RanGen: A Random Network Generator for Activity-on-the-Node Networks

Erik Demeulemeester, Mario Vanhoucke, Willy Herroelen

Institutions: Katholieke Universiteit Leuven

Published on: 01 Jan 2003 - Journal of Scheduling (Kluwer Academic Publishers)

Topics: Network topology, Node (networking), Random graph and Generator (computer programming)

Journal of Scheduling 6: 17–38, 2003.

RanGen: A RANDOM NETWORK GENERATOR FOR ACTIVITY-ON-THE-NODE NETWORKS

ERIK DEMEULEMEESTER[1,*], MARIO VANHOUCKE[2,3], AND WILLY HERROELEN[1]

[1] *Operations Management Group, Department of Applied Economics, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium*

[2] *Operations & Technology Management Center, Vlerick Leuven Gent Management School, Bellevue 6, B-9050 Gent, Belgium*

[3] *Faculty of Economics and Business Administration, Hoveniersberg 24, B-9000 Gent, Belgium*

ABSTRACT

In this paper, we describe *RanGen*, a random network generator for generating activity-on-the-node networks and accompanying data for different classes of project scheduling problems. The objective is to construct random networks which satisfy preset values of the parameters used to control the hardness of a problem instance. Both parameters which are related to the network topology and resource-related parameters are implemented. The network generator meets the shortcomings of former network generators since it employs a wide range of different parameters which have been shown to serve as possible predictors of the hardness of different project scheduling problems. Some of them have been implemented in former network generators while others have not.

KEY WORDS: project scheduling; network generator

1. INTRODUCTION

Since the beginning of the research in project scheduling, activity networks (AN) have become an important tool to visualize different kinds of projects. Rapid progress regarding exact and suboptimal procedures has created the need to differentiate between easy and hard project scheduling instances. The hardness of a problem instance is typically measured by the amount of CPU-time that a solution procedure needs to find an optimal solution for the problem at hand. It is, therefore, of great importance to possess a set of problem characteristics that discriminates between easy and hard instances and that acts as a predictor of the computational effort of the different solution procedures. If good predictions of the required CPU-time for each of these procedures were available, it would be possible to a priori select the fastest solution procedure, based on the simple calculation of these problem characteristics for the problem at hand. Moreover, some problem characteristics explain a larger part of the required CPU-time for a certain problem instance and in that sense we will say that one complexity measure outperforms another. Good random network generators are, therefore, indispensable in the construction of problem sets that span the complete range of complexity of the important problem characteristics.

* Correspondence to: Erik Demeulemeester, Operations Management Group, Department of Applied Economics, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium. E-mail: erik.demeulemeester@econ.kuleuven.ac.be

Quite a number of measures have been proposed in the literature (Davis, 1975; Patterson, 1976). Recently, Herroelen and De Reyck (1999) have shown the occurrence of phase transitions in project scheduling problems and call the attention to the importance of measures with sufficient discriminatory power to allow for the observation of these dramatic changes in problem difficulty. We can distinguish between measures capturing information about the size and the topological structure of the network and measures which are related to the different resources allocated to the project. Patterson (1984) was the first to satisfy the need for a standard set of problems with varying degrees of difficulty. His testset contains problems taken from different sources in the literature. Unfortunately, the testset lacks a controlled design of several parameters and has been recently labeled as rather easy. Motivated by this fact, the recognition for the need for networks with controllable measures of complexity arose, which has led to the construction of a few network generators and the generation of new standard datasets.

To the best of our knowledge, papers dealing with network generators for project scheduling problems are rather sparse. Demeulemeester, Dodin, and Herroelen (1993) have developed a random generator for activity-on-the-arc (AoA) networks. These networks are called *strongly random* since they can be generated at random from the space of all feasible networks with a specified number of nodes and arcs. Besides the number of nodes and the number of arcs, no other characteristics can be specified for describing the network topology. The number of renewable resource types as well as the resource availabilities and requirements either are constant or drawn from precoded distributions. Kolisch, Sprecher, and Drexl (1995) describe *ProGen*, a network generator for activity-on-the-node (AoN) networks, which takes into account network topology as well as resource-related characteristics. Schwindt (1995) extended *ProGen* to *ProGen/Max* which can handle three different types of resource-constrained project scheduling problems with minimal and maximal time lags. Agrawal, Elmaghraby, and Herroelen (1996) recognize the importance of the complexity index (CI) as a measure of network complexity and have developed an AoA network generator *DAGEN* for which this complexity measure can be set in advance. Neither of the networks generated by the last three generators can be called *strongly random* because they do not guarantee that the topology is a random selection from the space of all possible networks which satisfy the specified input parameters.

The aim of this paper is to present an AoN network generator which generates problem instances that span the full range of problem complexity (Elmaghraby and Herroelen, 1980). The generator uses a reliable set of complexity measures which have been shown in former studies to stand in clear and strong relation to the hardness of different project scheduling problems (De Reyck, 1995; De Reyck and Herroelen, 1996; Elmaghraby and Herroelen, 1980; Herroelen and De Reyck, 1999). The network generator also guarantees networks with a prespecified order strength (OS). Moreover, we seem to satisfy the need for a random AoN network generator with prespecified values of the network CI, as mentioned by De Reyck (1995), De Reyck and Herroelen (1996), Demeulemeester et al. (1998), and Herroelen and De Reyck (1999).

The organization of the paper is as follows. In Section 2, we briefly review the most important conclusions from the literature on the use of network-based measures of complexity. Section 3 introduces a procedure for the generation and unique representation of all networks, which satisfy a preset topological structure. Section 4 explains the basics of the network generator *RanGen*. In Section 5, we try to gain additional insight in the relation between two network topology measures: CI and OS. In Section 6, we briefly summarize the advantages of our new network generator. Section 7 reviews a number of existing resource measures and their implementation in our new generator. Section 8 provides a summary and overall conclusions. For an overview of the terminology used in the project scheduling literature, we refer to the excellent papers by Icmeli, Erengüç, and Zappe (1993), Elmaghraby (1995), Özdamar and Ulusoy (1995), Herroelen, De Reyck, and Demeulemeester (1998) and Brucker et al. (1999).

2. NETWORK TOPOLOGY MEASURES

Contributions to measures of network complexity have received attention from researchers since the mid-1960s and emanate from studies in the area of activity networks, assembly line balancing, and machine scheduling problems. Ideally, such measures should serve as predictors of the hardness of a problem as measured by the CPU-time. For a complete evaluation of contributions regarding these measures, we refer to Elmaghraby and Herroelen (1980).

Probably the best known measure for the topological structure of AoA networks is the *coefficient of network complexity* (CNC), defined by Pascoe (1966) as the number of arcs over the number of nodes, and redefined by Davies (1973) and Kaimann (1974, 1975). The measure has been adapted for AoN problems by Davis (1975) as the number of direct arcs over the number of activities (nodes) and has been used in the network generator *ProGen* (Kolisch, Sprecher, and Drexl, 1995). Since the measure relies totally on the count of the activities and the direct arcs of the network and as it is easy to construct networks with an equal CNC-value but a different degree of difficulty, Elmaghraby and Herroelen (1980) questioned the usefulness of the suggested measure. De Reyck and Herroelen (1996) and Herroelen and De Reyck (1999) conclude that the correlation of the CNC with the so-called CI is responsible for a number of misinterpretations with respect to the explanatory power of the CNC. Indeed, Kolisch, Sprecher, and Drexl (1995) and Alvarez-Valdes and Tamarit (1989) had revealed that resource-constrained project scheduling instances become easier with increasing values of the CNC, without considering the underlying effect of the CI. In conclusion, the CNC, by itself, fails to discriminate between easy and hard instances and can, therefore, not serve as a good measure for describing the impact of the network topology on the hardness of a project scheduling problem.

Another well-known measure of the topological structure of an AoN network is the OS (Mastor, 1970), defined as 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 nondummy activities in the network). It is sometimes referred to as the density (Kao and Queranne, 1982) or the restrictiveness (RT) (Thesen, 1977) and equals 1 minus the flexibility ratio (Dar-El, 1973). Herroelen and De Reyck (1999) conclude that the OS, density, RT, and the flexibility ratio constitute one and the same complexity measure. Schwindt (1995) uses RT in the problem generator *ProGen/Max* and argues that this measure plays an important role in predicting the difficulty of different resource-constrained project scheduling problems. De Reyck (1995) verified and confirmed the conjecture that the OS outperforms CI as a measure of network complexity for the resource-constrained project scheduling problem.

