected T and $\boldsymbol{n}$. In this case also, the algorithm must regenerate the project.

Once a network with $C_{des,l}$ has been generated, each activity is assigned a random number of required resources of each type. Again following Kurtulus and Davis (1982), we sample each r_{ilk} from integers in [1,9].

Table 3: Algorithm for allocating N activities to T tiers

1. Determine the number of front-half tiers, T_a = Integer(T / 2).
2. Designate half of the activities as "front-half" activities, N_a.† $N_a = N / 2$.
3. Ensure that each tier can have at least one activity:
   - If $N_a > N - T + T_a$, then $N_a = N - T + T_a$.
   - If $N_a < T_a$, then $N_a = T_a$.
4. Randomly allocate N_a activities to the first T_a tiers.
   - *Temp1* = 0.
   - For I = 1 to T_a:
     - T_i = Random Integer in [1,10]. (Assign a random number to each front-half tier.)
     - *temp1* = *temp1* + T_i. (Keep the total all of the random numbers.)
   - For I = 1 to T_a:
     - T_i = Closest Integer($T_i \cdot N_a$ / *temp1*). (Convert the random number to a percentage of the total, and assign that percentage of the front-half activities to tier T_i.)
   - *temp1* = 0.
   - For I = 1 to T_a:
     - If T_i = 0 then T_i = 1. (Add an activity to any front-half tier without one.)
     - *temp1* = *temp1* + T_i. (Count of the number of activities *actually* in the first T_a tiers.)
5. Check the allocation and correct it if necessary.
   - Do while *temp1* $\neq$ N_a: (i.e., if the actual number of activities in the first T_a tiers $\neq$ N_a)
     - *temp2* = Random Integer in [1,T_a]. (Pick a random front-half tier.)
     - *temp3* = *temp1* – N_a. (*temp3* will be positive if the actual number of activities is > N_a.)
     - If (*temp3* > 0) and (T_{temp2} > 1) then:
       - $T_{temp2} = T_{temp2} - 1$. (Subtract an activity from a random tier that has more than one.)
       - *temp1* = *temp1* – 1.
     - If *temp3* < 0 then:
       - $T_{temp2} = T_{temp2} + 1$. (Add an activity to a random tier.)
       - *temp1* = *temp1* + 1.
6. Randomly allocate $N - N_a$ activities to the other tiers.
   - *Temp1* = 0.
   - For $I = T_a + 1$ to T:
     - T_i = Random Integer in [1,10].
     - *Temp1* = *temp1* + T_i.
   - For $I = T_a + 1$ to T:
     - T_i = Closest Integer[$T_i \cdot (N - N_a)$ / *temp1*].
   - *Temp1* = 0.
   - For $I = T_a + 1$ to T:
     - If T_i = 0 then T_i = 1. (Add an activity to any tier without one.)
     - *temp1* = *temp1* + T_i. (*temp1* is a count of the number of activities *actually* in the last $T - T_a$ tiers.)
7. Check the allocation and correct it if necessary.
   - Do while *temp1* $\neq$ $N - N_a$: (i.e., if the actual number of activities outside the first T_a tiers $\neq$ $N - N_a$)
     - *temp2* = Random Integer in [T_a + 1,T].
     - *temp3* = *temp1* – N + N_a. (*temp3* will be positive if the actual number of activities is > $N - N_a$.)
     - If (*temp3* > 0) and (T_{temp2} > 1) then:
       - $T_{temp2} = T_{temp2} - 1$. (Subtract an activity from a random tier that has more than one.)
       - *temp1* = *temp1* – 1.
     - If *temp3* < 0 then:
       - $T_{temp2} = T_{temp2} + 1$. (Add an activity to a random tier.)
       - *temp1* = *temp1* + 1.

†An activity in a "front half" tier will not necessarily be in the front half of the project, especially for low-complexity projects, so the designation of N_a activities as "front-half" activities is only an approximation.

[8] The procedure described by Kolisch *et al.* (1992; 1995) requires an exhaustive checking for whether a randomly added arc is redundant or renders other existing arcs redundant. This operation could be time-consuming and yet may produce an infeasible network. Our simple procedure is more likely to generate a network with a pre-specified C_l.

3.3 Calibrating to the Desired *NARLF*

Having generated L project networks, the generator must now calibrate the problem's *NARLF* to $NARLF_{des}$. We accomplish this by adjusting the projects' resource requirements. Figure 4 provides an overview of this process, which begins by initializing a counter to track the number of adjustments. If this counter exceeds a threshold (we use 500), then the current set of projects is deemed too ill-conditioned to reach $NARLF_{des}$. In such cases an *NARLF* failure is flagged and a new set of project networks is generated.

The *NARLF* calibration algorithm first determines the critical path duration, CP_l, for each project by calculating the early start (ES), early finish (EF), and slack times for each activity in its network. The greatest CP_l is designated CP_{max}, and the ES schedules are used to calculate the *NARLF* using equation (10). This actual *NARLF* is compared with $NARLF_{des}$, and if the difference is greater than a user-specified error bound, α, then the *NARLF* must be increased or decreased.

The *NARLF* can be adjusted in a variety of ways. It depends on CP_{max}, each N_l, K, the r_{ilk}s, and, most importantly, the location of the point of usage of those resources relative to the mid-point of the problem, CP_{max} / 2. Thus, we can increase (decrease) the *NARLF* by adding activities or resources to the back (front) half of a problem, or by subtracting them from the front (back) half. If we want the capability to hold N_l constant, however, we must adjust the *NARLF* by changing r_{ilk}. Hence, to increase the *NARLF*, we select a random activity, project, and resource type—i, l, and k. If the activity is predominately in the front-half of the problem—i.e., if $ES_{il} + (d_{il} / 2) < \text{Integer}(CP_{max} / 2)$—then one unit of r_{ilk} is removed from it (if $r_{ilk} > 1$; otherwise the random selections recur). If the activity is predominately in the back-half of the problem, then r_{ilk} is incremented by one (if $r_{ilk} < 9$; otherwise the random selections recur). Decreasing the *NARLF* occurs in the opposite fashion. This approach to adjusting the *NARLF* provides finer tuning than does changing the number of activities or their durations.[9] Hence, we can choose a relatively small $\alpha = 0.025$ without much risk of an unreachably accurate $NARLF_{des}$.

[9] We originally tried a dual-adjustment algorithm, with both coarse- and fine-tuning mechanisms, depending on the magnitude of the difference between *NARLF* and $NARLF_{des}$. The course-adjustment approach reallocated the activities among the tiers. However, this requires re-determining the arcs as well, and the advantages are outweighed by the added computational intensity. Another approach to course adjustment involves changing the durations of activities, but this could also change the critical path duration.

