

Comments welcome

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

Tyson R. Browning*

M.J. 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

*Texas Christian University
M.J. Neeley School of Business
Working Paper*

Current version: January 23, 2007

*Corresponding author

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

Abstract

Many operations research problems in project management, manufacturing, and elsewhere require the generation of specified sets of activity networks for purposes of testing proposed solution approaches. To date, 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 usual single-project characteristics handled by existing network generators—such as network size, shape, and complexity, activity duration, resource types, and resource usage—the proposed generator produces multi-project portfolios with controlled resource distributions and amounts of resource contention. To enable the generation of networks with desired levels of complexity, we also develop several theoretical insights on network structure in terms of tiering. Finally, we statistically analyze 12,320 generated test problems and conclude that the generator quickly produces “near-strongly random” problems. Fully strongly random networks are possible but increase computational intensity.

Keywords: *Project Management; Networks/Graphs; Multi-Project Scheduling; Resource-Constraints; Network Generator; Network Complexity; Network Topology*

Nomenclature (optional section)

| | |
|--|--|
| L | Number of projects (independent networks) in a multi-project problem (portfolio) |
| l | Project index: $1 \leq l \leq L$ |
| N_l | Number of nodes (activities) in project network l |
| i | Activity index: $1 \leq i \leq N_l$ |
| $\mathbf{N}(\mathfrak{N}, \mathfrak{A})$ | A project network with node set \mathfrak{N} and arc set \mathfrak{A} (the subscript ' l ' denoting project l is suppressed) |
| T_l | Number of tiers in a project network |
| T_{min} | Minimum number of tiers in a network (subscript ' l ' suppressed) |
| T_{max} | Maximum number of tiers in a network (subscript ' l ' suppressed) |
| j | Tier index: $1 \leq j \leq T_l$ |
| n_{jl} | Number of activities in tier j of project l |
| \mathbf{n}_l | A vector of length T and components n_{jl} indicating the allocation of activities to tiers in project l |
| C_l | Complexity level of project l |
| A | Number of arcs (precedence relationships or dependencies) in a network (subscript ' l ' suppressed) |
| A' | Number of non-redundant arcs in a network, $A' \leq A$ (subscript ' l ' suppressed) |
| A'_{min} | Minimum number of non-redundant arcs in a network of size N |
| $A'_{max}(N, T, \mathbf{n})$ | Maximum number of non-redundant arcs in a network of size N with T tiers and activities distributed to tiers according to \mathbf{n} |
| d_{il} | Duration of activity i in project l |
| K_l | Number of types of resources used by project l |
| k | Resource type index: $1 \leq k \leq K_l$ |
| K_{il} | Number of types of resources used by activity i in project l |
| r_{ilk} | Amount of resource type k required by activity i in project l |
| S | Number of time intervals spanning a problem |
| s | Interval index: $1 \leq s \leq S$ |
| S_s | Length of interval s |
| R_k | Renewable amount of resource type k available in each time interval |
| t | Time period index |
| X_{ilt} | Boolean variable, true (equal to 1) if activity i of project l is active at time t |
| CP_l | Non-resource-constrained critical path duration of project l |
| CP_{max} | Non-resource-constrained CP duration of the longest project in a problem; $CP_{max} = \text{Max}(CP_1, \dots, CP_L)$ |
| Z_{ilt} | Equal to -1 if the part of activity i of project l is active at time $t \leq CP_l / 2$; otherwise equal to 1 |
| $ARLF_l$ | Average resource loading factor for project l |
| $NARLF$ | Normalized average resource loading factor for a problem |
| $NARLF_{des}$ | Desired $NARLF$ setting for a problem |
| σ^2_{NARLF} | Variance in projects' $ARLF$ values from problem's $NARLF$ |
| AUF_k | Average utilization factor for resource k |
| $MAUF_k$ | Modified average utilization factor for resource k |
| $MAUF$ | $MAUF$ for a problem; $MAUF = \text{Max}(MAUF_1, \dots, MAUF_k)$ |
| $MAUF_{des}$ | Desired $MAUF$ setting for a problem |
| $MAUF_{des,k}$ | Desired $MAUF$ setting for resource k |
| σ^2_{MAUF} | Variance in resources' $MAUF$ values from problem's $MAUF$ |
| $\sigma^2_{MAUF,des}$ | Desired $MAUF$ variance |
| α | $NARLF$ sensitivity threshold |
| β | $MAUF$ sensitivity threshold |
| P_{il} | Set of all immediate predecessors of activity i in project l |
| \hat{i} | An element in P |

1. Introduction

Activity networks (directed acyclic graphs) provide the basis for models of decision problems in a number of areas, including project management, production planning, research and development, and construction. In particular, resource-constrained project scheduling problems (RCPSPs) have been the focus of extensive research since about 1960 (for a review, see, e.g., Brucker *et al.* 1999). These problems are famous for their simple mathematical formulation and the difficulty of solving large instances to optimality; they are NP-hard (Lenstra and Rinnooy 1978). While optimal approaches are available for small problems, heuristic and meta-heuristic approaches are needed for large ones. In the case of resource-constrained multi-project scheduling problems (RCMPSPs), researchers have found conflicting results for which heuristic performs best (e.g., Davis and Patterson 1975; Lova and Tormos 2001). The literature suggests that performance depends on project and problem characteristics (Kurtulus and Davis 1982; Vanhoucke *et al.* 2004). Progress in the study of RCMPSPs requires the development and comparative testing of new heuristic and meta-heuristic approaches, and this in turn requires a means of generating a set of test problems with pre-specified characteristics.

RCMPSPs can represent, for example, a portfolio of product development projects, a set of information technology implementation projects, or a group of houses to be built by a single construction firm. There are two main approaches for solving a RCMPSP: (1) the single-project approach and (2) the multi-project approach (Kurtulus and Davis 1982). In the single-project approach, the projects are aggregated into a single mega-project using dummy activities, thereby reducing the RCMPSP to a RCPSP and leveraging the abundant research and progress reported in the literature. Meanwhile, the multi-project approach (what we term the RCMPSP) maintains the problem as a portfolio of individual projects (each with its own critical path) rather than artificially combining them into a single project. 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. For example, results with the minimum slack priority rule will vary dramatically depending on which approach is used (since combining all of the projects into one meta-project bases all of their slack calculations off of a single critical path). While the single-project approach is more efficient for minimizing a single project's duration (Davis and Patterson 1975), rules based on the multi-project approach perform better when minimizing average project delay (Kurtulus and Davis 1982). The multi-project approach, while more realistic, has received very

little attention in past research.

Motivated by the conflicting results on which rule performs best in the RCMPSP, researchers sought to improve understanding of the problem by identifying and developing measures for important problem characteristics. In general, these summary measures may be divided into two types: (a) measures of network size, shape, and connectivity, and (b) measures of resource demand, distribution, availability, and contention. With appropriate measures in hand, a set of benchmark projects is needed as a prerequisite to rating and comparing the performance of any solution procedure—optimal, heuristic, or meta-heuristic. 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 do not span the space of 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. Demeulemeester *et al.* (1993) developed the first random generator, which can specify only the number of nodes (activities) and arcs (precedence relationships among activities) 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, its users cannot intentionally generate special network structures. Agrawal *et al.* (1996) extended this work with a generator called *DAGEN*, in which the user can specify network complexity (a relationship between nodes and arcs, discussed below). 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 of multiple resource types. Unpublished work by Schwindt (1995; 1996; 1998) extended *ProGen* to *ProGen/Max*, which 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 all feasible networks. Hence, Demeulemeester *et al.* (2003) developed *RanGen*, which claims to generate strongly random networks that conform to desired values of complexity measures. 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 characteristics of the RCMPSP. Moreover, the results of these generators have not been analyzed for biases in their generation schemes.

This paper contributes what is, to our knowledge, the first random generator of RCMPSPs. It enables the formation of a controlled set of strongly random activity-on-node RCMPSPs with specified project, problem, and resource characteristics, including complexity, resource distribution, and resource contention. In the case of each characteristic, a measure proposed in the former literature is improved upon. For the characteristic of network tiering, we also make several theoretical contributions regarding its effects on network complexity. Both the problem generator and 12,320 resulting test problems are available from the authors upon request. We analyzed these test problems to explore their characteristics and any biases in the generator. We find that, while biases exist, they can be ameliorated at additional computational expense. Therefore, we term our generator “near-strongly random” and find that moving towards “strong randomness” increases computational intensity.

The rest of the paper is organized as follows. In next section, we provide a formal description of the RCMPSP. In §3, we discuss important characteristics and measures in the RCMPSP, upon which we base the proposed generator, which is detailed in §4. §5 describes using the generator to create sets of benchmark problems, and §6 mentions our computational experience with the generator. Our summary and conclusion are presented in §7.

2. Formal Description of the RCMPSP

The RCMPSP can be stated as follows. A set of $l = 2, \dots, L$ projects are to be performed. Each project consists of $i = 1, \dots, N_l$ activities with deterministic, non-preemptable duration d_{il} . The activities are interrelated by predecessor and resource constraints. Predecessor constraints keep activity i from starting until all of its predecessors have finished. Each activity requires r_{ik} 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 $(F_{1l}, \dots, F_{il},$

..., F_{N_l}). Let $A(t)$ be the set of activities in work at time instant t . P_{il} is the set of all immediate predecessors of activity i in project l . $\hat{i} \in P_{il}$. With these definitions, the problem can be formally stated as:

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

$$\text{Subject to: } \quad \forall i \in N_l, \hat{i} \in P_{il}, l \in L: \quad F_{\hat{i}l} \leq F_{il} - d_{il} \quad (2)$$

$$\forall i, l \in A(t): \quad \sum_{i, l \in A(t)} r_{ilk} \leq R_k \quad k \in K; t \geq 0 \quad (3)$$

$$\forall i \in N_l, l \in L: \quad F_{il} \geq 0 \quad (4)$$

The objective function (1) seeks to optimize a pre-specified performance measure. Constraints (2) impose the precedence relations between activities; constraints (3) limit the resource demand imposed by the activities being processed at time t to the capacity available; and constraints (4) force the finish times to be non-negative.

The basic problem can be expanded in several possible ways. Projects might not begin simultaneously, and new projects might arrive at various rates. Project interdependencies (beyond common resources) might exist. Activities could 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. To maximize our insights from the basic RCMPSP, we do not address these complicating features in this paper, although our approach could be extended to do so.

3. Important RCMPSP Measures

A number of problem, project, and resource characteristics affect the RCMPSP. In this section we describe the major ones identified in the literature and their associated measures. Moreover, where these measures are deficient, we present improved ones.

3.1 Problem Size

The number of projects, L , considered in a multi-project problem and the size (number of activities), N_l , of each project influence the degree of resource constraints. For example, all else being equal, a portfolio of five projects would expect greater resource contention and delay than a two-project problem. However, projects can come in various sizes (number of activities), with various amounts of concurrency among the activities, and with varied resource loadings (requirements for resources over time). 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 this reason, we propose holding the problem and project sizes constant, since allowing these to vary could obscure the insights from varying other factors such as network complexity and resource measures. These factors can be varied, however.

3.2 Network Topology and Complexity

Network topology and complexity (just complexity for short) affect RCMPSP results by accounting for the general flexibility in a project. That is, from a network topology perspective, a complex project has a higher ratio of activities to dependencies among them, whereas this ratio is lower in a simple project. A greater number of relationships among the activities leave less flexibility for doing them at a different time (because more things have to happen before them, and more things depend on them before they can occur). Therefore, in its most basic form, a measure of network complexity accounts for the number of activities or nodes¹, N , and precedence relationships or arcs, A .