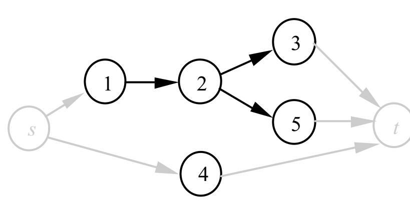
The CI was originally defined by Bein, Kamburowski, and Stallmann (1992) for two-terminal acyclic AoA networks as the *reduction complexity*, that is, the minimum number of node reductions which—along with series and parallel reductions—allow to reduce a two-terminal acyclic network to a single edge. As a consequence, the CI measures the closeness of a network to a series–parallel directed graph. Their approach for computing the reduction complexity consists of two steps. First, they construct the so-called complexity graph by means of a dominator and a reverse-dominator tree. Second, they determine the minimal node cover through the use of the maximum flow procedure by Ford and Fulkerson (1962). De Reyck and Herroelen (1996) adopted the reduction complexity as the definition of the CI of an activity network and have proven the CI to outperform other popular measures of performance, such as the CNC. On the other hand, they conclude that the OS outperforms the CI. These studies motivated the construction of an AoN problem generator for networks where both the OS and the CI can be specified in advance. To the best of our knowledge, *DAGEN* (Agrawal, Elmaghraby, and Herroelen, 1996) is the only generator which generates networks with prespecified CI. They construct problems in AoA format and do not take the OS as a measure of network topology into account. Unfortunately, the generated networks are not strongly random.

In the next section, we introduce an algorithm to represent networks in a unique fashion by means of an upper triangular matrix. In the succeeding section, this enumerative procedure will be used in our construction method for generating networks with a specified OS and CI.

3. A UNIQUE REPRESENTATION OF NETWORKS

A project network in AoN format $G = (N, A)$ where the set of nodes, N, represents activities and the set of arcs, A, represents precedence constraints can be represented by an upper triangular precedence relations matrix without the diagonal as given in Figure 1. This binary precedence matrix (PM) denotes whether or not a precedence relation exists between two nodes. If node j is a successor (either direct or transitive) of node i, then $\mathrm{PM}_{ij} = 1$; otherwise, it equals zero. Notice that in Figure 1 activity 1 has three successors: arc $(1, 2)$ represents the direct precedence relation, while $(1, 3)$ and $(1, 5)$ denote transitive precedence relations. Activity 2 has two immediate successors: activities 3 and 5. In line with the literature, we add a dummy start activity s and a dummy end activity t to visualize the network.

The representation of a network as an upper triangular matrix PM has serious advantages. First, it is never possible to have activities with a smaller number than one of its predecessors and second, it leads to a very easy and precise calculation of the OS. The number of elements in PM, either zero or one, denotes the maximal number of arcs in the considered network, while the number of ones denotes the number of precedence relations. From the definition of OS, we have to divide this number of ones by the number of precedence relations to obtain the value for the OS. Remark that the OS in our example equals $5/10 = 0.5$.



	2	3	4	5
1	1	1	-	1
2		1	-	1
3			-	-
4				-

Figure 1. An example network with its precedence matrix (PM) representation

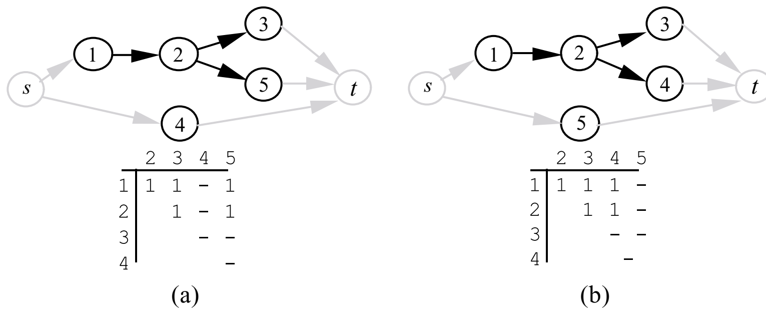


Figure 2. Two identical networks with a different precedence matrix (PM) representation

As shown in Figure 2, it is easy to construct networks with the same topological structure but a different PM. Although the two networks differ in the successors of node 2, the topology of both networks is the same. Since we are only interested in networks with a different topological structure, we have to detect such similarities one way or another. In the next section, we discuss a generally fast recursive procedure which transforms different matrices PM of networks with the same topological structure into one and the same standardized precedence matrix (SPM).

3.1. The recursive algorithm

The recursive algorithm implicitly enumerates all possible representations of a network which satisfy a given topological structure, the unique representation of which will be specified by its SPM. To that purpose we assign a weight w_i to each node i of the network AN with a PM based on its number of successors and predecessors (both direct and transitive) and create a recursively enumerated matrix (*rem*) by a recursive enumeration method in which the nodes are ranked according to specific criteria. This matrix *rem* will then be used to create the SPM of the network AN. During the recursive enumeration procedure, we create each time a set of eligible nodes (EN) for which all predecessor nodes are already enumerated. Among the set of EN, we choose the node with the highest weight w_i to include in the matrix *rem*, we update EN and continue our recursive enumeration method until we have enumerated (and ranked) all nodes of the network. The recursive method is a complete enumeration in the sense that, when tie breaks occur (nodes with an equal weight to choose from), we split the recursive enumeration method in a number of cases equal to the number of nodes with equal weight and continue our recursive enumeration method for each case, assuring that each possible rank order will be found.

Since each rank order given by its matrix *rem* corresponds to a network AN with the same topological structure, we can compute a value *ub* for each matrix *rem* (and its corresponding network), using the idea of binary digits. Each arc in the network has a value which is twice the value of the preceding arc in the PM. In doing so, we assure that each network can be identified in a unique way, similar to the binary representation of any integer number by a multiple of two. Among the possible rank orders found, we choose the one with the smallest bound *ub* which leads, by means of rearranging the nodes, to the SPM.

Let UN denote the set of unenumerated nodes, EN the set of eligible nodes, PM the precedence matrix of the AN in the AoN format, p the level of the recursive enumeration algorithm and *rem* the matrix in which we will rank the different nodes. $\bar{S}_i$ and $\bar{P}_i$ denote, respectively, the sets of successors and predecessors (including the transitive ones) of node $i \in$ PM, while S_i and P_i denote the sets of their immediate successors and predecessors. Remark that the variable E is a local auxiliary variable of the recursive procedure while SN denotes a set which stores the set of eligible nodes EN. The procedure to represent each network in a unique fashion, including the recursive enumeration algorithm, can be written as follows:

Procedure Unique_representation(AN);
  Initialize $UN = N$, $EN = \{i \mid P_i = \emptyset\}$, $ub = \infty$ and $\forall i \in N$, $w_i = a_1|\bar{S}_i| + a_2|S_i| + a_3|\bar{P}_i| + a_4|P_i|$;
    (with $a_1 \gg a_2 \gg a_3 \gg a_4$)
  Do RECURSIVE_ENUMERATION$(1, EN) \rightarrow rem$;
  Rearrange the Precedence Matrix PM according to *rem*.

RECURSIVE_ENUMERATION(p, EN)
If $p = n$ **then**
  $rem_i = p \mid i \in EN$;
  If $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2^{n(rem_i - 1) - \frac{rem_i(1+rem_i)}{2} + rem_j} PM_{ij} < ub$ **then** save $rem_i \mid i \in N$ in the matrix *opt_rem* and update ub;
else
  $E = \{i \mid w_i = \max(w_j \mid j \in EN)\}$;
  $\forall i \in E$
    $SN = EN$, $EN = EN \backslash \{i\}$, $UN = UN \backslash \{i\}$ and $rem_i = p$;
    $EN = EN \cup \{j \mid j \in S_i$ and $P_j \cap UN = \emptyset\}$;
    RECURSIVE_ENUMERATION$(p + 1, EN)$;
    $UN = UN \cup \{i\}$ and $EN = SN$;
Return;

Note that the expression

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2^{n(rem_i - 1) - \frac{rem_i(1+rem_i)}{2} + rem_j} \mathrm{PM}_{ij}$$

simply calculates a representation value using the assigned values of each arc in the PM. More precisely, the renumbered arc $(1, 2)$ has a value of 1 (2^0), the renumbered arc $(1, 3)$ has a value of 2 (2^1), the renumbered arc $(1, 4)$ has a value of 4 (2^2), the renumbered arc $(1, 5)$ has a value of 8 (2^3) and so on. If $n = 5$, then the renumbered arc $(2, 3)$ has a value of 16 (2^4). Each arc thus has a value which is twice the value of the preceding arc.

In order to clarify the procedure as described above, we apply the unique representation algorithm to the PM of Figure 1. Note that in the example we use $a_1 = 2^{24}$, $a_2 = 2^{16}$, $a_3 = 2^8$ and $a_4 = 1$.

