Control and Cybernetics

vol. **35** (2006) No. 3

Adaptive resource allocation to stochastic multimodal projects: a distributed platform implementation in Java

by

Anabela P. Tereso, João Ricardo M. Mota and Rui Jorge T. Lameiro

Universidade do Minho, Dep. de Produção e Sistemas
4800-058 Guimarães, Portugal

e-mail: anabelat@dps.uminho.pt, joao_mota@fastmail.fm, rjl@eurotux.com

Abstract: This paper presents the implementation of the dynamic programming model (introduced in a previous paper) for the resolution of the adaptive resource allocation problem in stochastic multimodal project networks. A distributed platform using an Object Oriented language, Java, is used in order to take advantage of the available computational resources.

Keywords: cluster, distributed platform, activity networks, resource allocation, dynamic programming.

1. Introduction

The problem addressed in this paper may be stated as follows: Given a multimodal activity network with stochastic work contents, the goal is to determine the optimal resource allocation to all the activities of the network, in order to minimize cost.

The cost is composed of two parts: (i) the resource cost, which is assumed to be proportional to the square of the intensity of resource usage for the duration of the activity, and a tardiness cost, which is proportional to the amount of tardiness from a specified *Due Date* (T), with the proportionality constant equal to the *Unit Delay Cost* (c_L) (representing the marginal cost per period).

This problem was first treated using a dynamic programming model and was implemented in Matlab (see Tereso et al., 2004). The results obtained (see Tereso and Araújo, 2003) demonstrated the need for further study, particularly concerning the implementation language and paradigms used. In this paper we explain how the problem was redefined, using an *Object Oriented* (OO) programming language, Java, and parallel programming to take advantage of a network of computers.

2. Review of prior work

Before introducing the new implementation and results obtained, we shall briefly define the problem and describe how it first was implemented in Matlab.

2.1. Problem definition

Given a multimodal activity network ('multimodal' means that each activity can be performed at any number of levels of resource intensity applied to it, with resulting shorter or longer duration), with a stochastic work content (W_a), we wish to decide on the amount of resource to apply to each activity (x_a), so that the total cost is minimized. This cost includes the resource cost and the delay cost. The duration of an activity depends on its work content and on the amount of resource allocated to it. To evaluate the delay cost, a due date must be specified (T), as well as the unit cost per period tardy (c_L). To the best of our knowledge this problem has never been treated before. Contributions to the classical 'resource constrained project scheduling problem' (RCPSP) and its variants are numerous; the interested reader may wish to consult the two most recent books on the subject by Demeulemeester and Herroelen (2002) and Neumann, Schwindt, and Zimmermann (2002), and the references cited therein, to gain a complete picture of developments in that aspect of project scheduling.

We imposed the following assumptions:

- The work content of each activity is a random variable ($r.v.$) exponentially distributed.
- The amount of resource applied to any activity is bounded from below and from above; $x_a \in [l_a, u_a]$, with $l_a < u_a$, for all activities a.
- The availability of the resource is unlimited, so it does not impose any limitations on the problem: any number of activities may run simultaneously.

The model developed to solve this problem will be reviewed, using a simple example with only three activities (see Fig. 1).

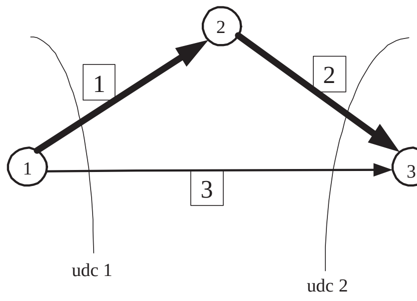

Figure 1. Example network with its uniformly directed cutsets

The due date of the project is $T = 16$, and the unit cost per period tardy is $c_L = 2$. The resource allocation to each activity is denoted by x_i for $i = 1, 2, 3$; with lower limit $l_i = 0.5$ and upper limit $u_i = 1.5$ for all i. The x_i's are the *decision variables* of this problem. The parameters $\{\lambda_i\}$ of the distributions of the work content of the activities are as shown in Table 1.