Pascoe (1966) introduced the first measure of network complexity, the coefficient of network complexity, *CNC*. The *CNC* is defined simply as the ratio of the number of arcs to the number of nodes. For a network with a fixed number of activities, 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-Valdes and Tamarit 1989; Kolisch *et al.* 1995). Elmaghraby and Herroelen (1980) questioned the ability of the *CNC* to reflect 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, network complexity should not be confused with problem difficulty.

Many other network complexity measures have been proposed since the *CNC*. Table 1 summarizes several. The Reduction Complexity measure describes how close a network is to being fully serial or parallel (Bein *et al.* 1992). De Reyck and Herroelen (1996) found that it outperformed the *CNC* in predicting the computational difficulty of RCPSPs. Mastor (1970) introduced another popular measure, network Restrictiveness, also called Order Strength (Thesen 1977). Thesen's definition is based on the number of feasible sequences in a graph, which is impossible to enumerate in practice for large graphs (Elmaghraby and

¹ We assume an activity-on-node representation.

Herroelen 1980). Therefore, Thesen proposed several estimators for his measure of restrictiveness, one of them being the one studied by Schwindt (1995) and De Reyck and Herroelen (1996). Schwindt (1995) conjectured that restrictiveness of the project network is a better measure of complexity than the *CNC* in the RCPSp.

Table 1: Summary of network complexity measures

| Complexity Measure | Definition / Formula | Comments / Reference |
|---|---|---|
| Coefficient of Network Complexity, <i>CNC</i> | $= A / N$, 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 <i>CNC</i> | $= A' / N$ | (Kaimann 1974; Kaimann 1975) |
| Network Complexity, C | $= A' / N$, where A' is the number of non-redundant arcs; $A' \leq A$ | Used by <i>ProGen</i> (Kolisch <i>et al.</i> 1995) |
| Total Activity Density, T-density | $= \sum_{i=1}^N \text{Max}(0, P_i - S_i)$, 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, <i>AAD</i> | $= \text{T-density} / N$ | (Patterson 1976) |
| Cyclomatic Number, S | $= A - N + 1$ | (Temperley 1976) |
| (Network) Restrictiveness, <i>RT</i> Order Strength, <i>OS</i> Network Density, <i>ND</i> | $= \frac{2 \sum a_{ij} - 6(N-1)}{(N-2)(N-3)}$, where a_{ij} is an element of the reachability matrix (defined as the transitive closure of the adjacency matrix), N does <i>not</i> 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, <i>OS</i> (Mastor 1970), and Network Density (Kao and Queranne 1982); related to Flexibility Ratio, F (Dar-El 1973), in that $RT = OS = 1 - F$ |
| Measures of Network Complexity, <i>MNC</i> | $= g(A - e_1, A - N + 1)$, 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, <i>RC</i> Complexity Index, <i>CI</i> | 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 <i>et al.</i> 1992); never adapted for activity-on-node representation (Vanhoucke <i>et al.</i> 2004); later adopted as the Complexity Index, <i>CI</i> (De Reyck and Herroelen 1996) |
| 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 <i>et al.</i> 2000) |

While the simplicity of measures such as the *CNC*, S , and T-density is attractive, their inability to distinguish between redundant² and non-redundant arcs is a critical inadequacy, since redundant arcs should not increase a network's complexity. Therefore, Kolisch *et al.* (1992; 1995) use C , which modifies the *CNC* merely by accounting only for the non-redundant arcs. Since C has seen extensive use in the RCPSp literature, and since it has been used in the few experimental studies of the RCMPSp (Kurtulus and Narula 1985; Lova and Tormos 2001), we propose a measure of network complexity adapted from C , but normalized over $[0,1]$:

$$C = \frac{A' - A'_{\min}}{A'_{\max^*} - A'_{\min}} \quad (5)$$

where A' is the number of non-redundant arcs, A'_{\min} is the lower bound on A' in a network of N nodes, and

² An arc $\langle h, u \rangle$ connecting activities h and u (in a project network) is redundant if there are arcs $\langle i_0, i_1 \rangle, \dots, \langle i_{s-1}, i_s \rangle$ with $i_0 = h$, $i_s = j$ and $s \geq 2$ (Kolisch *et al.* 1995).

A'_{max*} is the upper bound.³ (We will discuss the bounds on A' in the next sub-section.)

Nevertheless, C remains a single-project measure, not a problem measure. The development of a complexity measure for a portfolio of precedence-independent networks remains an area for further research. Therefore, we maintain a separate complexity measure, C_i , for each project and denote problem complexity as a vector of project complexities, $\{C_1, C_2, \dots, C_L\}$.

3.3 Network Tiering

Generating a network with a desired C requires a specific A' relative to N . However, the network topology can constrain A' , making certain levels of C unachievable. To explore this issue, we introduce the concept of tiers (or stages) in a network, which is similar to the concept of morphologic type or hierarchical level (Tavares *et al.* 1999). 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 earlier tiers. Thus, the tiers in a network can be identified with the following algorithm:

1. Assign all activities without any predecessors (in the network) to Tier 1.
2. Out of the unassigned activities, find all activities that (a) do not depend on each other and (b) depend only on activities already assigned to a tier. Assign these to the next tier.
3. Repeat step 2 until all activities are assigned to tiers.

When 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, one per activity. Alternatively, what we define as a maximally parallel network has $T = 2$ tiers, wherein all of the activities except one (either the start or finish activity) work concurrently.⁴ The number of activities in tier j (T_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 $\mathbf{n} = \{n_1, \dots, n_j, \dots, n_T\}$. Activities in T_1 are called start activities, and activities in T_T are called finish activities.

Figure 1 shows an example of the maximum number of non-redundant arcs in two four-tier networks. Figure 1(a) is simply a serial network consisting of four activities. Figure 1(b) has seven activities that are

³ We define a *project* as a network with $A' > 0$ and do *not* include a dummy start and finish node in any of the equations in this paper.

⁴ Since we define a project as a network with $A' > 0$, we do not account for the case where all N activities work in parallel. We are not sanguine that such a case represents a project, because the activities have no relationship to each other except a common start opportunity and the need for all to finish. Such a case corresponds to our definition of a problem (a portfolio of projects) rather than a project. Therefore, we define a project as a network with $A' > 0$ and $T \geq 2$.

distributed among four tiers, where each activity in one tier is connected to every activity in the subsequent tier. Figure 1(c) is the matrix representation of the network in Figure 1(b). In the matrix, diagonal cells represent the activities and sub-diagonal cells represent arcs—specifically, the dependence of the activity in row j on the activity in column i . While both representations convey the same observation regarding tiers and non-redundant arcs, the pattern for the identification and creation of tiers and non-redundant arcs is clearer in the matrix representation. For instance, the dashed boxes in Figure 1(c) show the tiers and the lightly shaded areas represent the location of the allowed links between the various tiers to preclude the production of redundant arcs. Thus, we have four parameters by which to specify a network: N , A' , T , and n . The network in Figure 1(b) and (c) is denoted by $(7, 11, 4, \{1,3,2,1\})$.

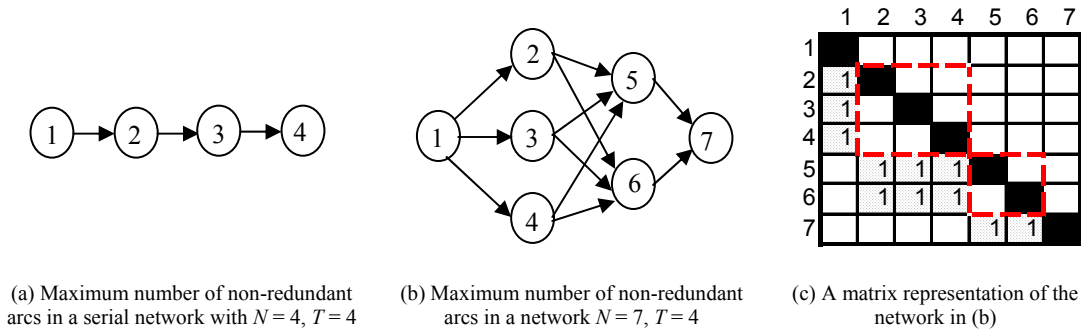


Figure 1: Generation of non-redundant arcs

Because of their importance in randomly generating project networks with desired levels of complexity, we explore some of the properties of tiers in the rest of this sub-section.

Lemma 1: Creating an arc between two activities in a tier increases the number of tiers.

Proof: See Appendix for all proofs.

Lemma 1 prohibits the connection of activities within the same tier if the desired number of tiers must be maintained throughout network generation. For example, consider any two activities from any two tiers, as shown in Figure 2a. These four activities can be connected to each other using a maximum of four non-redundant arcs, shown as solid arrows. An arc connecting activities 1 and 2 also results in the creation of an additional tier, as shown in Figure 2b (which does not show the resultant redundant arcs). Also, if we connect activities 1 and 2 (as shown with the dashed arrow), then this arc renders arcs 1-3 and 1-4 redundant, thus reducing the number of non-redundant arcs to three. (Connecting activities 3 and 4 would have an equivalent effect.)

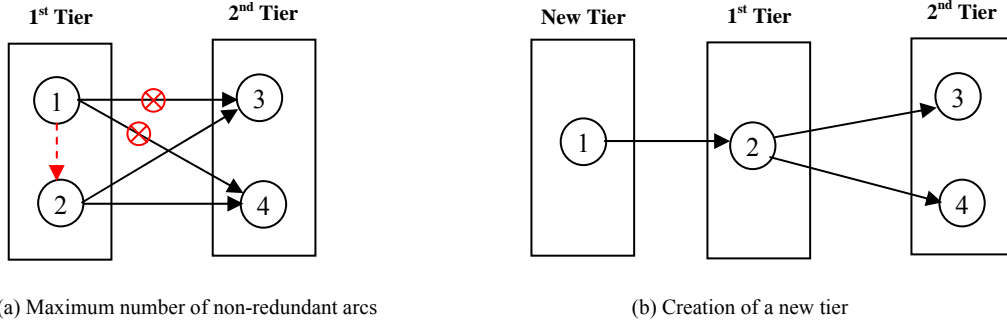


Figure 2: An example of the impact of tiering on non-redundant arc generation

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 $j + 1$.

For example, consider Figure 2b. If we add arc 1-3, then this arc will be redundant. To make this arc non-redundant, we would have to remove arc 1-2 or 2-3. If arc 1-3 existed prior to arcs 1-2 and 2-3, then a procedure would have to be in place to prevent the generation of either one of these arcs. However, if we only allow arc generation between consecutive tiers, then we obviate the need to perform a check of redundancy every 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, \mathbf{n} , is obtained when an arc connects each activity in tier j with each activity in tier $j + 1$:

$$A'_{max}(N, T, \mathbf{n}) = \sum_{j=1}^{T-1} n_j n_{j+1}, \text{ where } \sum_{j=1}^T n_j = N \text{ and each } n \text{ is an integer.}^5 \quad (6)$$

Theorem 1: For a network of size N and T tiers, $A'_{max}(N, T, \mathbf{n})$ is maximized by \mathbf{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 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 (T_1 or T_T). Thus, when using \mathbf{n}^* , the maximum A'_{max} is denoted by A'_{max^*} and is given by:

⁵ A form of Equation (6) 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.

$$A'_{max^*}(N, T, \mathbf{n}^*) = \begin{cases} N^2 / 4 & , \text{if } T \in [2,3] \text{ and } N \text{ is even} \\ (N^2 - 1) / 4 & , \text{if } T \in [2,3] \text{ and } N \text{ is odd} \\ \frac{(N - T + 2)^2}{4} + (N - 2) & , \text{if } T \geq 4 \text{ and } (N - T) \text{ is even} \\ \frac{(N - T + 1)(N - T + 3)}{4} + (N - 2) & , \text{if } T \geq 4 \text{ and } (N - T) \text{ is odd} \end{cases} \quad (7)$$

For example, $A'_{max^*}(20, 2, \{10,10\}) = 100$, by equation (6) or (7). Here, $\mathbf{n}^* = \{10,10\}$, and any other \mathbf{n} will not yield A'_{max^*} —e.g., $A'_{max}(20, 2, \{11,9\}) = 99$ by equation (6). In other examples, $A'_{max^*}(20, 3, \{1,10,9\}) = 100$, $A'_{max^*}(20, 4, \{1,9,9,1\}) = 99$, and $A'_{max^*}(20, 6, \mathbf{n}^*) = 82$, where $\mathbf{n}^* = \{1,8,8,1,1,1\}$ or any other arrangement where (a) the two 8s are adjacent and (b) no 8 is in the first or last tier. For the example of $A'_{max}(10, 2)$, Table 2 enumerates the (non-symmetric) possibilities for the allocation of activities across tiers, where it is obvious that A'_{max^*} is obtained only when $n_1 = n_2$.