Initialize $UN = \{1, 2, 3, 4, 5\}$, $\mathrm{EN} = \{1, 4\}$, $ub = \infty$ and $w_1 = 50{,}397{,}184$, $w_2 = 33{,}685{,}761$, $w_3 = 513$, $w_4 = 0$, and $w_5 = 513$. The calculations of the weights w_i are as follows:

i	$\lvert \bar{S}_i \rvert$	$\lvert S_i \rvert$	$\lvert \bar{P}_i \rvert$	$\lvert P_i \rvert$	w_i
1	3	1	0	0	50,397,184
2	2	2	1	1	33,685,761
3	0	0	2	1	513
4	0	0	0	0	0
5	0	0	2	1	513

Recursive_Enumeration(1, {1,4});
$E = \{\mathbf{1}\}$;
$EN = \{4\}$, $UN = \{2,3,4,5\}$ and $rem_1 = 1$;
$EN = \{2,4\}$;
Recursive_Enumeration(2, {2,4});
$E = \{\mathbf{2}\}$;
$EN = \{4\}$, $UN = \{3,4,5\}$ and $rem_2 = 2$;
$EN = \{3,4,5\}$;
Recursive_Enumeration(3, {3,4,5});
$E = \{\mathbf{3},5\}$;
$EN = \{4,5\}$, $UN=\{4,5\}$ and $rem_3 = 3$;
$EN = \{4,5\}$;
Recursive_Enumeration(4, {4,5});
$E = \{\mathbf{5}\}$;
$EN = \{4\}$, $UN = \{4\}$ and $rem_5 = 4$;
$EN = \{4\}$;
Recursive_Enumeration(5, {4});
$rem_4 = 5$;
Since $p = 5$ and $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2^{n(rem_i-1)-\frac{rem_i(1+rem_i)}{2}+rem_j}\ PM_{ij} = 55 < \infty$,
$ub = 55$ and save $opt\_rem = [1,2,3,5,4]$;
$UN = \{4,5\}$;
$UN = \{3,4,5\}$;
$E = \{3,\mathbf{5}\}$;
$EN = \{3,4\}$, $UN = \{3,4\}$ and $rem_5 = 3$;
$EN = \{3,4\}$;
Recursive_Enumeration(4, {3,4});
$E = \{\mathbf{3}\}$;
$EN = \{4\}$, $UN = \{4\}$ and $rem_3 = 4$;
$EN = \{4\}$;
Recursive_Enumeration(5, {4});
$rem_4 = 5$;
Since $p = 5$ and $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2^{n(rem_i-1)-\frac{rem_i(1+rem_i)}{2}+rem_j}\ PM_{ij} = 55 = ub$,
continue;
$UN = \{3,4\}$;

$UN = \{3,4,5\}$;
$UN = \{2,3,4,5\}$;
$UN = \{1,2,3,4,5\}$;
Return;

Rearrange the PM. Since $opt\_rem = [1, 2, 3, 5, 4]$ we have to switch rows 4 and 5 as well as columns 4 and 5 in order to obtain the SPM as shown in Figure 3.

In the example above, two possible rank orders, as given by their recursively enumerated matrices *rem*, have been found. Notice that the upper bound for both matrices is equal to 55. However, this is not always the case, as we will show by means of the network given in Figure 4. The network consists of 5 nodes and 2 dummy nodes.

During the recursive enumeration procedure, a number of representations will be found resulting in a value for its corresponding matrix *rem*. As shown in Figure 5, the matrices (a) $rem = [1, 2, 4, 3, 5]$ and (b) $rem = [2, 1, 5, 3, 4]$ have an upper bound $ub = 86$ while the upper bound of the matrices (c) $rem = [1, 2, 5, 3, 4]$ and (d) $rem = [2, 1, 4, 3, 5]$ equals $ub = 58$. Notice that these triangular matrices correspond to a network with the topological structure shown in Figure 4. The corresponding PM, however, will never be enumerated as in the recursion node 4 will always be considered before nodes 3 and 5. The enumeration procedure will store the representation with $ub = 58$.

3.2. *A dominance rule for the recursive algorithm*

In order to improve the efficiency of the enumeration algorithm, we use a modified definition of the auxiliary variable E used by the recursive enumeration method. Instead of simply

	2	3	4	5
1	1	1	-	1
2		1	-	1
3			-	-
4				-

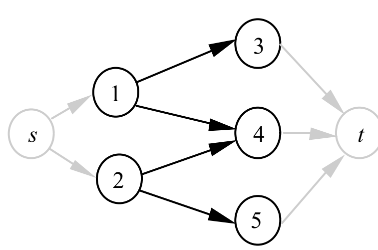
(a)

→ Unique representation (AN)→

	2	3	4	5
1	1	1	1	-
2		1	1	-
3			-	-
4				-

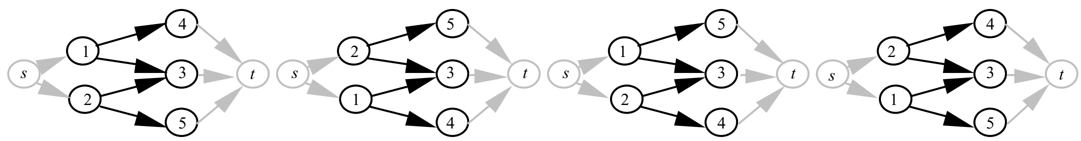
(b)

Figure 3. (a) Original precedence matrix (PM) and (b) its standardized version (SPM) after applying the unique representation algorithm



	2	3	4	5
1	-	1	1	-
2		-	1	1
3			-	-
4				-

Figure 4. An example network with its corresponding precedence matrix (PM)



	2	3	4	5
1	-	1	1	-
2		1	-	1
3			-	-
4				-

(a) $uv = 86$

	2	3	4	5
1	-	1	1	-
2		1	-	1
3			-	-
4				-

(b) $uv = 86$

	2	3	4	5
1	-	1	-	1
2		1	1	-
3			-	-
4				-

(c) $uv = 58$

	2	3	4	5
1	-	1	-	1
2		1	1	-
3			-	-
4				-

(d) $uv = 58$

Figure 5. Four networks with the same topological structure with their corresponding precedence matrices (PMs)

including the nodes $j \in$ EN with the highest weight w_j into the set E, we expand the condition as follows:

$$E = \{i \mid w_i = \max(w_j \mid j \in \text{EN}),\ \neg\exists j \in E\backslash\{i\} \mid P_i = P_j \text{ and } S_i = S_j\}$$

The first condition, $w_i = \max(w_j \mid j \in \text{EN})$, searches for the nodes with the maximum weight w_j as described above. The second condition, $\neg\exists j \in E\backslash\{i\} \mid P_i = P_j$ and $S_i = S_j$, assures that no node with the same successors and predecessors as an already included node will be selected to enter the set E.

Consider again the example of Figure 1. The new definition of the auxiliary variable E allows for a considerable reduction of the search effort. At level three of the enumeration (denoted by *Recursive_Enumeration*(3, {3,4,5})) we created the set $E = \{3, 5\}$ since both nodes 3 and 5 have a maximum weight of 513 among the nodes of EN. With the new definition things are different: both nodes 3 and 5 are still eligible to enter the set E. But once a node has been selected (e.g., node 3), the other node (node 5) does no longer satisfy the second condition, since it has the same set of successors and predecessors as an already selected node (node 3) of the set E. This results in the fact that one activity (3 or 5) will be randomly chosen and that we will find only one *ub* with the same value as found earlier.

In the next section, we explain the logic of the generation method used in *RanGen*. During the generation of the networks, the recursive enumeration method of Section 3.1 will be used in order to prevent the generation of two networks with the same topological structure.

4. GENERATION METHOD

4.1. The exhaustive generation of strongly random activity networks

It might be tempting to generate strongly random networks (Demeulemeester, Dodin, and Herroelen, 1993) by enumerating all possible network structures which satisfy preset values of the complexity parameters. The results of such an enumeration effort are shown in Figure 6. Unfortunately, as shown in the figure, both CPU-time and memory requirements render such a method inapplicable for networks with more than 10 activities.

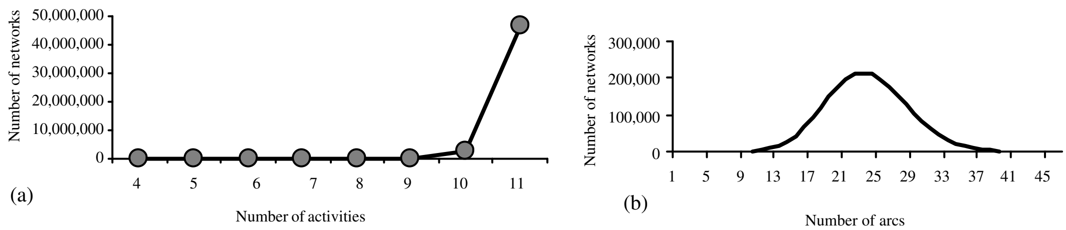


Figure 6. The impact of (a) the number of activities and (b) the number of precedence relations on the number of possible network structures

Figure 6(a) displays the number of different AoN networks for a number of nodes ranging from 4 to 11. Figure 6(b) shows the number of different 10-ANs as a function of the number of precedence relations. While there are somewhat less than 50,000,000 different networks containing 11 nodes, memory restrictions prohibited us from finding the exact number of networks with 12 or more nodes. Remark also that there are many more networks with an OS = 0.5 (i.e., either 22 or 23 arcs) than, for example, an OS = 0.25. Since we need one bit for the representation of a precedence constraint and the maximum number of precedence relations is $n(n-1)/2$, we need 6 bytes of memory to store a network of 10 activities, 24 bytes to store a 20-AN, network and 55 bytes to store a 30-AN. Suppose we have 64 MB RAM available and we want to enumerate all possible networks containing 30 activities, then we will run out of memory after 1,220,161 networks, which is only a very small fraction of the full space of possible networks. Due to these reasons, we were obliged to search for another generation method, which will be discussed in the next section.