Table 1. Parameters for the simple example

Activity i :	**1**	**2**	**3**
λ_i :	0.2	0.1	0.07

At any point in time the manager must cope with a subset of activities that lie on a uniformly directed cutset (*udc*) (a *udc* represents a set of possible 'active' activities during the life of the project). In this simple example there are only two *udc*'s: $C_1 = \{1, 3\}$ and $C_2 = \{2, 3\}$. At the start the project manager is concerned with activities 1 and 3, which lie on C_1. Then, depending on the progress in these two activities, he may eventually be concerned with activities 2 and 3 (if activity 1 completes before activity 3) which lie on C_2, or with activity 2 alone if activity 3 completes before activity 1. If the resource allocation to activity 3 is (temporarily) fixed at, say, $\hat{x}_3$, the problem reduces to the optimal determination of the resource allocation to activities 1 and 2, which can be readily resolved by standard DP recursion. The set of 'fixed' activities is denoted by $\mathcal{F}$; in this example $\mathcal{F} = \{3\}$. Finally, searching over the values of x_3 with repeated optimization at each value would yield the (unconditional) optimal allocation to all three activities. In general, our procedure determines the *udc*'s of the network (which define the stages of the DP iterative scheme), and the cutset intersection index (*cii*), which represents the variables to be (temporarily) fixed (see Tereso et al., 2004, for details).

The expected resource cost of the fixed variables is denoted by *rcf*, which in this case, is the expected cost of activity 3

$$rcf = \hat{x}_3 \cdot \mathcal{E}(W_3) = \frac{\hat{x}_3}{0.07}, \tag{1}$$

where W_3 is the work content of activity 3, $\mathcal{E}(W_3)$ denotes its expected value, and $\hat{x}_3$ the amount of resource allocated to it. Reverse numbering of the DP stages yields

$$f_1(t_2 \mid \mathcal{F} = \{3\}) = rcf + \min_{x_2 \in [0.5, 1.5]} \mathcal{E}\{x_2 W_2 + 2\mathcal{E}(U)\}, \tag{2}$$

where

$$U = \max\{0, \Upsilon_3 - T\}, \tag{3}$$

and

$$\Upsilon_3 = \max\{t_2 + \frac{W_2}{x_2}, \frac{W_3}{\hat{x}_3}\}. \tag{4}$$

The second and last stage would be defined as follows:

$$f_2(t_1 = 0 \,|\, \mathcal{F} = \{3\}) = \min_{x_1 \in [0.5, 1.5]} \mathcal{E}\{x_1 W_1 + \mathcal{E}[f_2(\Upsilon_2)]\} \tag{5}$$

where

$$\Upsilon_2 = \frac{W_1}{x_1}. \tag{6}$$

The solution for this network, obtained in 0.094 seconds, was:

$\{x_1^*, x_3^*\} = \{1.0, 1.0\}$
with an expected cost of 43.32. (7)

The optimal value of x_2 depends on the state of node 2, when it is reached, and can be obtained by the previously developed optimal policy for stage 1, as defined in equation (2). The time necessary to get results in this example is quite small, but for larger networks, this time increases exponentially, taking hours and even days to achieve (see the order of complexity of this problem in the paper by Tereso et Araújo, 2003).

2.2. Matlab implementation

This model was first implemented in Matlab. The pseudo-code can be accessed on the internet (*www.dps.uminho.pt/pessoais/anabelat*; Topic: research), or upon request by e-mail (*anabelat@dps.uminho.pt*). Details of the development of this application can be found in Tereso et Araújo (2003). Here, we will only refer to the more important aspects of this implementation. One of the main problems was that the code necessary to implement the DP iterations was dependent on the network topology and the derived number of stages of the DP iterations. To solve this problem we decided to compose the code dynamically, after the network was imputed. This was accomplished by using the procedures *generateMainCode*, *generateDps1Code* and *generateDpsNCode*.

For the example network introduced above with only three activities, the generated code (*main*, *dps1*, *dps2*) can be seen on the first author's web page.

After the code generation, the *main* program is called, which in turn calls *dps1*, to optimize stage 1 and *dps2* to optimize stage 2. The result is then displayed.

For a larger network, the number of *dps* procedures and the number of nested cycles inside them increases, rendering the program very slow.

3. Application development

In this section, we discuss the main issues that arose during the new development using Java.

3.1. The choice of a programming language

From the beginning, we wanted an OO language and the possibility to easily create applications for different architectures (ia32, ppc, etc.) and different *Operating Systems* (OSs) so that the doors are kept open to future developers. Nowadays, there are two powerful languages that fit these requirements, Java and C++, which means we had to choose between a quicker development time and a faster runtime, respectively.