Table 2: Five of nine possible allocations of ten activities across two tiers

| n_1 | n_2 | $A'_{max}(10, 2)$ |
|-------|-------|-------------------|
| 5 | 5 | 25 |
| 4 | 6 | 24 |
| 3 | 7 | 21 |
| 2 | 8 | 16 |
| 1 | 9 | 9 |

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

$$A'_{max^*} = \begin{cases} N - 2 + \left(\frac{N-2}{2}\right)^2 & , \text{if } N \text{ is even} \\ N - 2 + \left(\frac{N-1}{2}\right)\left(\frac{N-3}{2}\right) & , \text{if } N \text{ is odd} \end{cases} \quad (8)$$

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 (7) reduce to equation (8). When $T > 4$, however, it is not always possible to achieve the A'_{max^*} implied in equation (8).

Since we will use networks where $N = 20$ later in the paper, we note in this case equation (7) reduces to:

$$A'_{max^*}(20, T, \mathbf{n}^*) = \begin{cases} 100 & , \text{if } T \in [2,3] \\ \frac{(22 - T)^2}{4} + 18 & , \text{if } T \geq 4 \text{ and } T \text{ is even} \\ \frac{(23 - T)(21 - T)}{4} + 18 & , \text{if } T > 4 \text{ and } T \text{ is odd} \end{cases} \quad (9)$$

Corollary 1: $A'_{max^*}(N, T, \mathbf{n}^*)$ 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 (5), we let $A'_{min} = N - 1$ (which can occur in the fully serial case, where $T = N$, or the maximally parallel case, where $T = 2$ and $\mathbf{n} = \{N - 1, 1\} \neq \mathbf{n}^*$)⁶ and $A'_{max*} = N^2 / 4$, such that:

$$C(A', N) = \frac{A' - (N - 1)}{\left(\frac{N^2}{4}\right) - (N - 1)} = \frac{4A' - 4N + 4}{(N - 2)^2} \quad (10)$$

Solving for A' , we get:

$$A'(C, N) = C + N + \frac{CN^2}{4} - CN - 1 \quad (11)$$

where A' must be 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 $A' = 100$. So, a project network of size $N = 20$ and $C = 0.69$ requires the presence of $A'(0.69, 20) = 81(0.69) + 19 = 74.89 \rightarrow 75$ non-redundant arcs.

We seek to generate random networks of a specified size and complexity. Thus, N and C are given, and, according to equation (11), these imply a requisite A' , which increases linearly with C and quadratically with N . T and \mathbf{n} must be chosen so that the number of non-redundant arcs *possible* in the network is greater than this required number—i.e., so that $A'_{max}(N, T, \mathbf{n}) \geq A'$. Theorem 1 tells us that this possibility is maximized when $\mathbf{n} = \mathbf{n}^*$ 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—i.e., where $T > 3$, $\mathbf{n} < \mathbf{n}^*$, and $A' < A'_{max} < A'_{max*}$ —and thus the generation of strongly random networks. According to Corollaries 1 and 2, our choice of T is constrained by A'_{max} . Therefore, we need to understand the implications of A'_{max} for T .

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:

$$T_{max}(C, N) \leq \begin{cases} N + 2 - \sqrt{C(N - 2)^2 + 4} & , \text{ if } (N - T) \text{ is even} \\ N + 2 - \sqrt{C(N - 2)^2 + 5} & , \text{ if } (N - T) \text{ is odd} \end{cases} \quad (12)$$

Theorem 2 implies that generating a network with a desired N and C requires that T be randomly sampled from a distribution over $[2, T_{max}]$. For example, $T_{max}(0.69, 20) = 6$ by equation (12). By equation (11), $A'(0.69,$

⁶ Both the fully serial and maximally parallel cases have $C = 0$, since $A' = N - 1$, but these cases are distinguished by T .

20) = 75. By equation (9), $A'_{max^*}(20, 6, \mathbf{n}^*) = 82$. Since the maximum possible number of non-redundant arcs with a six-tier network is 82, and the network with $C = 0.69$ requires 75 non-redundant arcs, a six-tier network is feasible. Figure 3 shows the required number of non-redundant arcs for a desired C and the T_{max} that would result in an $A'_{max^*}(20, T, \mathbf{n}^*) \geq A'(C, 20)$.

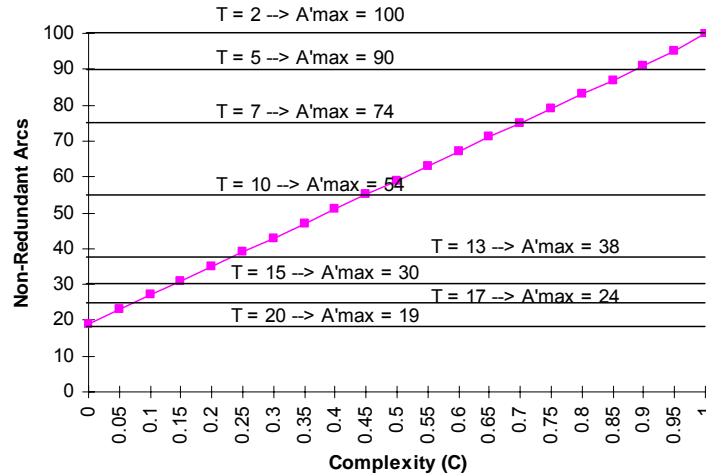


Figure 3: The function $A'(C, 20)$

High C constrains not only T_{max} but also the allocation of activities among the tiers, \mathbf{n} . For example, consider a project where $N = 10$ and $C = 0.75$. By equation (11), $A'(0.75, 10) = 21$. Even if $T = 2$, some of the combinations in Table 2 still do not allow for $A' \geq 21$. If $T = 3$, $A'_{max^*}(10, 3, \mathbf{n}^*) = 25$ with $\mathbf{n}^* = \{4,5,1\}$ or $\{1,5,4\}$. If $T = 4$, $A'_{max^*}(10, 4, \{1,4,4,1\}) = 24$. If $T = 5$, $A'_{max^*}(10, 5, \mathbf{n}^*) = 20$ with $\mathbf{n}^* = \{1,3,4,1,1\}$ or $\{1,1,4,3,1\}$.⁷ Thus, when attempting to generate a network, choosing T near T_{max} drastically limits the possible allocations of activities to tiers.

The number of possible allocations of N activities to any possible number of tiers is $2^{N-1} - 1$. The number of possible \mathbf{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):

$$\mathbf{n}(N, T_{max}) = \sum_{i=1}^{T_{max}-1} \frac{(N-1)!}{(N-i-1)!i!} \quad (13)$$

For example, for $N = 20$, there are 524,287 possible \mathbf{n} s. However, choosing $C = 0.14$ implies that $T_{max} = 14$, which reduces this number to 507,623, and $C = 0.69$ ($T_{max} = 6$) reduces this number to 16,663. Figure 4 shows a plot of $\mathbf{n}(N, T_{max})$ ("Possible"). It also shows the subset of these \mathbf{n} s for which $A'_{max} \geq A'(0.69, 20) = 75$

⁷ In each of these cases, the reverse order of each vector provides an equivalent A'_{max} —e.g., $\{4,5,1\}$ is equivalent to $\{1,5,4\}$.

(“Acceptable”) and the ratio of these two functions as a percentage. 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 T to something even slightly less than T_{max} will dramatically decrease the number of generation failures and thus the computational intensity. Generating strongly random networks (in the purest sense) requires searching a landscape largely devoid of successful allocations.

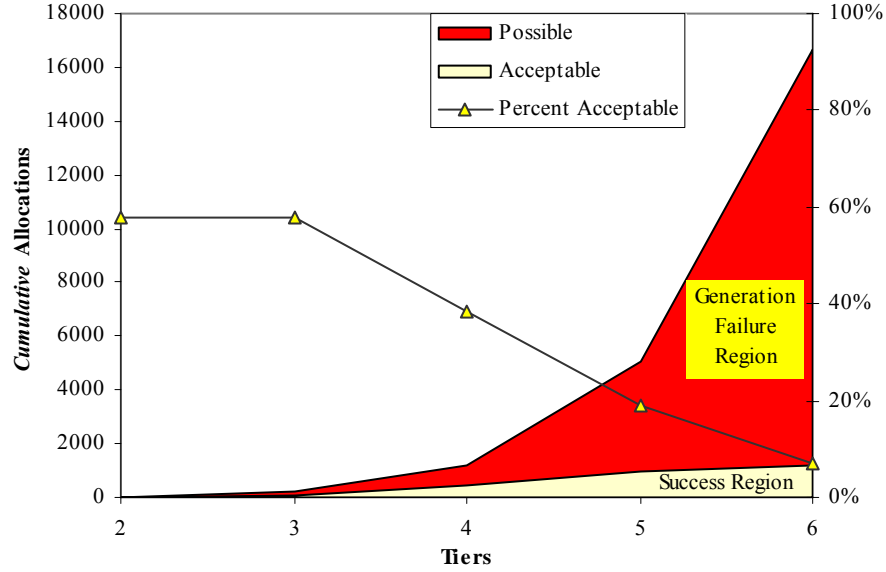


Figure 4: Graphs of cumulative number of allocations of 20 activities to tiers (with $C = 0.69$)

3.4 Resource Measures

Several measures for the availability and distribution of project resources have been developed for the RCPSp. For example, two of the many RCPSp measures are the resource factor, RF , which indicates the average number of resources used by an activity (Pascoe 1966), and the resource strength, RS , which expresses the relationship between resource requirements and resource availability (Cooper 1976; Kolisch *et al.* 1995). They are respectively defined in the following two equations:

$$RF = \frac{1}{NK} \sum_{i=1}^N \sum_{k=1}^K \begin{cases} 1 & \text{if } r_{ik} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

$$RS_k = \frac{R_k - r_k^{\min}}{r_k^{\max} - r_k^{\min}} \quad (15)$$

where R_k is the total availability of resource k , $r_k^{\min} = \max_{i=1}^N r_{ik}$, and r_k^{\max} is the peak demand of resource k .

However, Kurtulus and Davis (1982) note that RF and RS are not very useful for a multi-project environment. For the RCMPSP, they defined two related but alternative measures.