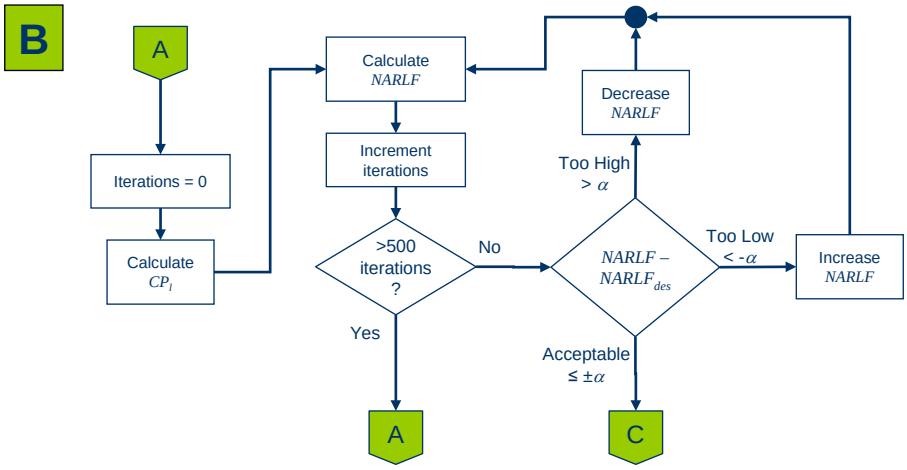


Figure 4: Overview of algorithm (part B) for calibrating a problem to $NARLF_{des}$

3.4 Calibrating to the Desired *MAUF*

Next, the problem must satisfy the $MAUF_{des}$ by setting the number of resources available. Figure 5 provides an overview of this portion of the algorithm. First, the algorithm determines $MAUF_{des,k}$, for each resource type, k. If $\sigma^2_{MAUF,des} = 0$, then $MAUF_{des,k} = MAUF_{des}\ \forall\ k$. If $\sigma^2_{MAUF,des} > 0$, however, then $MAUF_{des,1} = MAUF_{des}$ and

$$MAUF_{des,k} = MAUF_{des} - \sqrt{\sigma^2_{MAUFdes}}\ \ \forall\ k > 1. \tag{16}$$

Since equation (14) is a maximization function, equation (16) is not a function of k. Thus, we let the first resource type have the greatest $MAUF_{des}$ and let the other resource types provide any needed difference so as to yield $\sigma^2_{MAUF,des}$.[10]

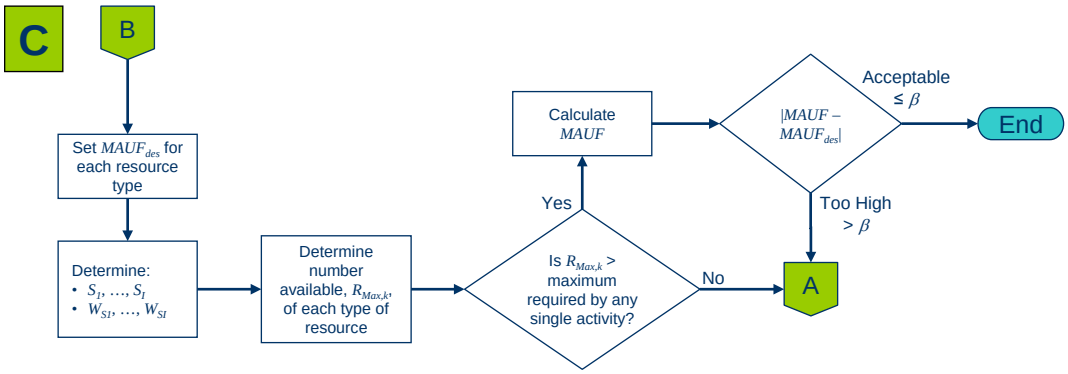


Figure 5: Overview of algorithm (part C) for calibrating a problem to $MAUF_{des}$

After determination of $MAUF_{des,k}\ \forall\ k$, the algorithm calculates each W_{sk} according to equation (13). To get $AUF_{des,k}$ we solve equation (12) for R_k:

[10] Note, however, that if $\sigma^2_{MAUFdes} = 0$, then $MAUF_{des,1} = \text{Max}(MAUF_{des,k})$ is not guaranteed to high precision, since the precision is limited by whole unit resource amounts in the determination of $MAUF_{des,k}\ \forall\ k$, as discussed below.

$$R_k = \text{Closest Integer}\left[\frac{\sum_{s=1}^{S}\left(W_{sk}\prod_{j=1,\, j\neq s}^{S} D_j\right)}{S \cdot AUF_{des,k}\prod_{s=1}^{S} D_s}\right]. \quad (17)$$

When using *MAUF* instead of *AUF*, we let S = Closest Integer(CP_{max}) so that $D_s = 1 \ \forall \ s$, and this equation simplifies to:

$$R_k = \text{Closest Integer}\left(\frac{\sum_{s=1}^{S} W_{sk}}{S \cdot MAUF_{des,k}}\right). \quad (18)$$

This R_k is then used in equation (12) to determine $MAUF_k \ \forall \ k$, with which the problem's *MAUF* and σ^2_{MAUF} can be found using equations (14-15). Sufficiently small differences between σ^2_{MAUF} and $\sigma^2_{MAUF,des}$ are not regarded since they tend to be insignificant in analyses of the test problems.

The algorithm must deal with two potential *MAUF* generation failures. First, if R_k is less than the maximum amount of resource type k required by any single activity, then the problem is infeasible and must be regenerated. This type of failure is rare, however, as long as $MAUF_{des} \leq 1.6$ and $r_{ilk} \leq 9$. The second and more common type of failure is due to the rounding of R_k to the closest integer: if R_k implies a $MAUF_k$ such that $|MAUF_k - MAUF_{des,k}| > \beta$, where β is a measure of the desired precision, then the problem must be regenerated. This failure increases in likelihood as $MAUF_{des,k}$ increases (which is why we begin with the maximum $MAUF_{des}$ when generating a bank of problems with varied *MAUF*s, as discussed in the next section). Unfortunately, this failure also requires returning to near the beginning of the procedure.

The choice of β depends on the incremental difference in $MAUF_{des}$ in a bank of test problems and must be chosen to allow sufficient distinction between them. For example, when varying $MAUF_{des}$ in increments of 0.1, we set $\beta = 0.025$, which implies that a problem with $MAUF_{des} = 1.6$ could actually have $1.575 \leq MAUF \leq 1.625$. However, this still provides ample separation from the next value of $MAUF_{des} = 1.5$. Decreasing β increases computational time by increasing the likelihood of *MAUF* failure and consequential iteration in the algorithm. If these failures do not occur, then we have now generated a problem with desired numbers of constituent projects, resource types used, and activities in each project. The projects each have a desired complexity and resource loading profiles, and the overall problem has a desired average utilization for each resource type.