4.2. *The* RanGen *procedure*

RanGen imposes both a CPU time limit and a limit on the number of networks which may be generated. As many networks as possible are generated within these limits and a number of networks satisfying the preset parameter values are then randomly generated from the obtained set. Using GN to denote the set of already generated networks and AN to denote a generated activity network in AoN format, the overall procedure can be written in pseudocode as follows:

---

procedure generate(OS);
$GN = \emptyset$;
Repeat
    $AN =$ remove_arcs(OS);
    Unique_representation(AN);
    If ($AN \notin GN$) **then**
        Save the network: $GN = GN \cup \{AN\}$;
        Transform AN into an AoA format;
        Compute CI;
Until bound or time limit;
Select a number of networks with prespecified OS and CI;
Return;

4.2.1. *The procedure remove_arcs*(OS)*: generating a network with prespecified OS*

The generator starts for each generation with a completely connected network with $OS = 1$ and removes nontransitive arcs until it obtains a network with specified OS. The set of nontransitive arcs is updated each time an arc is removed.

Figure 7 shows the different steps needed to generate the network of Figure 1.

Figure 7(a) displays the completely connected network for which $OS = 1$. The four (in general $(n - 1)$) nontransitive *direct* arcs which are *eligible* for selection are shown in bold. We randomly select arc $(3, 4)$ and remove it from the PM. In updating the set of direct arcs, both arc $(2, 4)$ and arc $(3, 5)$ are made eligible for selection (the corresponding 1s are shown in bold in Figure 7(b)). Again, we randomly choose an eligible arc $(3, 5)$ and update the set of eligible arcs as given in Figure 7(c). Since the maximal number of arcs equals 10 and we search for a network with an $OS = 0.5$, we repeat this random selection method five times until we obtain the network in Figure 7(f) with the given OS.

Observe that the generation of a network with prespecified OS value boils down to the deletion of arcs. This simple logic yields *RanGen* a competitive advantage over the existing generators *ProGen* and *ProGen/Max*, which do not allow the generation of networks with small OS value.

4.2.2. *Checking for uniqueness: procedure Unique_representation*(AN)

Upon the generation of a network with preset OS value, it must be checked for uniqueness by the recursive enumeration procedure described in Section 3. If the network is entirely new, it is added to the set of generated networks GN (initially $GN = \emptyset$).

4.2.3. *Compute the complexity index CI*

Each time a new activity network AN has been generated, its CI value must be calculated using the algorithm developed by Bein, Kamburowski, and Stallmann (1992). Since this algorithm works on networks in AoA format, we first have to transform the generated AoN network into an equivalent AoA network with minimal CI using the algorithm of Kamburowski, Michael, and Stallmann (1992).

4.2.4. *The random selection of networks*

Upon completion, the program yields a set of networks which satisfy preset OS and CI values from which a desired number of networks may be randomly selected.

4.2.5. *Truncating procedure Unique_representation*(AN)

For the generation of networks with a small OS (i.e., for $OS \leq 0.2$, as obtained from our experiments), the recursive enumeration procedure Unique_representation(AN) needs a large

(a)

	2	3	4	5
1	**1**	1	1	1
2		**1**	1	1
3			***1***	1
4				**1**

(b)

	2	3	4	5
1	**1**	1	1	1
2		**1**	**1**	1
3			-	***1***
4				**1**

(c)

	2	3	4	5
1	**1**	1	1	1
2		**1**	***1***	1
3			-	-
4				**1**

(d)

	2	3	4	5
1	**1**	1	***1***	1
2		**1**	-	**1**
3			-	-
4				**1**

(e)

	2	3	4	5
1	**1**	1	-	1
2		**1**	-	**1**
3			-	-
4				***1***

(f)

	2	3	4	5
1	**1**	1	-	1
2		**1**	-	**1**
3			-	-
4				-

Figure 7. Remove_arcs(0.5): generating the network of Figure 1

amount of time to compute the lowest upper bound *ub*. Therefore, we provide the option to truncate the search whenever a certain number of matrices *rem* are found. Although there is still a good chance that the optimal rank order, denoted by its matrix *opt_rem*, will be found after a limited number of searching steps, this method does no longer assure to find a unique representation of each network.

In the following section, we elaborate on the relation between the OS and the CI.

5. CI AND OS: AN OVERVIEW

The network generator *RanGen* is developed for generating different networks, which allows the user to perform several validation experiments and to determine the effectiveness and efficiency of different project scheduling procedures in relation to several complexity measures. Since we claim to generate networks with prespecified values for both OS and CI, realistic input values for both parameters must be entered in order to allow the generator to obtain sufficient problem instances during generation. These values must be selected with sufficient care, as shown in the following example. Suppose a user wants to validate an algorithm by means of a full factorial experiment on a randomly generated dataset. Assume *RanGen* is provided with two settings for the number of activities, three settings for the OS and four settings for the CI as follows:

Number of activities	10 or 20
Order strength (OS)	0.25, 0.50 or 0.75
Complexity index (CI)	3, 6, 9 or 12

Although it is perfectly possible to generate networks with 10 activities, $OS = 0.25$ and $CI = 3$, *RanGen* will not be able to generate networks with, for example, $OS = 0.25$ and $CI = 12$. Apparently, the range for the CI was chosen too large to allow for the generation of a dataset with the preset parameter values.

In order to provide the user of *RanGen* with a guideline for presetting the input parameters in a full factorial experiment, we have set up the following experiment which provides additional insight in the relation between the OS and the CI. We have generated a large number of networks with different input parameters for both the number of activities and the OS as follows:

Number of activities	5, 10, 15, 20,... or 120
Order strength (OS)	0.05, 0.10, 0.15,... or 1.00

For each network, we have calculated the CI. The input parameters are set up as follows: the number of activities is varied from 5 to 120 in steps of 5, resulting in 24 settings while the OS has 20 settings, varying between 0.05 and 1.00 in steps of 0.05. We have generated networks with a 1 hr time limit for each class. Consequently, the experiment took 20 days of CPU-time. Table I summarizes the results found from the experiment. This table displays the CI values that were found for each setting of the OS and the number of activities. We only list the CI values for OS activity combinations for which 10 or more instances were found. Thus, if one wants to create a problem set with 10 activities and an OS of 0.35, the complexity index may be varied between 0 and 5 as the experiment demonstrated that, after an hour of computation, at least 10 different networks were obtained for each value of the complexity index between 0 and 5. Note that, because of the simple logic, each network can be generated in a very small amount of time. This is exactly the purpose of our generator: generate a large amount of networks (within a reasonable time limit) in order to make our network generator "as strongly random as possible." It has been previously mentioned that the ultimate target of this table is to provide a guideline for presetting the input parameters in a full factorial experiment. For an example, we refer to Vanhoucke (2001) in which the author has used Table I as a guideline for establishing a dataset for resource-constrained project scheduling problems with 15 activities, an OS of 0.50, and different values for the CI varying between 2 and 9.

Thanks to Table I, the author knew which values for the CI could be generated taking the other parameters (number of activities and OS) into account. Of course, the reader can find many other examples in Vanhoucke (2001) in which datasets are generated based on the observations of Table I.