3.4.1 Multi-Project Resource Distribution

To measure resource distribution, Kurtulus and Davis (1982) proposed the average resource loading factor, $ARLF$, which identifies whether the bulk of a project's total resource requirements are in the front or back half of its critical path duration⁸ and the relative size of the disparity. For project l , it is defined as:

$$ARLF_l = \frac{1}{CP_l} \sum_{t=1}^{CP_l} \sum_{k=1}^{K_{il}} \sum_{i=1}^{N_l} Z_{ilt} X_{ilt} \left(\frac{r_{ilk}}{K_{il}} \right) \quad (16)$$

$$\text{where } Z_{ilt} = \begin{cases} -1 & t \leq CP_l/2 \\ 1 & t > CP_l/2 \end{cases} \quad (17)$$

$$X_{ilt} = \begin{cases} 1 & \text{if activity } i \text{ of project } l \text{ is active at time } t \\ 0 & \text{otherwise} \end{cases}, \quad (18)$$

$Z_{ilt}X_{ilt} \in \{-1, 0, 1\}$, N_l is the number of activities in project 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 . Projects with $ARLF < 0$ are “front loaded” in their resource requirements, while projects with $ARLF > 0$ are “back loaded.”

However, as we discuss in detail elsewhere (Browning and Yassine 2007), $ARLF$ is only a rough indicator of a project's resource distribution. It can fail to distinguish between significantly different cases. For example, two projects with dramatically different resource distributions, a uniform distribution and a bi-modal distribution, may both have $ARLF_l \approx 0$. Furthermore, K&D's definition of a multi-project problem's $ARLF$ as the average of its constituent projects' $ARLFs$ is problematic, especially because differences in the lengths of a problem's projects are not considered in equation (16). To address this issue, we proposed (Browning and Yassine 2007) a new way to measure a problem's resource distribution by normalizing over its critical path duration:

$$NARLF = \frac{1}{L \cdot CP_{max}} \sum_{l=1}^L \sum_{t=1}^{CP_l} \sum_{k=1}^{K_{il}} \sum_{i=1}^{N_l} Z_{ilt} X_{ilt} \left(\frac{r_{ilk}}{K_{il}} \right) \quad (19)$$

where $NARLF$ stands for *normalized ARLF* and $CP_{max} = \text{Max}(CP_1, \dots, CP_L)$.¹⁰

⁸ Based on scheduling each activity at its early start time (the “all EST” schedule)

⁹ The original definition of Z_{nlt} in (Kurtulus and Davis 1982) and (Kurtulus 1978, p. 59) assumes activities are indexed from 0 to $N_l - 1$ and therefore puts the equal sign in the second case rather than the first—i.e., $t \geq CP_l / 2$. However, it seems more intuitive to index activities from 1 to N_l . For example, in a 10-day project, with days numbered 1-10, activities on day 5 ($10 / 2 = 5$) are assigned to the first half of the project.

3.4.2 Multi-Resource Contention

To measure resource contention, Davis (1975) proposed the *utilization factor*, UF , which is calculated for each resource type as the ratio of the total amount required to the amount available in each time period, based on the problem's critical path duration. If $UF_k < 1 \forall k$ in each time period, then there is no resource contention. To reduce computational intensity, Kurtulus and Davis (1982) proposed averaging UF over intervals to get an *average utilization factor*, AUF . Using Figure 5 as an example, they proposed using $S = 3$ intervals, where $S_1 = CP_1$, $S_2 = CP_2 - CP_1$, and $S_3 = CP_3 - CP_2$, once the projects have been sorted from shortest to longest, such that $CP_1 \leq CP_2 \leq CP_3$.¹¹ The total amount of resource k required over any interval s is given by:

$$W_{sk} = \sum_{l=a}^b \sum_{l=1}^L \sum_{i=1}^{N_l} r_{ilk} X_{ilt} \quad (20)$$

where $a = CP_{s-1} + 1$, $b = CP_s$, r_{ilk} is the amount of resource k required by the i^{th} activity in project l , and X is defined in equation (18). The AUF indicates the average tightness of the constraints on (i.e., the average amount of contention for) each resource type:

$$AUF_k = \frac{1}{S} \sum_{s=1}^S \frac{W_{sk}}{R_k S} \quad (21)$$

where R_k is the (renewable) amount of resource type k available at each interval. Since the AUF is essentially a ratio of resources required to resources available, averaged across intervals of problem time, $AUF_k > 1$ indicates that resource type k is, on average, constrained over the course of a problem. To get the AUF for a problem involving K types of resources:

$$AUF = \text{Max}(AUF_1, AUF_2, \dots, AUF_K) \quad (22)$$

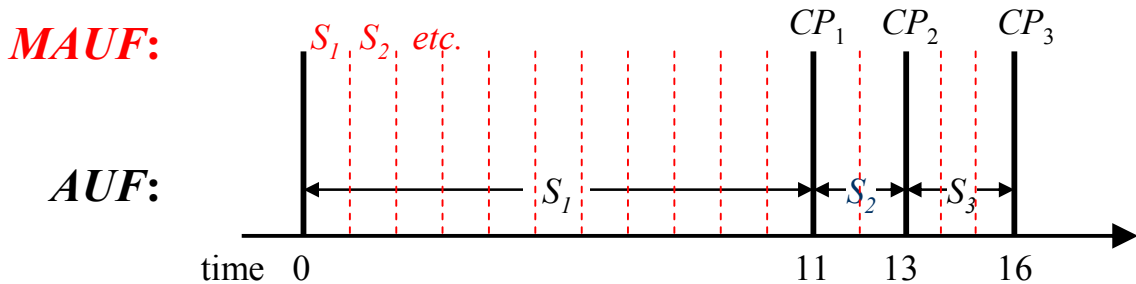


Figure 5: Example of time intervals formed by three projects in a problem

¹⁰ Since $ARLF$ and $NARLF$ assume fungible resources, they are sensitive to disparities in the number of types, K . For example, in a three-project problem, if one of the projects uses four types of resources and the other two projects use only one of those types, then $K = 4$. If all three projects use four types of resources, then $K = 4$ also.

¹¹ K&D (1982, p. 163) use M instead of S and define it as the number of projects, but if projects have equal durations, actually $S \leq L$.

In (Browning and Yassine 2007) we identify two issues with the AUF . First, if the projects constituting a problem are approximately equal in duration (which is especially likely to occur when the number of activities per project is equal), then $S_1 \gg S_{s>1}$, and averaging over these disproportionate intervals can obscure the situation. To ameliorate this issue, we average over equal intervals of problem time, such as the integer units indicated by the dashed, vertical lines in Figure 5.¹² Thus, in Figure 5's example, $S = 16 = CP_{max}$. We call this measure the *modified AUF* or $MAUF$, and equations (20-22) hold, although S is determined differently.

Second, determining a problem's AUF or $MAUF$ using equation (22) fails to distinguish between significantly different problems. For example, if a problem with three types of resources has $AUF_1 = 1.6$, $AUF_2 = 1.58$, and $AUF_3 = 1.59$, then $AUF = 1.6$ by equation (22). Since these three AUF_k s are almost equal and all greater than one, all three types of resources are highly constrained, and any set of activities that is unconstrained by the first type of resource is very likely to be constrained by one or both of the other types. However, if another problem has $AUF_1 = 1.6$, $AUF_2 = 0.6$, and $AUF_3 = 0.6$, then only the first type of resource is highly constrained, although overall problem's AUF is still 1.6. Hence, two problems can have very different amounts of resource contention yet identical AUF s and $MAUF$ s. To provide a clearer picture of resource contention, we augment the $MAUF$ with a measure of the variance in the $MAUF_k$ s:¹³

$$\sigma_{MAUF}^2 = \frac{\sum_{k=1}^K (MAUF - MAUF_k)^2}{K} \quad (23)$$

4. The Multi-Project Problem Generator

4.1 Setting Input Values

The user specifies the problem inputs shown in Table 3. The number of projects must be ≥ 1 and can be chosen randomly or set by the user.¹⁴ In our study of RCMPSP priority rules (Browning and Yassine 2007), we found it helpful to control for L . The number of activities is typically ≥ 2 , although single-activity projects are not prohibited. The generator can choose N randomly or the user can specify a desired N . The number of resource types, K , can vary by project and be set by the user or determined at random (typically in the range [1,10]). However, since K affects the $NARLF$, there is advantage in holding it constant also. $K = 4$ seems to

¹² The actual size of these intervals can be chosen to limit the computational intensity as and if necessary.

¹³ Note that this is a variance from the maximum, not from the mean.

¹⁴ This generator can also be used for single-problem studies.

allow enough variety and leaves room for the resource usage manipulations done by the algorithm (discussed below). The user may choose a desired complexity, C , for each individual project in the problem, or it can be 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 $NARLF_{des} > 0$.¹⁵ Similarly, solutions with $MAUF_{des} > 1.6$ become difficult to generate (especially if $\sigma_{MAUF_{des}}^2 = 0$), while problems with $MAUF_{des} < 0.6$ usually cease to be interesting (because the resource constraints mostly disappear).¹⁶ The desired variance in a problem's $MAUF$ s, $\sigma_{MAUF_{des}}^2$, can be set = 0, indicating a desire for all resource types in the problem to be similarly constrained, or > 0 , meaning that one resource type will tend to be more constrained than the others. (Below, we will discuss the rationale for $\sigma_{MAUF_{des}}^2 = 0.25$ as an upper bound.) The other input variables listed in Table 3 are discussed below, at the points they are used in the generator algorithm.

Table 3: User input variables

| 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 |
| $C_{des,l}$ | Desired complexity of project l | $[0,1]$ |
| $NARLF_{des}$ | Desired $NARLF$ | $[-3,3]$ |
| α | $NARLF$ sensitivity threshold | ~ 0.025 |
| $MAUF_{des}$ | Desired $MAUF$ | $[0.6,1.6]$ |
| β | $MAUF$ sensitivity threshold | ~ 0.025 |
| $\sigma_{MAUF_{des}}^2$ | Desired $MAUF$ variance | $[0,0.25]$ |

4.2 Generating the Basic Project Networks

Figure 6 provides an overview of the algorithm for generating the basic project networks. 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 K&D (1982), we let d_{il} equal a random integer in $[1,9]$.¹⁷ To enable the establishment of non-redundant precedence relationships between the activities, the activities must be allocated to tiers. The number of tiers, T_l , is chosen based on the desired complexity level, as described in §3.3. For example, if $C_{des,l} = 0.69$, then $T_{max} = 6$. However, to reduce computational intensity, we set the generator to sample T_l from $[2, T_{max} - 1]$.

¹⁵ In equation (18), $NARLF$ 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 at work towards the end. This effect is ameliorated somewhat by using a constant N for each project.

¹⁶ K&D (1982) used $ARLF$ settings over $[-3,3]$ and AUF settings over $[0.6,1.6]$.

¹⁷ This and many other parameters in the generator are easily adjustable by the user.