Note on choosing $\sigma^2_{MAUF,des}$: Since $\sigma^2_{MAUF,des}$ is a variance from a maximum (instead of from a mean), as that max increases, so does the upper bound on the feasible $\sigma^2_{MAUF,des}$. That is, if $MAUF_{des}$ is high—e.g., > 1—then $MAUF_{des,1}$ is also high and the other resource types have more "room" to vary (always on the low side). However,

if $MAUF_{des}$ is low (e.g., 0.6), then $MAUF_{des,1}$ is low and the R_k for $k > 1$ will have to be very large to provide the extremely low values of $MAUF_{des,k}$ (for $k > 1$) required to achieve the desired $\sigma^2_{MAUF,des}$. That is, equation (18) will fail as $MAUF_{des,k} \rightarrow 0$. Thus, when setting $\sigma^2_{MAUF,des}$ for a test bank in which problems will vary over a wide range of $MAUF_{des}$, $\sigma^2_{MAUF,des}$ must be kept low enough to accommodate the lowest values of $MAUF_{des}$. For typical *MAUF*s ranging from 0.6 to 1.6, we find that $\sigma^2_{MAUF,des} = 0.25$ works well.

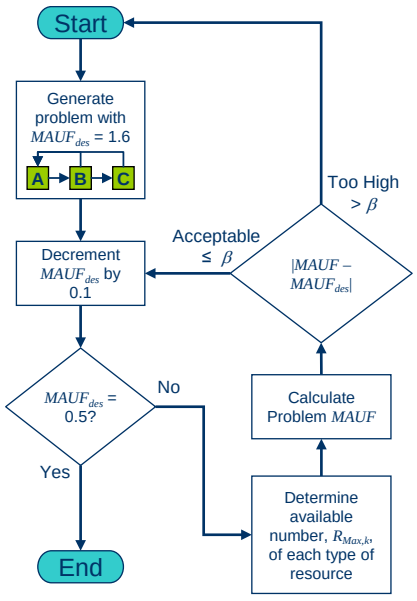


Figure 6: Overview of algorithm for generating a set of 11 problems with varied *MAUF*s

4. Generating a Set of Test Problems to Desired Specifications

To generate a set of random problems with specified *NARLF*s and *MAUF*s, we add two outer loops to the individual problem generator. The inner of these loops addresses *MAUF* and is shown in Figure 9. As discussed above, problems become more difficult to generate as $MAUF_{des}$ increases, so we start by attempting to generate a problem with the highest $MAUF_{des}$—e.g., 1.6. With this problem in hand, the loop decrements $MAUF_{des}$—e.g., by 0.1 (to 1.5)—and resolves for $R_k \; \forall \; k$ using equation (18). (Note that the projects' activity network structures are retained and do not need to be regenerated; we only need to vary the number of resources available to vary the *MAUF*.) As $MAUF_{des}$ decreases, so does the likelihood of a failure to find a *MAUF* that satisfies $|MAUF_k - MAUF_{des,k}| \leq \beta \; \forall \; k$. If such a failure occurs, then the *MAUF* loop must restart by generating a new problem with the highest $MAUF_{des}$.

After the problem with lowest $MAUF_{des}$—e.g., 0.6—is generated, then the *MAUF* loop is complete. In this example, we would now have a set of 11 test problems with identical characteristics, except for varied R_k values implying *MAUF* values that vary over 0.6 to 1.6 in increments of approximately 0.1.

The outer of the two additional loops addresses *NARLF*. This loop is placed on the outside because the test problems with different *NARLF*s are unrelated (unlike the test problems with varied *MAUF*s, which utilize the same basic problem structure except for changes in R_k). This loop varies $NARLF_{des}$ over a desired range, such as [-3,3] in increments of 1.0, although finer-grained increments may be used as long as they are sufficiently greater than α. In our example, we now have seven sets of 11 problems, for a total of 77.

5. Computational Results

Generating a set of 11 problems with a single *NARLF* value and 11 *MAUF* values takes between a few seconds

(for high-complexity problems) and about six minutes (for low-complexity problems) with an Intel Pentium 4M 2GHz processor. The time variance mostly depends on the number of times the algorithm must restart, which depends on the chosen levels of the summary measures. Our initial focus has been on successful problem generation rather than absolute efficiency, which is a secondary concern and can be improved later.

In this section, we explore how the settings affect the generation process and its results. We designed a full-factorial experiment to test the influence of the factors listed in Table 2. To maximize the insights from varying the other factors, we held three variables constant: $N = 20$, $L = 3$ (for the reasons discussed at the beginning of §2), and $K = 4$. The choices for *NARLF* and *MAUF* levels follow Kurtulus and Davis (1982), with *NARLF* in seven integer levels over [-3,3] and *MAUF* in eleven 0.1 increment levels over [0.6,1.6]. We designated two settings for project complexity, "high" ($C = 0.69$) and "low" ($C = 0.14$), and used these to form four levels of $\boldsymbol{C}$: all high-complexity projects ("HHH"), all low-complexity projects ("LLL"), and two intermediate combinations ("HHL" and "HLL"). Furthermore, we wanted some problems where all $MAUF_k$s were equal and others where one resource's *MAUF* determined the overall problem's *MAUF* (while the other three types of resources had significantly lower *MAUF*s)—so we used two levels of σ^2_{MAUF}, 0 and 0.25. Thus, we needed $7 \times 11 \times 4 \times 2 = 616$ problems. To enable identification of random effects, we used 20 replications for each setting, thus generating 12,320 test problems.

We present several analyses of means (ANOMs) of this data set: (1) average number of generated tiers (T), (2) average project duration (APD), (3) variance in project duration (VPD), (4 and 5) number of *MAUF* and *NARLF* failures during generation, and (6) total generation time, each as a function of C, *NARLF*, and σ^2_{MAUF}.[11] Generation time is driven by the number of attempts until a successful problem, which is equal to *NARLF* failures + *MAUF* failures + 1. Table 4 (which is presented in two parts) shows the ANOM results.

5.1 Effects of Network Complexity

Inspecting column one in both parts of Table 4 reveals that all six factors generally increase (although not necessarily monotonically) as $\boldsymbol{C}$ decreases. This is because high-complexity problems are very inflexible (due to the high number of predecessor constraints). Because they tend to have more tiers on average, low-complexity problems are also longer (APD) and more varied in the lengths of their constituent projects (VPD). As discussed in §2.3, the search space for low-complexity problems is much larger, as is the percentage of that space with

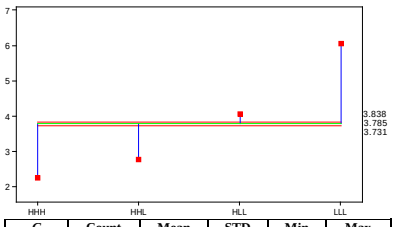
[11] Since all *MAUF* levels are based on a single, common network, we generally found the *MAUF* results uninteresting. Also, since each set of 11 problems with varied *MAUF*s were created as a batch, we did not distinguish the generation times by *MAUF* level. However, since the highest and most difficult *MAUF* setting (here, 1.6) is tried first, most failures that occur would tend to happen here, and the lower *MAUF*-level problems are only attempted once the most difficult one has been successful.

unsuccessful outcomes.