Another important requirement is the existence of open source platforms to develop applications, which is true for both languages. Among other software, in the *GNU Project* (*http://www.gnu.org*), we can find open source implementations of a Java *Virtual Machine* (JVM) and a Java and C++ compiler.

At a first glance C++ appears to be the right choice because this algorithm is an approach to solve *NP-Hard* problems, and it is important to choose a language that generates fast applications. But there are other factors to consider, such as the deadline for the project completion and our experience with each technology. We chose Java and Sun Microsystems' JVM (*http://java.sun.com*) (which is not open source but is free of charge).

3.2. From Matlab to Java

The first step was the conversion of the existing implementation in Matlab to Java, taking advantage of the OO-model, keeping the code as clean as possible, and easy to read and understand.

3.2.1. Data structures and input parameters

The complexity of the code usually grows if we use simple data structures, so we replaced the list of activities containing five fields per activity (the source node, the target node, the parameter λ, the lower bound, and the upper bound on the resource allocation), Tereso and Araújo (2003), by a more complex structure.

To represent the 'list of activities', we defined three Classes in Java (which are our main data structures):

- Node - to represent each node of the graph with information about immediately preceding and immediately succeeding nodes and the activities that connect to the node;
- Activity - to represent one activity with information about the parameter λ, the lower and upper bounds on the resource allocation; and
- Network - which contains a list of activities and a list of nodes.

Another important Class is ResourceCombination. This object is passed to the algorithm in order to calculate the optimum expected value of that combination. At each stage this object is cloned. There is a clone for every combination of the node time values belonging to that *Uniformly Directed Cutset* (UDC).

The number of resource combinations for a specific problem is determined by the following expression:

$$\#Resource\ Combinations = {k_2}^{\#\mathcal{F}} \tag{8}$$

where k_2 is the number of discretized points used for the fixed activities and $\#\mathcal{F}$ is the number of fixed activities (see Sections 3.2.2 and 3.2.3).

At the end of each stage, the clones will contain all the possible time values of the nodes in the current UDC. Each clone will also contain the optimal value of the decision variable and the expected value of the cost for that stage. This list is implemented through another class (BestResults) in order to allow a fast method of search for time values (see section 3.3).

When the program finishes testing all the resource combinations (for the fixed activities), the BestResults of all stages are shown for the best combination found. The class ResourceCombination implements methods to generate the next combination of:

- Resources for the fixed activities,
- Resources for the decision activities and
- Time values for the nodes.

This way the nested cycles for the implementation of the combinations present in the Matlab version are eliminated as well as the need for problem specific code generation.

Finally, there is the Class Problem which joins all data and metadata to represent our problem.

Fig. 2 shows the relations between these classes in a simplified model. Not all the classes are presented and some fields along with all the methods have been removed due to space limitations.

Appendix A contains an extract of the most relevant code of the Java implementation. The first section (A.1) shows a simplified version of the main program, and how the tasks are distributed between the threads. Each thread evaluates the cost of a combination (see code in section A.2) and returns its value. After that a new thread is started to evaluate the next combination. This process is repeated until there are no more resource combinations. At this point, the main program chooses the best result and the associated combination.

3.2.2. The stages of the dynamic program

The number of stages of the *Dynamic Program* (DP) iterations is the same as the number of decision variables. This is determined by evaluating the longest path (by the number of activities) in the network. This is easier to do in Java with the previously summarized data structure. The variables along this path define the set $\mathcal{D}$ of decision variables, and the complementary set of activities defines the set $\mathcal{F}$ of 'fixed' variables.