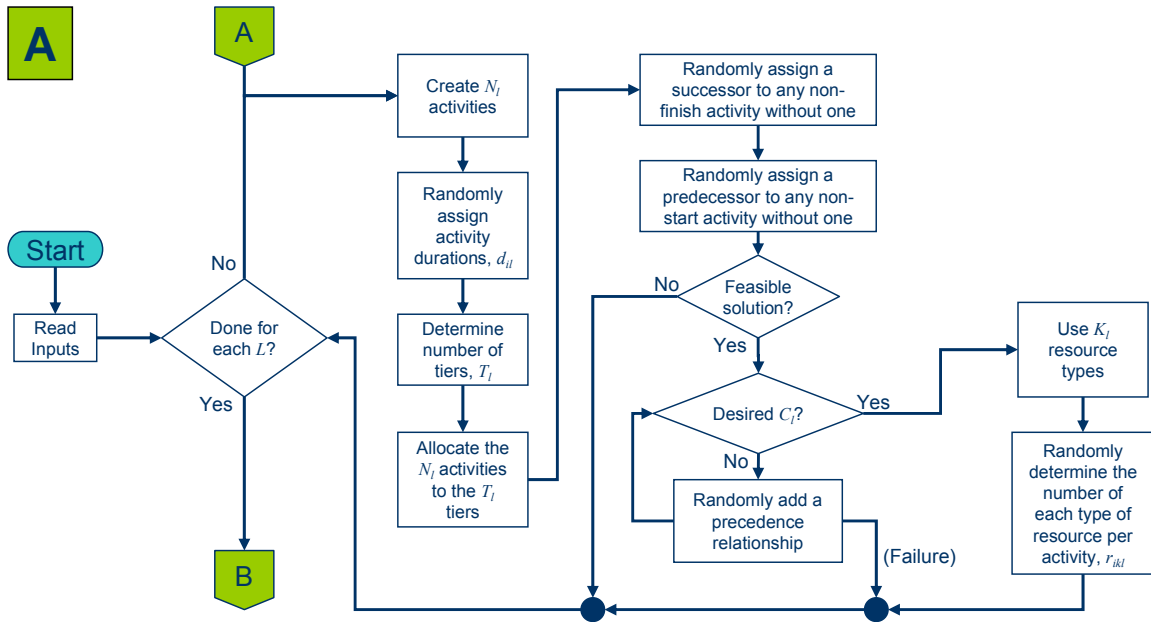


Figure 6: Overview of algorithm (part A) for generating the basic project networks

Once T_i is selected, the N_i activities are allocated to the tiers. After some efforts 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. The allocation is accomplished by the algorithm in Table 4, which uses N and T instead of N_i and T_i for simplicity.

Once the activities have been allocated to the tiers, non-redundant arcs are created. Each non-finish activity is assigned a random successor activity 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 the desired complexity is low. That is, randomly assigning successors and predecessors in this way may result in A' greater than the desired number (which is based on the desired complexity, as described in §3.2). In such a case, the algorithm returns to the beginning and regenerates the current project. In most cases, however, the A' established by this procedure will be less than the desired number. In this event, additional successors (always in the adjacent tier, so as to ensure non-redundancy) are added at random until the desired complexity is achieved.¹⁸ Here it is also possible for the algorithm to fail if there are no remaining positions to create non-

¹⁸ The procedure described by Kolisch *et al.* (1992; 1995) to generate project networks with a specified C requires an exhaustive checking for whether a randomly added arc is redundant or renders other existing arcs redundant. This operation is time-consuming and

redundant dependencies, given the selected tier structure and activity allocation to it. In this case also, the algorithm returns to the beginning and regenerates the current project.

Table 4: Algorithm for allocating N activities to T tiers

| |
|---|
| <ol style="list-style-type: none"> 1. Determine the number of front-half tiers, $T_a = \text{Integer}(T / 2)$. 2. Designate half of the activities as “front-half” activities, N_a.¹⁹ $N_a = N / 2$. 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: $T_i = \text{Random Integer in } [1, 10]$. (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: $T_i = \text{Closest Integer}(T_i \cdot N_a / temp1)$. (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 <i>actually</i> 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 $\neq N_a$) $temp2 = \text{Random Integer in } [1, T_a]$. (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: $T_i = \text{Random Integer in } [1, 10]$. $temp1 = temp1 + T_i$. For $i = T_a + 1$ to T: $T_i = \text{Closest Integer}[T_i \cdot (N - N_a) / temp1]$. $temp1 = 0$. 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 <i>actually</i> 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 $\neq N - N_a$) $temp2 = \text{Random Integer in } [T_a + 1, T]$. $temp3 = temp1 - N + N_a$. ($temp3$ 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.) $temp1 = temp1 + 1$. |
|---|

Once a network with the desired complexity has been generated, each activity is assigned a random

in the worst case may produce an infeasible network (in which case the procedure must be aborted and restarted). Our simple procedure is more likely to generate a network with a pre-specified complexity. In contrast, Demeulemeester *et al.* (2003) start with a completely connected network and remove redundant arcs one at a time to get a desired OS value, after which they calculate CI for the obtained network.

¹⁹ 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.

number of required resources (we use integers in $[1,9]$) of each type.

4.3 Calibrating to the Desired $NARLF$

Having generated L project networks, the program then calibrates the problem's $NARLF$ to $NARLF_{des}$ by adjusting its projects' resource requirements. Figure 7 provides an overview of this process, which begins by initializing an iteration counter that will track the number of adjustments. If this number exceeds a threshold (we use 500),²⁰ 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_i , for each project by calculating the early start (ES), early finish (EF), and slack times for each activity in its network. The greatest CP_i is designated CP_{max} , and the ES schedules are used to calculate the $NARLF$ using equation (18). 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.

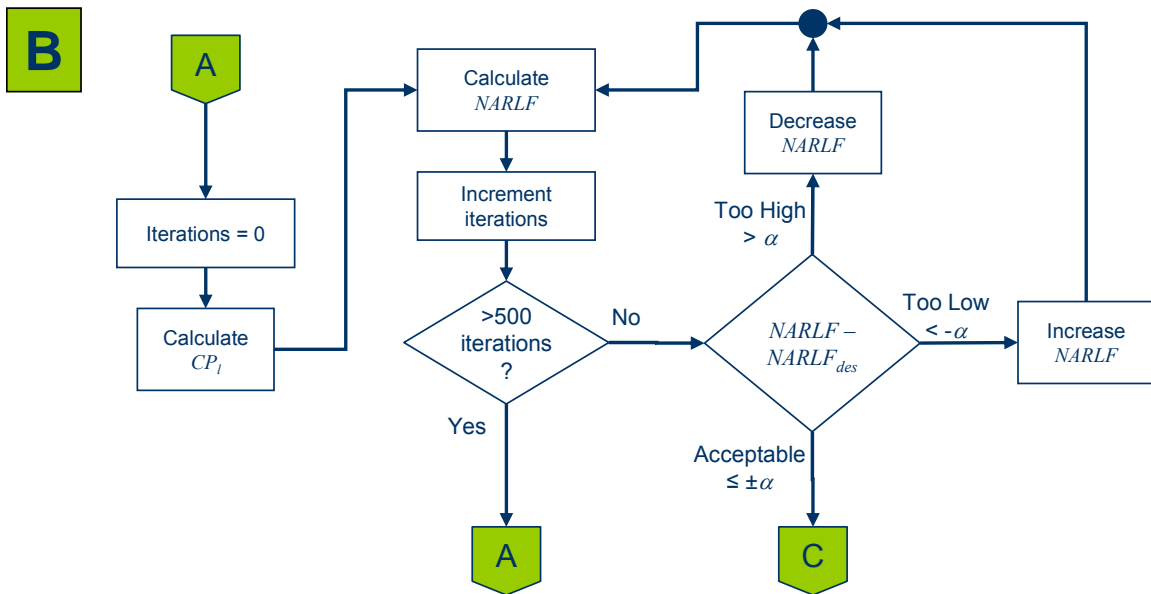


Figure 7: Overview of algorithm (part B) for calibrating a problem to $NARLF_{des}$

The $NARLF$ can be adjusted in a variety of ways. It depends on CP_{max} , the numbers of activities, the number of types of resources, the amount of each type of resource used, and, most importantly, the location of the point of usage of those resources relative to the mid-point of the problem. Thus, we can increase (decrease)

²⁰ The appropriate limit on the number of iterations depends on factors that figure into the determination of $NARLF$. We set the iteration limit $\approx \text{Max}(r_{ik})/2 \cdot (N/2) \cdot KL = \text{Max}(r_{ik})NKL/4$, with the rationale that too many adjustments might lead to overly biased results. Therefore, we base the appropriate limit on changes on half of the activities being in the front and back halves of the project ($N/2$), each activity having an average of $\text{Max}(r_{ik})/2$ resources of each type K , and the presence of L projects in the problem.

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 the number of activities constant, however, we must adjust the *NARLF* by changing the number of resources required by the activities. To increase the *NARLF*, we select (1) a random activity, i , from a random project and (2) a random resource type, k . If the activity is predominately in the front-half of the problem—i.e., if $ES_i + (d_i / 2) < \text{Integer}(CP_{max} / 2)$ —then one unit of the selected resource is removed from it (if its requirement for resource type k was already greater than one; otherwise the random selections recur). If the activity is predominately in the back-half of the problem, then one unit of required resource type k is added to it (if the requirement was already less than nine; 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.²¹ Hence, we can choose a relatively small $\alpha = 0.025$ without much risk of an unreachably accurate $NARLF_{des}$.

4.4 Calibrating to the Desired *MAUF*

Next, the problem must be made to satisfy the $MAUF_{des}$ by setting the number of resources available. Figure 8 provides an overview of this portion of the algorithm.²² First, the algorithm determines $MAUF_{des,k}$ for each resource type, k . If $\sigma^2_{MAUFdes} = 0$, then $MAUF_{des,k} = MAUF_{des} \forall k$. If $\sigma^2_{MAUFdes} > 0$, then $MAUF_{des,1} = MAUF_{des}$ and

$$MAUF_{des,k} = MAUF_{des} - \sqrt{\sigma^2_{MAUFdes}} \quad \forall k > 1. \quad (24)$$

Since equation (22) is a maximization function, equation (24) does not depend on k . Thus, we let the first resource type, $k = 1$, have the greatest $MAUF_{des}$, and let the other resource types, $k > 1$, provide any needed difference in their $MAUF_{des}$ so as to yield $\sigma^2_{MAUFdes}$.²³

After determination of $MAUF_{des,k} \forall k$, the algorithm calculates each W_k according to equation (20). To get $AUF_{des,k}$ we solve equation (21) for R_k :

²¹ 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 (non-critical) activities.

²² This algorithm can also be used for *AUF* instead of *MAUF*.

²³ Note, however, that if $\sigma^2_{MAUFdes} = 0$, then $MAUF_{des,1} = \text{Max}(MAUF_{des,k})$ is not guaranteed to high precision, since precision is limited by whole unit resource amounts in the determination of $MAUF_{des,k} \forall k$, as discussed below.

$$R_k = \text{Closest Integer} \left[\frac{\sum_{s=1}^S \left(W_{sk} \prod_{j=1, j \neq s}^S S_j \right)}{S \cdot AUF_{des,k} \prod_{s=1}^S S_s} \right]. \quad (25)$$

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

$$R_k = \text{Closest Integer} \left(\frac{\sum_{s=1}^S W_{sk}}{S \cdot MAUF_{des,k}} \right). \quad (26)$$

This R_k is then used in equation (21) to determine $MAUF_k \forall k$, with which the problem's $MAUF$ and σ^2_{MAUF} can be found using equations (22-23). Sufficiently small differences between σ^2_{MAUF} and $\sigma^2_{MAUF_{des}}$ are not regarded since they tend to be less important in the subsequent analysis of the test problems.