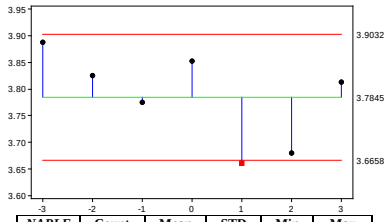
The relationship between ***C*** and *T* is especially interesting. From equation (13) and §2.3 we know that T_{max} = 6 for the HHH problems (C_l = 0.69), although we actually used T_{max} = 5 to reduce computational intensity. Therefore, for the HHH problems, the generator samples *T* from a uniform distribution (the most strongly random allocation) over [2,5], which has an expected value of 3.5 tiers. However, the average *T* for the successfully generated HHH problems is actually 2.26. Figure 7a shows the distribution of *T* by ***C*** for the

Table 4 (Part 1 of 2): ANOM results with main factors (statistics computed with α = 0.1)

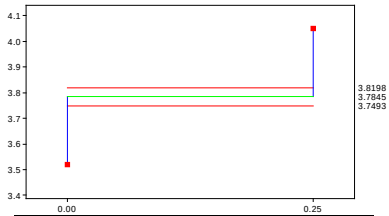
	C	***NARLF***	σ^2_{MAUF}
Avg. *T*	**Increases as *C* decreases**	**No significant relationship†**	**Increases as σ^2_{MAUF} increases**
APD	**Increases as *C* decreases**	**No significant relationship**	**Increases as σ^2_{MAUF} increases**
VPD	**Increases as *C* decreases**	**Decreases as *NARLF* increases**	**Increases as σ^2_{MAUF} increases**



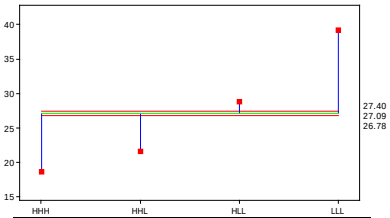
C	Count	Mean	STD	Min	Max
HHH	3080	2.26	0.52	2	4
HHL	3080	2.78	0.84	2	5
HLL	3080	4.06	1.37	2	8
LLL	3080	6.04	2.20	2	12



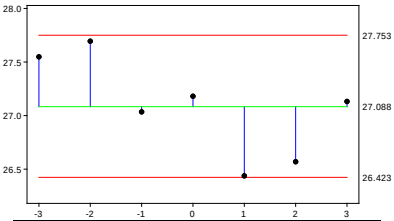
NARLF	Count	Mean	STD	Min	Max
-3	1760	3.89	2.02	2	10.33
-2	1760	3.82	1.79	2	9.33
-1	1760	3.78	1.83	2	8.67
0	1760	3.85	2.30	2	11.33
1	1760	3.66	2.02	2	11.33
2	1760	3.68	2.06	2	11.33
3	1760	3.81	2.02	2	11.67



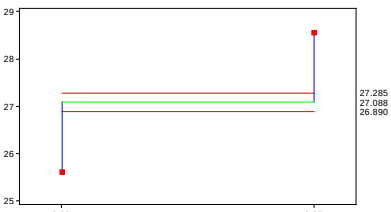
σ^2_{MAUF}	Count	Mean	STD	Min	Max
0	6160	3.52	1.76	2	11.67
0.25	6160	4.05	2.21	2	11.33



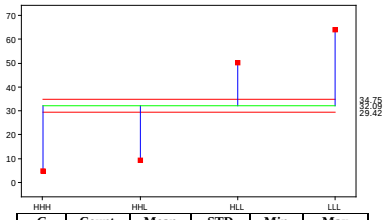
C	Count	Mean	STD	Min	Max
HHH	3080	18.64	3.70	15.33	33.67
HHL	3080	21.58	5.23	15.33	35.00
HLL	3080	28.87	8.19	15.00	55.00
LLL	3080	39.26	12.16	16.00	75.33



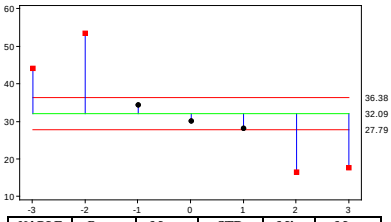
NARLF	Count	Mean	STD	Min	Max
-3	1760	27.55	10.86	15.00	59.33
-2	1760	27.70	10.68	15.67	64.67
-1	1760	27.03	10.24	15.33	56.67
0	1760	27.18	12.53	15.00	65.00
1	1760	26.44	11.52	15.00	71.33
2	1760	26.57	11.68	15.33	75.33
3	1760	27.13	11.25	15.00	71.00



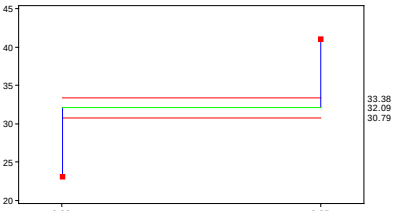
σ^2_{MAUF}	Count	Mean	STD	Min	Max
0	6160	25.62	9.98	15.00	71.33
0.25	6160	28.56	12.27	15.00	75.33



C	Count	Mean	STD	Min	Max
HHH	3080	4.76	12.29	0	80.22
HHL	3080	9.40	16.22	0	98.00
HLL	3080	50.10	96.09	0	686.00
LLL	3080	64.09	97.84	0	601.56