6. ADVANTAGES OF RanGen

In this section, we briefly summarize the reasons why we believe that *RanGen* outperforms other network generators. Our generating procedure boils down to removing arcs and is, therefore, fairly straightforward. Moreover, the recursive procedure to represent a network in a unique fashion, which constitutes an essential part of our generating process, is generally very efficient. Consequently, we are able to generate networks in a very efficient way. The reasons why *RanGen* outperforms other network generators can be summarized as follows:

- *RanGen* uses a reliable set of complexity measures which have been shown in former studies to stand in clear and strong relation to the hardness of different project scheduling problems. Previous research by Elmaghraby and Herroelen (1980), De Reyck (1995), De Reyck and Herroelen (1996), and Herroelen and De Reyck (1999) illustrates the need for such complexity measures (such as the OS and the CI).
- The simplicity of our process of generating networks with a prespecified value of the OS, which boils down to the deletion of arcs, yields *RanGen* a competitive advantage over the existing generators *ProGen* and *ProGen/Max*. Indeed, these two network generators do not allow the generation of networks with small OS value, as indicated in Section 4.2.1.
- In many papers (such as De Reyck (1995), De Reyck and Herroelen (1996), Demeulemeester et al. (1998), and Herroelen and De Reyck (1999)), it has been shown that there is a need for a random AoN network generator with prespecified values of the network CI. We are the first to present a network generator which uses both the OS and the CI as input values. Notice also that our network generator can consider every other parameter (instead of the CI) since we generate a large amount of networks with a given OS value and afterwards, we calculate the CI value.
- *RanGen* does not incorporate superfluous parameters, such as the maximal number of start activities in the network and the maximal number of successor and predecessor nodes. In some other network generators, these network parameters are not used to predict the difficulty of the underlying problem (expressed in the amount of CPU-time to find an optimal solution) but they are used to make the generating process more easy. As a consequence, these networks are no longer strongly random because the use of these superfluous parameters dramatically restricts the domain from which they are generated.

Table 1. Complexity index (CI) and order strength (OS): An overview

act[b]	OS[a]																			
	0, 05	**0, 10**	**0, 15**	**0, 20**	**0, 25**	**0, 30**	**0, 35**	**0, 40**	**0, 45**	**0, 50**	**0, 55**	**0, 60**	**0, 65**	**0, 70**	**0, 75**	**0, 80**	**0, 85**	**0, 90**	**0, 95**	**1, 00**
10		[0, 0]	[0, 2]	[0, 3]	[0, 4]	[0, 5]	[0, 5]	[0, 5]	[0, 5]	[0, 6]	[0, 6]	[0, 6]	[0, 6]	[0, 6]	[0, 6]	[0, 5]	[0, 4]	[0, 2]	[0, 0]	
15	[0, 1]	[0, 4]	[0, 6]	[0, 7]	[1, 7]	[1, 8]	[1, 8]	[2, 9]	[1, 9]	[2, 9]	[1, 10]	[1, 10]	[0, 10]	[0, 10]	[0, 10]	[0, 11]	[0, 11]	[0, 8]	[0, 3]	
20	[0, 3]	[1, 7]	[2, 8]	[3, 9]	[3, 10]	[4, 11]	[4, 11]	[4, 12]	[5, 12]	[5, 13]	[4, 14]	[4, 14]	[3, 14]	[2, 15]	[2, 15]	[1, 15]	[0, 14]	[0, 13]	[0, 7]	
25	[0, 6]	[2, 9]	[3, 11]	[5, 12]	[6, 13]	[7, 14]	[7, 15]	[7, 15]	[8, 16]	[8, 17]	[8, 17]	[7, 18]	[7, 18]	[6, 19]	[5, 19]	[4, 19]	[2, 19]	[0, 17]	[0, 10]	
30	[1, 7]	[4, 11]	[5, 14]	[7, 15]	[8, 16]	[9, 17]	[10, 18]	[11, 19]	[11, 20]	[11, 21]	[11, 21]	[11, 22]	[11, 22]	[10, 23]	[9, 24]	[8, 24]	[6, 23]	[2, 21]	[0, 13]	
35	[2, 9]	[5, 14]	[8, 16]	[9, 18]	[11, 19]	[12, 20]	[13, 21]	[14, 23]	[15, 24]	[15, 24]	[15, 25]	[15, 26]	[15, 27]	[14, 27]	[13, 28]	[11, 28]	[9, 28]	[5, 25]	[0, 17]	
40	[3, 12]	[7, 16]	[10, 19]	[11, 21]	[13, 22]	[15, 24]	[16, 25]	[17, 26]	[18, 27]	[19, 28]	[19, 29]	[19, 30]	[19, 31]	[19, 32]	[17, 32]	[16, 33]	[13, 32]	[8, 31]	[2, 22]	
45	[5, 13]	[9, 19]	[12, 21]	[14, 23]	[16, 26]	[18, 27]	[19, 28]	[21, 30]	[21, 31]	[22, 32]	[23, 33]	[23, 34]	[24, 35]	[23, 36]	[22, 37]	[21, 37]	[18, 37]	[12, 36]	[4, 27]	
50	[6, 15]	[11, 21]	[14, 24]	[17, 26]	[19, 29]	[20, 30]	[22, 32]	[24, 33]	[25, 35]	[26, 36]	[27, 37]	[27, 38]	[28, 39]	[28, 40]	[27, 41]	[25, 42]	[22, 42]	[17, 40]	[7, 31]	
55	[7, 18]	[13, 23]	[17, 27]	[19, 29]	[22, 32]	[24, 34]	[25, 35]	[27, 37]	[28, 38]	[30, 40]	[31, 41]	[32, 42]	[33, 44]	[33, 45]	[32, 46]	[30, 47]	[27, 47]	[22, 45]	[10, 36]	
60	[9, 20]	[15, 26]	[19, 30]	[22, 32]	[24, 35]	[27, 37]	[28, 39]	[30, 41]	[32, 42]	[34, 44]	[35, 45]	[37, 46]	[37, 48]	[37, 49]	[37, 50]	[35, 51]	[33, 52]	[26, 50]	[13, 41]	
65	[11, 22]	[17, 28]	[22, 32]	[25, 35]	[28, 38]	[30, 40]	[32, 42]	[34, 44]	[36, 46]	[38, 47]	[39, 49]	[40, 51]	[42, 52]	[42, 53]	[42, 55]	[40, 56]	[36, 56]	[32, 55]	[17, 45]	
70	[12, 24]	[20, 31]	[24, 35]	[28, 38]	[31, 41]	[33, 43]	[35, 45]	[37, 47]	[39, 49]	[41, 51]	[43, 53]	[45, 55]	[46, 56]	[47, 58]	[47, 59]	[45, 60]	[43, 61]	[35, 60]	[21, 50]	
75	[14, 26]	[22, 33]	[27, 37]	[31, 41]	[34, 44]	[36, 47]	[39, 49]	[41, 51]	[43, 53]	[45, 55]	[47, 57]	[49, 58]	[51, 60]	[51, 62]	[52, 63]	[51, 65]	[48, 66]	[41, 65]	[25, 55]	
80	[16, 28]	[24, 35]	[29, 40]	[34, 44]	[36, 47]	[40, 50]	[42, 52]	[44, 55]	[47, 57]	[49, 59]	[51, 61]	[53, 62]	[55, 64]	[56, 66]	[56, 68]	[56, 69]	[53, 70]	[46, 70]	[29, 60]	
85	[18, 30]	[26, 38]	[32, 43]	[36, 47]	[40, 50]	[42, 53]	[45, 56]	[48, 58]	[50, 60]	[53, 62]	[55, 64]	[57, 66]	[59, 68]	[61, 70]	[62, 72]	[61, 74]	[59, 75]	[52, 74]	[34, 65]	
90	[19, 32]	[29, 40]	[35, 45]	[39, 49]	[43, 53]	[46, 56]	[48, 59]	[51, 62]	[54, 64]	[57, 66]	[59, 68]	[61, 70]	[63, 72]	[65, 74]	[66, 76]	[66, 78]	[64, 80]	[57, 79]	[38, 70]	
95	[22, 34]	[31, 43]	[37, 48]	[42, 52]	[46, 56]	[49, 59]	[52, 63]	[55, 65]	[58, 68]	[60, 70]	[63, 72]	[65, 74]	[68, 77]	[70, 78]	[71, 81]	[71, 83]	[69, 84]	[63, 84]	[43, 74]	
100	[24, 36]	[33, 45]	[40, 51]	[45, 55]	[49, 60]	[53, 63]	[56, 66]	[58, 69]	[61, 71]	[64, 74]	[67, 76]	[69, 78]	[72, 81]	[74, 83]	[75, 85]	[76, 87]	[74, 89]	[67, 89]	[47, 80]	
105	[26, 39]	[36, 48]	[43, 54]	[48, 58]	[52, 62]	[56, 66]	[60, 69]	[62, 72]	[65, 75]	[69, 77]	[71, 80]	[74, 82]	[76, 85]	[78, 87]	[80, 89]	[81, 92]	[80, 93]	[73, 94]	[54, 84]	
110	[27, 41]	[39, 50]	[46, 57]	[51, 61]	[56, 65]	[60, 69]	[62, 73]	[66, 76]	[69, 79]	[72, 82]	[75, 84]	[78, 86]	[80, 89]	[83, 91]	[85, 94]	[86, 96]	[84, 98]	[79, 99]	[59, 89]	
115	[29, 43]	[41, 53]	[48, 59]	[54, 64]	[58, 68]	[63, 72]	[66, 76]	[69, 79]	[73, 82]	[76, 85]	[79, 88]	[82, 90]	[85, 93]	[88, 96]	[90, 98]	[90, 100]	[90, 103]	[85, 103]	[63, 94]	
120	[33, 44]	[44, 55]	[52, 62]	[57, 67]	[62, 71]	[68, 75]	[71, 79]	[74, 82]	[78, 85]	[81, 88]	[84, 91]	[86, 95]	[89, 97]	[92, 100]	[94, 102]	[96, 105]	[95, 107]	[90, 108]	[69, 99]	

[a]OS: order strength.
[b]act: number of activities.