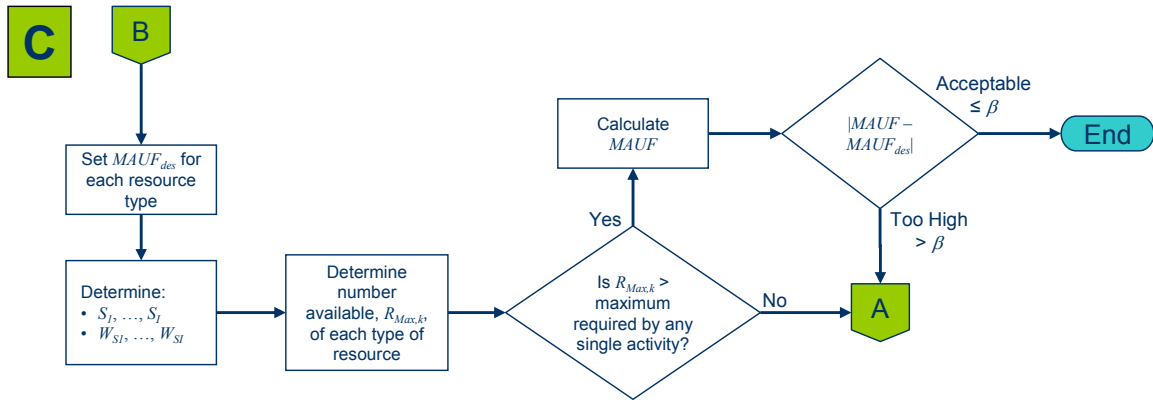


Figure 8: Overview of algorithm (part C) for calibrating a problem to $MAUF_{des}$

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 the resulting value of R_k implies an actual $MAUF_k$ such that $|MAUF_k - MAUF_{des,k}| > \beta$, where β is a measure of the desired precision, then the problem must be regenerated. This type of failure occurs often, and 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 almost the very

beginning of the algorithm. 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. 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 $MAUF_{des} = 1.5$, which could actually have $1.475 \leq MAUF \leq 1.525$. Decreasing β increases computational time by increasing the likelihood of a $MAUF$ 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 profile, and the overall problem has a desired amount of average resource utilization for each resource type.

Note on choosing $\sigma_{MAUF_{des}}^2$: Since $\sigma_{MAUF_{des}}^2$ is a variance from a maximum (instead of from a mean), as that max grows (away from zero), so does the upper bound on the feasible $\sigma_{MAUF_{des}}^2$. 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). If $MAUF_{des}$ is low (e.g., 0.6), however, 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 $\sigma_{MAUF_{des}}^2$. That is, equation (26) will fail as $MAUF_{des,k} \rightarrow 0$. Thus, when setting $\sigma_{MAUF_{des}}^2$ for a test bank in which problems will vary over a wide range of $MAUF_{des}$, $\sigma_{MAUF_{des}}^2$ 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 $\sigma_{MAUF_{des}}^2 = 0.25$ works well.

5. Generating a Set of Problems to Desired Specifications

To generate a set of problem instances with varied $NARLF$ s and $MAUF$ s, we add two outer loops to the individual problem generator. The inner of these two additional loops addresses $MAUF$ and is shown in Figure 9. As discussed in the last section, 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 (26). (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 in the problem to vary the MAUF.²⁴) As $MAUF_{des}$ decreases, so does the likelihood of a failure to find a MAUF that satisfies $|MAUF_k - MAUF_{des,k}| \leq \beta \forall k$, although such a failure is still possible. 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 these two additional loops addresses NARLF. This loop is placed on the outside because the test problems with different NARLFs are unrelated (unlike the test problems with varied MAUFs, 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.

6. Computational Results

Although we had a variety of platforms to select from, we implemented the generator in Microsoft Excel and Visual

Basic. While this limits our ability to analyze absolute computational intensity, it provides advantages in simplicity, ease of use, and likelihood of dissemination into industrial practice. Roughly speaking, generating a set of 11 problems with a single NARLF value and 11 MAUF values takes about 1 to 9 minutes on a laptop computer with an Intel Pentium 4M 2GHz processor, although most of this time is spent writing the output to the spreadsheets. The actual computational time varies from a few seconds to about 6 minutes, depending on the

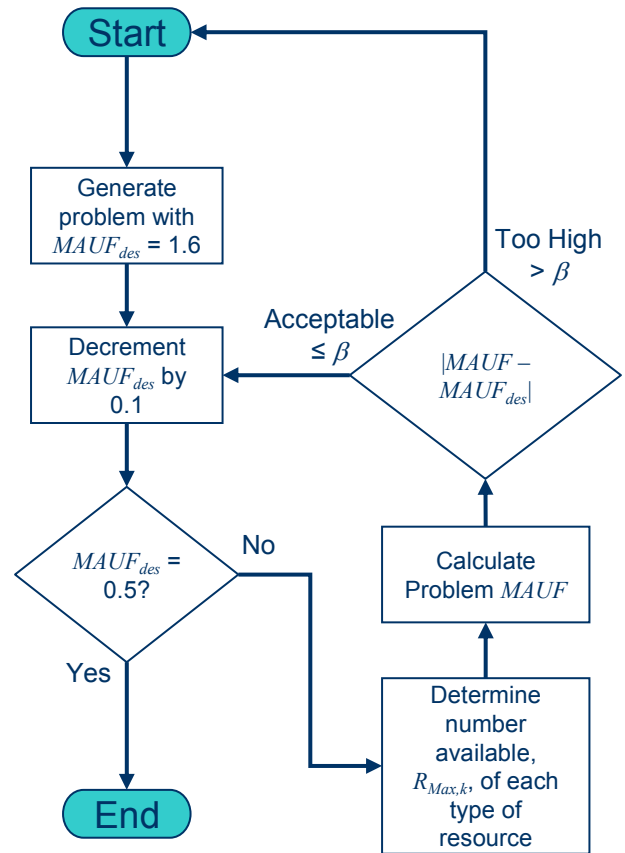


Figure 9: Overview of algorithm for generating a set of 11 problems with varied MAUFs

²⁴ Of course, it is possible to regenerate the entire problem for each new value of $MAUF_{des}$, but this approach yields test problems that differ in other ways besides MAUF, thus complicating their direct comparison.

number of times the algorithm must restart. A problem’s desired settings influence the actual time considerably. Because they involve fewer constraints, low-complexity problems tend towards six minutes per set of 11, while high-complexity problems tend to require only a few seconds to generate. Since our focus was on successful problem generation rather than efficiency, the generator could certainly be sped up, e.g., by implementing the algorithm in a stand-alone application and writing entire sets of problems to a single output file. In the rest of this section, we investigate how the settings affect the generation process and the types of networks generated.

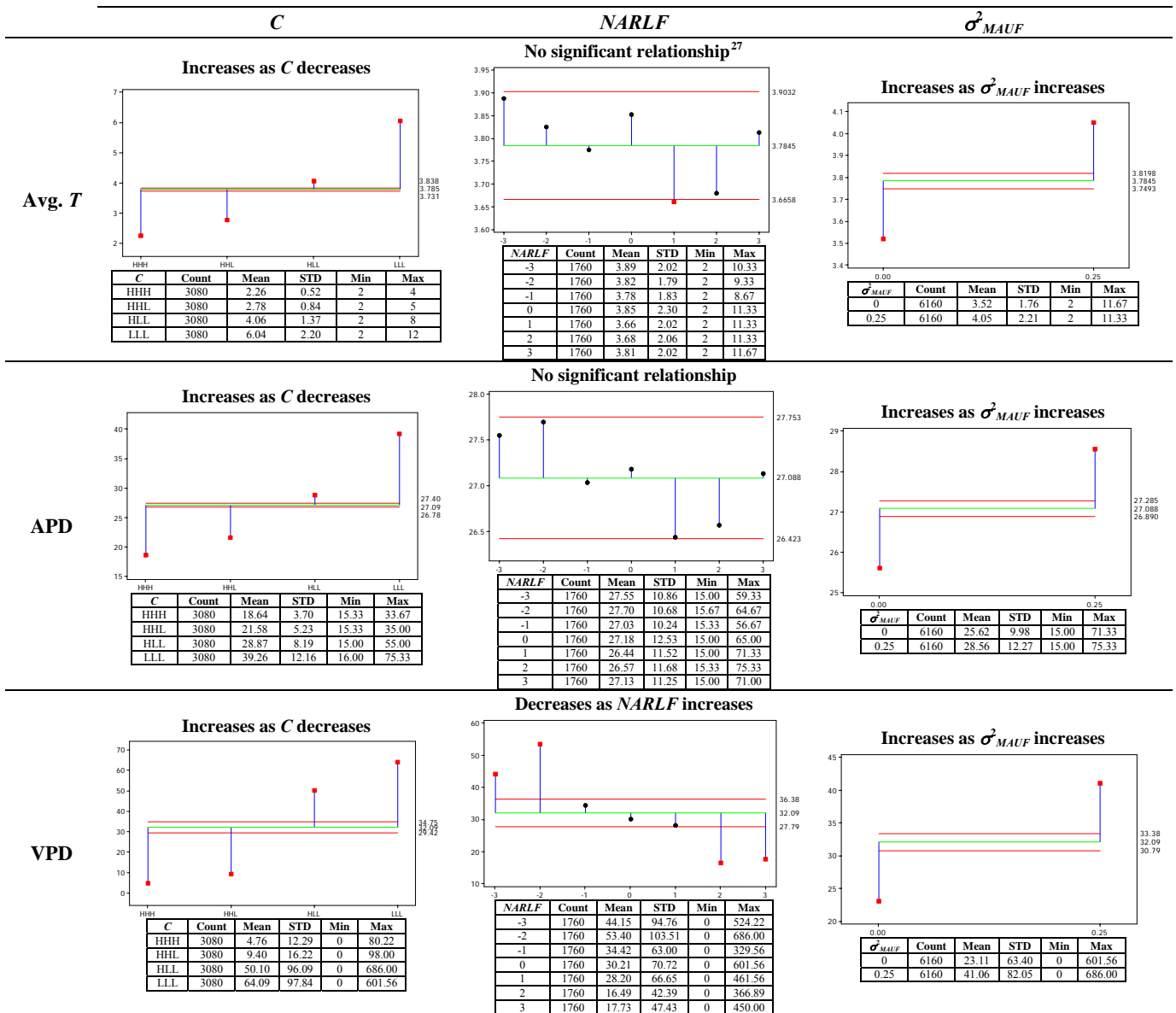
We designed a full factorial experiment to test the influence of the factors listed in Table 3. To maximize the insights from varying the other factors, we held $N = 20$, $L = 3$, and $K = 4$ constant.²⁵ The choices for *NARLF* and *MAUF* levels follow K&D (1982), with *NARLF* in seven integer increments over $[-3,3]$ and *MAUF* in eleven 0.1 increments over $[0.6,1.6]$. We designated two levels of project complexity, “high” ($C = 0.69$) and “low” ($C = 0.14$). We used these to form four variations in problem complexity: all high-complexity projects (“HHH”), all low-complexity projects (“LLL”), and two intermediate combinations (“HHL” and “HLL”). Furthermore, we wanted some problems where all of the individual resources’ *MAUF*s were equal (i.e., where $\sigma_{MAUF,des}^2 = 0$) and others where one resource’s *MAUF* determined the overall problem’s *MAUF_{des}* while the other three types of resources had a significantly different *MAUF*. 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 a series of analyses of means (ANOMs) of the attributes of the generated problems: average number of generated tiers (T), average project duration (APD), variance in project duration (VPD), number of *MAUF* and *NARLF* failures during generation, and total generation time, each as a function of C , *NARLF*, and σ_{MAUF}^2 .²⁶ Note that generation time is driven by the number of attempts required to generate a successful problem, and the number of attempts is equal to *NARLF* failures + *MAUF* failures + 1. These ANOMs lead to the results shown in Table 5.

²⁵ No specific relationship has been reported between portfolio size or project size and the solution quality obtained by the various heuristics (Hartmann and Kolisch 2000; Kurtulus and Davis 1982; Lova and Tormos 2001). Meanwhile, since K is used to determine *NARLF* (equation (9)), its variation would be confounded with *NARLF*.

²⁶ Since all *MAUF* levels are based on a single, common network, we generally found the *MAUF* results uninteresting. 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.