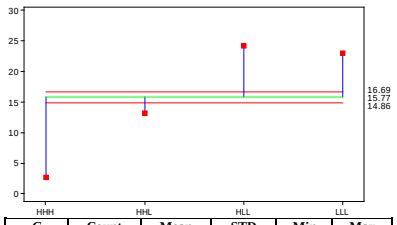
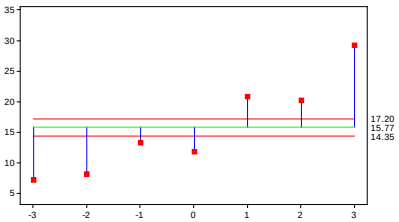
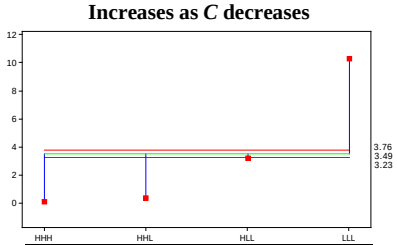
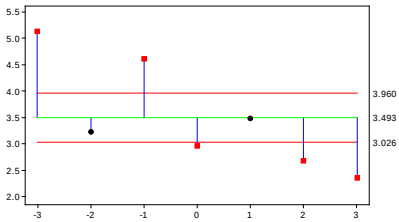
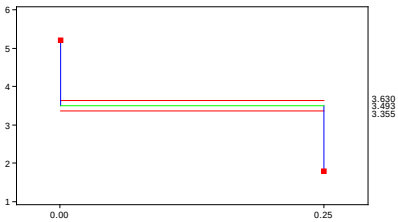
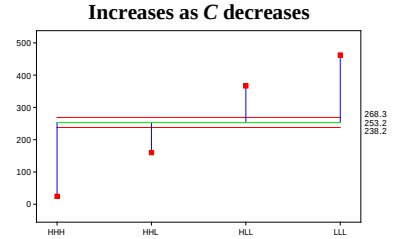
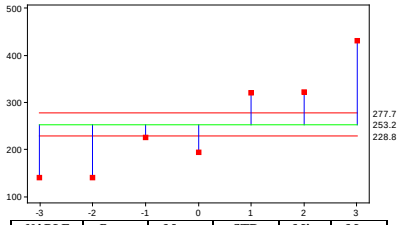
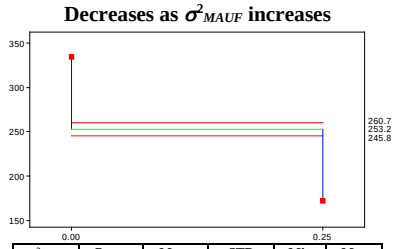
NARLF	Count	Mean	STD	Min	Max
-3	1760	44.15	94.76	0	524.22
-2	1760	53.40	103.51	0	686.00
-1	1760	34.42	63.00	0	329.56
0	1760	30.21	70.72	0	601.56
1	1760	28.20	66.65	0	461.56
2	1760	16.49	42.39	0	366.89
3	1760	17.73	47.43	0	450.00



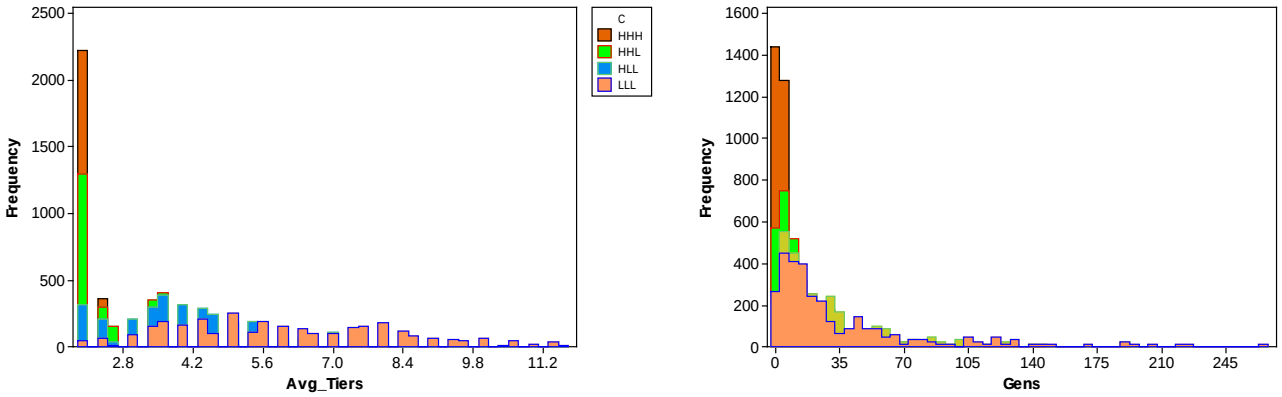
σ^2_{MAUF}	Count	Mean	STD	Min	Max
0	6160	23.11	63.40	0	601.56
0.25	6160	41.06	82.05	0	686.00

†Looking at only the HLL and LLL problems did not change this, except the *NARLF* = 0 problems took significantly longer.

Table 4 (Part 2 of 2): ANOM results with main factors (statistics computed with α = 0.1)

	C	*NARLF*	σ^2_{MAUF}
NARLF Fails	**Increases as *C* decreases**  16.69 15.77 14.86 HHH HHL HLL LLL	**Increases as *NARLF* increases**  17.20 15.77 14.35	**Decreases as σ^2_{MAUF} increases**  16.21 15.77 15.33
	C Count Mean STD Min Max HHH 3080 2.67 3.15 0 15 HHL 3080 13.24 14.98 0 87 HLL 3080 24.14 29.50 0 184 LLL 3080 23.05 33.88 0 226	*NARLF* Count Mean STD Min Max -3 1760 7.15 9.75 0 72 -2 1760 8.10 10.89 0 60 -1 1760 13.19 15.69 0 76 0 1760 11.82 14.78 0 80 1 1760 20.76 30.28 0 180 2 1760 20.24 29.20 0 165 3 1760 29.15 40.53 0 226	σ^2_{MAUF} Count Mean STD Min Max 0 6160 20.31 30.51 0 226 0.25 6160 11.24 17.46 0 139
MAUF Fails	**Increases as *C* decreases**  3.76 3.49 3.23 HHH HHL HLL LLL	**Decreases as *NARLF* increases**  3.960 3.493 3.026	**Decreases as σ^2_{MAUF} increases**  3.630 3.493 3.355
	C Count Mean STD Min Max HHH 3080 0.11 0.35 0 2 HHL 3080 0.37 0.80 0 6 HLL 3080 3.17 3.96 0 22 LLL 3080 10.32 13.05 0 99	*NARLF* Count Mean STD Min Max -3 1760 5.13 11.51 0 99 -2 1760 3.22 6.56 0 52 -1 1760 4.61 9.17 0 56 0 1760 2.97 5.68 0 30 1 1760 3.48 8.27 0 54 2 1760 2.68 6.38 0 37 3 1760 2.36 6.16 0 40	σ^2_{MAUF} Count Mean STD Min Max 0 6160 5.20 10.47 0 99 0.25 6160 1.78 3.45 0 33
Gen. Time (in sec.)	**Increases as *C* decreases**  268.3 253.2 238.2 HHH HHL HLL LLL	**Increases as *NARLF* increases**  277.7 253.2 228.8	**Decreases as σ^2_{MAUF} increases**  260.7 253.2 245.8
	C Count Mean STD Min Max HHH 3080 23.59 24.83 0 138 HHL 3080 160.47 179.61 0 1151 HLL 3080 366.75 420.24 3 2980 LLL 3080 462.20 635.10 2 4512	*NARLF* Count Mean STD Min Max -3 1760 139.85 206.35 0 1579 -2 1760 140.14 194.33 1 1230 -1 1760 225.03 288.19 0 1620 0 1760 194.34 279.76 0 2192 1 1760 320.10 498.60 0 3047 2 1760 321.90 546.70 0 3938 3 1760 431.40 647.30 0 4512	σ^2_{MAUF} Count Mean STD Min Max 0 6160 334.35 527.85 0 4512 0.25 6160 172.15 271.13 0 2238

generated problems. In each case, the average T is significantly less than the expected value of the uniform sampling distribution. These gaps indicate a bias towards projects with fewer tiers, because projects with smaller T are more likely to be generated successfully, as indicated by Figure 2. (Figure 7b shows the average number of attempts until successful generation, which grows with T and thus decreases with ***C***.) The bias in T could be compensated for by sampling from a non-uniform distribution, weighted towards T_{max}, but this would drastically increase computational time. To decrease computational time requirements, we set the generator to sample T_l from [2, T_{max} – 1], thereby sacrificing a small amount of "strong randomness" for a large gain in efficiency, as demonstrated by Figure 2. Note that this decision only directly impacts T, which is not one of the primary summary measures specified by the user (Table 2) and which, in limited past research (Tavares *et al.* 1999), has not been shown to have a direct effect on problem solution complexity.