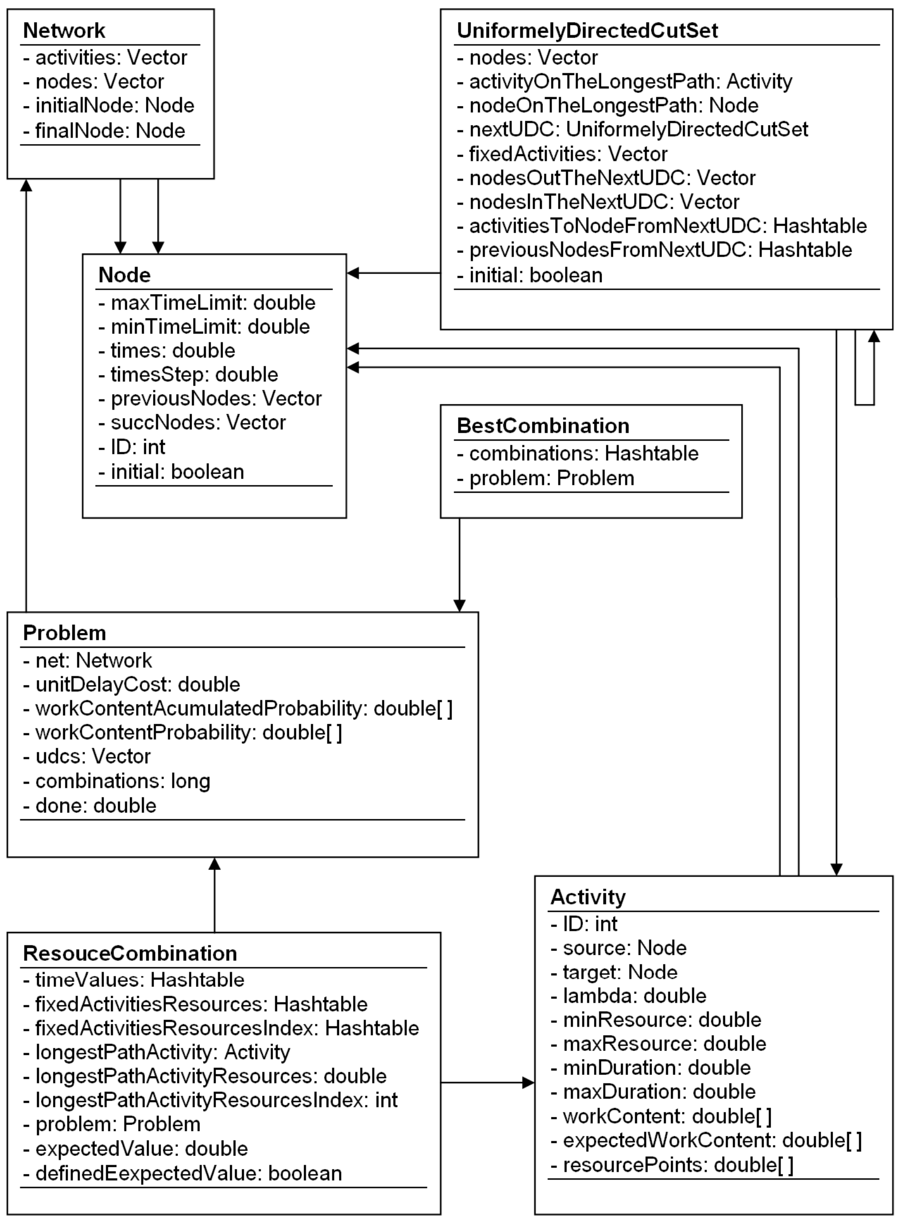

Figure 2. Simplified class diagram

To accomplish this, we constructed a set of methods associated with the Class Network; they are: getNodesOnLongestPath() and getActivitiesOnLongestPath(). With the first we retrieve the topology of the longest path and, with the second, we can populate the set $\mathcal{D}$.

3.2.3. The discretization of the work content

For the sake of computational feasibility, it is necessary to discretize the work content of each activity, using a finite number (k) of values of equal probability. These values represent the exponential distribution, all having the same probability (of $\frac{1}{k}$). The mean of these values is equal to $\frac{1}{\lambda}$ (the mean of the distribution).

For the decision activities, we used $k_1 = 5$. For the fixed activities we used a variable k_2. The previous Matlab implementation used $k_2 = 3$ or $k_2 = 2$ depending on the size of the network. In the Java implementation we used $k_2 = 3$ and for the cluster version also $k_2 = 5$.

3.3. Code optimization

The search method, used to find the closest combination of node due times (from the next UDC), was the most important code optimization of this new implementation. In Matlab this was performed applying the command 'find' to the 'best-result' matrix. This led to a full sequential search of this matrix. In our new approach, we used a hash table to accomplish faster access times. The hash function is as follows:

$$\textit{Hash Function: Combination} \rightarrow \textit{Integer}$$

$$\textit{hash(combination)} = \sum_{i=1}^{N} T_i \times MTV^i \tag{9}$$

where

- MTV = maximum size of 'Time Values',
- T_i = index value of 'Time Values' of the node i, and
- N = number of nodes.