Table 5 (Part 1 of 2): ANOM results with main factors (statistics computed with $\alpha = 0.1$)

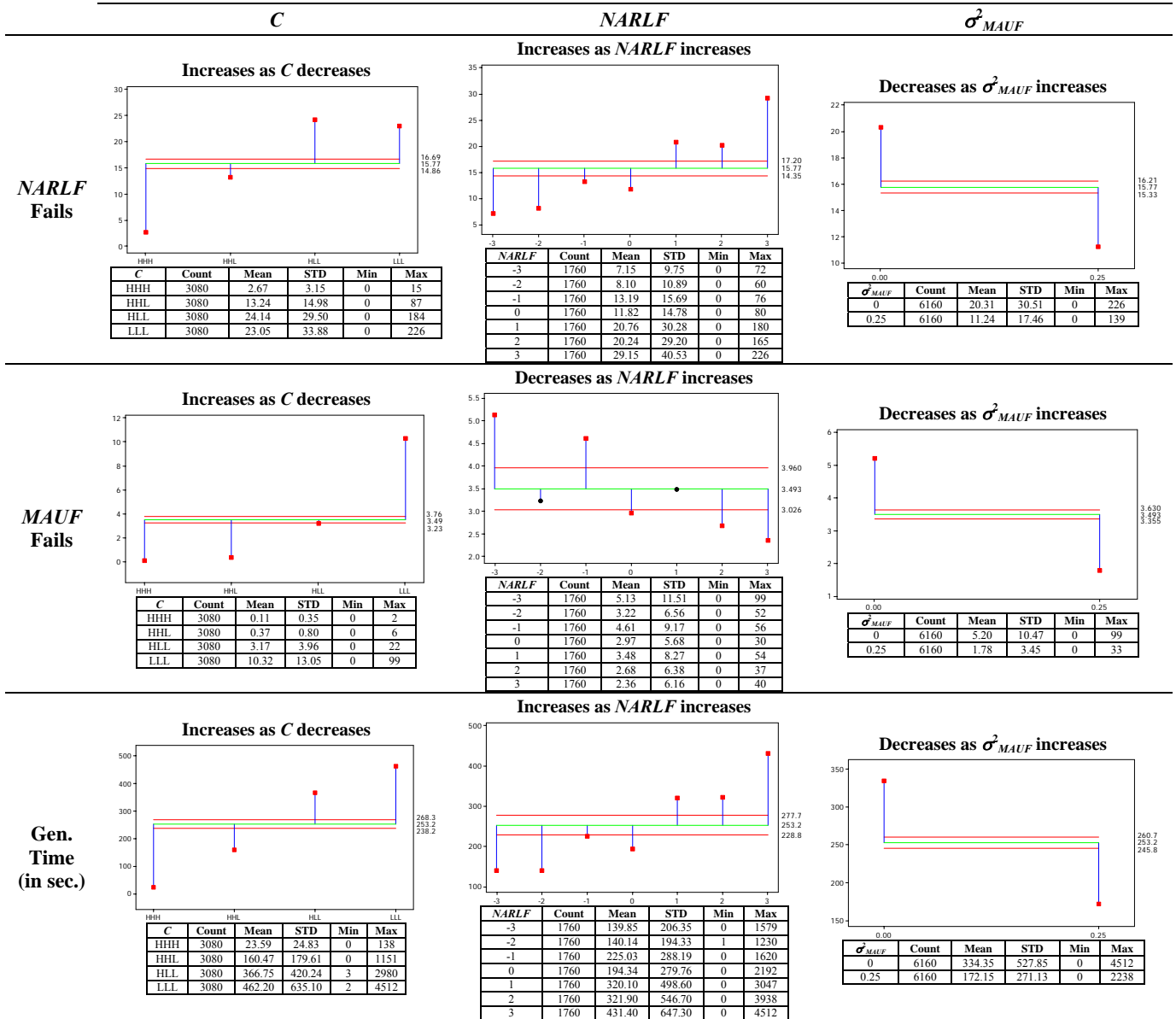


Looking at both parts of Table 5, all six row factors increase as C decreases. This is because high-complexity problems are very inflexible (due to the high number of predecessor constraints). Because they tend to have more tiers, low-complexity problems are also longer and more varied in the lengths of their constituent projects. Thus, the search space for low-complexity problems is much larger, as is the percentage of that space with unsuccessful outcomes.

As $NARLF$ increases, the durations of the projects in a problem become more varied. Since (1) all projects in a problem begin simultaneously and (2) the duration of a problem is determined by the duration of its longest

²⁷ Looking at only the HLL and LLL problems did not change this, except the $NARLF = 0$ problems took significantly longer.

Table 5 (Part 2 of 2): ANOM results with main factors (statistics computed with $\alpha = 0.1$)



project, a bias exists towards problems with low *NARLF*. High-*NARLF* problems have to be attained through a relatively greater loading of resource needs from the later activities in the longest project. This bias seems small enough when the number of activities is held constant, because this ensures projects with relatively similar durations (especially when complexity is high). The bias becomes more prominent, however, 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, some of the projects are carrying on much longer than the others. Hence, as *NARLF* increases, meaning back-loaded problems are desired, so does generation time, mainly due to more *NARLF* failures. More *NARLF* failures reduce the chances of a network with high variance in project durations

being generated successfully. Meanwhile, *MAUF* failures become less likely as *NARLF* increases, probably due to the fact that 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.

As σ_{MAUF}^2 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 $\sigma_{MAUF}^2 = 0.25$ only one type of resource is so constrained. Thus, when $K = 4$, a *MAUF* failure is four times more likely with $\sigma_{MAUF}^2 = 0$ than with $\sigma_{MAUF}^2 = 0.25$.

The relationship between C and T is especially interesting. From equation (12) we know that for the HHH problems ($C = 0.69$) $T_{max} = 6$. But we used $T_{max} = 5$ to reduce computational intensity. Therefore, for the HHH problems, the generator samples T from a uniform distribution over $[2,5]$, which has an expected value of 3.5 tiers. However, the average T for the HHH problems is actually 2.26 (Table 5). Figure 10a shows the distribution of T by C for the 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 4. (Figure 10b shows the average number of attempts until successful generation, which grows with T .) This bias can be compensated for by sampling from a non-uniform distribution, weighted towards T_{max} , although this would even more drastically increase computational time.

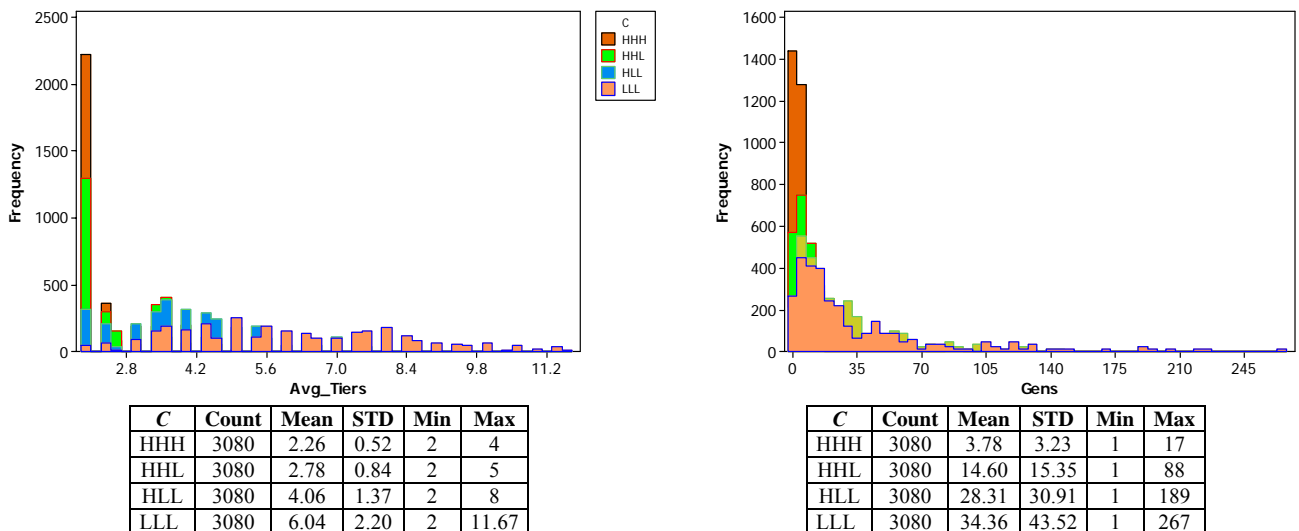


Figure 10: Stacked histograms of average tiers and number of generation attempts by problem complexity

In addition, we analyzed the two-way interactions for the generation times by C , $NARLF$, and σ^2_{MAUF} as shown in Figure 11. The figure shows 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 (HHH), $NARLF$ level does not influence generation time, but it does when C is low. Similar interaction plots were made for the average number of tiers but these showed 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 .

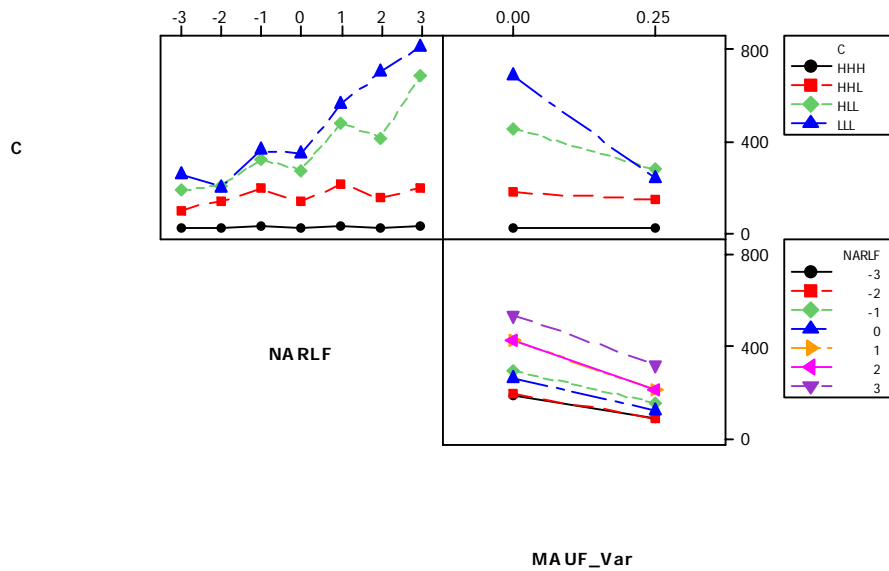


Figure 11: C , $NARLF$, and σ^2_{MAUF} interaction plots in terms of generation time

These results indicate that, while the generator *can* produce strongly random networks—which requires 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. Therefore, faster generation requires biasing the generator towards what we call “near-strongly random” networks.

7. Conclusion

This paper presented the first random generator of multi-project network test problems. The generator allows a user to control for important multi-project problem characteristics—including $NARLF$, AUF or $MAUF$, AUF variance or $MAUF$ variance, and project complexity. The generator incorporates several theoretical contributions, including a new approach to allocating activities to tiers in the network and improved multi-

project resource distribution and contention measures. The generator produces “near-strongly random” networks quickly, and can produce increasingly strongly random networks with greater computational expense. We therefore identify a tradeoff between the degree of randomness and computational time. The generated networks provide the basis for much further research into the effects of the factors on project duration, resource contention, etc. For example, in a separate paper (Browning and Yassine 2007), we analyze the performance of 20 heuristic priority rules on the 12,320 test problems created with this generator. We have no doubt that this basic generator can be improved, both in terms of efficiency and effectiveness, and that this will provide opportunities for fruitful research in the new realm of multi-project network problem generation.