In order to document that *RanGen* outperforms other generators, we refer to an example of the doctoral dissertation of Vanhoucke (2001). In Chapter 12, the author has solved the well-known *discrete time/cost trade-off problem* (DTCTP). In the paper by Demeulemeester et al. (1998), the authors were only able to generate a dataset with fragmentary results (i.e., for some values of the CI, they were not able to generate networks). This is because the problem generator *ProGen* used in that paper does not allow for the generation of networks satisfying preset values of the CI. Moreover, De Reyck and Herroelen (1996) encountered similar problems in generating sufficient problem instances over the full range of CI values. In the dissertation, the author does not show fragmentary results, but instead, he was able to generate sufficient problem instances for each value of the CI. The results of this observation can be seen in Figures 8 and 9. In Figure 8, we display the results of the computational experience obtained by solving a testset generated by *RanGen* for the DTCTP, using two different solution

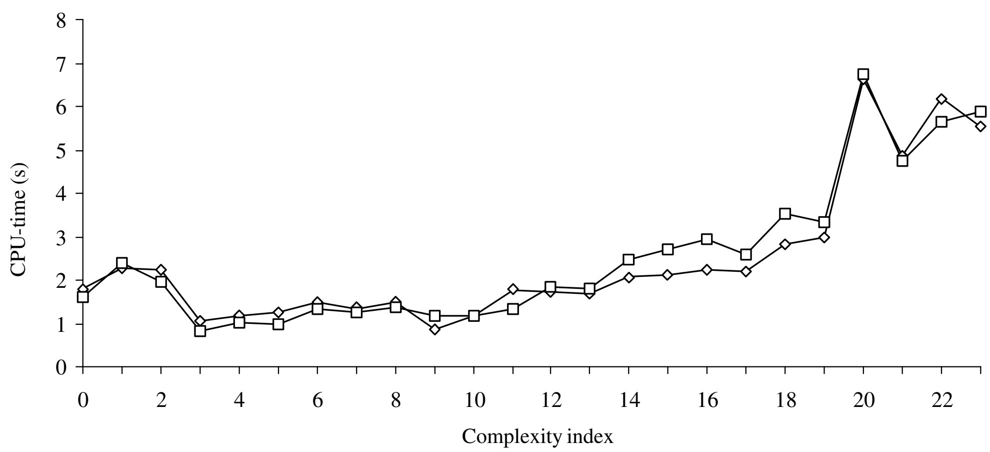


Figure 8. Computational experience for the discrete time/cost trade-off problem on a randomly generated dataset generated by *RanGen* (Source: Vanhoucke (2001))

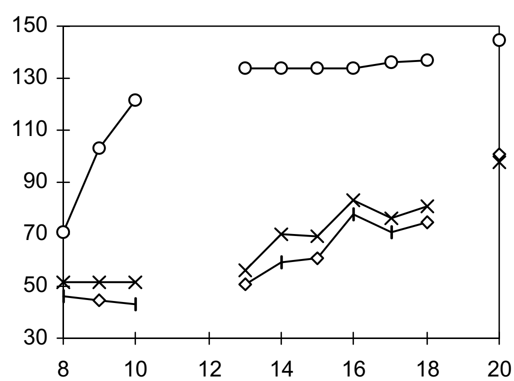


Figure 9. Computational experience for the discrete time/cost trade-off problem on a randomly generated dataset generated by *ProGen* (Source: Vanhoucke (2001) and Demeulemeester et al. (1998))

procedures. As you may notice, the CI value varies between 0 and 23. In Figure 9 (from the original paper by Demeulemeester et al. (1998)), we show the computational results obtained by three different solution procedures for the DTCTP (albeit on another, slower computer) on a randomly generated testset of *ProGen* (since at that time, *RanGen* was not yet developed). Unfortunately, the authors were not able to generate networks with a CI value varying between 0 and 23. Indeed, they have generated a testset with unknown values for the CI and afterwards, they have looked at the effect of the CI. In doing so, they have obtained networks with a CI value between 8 and 20, except for CI values equal to 11, 12, and 19, for which no networks could be generated.

Another example, which illustrates the importance of the CI as a predictive complexity measure, deals with the well-known unconstrained max-*npv* problem (for an overview of the literature, we refer to Vanhoucke, Demeulemeester, and Herroelen (2001)). In Figures 10 and 11, we show the impact of the CI on the computational effort needed by the recursive search procedure of Vanhoucke, Demeulemeester, and Herroelen (2001) for solving the the max-*npv* problem for different percentages of negative cash flows. It appears that the CI is positively correlated with the computational effort, that is, the larger the CI, the more difficult the problem.

In order to confirm our conjecture, we performed a loglinear regression for every setting of the percentage of negative cash flows, resulting in 11 equations. Apart from the settings in which the percentage of negative cash flows amounts to 0% or 100%, the results indicate that the CI is positively correlated with the computational effort needed to solve the max-*npv* problem. All p-values are small enough to confirm this conjecture. Moreover, we verified the assumption of normality by inspecting the normality plots of the residuals. The rather small values for the R^2 statistic (19% on the average) indicate that the variability for each setting is rather high. This leads to the conclusion that although the complexity index can be used as a predictor for the

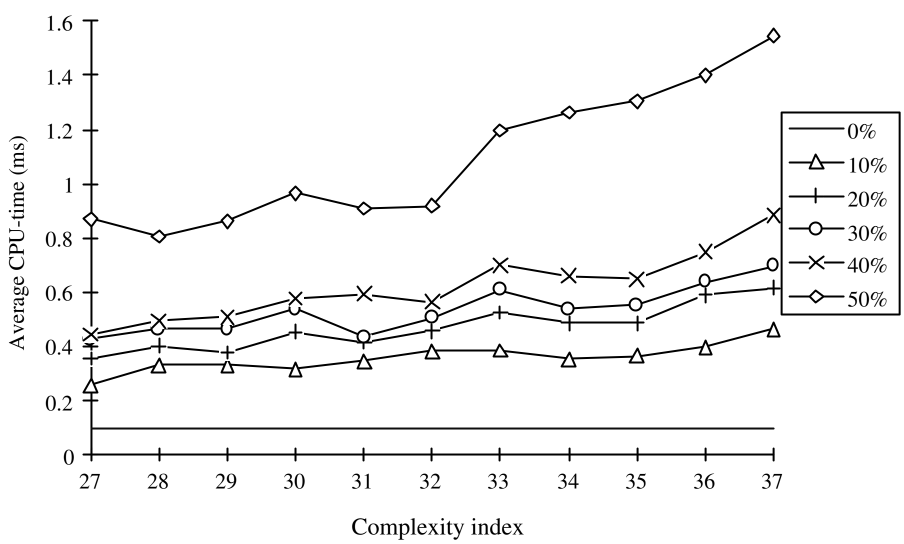


Figure 10. Effect of the complexity index (CI) and the percentage of negative cash flows (0–50%) for the max-*npv* problem

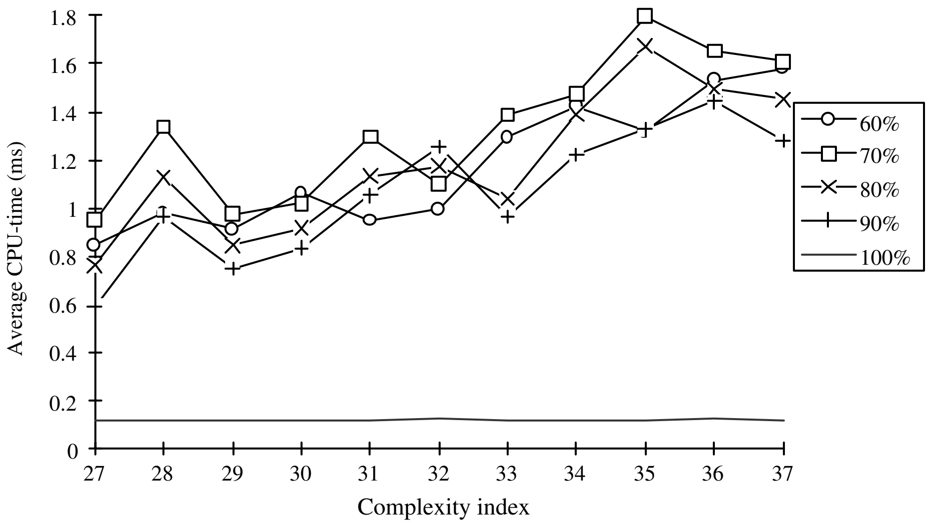


Figure 11. Effect of the complexity index (CI) and the percentage of negative cash flows (60–100%) for the max-*npv* problem

computational effort of the max-*npv* problem, it explains only a small portion of the total variability.