In order to search a similar combination we first have to calculate the approximate T_i's of the one being searched. This is done by choosing the nearest 'Time Value' to the one being searched for each node. After this we use the same hash function to retrieve the right combination calculated in the following stage.

This optimization drastically reduced computation time (the application became $\approx$ 10 times faster) but the search method is still the largest *Central Processing Unit* (CPU) consumer of the entire program, accounting for about one half of the total CPU time consumed.

3.4. Cluster

The distributed application works in the client/server model. The philosophy is similar to the *SETI at Home* (*http://setiathome.ssl.berkeley.edu/*) project. The server divides the problem into small parts or chunks (sets of 250 resource combinations by default) to be processed by the clients. As the server has more sets of resource combinations to process, each client asks for a set, processes it, and returns the result to the server. In the end, the server joins everything and returns the best solution found.

Besides the network related code, the algorithm for the cluster version is similar to the single machine code, presented in Appendix A. The main difference is the number of combinations processed by each thread, or, in this case, by each cluster node.

All communication between the server and each client is made using TCP/IP sockets, so we are not restricted to a *Local Area Network* (LAN) but we can use the Internet to establish a client/server connection. If the computers are not on the same LAN and there is a slow connection (over the Internet) between them, it is a good idea to increase the number of resource combinations per set to decrease the overhead introduced by the network.

3.4.1. Example

Let us consider the network of Fig. 1. In this case, we only have one fixed activity (x3) with three possible values: 0.5, 1.0 and 1.5. Since we used 250 resource combinations per chunk, and in this network we only have three resource combinations, only one cluster node will process this problem.

Since this is a very small network, we did not use a cluster to solve the problem. A computer with a single processor sequentially evaluates all the combinations in a fraction of a second.

To better illustrate this issue, we are going to present an example where we have three fixed activities discretized in three points (0.5, 1.0, 1.5). We will have $3^3 = 27$ combinations. Consider that the time unit is the time needed to process each combination.

In the Matlab implementation, the combinations would be processed sequentially (see Table 2).

In the Java implementation, we use two threads in parallel computation to process the resource combinations. In this case we would have the work distribution described in Table 3.

Finally, for the cluster version, let us consider that we have three cluster clients (A, B and C) and that A has two times the computation power of B, and four times the computational power of C. A takes 1 second to process each chunk (assume the time lost between process communications to be negligible). When the clients connect to the server, they ask for chunks (sequential combinations of the fixed activities). When each node finalizes processing a chunk, it returns the result and asks for a new one. Suppose they all connect at the same time. The sequence of chunk requests would be in Table 4.

Table 2. Work distribution in the sequential version

t	Resource Combination
0	(0.5, 0.5, 0.5)
1	(0.5, 0.5, 1.0)
2	(0.5, 0.5, 1.5)
...	...
26	(1.5, 1.5, 1.5)

Table 3. Work distribution in the dual thread version

	Resource Combinations	
t	Thread 1	Thread 2
0	(0.5, 0.5, 0.5)	(0.5, 0.5, 1.0)
1	(0.5, 0.5, 1.5)	(0.5, 1.0, 0.5)
...	...	...
13	(1.5, 1.5, 1.0)	(1.5, 1.5, 1.0)

Table 4. Work distribution in the cluster version

t (s)	Chunk	Client
0	{(0.5, 0.5, 0.5), (0.5, 0.5, 1.0), (0.5, 0.5, 1.5)}	A
	{(0.5, 1.0, 0.5), (0.5, 1.0, 1.0), (0.5, 1.0, 1.5)}	B
	{(0.5, 1.5, 0.5), (0.5, 1.5, 1.0), (0.5, 1.5, 1.5)}	C
1	{(1.0, 0.5, 0.5), (1.0, 0.5, 1.0), (1.0, 0.5, 1.5)}	A
2	{(1.0, 1.0, 0.5), (1.0, 1.0, 1.0), (1.0, 1.0, 1.5)}	A
	{(1.0, 1.5, 0.5), (1.0, 1.5, 1.0), (1.0, 1.5, 1.5)}	B
3	{(1.5, 0.5, 0.5), (1.5, 0.5, 1.0), (1.5, 0.5, 1.5)}	A
4	{(1.5, 1.0, 0.5), (1.5, 1.0, 1.0), (1.5, 1.0, 1.5)}	A
	{(1.5, 1.5, 0.5), (1.5, 1.5, 1.0), (1.5, 1.5, 1.5)}	B
	{no more chunks}	C
5	process concluded.	