C	Count	Average *T*	STD	Min	Max
HHH	3080	2.26	0.52	2	4
HHL	3080	2.78	0.84	2	5
HLL	3080	4.06	1.37	2	8
LLL	3080	6.04	2.20	2	11.67

C	Count	Average *T*	STD	Min	Max
HHH	3080	3.78	3.23	1	17
HHL	3080	14.60	15.35	1	88
HLL	3080	28.31	30.91	1	189
LLL	3080	34.36	43.52	1	267

Figure 7: Stacked histograms of average tiers and number of generation attempts by problem complexity

5.2 Effects of *NARLF*

As *NARLF* increases, the durations of the projects in a problem become more varied (VPD). Since (1) all projects in a problem begin simultaneously and (2) the duration of a problem is determined by the duration of its longest project, a bias exists towards problems with low *NARLF*. That is, high-*NARLF* problems have to be attained through a relatively greater loading of resource requirements in the later activities of the longest project. This bias is ameliorated when the number of activities is held constant (because this ensures projects with relatively similar durations, especially when complexity is high), but it becomes more prominent as *NARLF* increases. This effect is moderated by VPD, because if the variance is low, the projects are finishing together, whereas if the variance is high, then some of the projects are carrying on much longer than the others. Hence, as *NARLF* increases (meaning problems with back-loaded resources are desired), so does generation time, mainly due to more *NARLF* failures. More *NARLF* failures reduce the chances of a network with high VPD being generated successfully. Meanwhile, *MAUF* failures become less likely as *NARLF* increases, probably because less variance in project durations provides more options for resource allocation across concurrent projects, which, all else being equal, increases the appropriate R_k. Overall, generation time increases as *NARLF* increases, so the decrease in *MAUF* failures is more than offset by the increase in *NARLF* failures.

5.3 Effects of σ^2_{MAUF} and Two-way Interactions

As σ^2_{MAUF} increases, so do average *T*, APD, and VPD, while generation time and both types of failures decrease. It is harder to get a successful network with no variance in *MAUF*, because all resource types are equally constrained, whereas when $\sigma^2_{MAUF} = 0.25$ only one type of resource is so constrained. Thus, when $K = 4$, a *MAUF* failure is four times more likely with $\sigma^2_{MAUF} = 0$ than with $\sigma^2_{MAUF} = 0.25$.

We analyzed the two-way interactions for the generation times by ***C***, *NARLF*, and σ^2_{MAUF} and found significant interactions between ***C*** and both *NARLF* and σ^2_{MAUF}, but no significant interaction between *NARLF* and σ^2_{MAUF}. This indicates, e.g., when ***C*** is high, *NARLF* level does not influence generation time, but it does when ***C*** is low. We also examined interactions with the average number of tiers but found no significant interactions between ***C*** and *NARLF* or between *NARLF* and σ^2_{MAUF}, only a small interaction between ***C*** and σ^2_{MAUF}. Finally, we note that APD increases linearly with an increase in the average *T*.

Overall, these results indicate that, while the generator can produce strongly random networks (by sampling from the full space of feasible settings), certain combinations of settings are much more likely than others to yield a valid test problem quickly. Faster generation requires biasing the generator towards "near-strongly random" networks.

6. Conclusion

This paper has presented the first random generator of multi-project network test problems. The generator allows a user to control for important multi-project *problem characteristics*, using the *summary measures NARLF*, *MAUF* (or *AUF*), *MAUF* variance, and network complexity. The generator is made possible by a theoretical contribution, a new approach to allocating activities to tiers in the network topology. The generator produces "near-strongly random" networks quickly, and can produce increasingly more strongly random networks at greater computational expense. We therefore identify a tradeoff between the degree of randomness and computational time.

The generation of RCMPSPs is an area for continued research. This paper takes an initial step in this area, but several avenues for further work remain. Further work could explore the power of *T* as a separate summary measure. Past research (Tavares *et al.* 1999) has not shown any direct effect of *T* on problem solution difficulty, but this should be confirmed. As a next step, further research should address the computational effort required to solve the generated problems optimally. It would be interesting to see which of the multi-project problem characteristics and summary measures are best able to predict the computational effort required by solution procedures. We suspect that no single measure will perform as well as certain combinations of measures. Browning and Yassine (2009) use several analysis of variance (ANOVA) models to demonstrate the significance and superiority of the new measures over the traditional ones. However, this superiority could be augmented with a study of the measures' ability to predict computational difficulty. Also, we expect that the efficiency of the proposed generator could be improved.

Appendix A: Formal Description of the RCMPSP

The RCMPSP can be stated as follows. A portfolio or set of $l = 2, \ldots, L$ projects, *collectively* referred to as a *problem*, are to be performed. Each project consists of $i = 1, \ldots, N_l$ activities with deterministic, non-preemptable duration d_{il}. The activities are related by both intra-project predecessor constraints and inter-project resource constraints. Predecessor constraints keep activity i from starting until all of its predecessors in project l have finished. Each activity requires r_{ilk} units of resource type $k \in K$ during every period of its duration. Resource k has a renewable capacity of R_k. At any time, if the set of eligible (precedence unconstrained) activities requires more than R_k for any k, then some activities will be delayed. The RCMPSP entails finding a schedule for the activities (i.e., determining the start or finish times) that optimizes a performance measure, such as minimizing the average delay in all projects. Each project is associated with a due date, set by its resource-unconstrained duration, which is used to measure delays. Let F_{il} represent the finish time of activity i in project l, such that a schedule can be represented by a vector of finish times ($F_{1l}, \ldots, F_{il}, \ldots, F_{N_l l}$). Let $A(t)$ be the set of activities in progress at time instant t. P_{il} is the set of all immediate predecessors of activity i in project l. With these definitions, the problem can be formally stated as:

$$\text{Optimize: Performance Measure } (\forall i \in N_l, l \in L: F_{1l}, \ldots, F_{il}, \ldots, F_{N_l l}) \tag{19}$$

$$\text{Subject to:} \quad \forall i \in N_l,\ \hat{i} \in P_{il}, l \in L: \quad F_{\hat{i}l} \leq F_{il} - d_{il} \tag{20}$$

$$\forall i, l \in A(t): \quad \sum_{i,l \in A(t)} r_{ilk} \leq R_k \quad k \in K; t \geq 0 \tag{21}$$

$$\forall i \in N_l, l \in L: \quad F_{il} \geq 0 \tag{22}$$

The objective function (18) seeks to optimize a pre-specified performance measure. Constraints (19) impose the precedence relations between activities; constraints (20) limit the resource demand imposed by the activities being processed at time t to the capacity available; and constraints (21) force the finish times to be non-negative. Examples of common objective functions are to minimize the average delay to each project in the problem or to minimize the delay to the overall problem.