8. Appendix (Proofs)

Proof of Lemma 1: This follows directly from the definition of a tier. \square

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. \square

Proof of Theorem 1:

Maximize $A' = (n_1 n_2) + \dots + (n_{T-1} n_T)$, subject to: $\sum_{j=1}^T n_j = N$, and n_j are non-zero positive integers, $1 \leq j \leq T$.

If $T = 2$, then the above problem becomes: $\text{Max}(n_1 n_2)$, subject to $n_1 + n_2 = N$, where both n_1 and n_2 are non-zero positive integers. 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 $n_2 = (N - 1) / 2$.

If $T = 3$, then the problem becomes: $\text{Max}(n_1 n_2 + n_2 n_3)$, subject to $n_1 + n_2 + n_3 = N$, where n_1, n_2 and n_3 are non-zero positive integers. Again, taking the Lagrangian of the objective function and solving the resultant equation yields the following optimal solution:

$$\begin{array}{l} n_1 = 1 \\ n_2 = n_3 = \frac{N-1}{2} \end{array} \quad \text{or} \quad \begin{array}{l} n_1 = n_2 = \frac{N-1}{2} \\ n_3 = 1 \end{array}$$

For $T \geq 4$, we will prove the theorem by induction. We start with the hypothesized optimal arrangement. Then, we will show that any deviation from this arrangement will result in a reduction of non-redundant arcs, or at least no increase. We will call the two consecutive tiers (where the majority of nodes are allocated) “major tiers” and the rest “minor tiers.” Then, removing a single node from any major tier and placing it in any minor tier results in three possible location

categories: (a) boundary minor tier, (b) before or after a major tier, or (c) anywhere else other than locations (a) or (b). Placing the node in locations (a) gains one non-redundant arc but at the expense of losing at least $(N - T + 2) / 2$ non-redundant arcs, which is greater than or equal to the gain (equal when $N = T$). Placing the node in locations (c) gains two non-redundant arcs but again at the expense of losing at least $(N - T + 2) / 2$ non-redundant arcs. Placing the node in locations (b) gains up to $[(N - T + 2) / 2] + 1$ non-redundant arcs but at the expense of losing the same amount of non-redundant arcs. Finally, the case of moving the node from one major tier to the other is already covered by the above case when $T = 2$, where it is proven that an equal assignment between the major tiers is optimal. Therefore, in all cases, deviating from the hypothesized optimal arrangement will reduce the maximum number of non-redundant arcs or leave it unchanged.

Now, we explain what happens if we move k nodes from a major tier and place them in any single minor tier. The loss (of non-redundant arcs) from removing k nodes from any one of the major tiers (or a combination of both major tiers) is at least $k\{[(N - T + 2) / 2] + 1\}$ (when all k is removed from one major tier). Placing the k nodes in location (a) gains k non-redundant arcs, which is smaller than the loss. Placing the k nodes in locations (c) gains $2k$ non-redundant arcs, which is also less than or equal to the loss (equal when $T = N$). Finally, placing the node in locations (b) gains up to $k\{[(N - T + 2) / 2] + 1\}$ non-redundant arcs, which is, at most, equal to the minimum loss.

The last thing to check is when we distribute the k nodes to more than one minor tier. The loss from doing so is still at least $k\{[(N - T + 2) / 2] + 1\}$. Assume that the k nodes are added to all minor tiers as follows: $k_1, k_2, k_3, \dots, k_{T-1}$, and k_T , such that $\sum_{j=1}^T k_j = k$, where k_j is the number of nodes added to tier j , $k_M = k_{M+1} = 0$, and M and $M+1$ are the locations of the two consecutive major tiers. Then, the total gain from this new arrangement is:

$$\begin{aligned} & (1 + k_2)k_1 + (2 + k_1 + k_3)k_2 + \dots + (2 + k_{j-1} + k_{j+1})k_j + \dots + (k_{M-1} + k_{M+2})\{(N - T + 2) / 2\} - k + \dots + (1 + k_{T-1})k_T \\ & = 2(k_1 + k_2 + \dots + k_{T-1} + k_T) + 2(k_1k_2 + k_2k_3 + \dots + k_{T-1}k_T) + (k_{M-1} + k_{M+2})\{(N - T + 2) / 2\} - k \end{aligned}$$

which is less than or equal to the loss. \square

Proof of Corollary 1: In equation (7), it is obvious that $A'_{max^*}(N, T \in [2, 3], \mathbf{n}^*) \geq A'_{max^*}(N, T \geq 4, \mathbf{n}^*)$. Furthermore, equations (7c) and (7d) are decreasing functions in T . Thus, A'_{max^*} is a decreasing function of T . \square

Proof of Theorem 2: We prove only the limited case where $A'_{max} = A'_{max^*}$ —i.e., where $\mathbf{n} = \mathbf{n}^*$. We must show that any $T > T_{max}$ fails to allow $A'_{max^*} \geq A'$. Substituting equations (7) and (11) into this inequality yields:

$$\left(\frac{N - T + 2}{2} \right)^2 + N - 2 \geq C + N + \frac{CN^2}{4} - CN - 1.$$

Arranging terms yields the following quadratic equation (in T):

$$T^2 - (2N + 4)T - (CN^2 - N^2 - 4CN + 4C - 4N) \geq 0.$$

Let $\delta \equiv CN^2 - N^2 - 4CN + 4C - 4N$, which is a constant when given N and C . Therefore,

$$T \leq \frac{(2N + 4) \pm \sqrt{(2N + 4)^2 + 4\delta}}{2}.$$

Since $T \leq N$, we choose the negative sign in front of the square root, and transform the function to:

$$T(C, N) \leq N + 2 - \sqrt{C(N - 2)^2 + 4}, \quad (12a)$$

where T must be rounded *down* to an integer and $N - T$ is even. When $N - T$ is odd, then the function is:

$$T(C, N) \leq N + 2 - \sqrt{C(N - 2)^2 + 5} \quad (12b)$$

Therefore, T_{max} is the largest value of T that satisfies equation (12a) or (12b). Again, this assumes that $n = n^*$. If not, then T_{max} will be even lower. \square

9. References

- Agrawal, M.K., S.E. Elmaghraby and W.S. Herroelen. 1996. DAGEN: A Generator of Testsets for Project Activity Nets. *Eur. J. of Op. Res.*, **90**: 376-382.
- Alvarez-Valdes, R. and J.M. Tamarit. 1989. "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis" in Slowinski, R. and J. Weglarz, Eds., *Advances in Project Scheduling*, Elsevier, Amsterdam, 113-134.
- Bein, W.W., J. Kamburowski and M.F.M. Stallmann. 1992. Optimal Reduction of Two-Terminal Directed Acyclic Graphs. *SIAM Journal on Computing*, **21**(6): 1112-1129.
- Browning, T.R. and A.A. Yassine. 2007. Resource-Constrained Multi-Project Scheduling: Priority Rule Performance Revisited, Working Paper.
- Brucker, P., *et al.* 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *Eur. J. of Op. Res.*, **112**(1): 3-41.
- Cooper, D.F. 1976. Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation. *Management Sci.*, **22**(11): 1186-1194.
- Dar-El, E.M. 1973. MALB-A Heuristic Technique for Balancing Large Single-Model Assembly Lines. *AIIE Trans.*, **5**(4): 343-356.
- Davis, E.W. 1975. Project Network Summary Measures and Constrained Resource Scheduling. *IIE Trans.*, **7**(2): 132-142.
- Davis, E.W. and J.H. Patterson. 1975. A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling. *Management Sci.*, **21**(8): 944-955.
- De Reyck, B. and W. Herroelen. 1996. On the Use of the Complexity Index as a Measure of Complexity in Activity Networks. *Eur. J. of Op. Res.*, **91**(2): 347-366.
- Demeulemeester, E., B. Dodin and W. Herroelen. 1993. A Random Activity Network Generator. *Operations Res.*, **41**(5): 972-980.
- Demeulemeester, E. and W. Herroelen. 1992. A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem. *Management Sci.*, **38**(12): 1803-1818.
- Demeulemeester, E., M. Vanhoucke and W. Herroelen. 2003. A Random Network Generator for Activity-on-the-Node Networks. *Journal of Scheduling*, **6**(1): 13-34.
- Elmaghraby, S.E. and W.S. Herroelen. 1980. On the Measurement of Complexity in Activity Networks. *Eur. J. of Op. Res.*, **5**(4): 223-234.
- Gutiérrez, M., A. Durán, D. Alegre and F. Sastrón. 2004. Hiergen: A Computer Tool for the Generation of Activity-on-the-Node Hierarchical Project Networks. *Proc. of the Computational Science and Its Applications - ICCSA, Part III*, Assisi, Italy, .
- Haberle, K., R. Burke and R. Graves. 2000. A Note on Measuring Parallelism in Concurrent Engineering. *Int. J. of Production Res.*, **38**(8): 1947-1952.

- Hartmann, S. and R. Kolisch. 2000. Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem. *Eur. J. of Op. Res.*, **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 Sci.*, **21**(2): 172-177.
- Kaimann, R.A. 1975. Coefficient of Network Complexity: Erratum. *Management Sci.*, **21**(10): 1211-1212.
- Kao, E.P.C. and M. Queranne. 1982. On Dynamic Programming Methods for Assembly Line Balancing. *Operations Res.*, **30**(22): 375-390.
- Kolisch, R., A. Sprecher and A. Drexel. 1992. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. Kiel, Germany.
- Kolisch, R., A. Sprecher and A. Drexel. 1995. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Sci.*, **41**(10): 1693-1703.
- Kurtulus, I. 1978. *An Analysis of Scheduling Rules for Multi-Project Scheduling*, Ph.D. Thesis (Business), University of North Carolina at Chapel Hill, Chapel Hill.
- Kurtulus, I. and E.W. Davis. 1982. Multi-Project Scheduling: Categorization of Heuristic Rules Performance. *Management Sci.*, **28**(2): 161-172.
- Kurtulus, I.S. and S.C. Narula. 1985. Multi-Project Scheduling: Analysis of Project Performance. *IIE Trans.*, **17**(1): 58-65.
- Lenstra, J. and K. Rinnooy. 1978. Complexity of Scheduling under Precedence Constraints. *Operations Res.*, **26**(1): 22-35.
- Lova, A. and P. Tormos. 2001. Analysis of Scheduling Schemes and Heuristic Rules Performance in Resource-Constrained Multi-project Scheduling. *Annals of Operations Research*, **102**: 263-286.
- Mastor, A.A. 1970. An Experimental and Comparative Evaluation of Production Line Balancing Techniques. *Management Sci.*, **16**(22): 728-746.
- Pascoe, T.L. 1966. Allocation of Resources - CPM. *Revue Française de Recherche Opérationnelle*, **38**: 31-38.
- Patterson, J.H. 1976. Project Scheduling: The Effects of Problem Structure on Heuristic Performance. *Naval Research Logistics*, **23**(1): 95-123.
- Patterson, J.H. 1984. A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. *Management Sci.*, **30**(7): 854-867.
- Schwindt, C. 1995. A New Problem Generator for Different Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags, Institut für Wirtschaftstheorie und O.R., 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, 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 WIOR-543.
- Tavares, L.V. 1998. *Advanced Models for Project Management*. Kluwer Academic Publishers, Boston.
- Tavares, L.V., J.A. Ferreira and J.S. Coelho. 1999. The Risk of Delay of a Project in Terms of the Morphology of Its Network. *Eur. J. of Op. Res.*, **119**: 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 and L.V. Tavares. 2004. On the Morphological Structure of a Network, Vlerick Leuven Gent Management School, Working Paper no. 2004/9.