In the next section, we review a number of resource measures from the literature and their implementation in *RanGen*.

7. RESOURCE MEASURES

Several resource measures have been introduced in the literature to describe the relation between the presence of different resources and their impact on the hardness of a project scheduling instance. Patterson (1976) has listed a large number of resource utilization parameters reflecting the tightness of certain resource types as well as different constrainedness parameters. Other famous attempts in describing resource parameters have been made by Pascoe (1966), Cooper (1976), Alvarez-Valdes and Tamarit (1989), and Kolisch, Sprecher, and Drexl (1995).

In generating the resource measures for a project, several decisions have to be made. First, we determine the density of the resource demand matrix in order to specify whether an activity uses a particular resource or not. This is done by computing the resource factor (RF) and the resource use (RU). Second, the resource demand and resource availability for each activity are generated. Therefore, we either determine the resource strength (RS) or the resource constrainedness (RC). In the following we make a distinction between the single-mode and the multimode case.

7.1. Single-mode case

ProGen (Kolisch, Sprecher, and Drexl, 1995) and *ProGen/Max* (Schwindt, 1995) use the RF. This parameter, introduced by Pascoe (1966) and utilized in studies by Cooper (1976) and

Alvarez-Valdes and Tamarit (1989), can be calculated as follows:

$$\mathrm{RF} = \frac{1}{nK}\sum_{i=1}^{n}\sum_{k=1}^{K}\begin{cases}1, & \text{if } r_{ik} > 0\\ 0, & \text{otherwise}\end{cases} \tag{1}$$

where n denotes the number of activities (excluding dummy activities), K denotes the number of resource types, and r_{ik} denotes the amount of resource type k required by activity i. The resource factor RF reflects the average portion of resource types requested per activity and consequently measures the density of the matrix r_{ik}. According to Kolisch, Sprecher, and Drexl (1995), there is a positive relation between the required CPU-time to solve the well-known resource-constrained project scheduling problem and the RF while Alvarez-Valdes and Tamarit (1989) have observed that problems with $\mathrm{RF} = 1.0$ were easier to solve than problems with $\mathrm{RF} = 0.5$.

However, a few remarks should be made on the use of RF as a measure of the density of the matrix r_{ik}. When implementing the RF as defined in (1), it is possible that no resource requirement will be generated for some activities. This is certainly true when RF < 1/(number of resources), but it can also happen in other cases (e.g., $\mathrm{RF} = 0.5$ and half of the number of activities use all resource types while the other half do not require any resources).

ProGen/Max (Schwindt, 1995) uses a lower bound equal to 1/(number of resources) for RF to assure that all activities use at least one resource type. We instead keep the original definition of RF as given in (1) and introduced a new measure of resource density, the RU. RU varies between zero and the number of resource types available and measures for each activity the number of resource types used in the following way:

$$\mathrm{RU}_i = \sum_{k=1}^{K}\begin{cases}1, & \text{if } r_{ik} > 0, \; {\scriptstyle i=1,\ldots,n}\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

In *RanGen*, $\mathrm{RU}_i = \mathrm{RU}$, where RU is a positive constant, to assure that each activity uses at least one resource type. Moreover, the impact of the number of resources on problem hardness can be studied by varying the number of resource types K for the set of networks with $\mathrm{RF} = 1$ (or RU equal to the number of resource types).

RanGen relies on two resource measures for generating the resource availability and the resource requirements: the resource strength (RS) and the resource-constrainedness (RC). RS was first introduced by Cooper (1976) and later used by Alvarez-Valdes and Tamarit (1989). We use the new definition introduced by Kolisch, Sprecher, and Drexl (1995):

$$\mathrm{RS}_k = \frac{a_k - r_k^{\min}}{r_k^{\max} - r_k^{\min}} \tag{3}$$

where a_k denotes the total availability of renewable resource type k, $r_k^{\min}$ equals $\max_{i=1,\ldots,n} r_{ik}$ and $r_k^{\max}$ denotes the peak demand of resource type k in the precedence preserving earliest start schedule. Elmaghraby and Herroelen (1980) were the first to conjecture that the relationship between the complexity of a resource-constrained project scheduling problem and the resource availability varies according to a bell-shaped curve. De Reyck and Herroelen (1996) and Herroelen and De Reyck (1999) confirmed this conjecture and rejected the negative correlation between problem difficulty and the RS found by Kolisch, Sprecher, and Drexl (1995).

RC has been introduced by Patterson (1976):

$$\mathrm{RC}_k = \frac{\bar{r}_k}{a_k} \tag{4}$$

where a_k is defined as above and $\bar{r}_k$ denotes the average quantity of resource type k demanded when required by an activity, that is, $\bar{r}_k = \sum_{i=1}^{n} r_{ik} / \sum_{i=1}^{n} \{1, \text{if } r_{ik} > 0; \text{ 0 otherwise}\}$.

The arguments for using either the RS or the RC are often confounding. Kolisch, Sprecher, and Drexl (1995) argue that the major advantage of the RS lies in the incorporated information about the precedence structure of the network, while De Reyck and Herroelen (1996) find this a drawback since then it cannot be considered as a pure measure anymore. In addition, these authors have shown occasions where the RS can no longer distinguish between easy and hard problem instances while RC continues to do so. Furthermore, Herroelen and De Reyck (1999) restrict the use of these parameters to the case where there is only one resource type or when RS and RC are constant over all resources.

In summary, *RanGen* needs a number of inputs for the resource measures. First, we need to specify the number of resource types K and a value of either the RF or the RU. Alternatively, we can choose to vary the number of resources in an interval while, consequently, the RF is set automatically to one. Second, a value for either the RS or the RC is needed in order to generate the resource demand and availability.

7.2. Multi-mode case

In Section 7.1, we assumed a project has only one way of performing each activity. In what follows, we broaden this scope to the introduction of several execution modes for each activity. In this multi-mode project scheduling problem, the activities possess different execution modes reflecting different ways of performing the activities. Each mode is a tuple denoting an activity duration with corresponding resource demand. The interrelation $r_{ikm} = g_{ikm}(d_{im})$ between the durations of the modes and the resource demand is reflected by two types of functions g_{ikm} (Kolisch, Sprecher, and Drexl, 1995). r_{ikm} denotes the resource demand of mode m of activity i for resource type k while d_{im} denotes the duration of mode m of activity i. When the resource demand is a decreasing function of the durations, either a time/cost trade-off or a time/resource trade-off is involved. In the former case, the resource is of the nonrenewable type, while in the latter case it is a renewable resource. In resource/resource trade-off problems, the resource demand does not depend on the activity duration. We discuss these two interrelations in the following subsections.

7.2.1. Time/cost and time/resource trade-off functions

If the resource demand is decreasing with the duration of the modes, the function g_{ikm} can be either linear, convex, concave, or randomly chosen. In order to capture these four cases, we need as an input an interval $\mathrm{SI} = [a, b]$ for the slope connecting two adjacent modes. Starting with a randomly chosen resource demand for the mode with the highest duration (normal duration), we randomly choose a value for the slope $s \in [a, b]$. In doing so, we are able to generate the resource demand for the next mode and we update the interval SI' as follows:

- random: $SI' = [a, b]$
- linear: $SI' = [s, s]$

- convex: $SI' = [s, b + s - a]$
- concave: $SI' = [a - b + s, s]$

We repeat this stepwise generation method until the crash duration is reached, which corresponds to a maximum allocation of resources.

Suppose we generate a convex time/resource trade-off function for an activity i with four modes with corresponding durations $d_{i1} = 10$ (normal duration), $d_{i2} = 9$, $d_{i3} = 7$, and $d_{i4} = 5$ (crash duration). The $\mathrm{SI} = [1, 3]$. The randomly chosen resource demand for the normal duration equals 3, that is, $\mathrm{mode}_1 = (10, 3)$. We select randomly a slope $s = 2 \in [1, 3]$, update the interval $\mathrm{SI}' = [2, 4]$ and determine the next mode as follows: $\mathrm{mode}_2 = (9, 5)$. Again, we select randomly slope s equals $2 \in [2, 4]$. The updated $\mathrm{SI}' = [2, 4]$ and the new $\mathrm{mode}_3 = (7, 9)$. The last randomly chosen slope s equals $3 \in [2, 4]$ which results in an updated $\mathrm{SI}' = [3, 5]$ and the crash $\mathrm{mode}_4 = (5, 15)$. The four modes for activity i and one renewable resource type k, $\{(d_{im}, r_{ikm})\} = \{(5, 15), (7, 9), (9, 5), (10, 3)\}$, clearly represent a convex time/resource trade-off function. In the case of a nonrenewable resource k, the four modes represent a time/cost trade-off function.