Appendix B: Proofs

Proof of Lemma 1: Let us consider two situations. First, in a network that already contains the maximum number of non-redundant arcs, adding an arc between two activities in the same tier will result in the creation of a new tier. This follows directly from the definition of a tier. Second, if the network does not already contain the maximum number of non-redundant arcs, then it may be possible to add an arc within a tier such that an activity can be moved to the next tier without increasing the number of tiers, but this results in a redistribution of the activities among the tiers (i.e., changing $\boldsymbol{n}$). □

Proof of Lemma 2: To do otherwise, i.e., to connect nodes in tier j to nodes in tiers $j + j'$ (where $j' > 1$), will result in either (a) the addition of a redundant arc or (b) the rendering of a previously non-redundant arc redundant (i.e. addition of a redundancy-causing arc). First, we define $T^{(i)}$ as the tier of activity i, $T^{(h)}$ as the tier of activity h, where $T^{(i)} < T^{(h)}$, $\langle i,h \rangle$ as an arc between activities i and h, and $L(i,h)$ as the length of arc $\langle i,h \rangle$, such that $L(i,h) = T^{(h)} - T^{(i)}$. A randomly generated arc $\langle i,h \rangle$ has length $L(i,h) = 1$, if tiers $T^{(i)}$ and $T^{(h)}$ are consecutive, or $L(i,h) \geq 2$, if tiers $T^{(i)}$ and $T^{(h)}$ are not consecutive. Thus, two scenarios are possible: (a) $L(i,h) = 1$, when there exists an arc $\langle i,s \rangle \in \mathfrak{A}$ such that $L(i,s) \geq 2$, and an arc $\langle h,s \rangle \in \mathfrak{A}$ such that $L(h,s) \geq 1$. This makes arc $\langle i,h \rangle$ redundant. (b) $L(i,h) \geq 2$, when there exists an arc $\langle i,s \rangle \in \mathfrak{A}$ which makes arc $\langle i,h \rangle$ redundant. Therefore, to avoid the above two cases for generating redundant arcs, arcs of $L \geq 2$ are prohibited. □

Proof of Theorem 1: Maximize $A' = (n_1 n_2) + \ldots + (n_{T-1} n_T)$, s.t.: $\sum_{j=1}^{T} n_j = N$, and n_j are non-zero positive integers, $1 \leq j \leq T$.

If $T = 2$, then the above problem becomes: Max($n_1 n_2$), subject to $n_1 + n_2 = N$. Taking the Lagrangian of this nonlinear integer optimization problem and assuming N is even yields the optimal answer $n_1 = n_2 = N / 2$. If N is odd, then solving this optimization problem yields: $n_1 = (N + 1) / 2$ and $n_2 = (N - 1) / 2$, or vice-versa. Similarly, if $T = 3$, then solving the resultant optimization problem yields the following optimal solutions: $\{n_1 = 1$ and $n_2 = n_3 = (N - 1) / 2\}$ or $\{n_1 = n_2 = (N - 1) / 2$ and $n_3 = 1\}$, if N is even, and solutions similar to the $T = 2$ case when N is odd. For $T \geq 4$, we provide proof by induction. Starting with the hypothesized optimal arrangement, we show that any deviation from this arrangement results in a reduction of non-redundant arcs, or at least no increase. We call the two consecutive tiers where the majority of nodes are allocated the "major tiers" and the others the "minor tiers." Assume that x activities ($x \geq 0$) are removed from one or both major tiers and reallocated to one or more minor tiers as follows: x_j is the number of activities added to or subtracted from tier j; $\sum_{j=1}^{T} x_j = 2x$; $x_j \geq 0$; M and $M+1$ are indices for the major tiers; and $x_M + x_{M+1} = x$. (Note that the case of moving activities between the two major tiers is not considered since we showed above that an equal assignment between the major tiers is optimal.) Let $\phi = (N - T + 2) / 2$ and assume that $(N - T)$ is even. Then, the reallocation of activities to tiers causes:

$$Gain = 2x - (x_1 + x_{M-1} + x_{M+2} + x_T) + \sum_{i=1}^{T-1} x_i x_{i+1} + \phi(x_{M-1} + x_{M+2}) - 2(x_{M-1} x_M + x_{M+1} x_{M+2}) - x_M x_{M+1} \;\&\; Loss = (1 + \phi)x - x_M x_{M+1}.$$

$\Delta = Gain - Loss = (1-\phi)(x - x_{M-1} - x_{M+2}) - x_1 - x_T - 2(x_{M-1}x_M + x_{M+1}x_{M+2}) + \sum_{i=1}^{T-1} x_i x_{i+1}$.

Since $(1-\phi) \le 0$, $(x - x_{M-1} - x_{M+2}) \ge 0$, $(\phi - 1)(x_{M-1} + x_{M+2}) \le 2(x_{M-1}x_M + x_{M+1}x_{M+2})$, and $\sum_{i=1}^{T-1} x_i x_{i+1} \le (\phi - 1)x$, then $\Delta \le 0$. Note that $\sum_{i=1}^{T-1} x_i x_{i+1} \le \frac{x^2}{2}$ when exactly 2 activities are removed from tiers M and $(M+1)$ and placed in tiers $(M-1)$ and $(M+2)$. Also, $2(\phi - 1) \ge x$, which makes $(\phi - 1)x \ge (x^2/2)$. A similar proof may be developed for the case where $(N - T)$ is odd, in which we round the *Loss* and *Gain* expressions down and up, respectively, to the nearest integer. □

Proof of Corollary 1: In equation (3), it is obvious that $A'_{max*}(N, T \in [2,3], \boldsymbol{n}^*) \ge A'_{max*}(N, T \ge 4, \boldsymbol{n}^*)$. Furthermore, equations (3c) and (3d) are decreasing functions in T. Thus, A'_{max*} is a decreasing function of T. □

Proof of Theorem 2: We prove only the limited case where $A'_{max} = A'_{max*}$—i.e., where $\boldsymbol{n} = \boldsymbol{n}^*$. We must show that any $T > T_{max}$ fails to allow $A'_{max*} \ge A'$. Substituting equations (3) and (7) into this inequality yields: $\left(\frac{N-T+2}{2}\right)^2 + N - 2 \ge C + N + \frac{CN^2}{4} - CN - 1$. Arranging terms yields the following quadratic equation (in T): $T^2 - (2N+4)T - (CN^2 - N^2 - 4CN + 4C - 4N) \ge 0$. Let $\delta \equiv CN^2 - N^2 - 4CN + 4C - 4N$, which is a constant when given N and C. Therefore, $T \le \frac{(2N+4) \pm \sqrt{(2N+4)^2 + 4\delta}}{2}$. Since $T \le N$, we choose the negative sign in front of the square root, and transform the function to: $T(C,N) \le N + 2 - \sqrt{C(N-2)^2 + 4}$, where T must be rounded *down* to an integer and $N - T$ is even. If $N - T$ is odd, then the function is: $T(C,N) \le N + 2 - \sqrt{C(N-2)^2 + 5}$. Therefore, T_{max} is the largest value of T that satisfies equation (8a) or (8b). This assumes that $\boldsymbol{n} = \boldsymbol{n}^*$. If not, then T_{max} will be even lower. □

References

Agrawal, M.K., S.E. Elmaghraby, W.S. Herroelen. 1996. DAGEN: A Generator of Testsets for Project Activity Nets. *European Journal of Operational Research,* **90**(2) 376-382.

Alvarez-Valdés, R., J.M. Tamarit. 1989. Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis. in Slowinski, R., J. Weglarz, Eds., *Advances in Project Scheduling*, Elsevier, New York, 113-134.

Bein, W.W., J. Kamburowski, M.F.M. Stallmann. 1992. Optimal Reduction of Two-Terminal Directed Acyclic Graphs. *SIAM Journal on Computing,* **21**(6) 1112-1129.

Browning, T.R., A.A. Yassine. 2009. Resource-Constrained Multi-Project Scheduling: Priority Rule Performance Revisited. TCU Neeley School of Business, Working Paper.

Dar-El, E.M. 1973. MALB-A Heuristic Technique for Balancing Large Single-Model Assembly Lines. *AIIE Transactions,* **5**(4) 343-356.

Davis, E.W. 1975. Project Network Summary Measures and Constrained Resource Scheduling. *IIE Transactions,* **7**(2) 132-142.

Davis, E.W., J.H. Patterson. 1975. A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling. *Management Science,* **21**(8) 944-955.

De Reyck, B., W. Herroelen. 1996. On the Use of the Complexity Index as a Measure of Complexity in Activity Networks. *European Journal of Operational Research,* **91**(2) 347-366.

Demeulemeester, E., B. Dodin, W. Herroelen. 1993. A Random Activity Network Generator. *Operations Research,* **41**(5) 972-980.

Demeulemeester, E., W. Herroelen. 1992. A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem. *Management Science,* **38**(12) 1803-1818.

Demeulemeester, E., M. Vanhoucke, W. Herroelen. 2003. A Random Network Generator for Activity-on-the-Node Networks. *Journal of Scheduling,* **6**(1) 13-34.

Elmaghraby, S.E. 1977. *Activity Networks: Project Planning and Control by Network Models*. Wiley, New York, NY.

Elmaghraby, S.E., W.S. Herroelen. 1980. On the Measurement of Complexity in Activity Networks. *European Journal of Operational Research,* **5**(4) 223-234.

Gutiérrez, M., A. Durán, D. Alegre, F. Sastrón. 2004. Hiergen: A Computer Tool for the Generation of Activity-on-the-Node Hierarchical Project Networks. *Proceedings of the Computational Science and Its Applications - ICCSA, Part III,* Assisi, Italy, 857-866.

Haberle, K., R. Burke, R. Graves. 2000. A Note on Measuring Parallelism in Concurrent Engineering. *International Journal of Production Research,* **38**(8) 1947-1952.

Hartmann, S., R. Kolisch. 2000. Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research,* **127**(2) 394-407.

Johnson, T.J.R. 1967. *An Algorithm for the Resource-Constrained Project Scheduling Problem*. Ph.D. Thesis MIT, .

Kaimann, R.A. 1974. Coefficient of Network Complexity. *Management Science,* **21**(2) 172-177.

Kaimann, R.A. 1975. Coefficient of Network Complexity: Erratum. *Management Science,* **21**(10) 1211-1212.

Kao, E.P.C., M. Queranne. 1982. On Dynamic Programming Methods for Assembly Line Balancing. *Operations Research,* **30**(22) 375-390.

Kolisch, R., A. Sprecher, A. Drexl. 1992. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. Kiel, Germany.

Kolisch, R., A. Sprecher, A. Drexl. 1995. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science,* **41**(10) 1693-1703.

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

Kurtulus, I.S., S.C. Narula. 1985. Multi-Project Scheduling: Analysis of Project Performance. *IIE Transactions,* **17**(1) 58-65.

Liberatore, M.J., B. Pollack-Johnson. 2003. Factors Influencing the Usage and Selection of Project Management Software. *IEEE Transactions on Engineering Management,* **50**(2) 164-174.

Lova, A., P. Tormos. 2001. Analysis of Scheduling Schemes and Heuristic Rules Performance in Resource-Constrained Multi-Project Scheduling. *Annals of Operations Research,* **102**(1-4) 263-286.

Maroto, C., P. Tormos, A. Lova. 1999. The Evolution of Software Quality in Project Scheduling. in Weglarz, J., Ed., *Project Scheduling: Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 239-259.

Mastor, A.A. 1970. An Experimental and Comparative Evaluation of Production Line Balancing Techniques. *Mgmt Sci,* **16**(22) 728-746.

Pascoe, T.L. 1966. Allocation of Resources - CPM. *Revue Française de Recherche Opérationelle,* **38**31-38.

Patterson, J.H. 1976. Project Scheduling: The Effects of Problem Structure on Heuristic Performance. *Naval Research Log,* **23**(1) 95-123.

Patterson, J.H. 1984. A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. *Management Science,* **30**(7) 854-867.

Payne, J.H. 1995. Management of Multiple Simultaneous Projects: A State-of-the-Art Review. *Int J of Project Mgmt,* **13**(3) 163-168.

Schwindt, C. 1995. A New Problem Generator for Different Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, WIOR-Report-449.

Schwindt, C. 1996. Generation of Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, Report WIOR-Report-489.

Schwindt, C. 1998. Generation of Resource-Constrained Project Scheduling Problems Subject to Temporal Constraints. Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, Report Report WIOR-543.

Tavares, L.V. 1998. *Advanced Models for Project Management*. Kluwer Academic Publishers, Boston.

Tavares, L.V., J.A. Ferreira, J.S. Coelho. 1999. The Risk of Delay of a Project in Terms of the Morphology of Its Network. *European Journal of Operational Research,* **119**(2) 510-537.

Temperley, H.M. 1976. *Graph Theory and Applications*. Ellis Horwood Ltd., England.

Thesen, A. 1977. Measures of the Restrictiveness of Project Networks. *Networks,* **7**193-208.

Vanhoucke, M., J. Coelho, D. Debels, L.V. Tavares. 2004. On the Morphological Structure of a Network. Vlerick Leuven Gent Management School, Working Paper no. 2004/9.