7.2.2. Resource/resource trade-off functions

If the resource demand is independent of the duration, the resource requirement for each activity is randomly generated as follows. We assign to each resource type k a weight wr_k and to each activity a total work content $W_i = \sum_{k=1}^{K} wr_k r_{ikm}$. During the generation of the resource demand for each mode m of activity i, the work content W_i is held constant. To that purpose, we randomly generate resource demands for the first mode and for $K - 1$ resource types and fix the demand of the last resource type such that the work content is held constant. During the generation of the other modes, we increase the demand for a randomly chosen resource type and subsequently determine the demands for the other resource types. We repeat this generation method until all the modes of the activities contain resource demands for each resource type, as will be illustrated in the following example.

Suppose we generate a resource/resource trade-off function of an activity i with three renewable resource types and three modes. The work content W_i equals 100 and the weights of the resources are $wr_1 = 10$, $wr_2 = 3$, and $wr_3 = 5$. First, we randomly generate two $(K - 1)$ numbers for the resource demands of the first mode, $r_{i11} = 3$ and $r_{i21} = 5$ and subsequently, $r_{i31} = (100 - 3 * 10 - 5 * 3)/5 = 11$. We randomly select the resource type 1 and increase the demand for the second mode with a randomly generated number 2. Therefore, the second mode contains the following resource demands: $r_{i12} = 5$ and $r_{i22} = 5$ and subsequently, $r_{i32} = (100 - 5 * 10 - 5 * 3)/5 = 7$. In generating the third mode, we now randomly select resource type 2 and increase its demand with 3 units, that is, $r_{i13} = 5$ and $r_{i23} = 8$ and subsequently, $r_{i31} = (100 - 5 * 10 - 8 * 3)/5 = 5$. The three modes for activity i with three renewable resource types, $\{(r_{i1m}, r_{i2m}, r_{i3m})\} = \{(3, 5, 11), (5, 5, 7), (5, 8, 5)\}$, clearly represent a resource/resource trade-off function.

In order to generate the availabilities a_k for each resource type k, we define the RS for the multimode case as follows:

$$\mathrm{RS}_k = \frac{a_k - r_k^{\min}}{r_k^{\max} - r_k^{\min}}$$

where $r_k^{\min}$ equals $\max_{i=1,\ldots,n\ m=1,\ldots,M} r_{ikm}$ and $r_k^{\max}$ denotes the peak demand of resource type k in the precedence preserving earliest start schedule, where each activity has a duration which corresponds to a maximum allocation of resources. Notice that this definition does not correspond to the definition by Kolisch, Sprecher, and Drexl (1995). In their definition, $r_k^{\min}$ equals $\max_{i=1,\ldots,n}\{\min_{m=1,\ldots,M} r_{ikm}\}$. Consequently, the feasibility of the problem cannot be assured since low values for the RS will lead to many infeasible modes, that is, modes for which the resource demand exceeds the availability a_k.

8. CONCLUSION

In this paper, we discussed the logic of *RanGen*, a new generator of AoN networks which can be used to represent the underlying project for different classes of project scheduling problems. *RanGen* avoids the shortcomings of existing network generators since it employs the OS as well as the CI, which have been shown in previous experiments to serve as reliable predictors of the hardness of various types of project scheduling problems.

We equipped *RanGen* with a number of resource measures taken from the literature. Both single-mode and multi-mode measures have been implemented in order to describe the relation between the presence of different resource types and their impact on problem hardness.

REFERENCES

Alvarez-Valdes, R. and J. M. Tamarit, "Heuristic algorithms for resource-constrained project scheduling: A review and empirical analysis," in R. SlowinskiJ. Weglarz (eds.), *Advances in Project Scheduling*, Elsevier, Amsterdam, 1989.

Agrawal, M. K., S. E. Elmaghraby, and W. S. Herroelen, "*DAGEN*: A generator of testsets for project activity nets," *Eur. J. Oper. Res.*, **90**, 376–382 (1996).

Bein, W. W., J. Kamburowski, and M. F. M. Stallmann, "Optimal reduction of two-terminal directed acyclic graphs," *SIAM J. Comput.*, **21**, 1112–1129 (1992).

Brucker, P., A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *Eur. J. Oper. Res.*, **112**, 3–41 (1999).

Cooper, D. F., "Heuristics for scheduling resource-constrained projects: An experimental investigation," *Manag. Sci.*, **22**, 1186–1194 (1976).

Dar-El, E. M., "MALB—A heuristic technique for balancing large single-model assembly lines," *IIE Trans.*, **5**, 343–356 (1973).

Davis, E. W., "Project network summary measures and constrained resource scheduling," *IIE Trans.*, **7**, 132–142 (1975).

Davies, E. M., "An experimental investigation of resource allocation in multiactivity projects," *Oper. Res. Quart.*, **24**, 587–591 (1973).

Demeulemeester, E., B. Dodin, and W. Herroelen, "A random activity network generator," *Oper. Res.*, **41**, 972–980 (1993).

Demeulemeester, E., B. De Reyck, B. Foubert, W. Herroelen, and M. Vanhoucke, "New computational results on the discrete time/cost trade-off function," *J. Oper. Res. Soc.*, **49**, 1153–1163 (1998).

De Reyck, B., "On the use of the restrictiveness as a measure of complexity for resource-constrained project scheduling," *Research Report 9535*. Department of Applied Economics, Katholieke Universiteit Leuven, Belgium, 1995.

De Reyck, B. and W. Herroelen, "On the use of the complexity index as a measure of complexity in activity networks." *Eur. J. Oper. Res.*, 91, 347–366 (1996).

Elmaghraby, S. E., "Activity nets: A guided tour through some recent developments," *Eur. J. Oper. Res.*, **82**, 383–408 (1995).

Elmaghraby, S. E. and W. S. Herroelen, "On the measurement of complexity in activity networks," *Eur. J. Oper. Res.*, **5**, 223–234 (1980).

Ford, J. E. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, New Jersy, 1962.

Herroelen, W. and B. De Reyck, "Phase transitions in project scheduling," *J. Oper. Res. Soc.*, **50**, 148–156 (1999).

Herroelen, W., B. De Reyck, and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments," *Comput. Oper. Res.*, **25**, 279–302 (1998).

Icmeli, O., S. S. Erengüç, and C. J. Zappe, "Project scheduling problems: A survey," *International J. Oper. Prod. Manag.*, **13**, 80–91 (1993).

Kaimann, R. A., "Coefficient of network complexity." *Manag. Sci.*, **21**, 172–177 (1974).

Kaimann, R. A., "Coefficient of network complexity: erratum." *Manag. Sci.*, **21**, 1211–1212 (1975).

Kamburowski, J., D. J. Michael and M. F. M. Stallmann, "Optimal construction of project activity networks," *Proceedings of the 1992 Annual Meeting of the Decision Sciences Institute*, San Francisco, 1992, pp. 1424–1426.

Kao, E. P. C. and M. Queranne, "On dynamic programming methods for assembly line balancing," *Oper. Res.*, **30**, 375–390 (1982).

Kolisch, R., A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Manag. Sci.*, **41**, 1693–1703 (1995).

Mastor, A. A., "An experimental and comparative evaluation of production line balancing techniques," *Manag. Sci.*, **16**, 728–746 (1970).

Özdamar, L. and G. Ulusoy, "A survey on the resource-constrained project scheduling problem," *IIE Trans.*, **27**, 574–586 (1995).

Pascoe, T. L., "Allocation of resources—CPM," *Revue Française de Recherche Opérationelle*, **38**, 31–38 (1966).

Patterson, J. H., "Project scheduling: The effects of problem structure on heuristic scheduling," *Naval Res. Logist.*, **23**, 95–123. (1976).

Patterson, J. H., "A comparison of exact procedures for solving the multiple-constrained resource project scheduling problem," *Manag. Sci.*, **20**, 767–784 (1984).

Schwindt, C., "A new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags," *WIOR-Report-449*, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe (1995).

Thesen, A., "Measures of the restrictiveness of project networks," *Networks*, **7**, 193–208 (1977).

Vanhoucke, M., "Exact algorithms for various project scheduling problems: Nonregular measures of performance and time/cost trade-offs." *Unpublished PhD Dissertation* (2001).

Vanhoucke, M., E. Demeulemeester, and W. Herroelen, "On maximizing the net present value of a project under renewable resource constraints," *Manag. Sci.*, **47**, 1113–1121 (2001).