In this simple example, there is an under-utilization of client C (in t=4). In order to solve this problem, we would have to configure the faster nodes to ask for bigger chunks, or define a smaller chunk size (this would decrease the processing time for each chunk). For larger networks (from network 7 onwards in our experiments) this phenomenon represents a negligible difference in processing time (<0.1% of the total time), even with the chunk size used in the benchmark (250 combinations).

3.4.2. Differences between cluster and a single station

The non distributed version in JAVA is similar to the distributed one (cluster version). The only difference is that for the single station version, there is no need for the use of network protocols, even if using more than one thread. For more than one thread, the work division is made in the same way (see section above).

Figs. 3 and 4 show, in an illustrative scheme, the difference between the work flow on a single station and on a cluster. Apart from the existence of threads, in the single machine application, the evaluation of each resource combination is made sequentially. For the cluster, this computation is made in parallel.

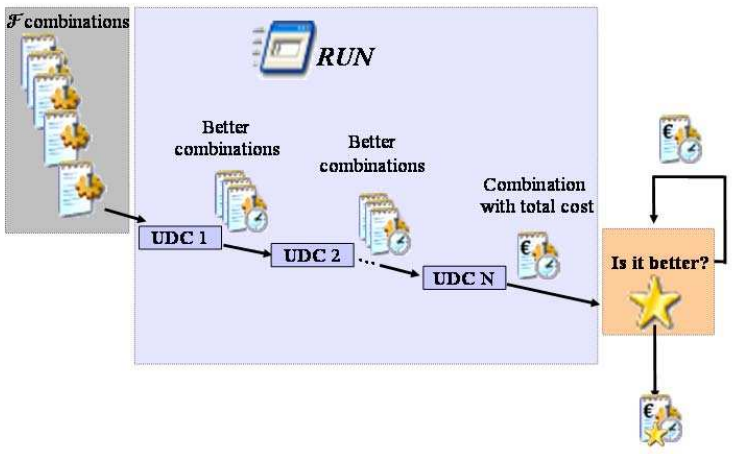

Figure 3. Single station work flow

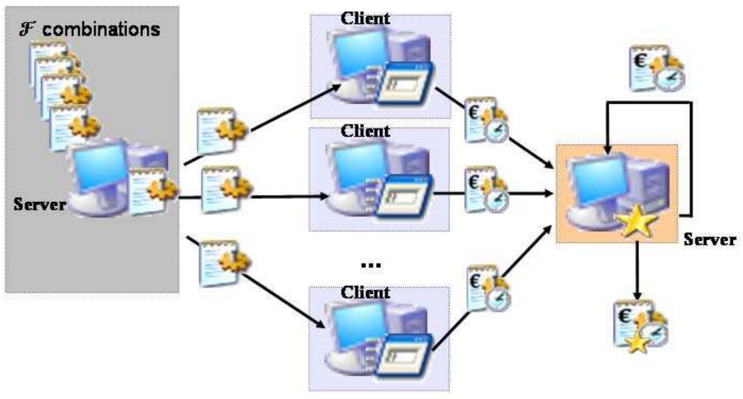

Figure 4. Cluster work flow

4. Results

The program outlined above was tested on a set of eleven projects that range in size from 3 to 18 activities (see Appendix B for details).

4.1. Benchmark results

The values relative to a single machine in Table 5 were obtained using an *Intel Pentium IV E* 3.0GHz with 1GB of RAM under *Microsoft Windows XP Professional SP2*. The cluster was formed by 13 computers (one of them acting as server and client at the same time) with an *Intel Celeron* 2.4GHz with 512MB of RAM using *Fedora Core 2* running a 2.4 *kernel* version on the same LAN connected at 100Mbps. In both cases we used Sun Microsystems' *Java Virtual Machine* (JVM) version 1.4.2.

This application was design, from the beginning, to take advantage of Intel's *HyperThreading Technology* (*http://www.intel.com/technology/hyperthread/*), common to today's computers. Even in a computer without support for the execution of instructions in parallel, this application would run without problems.

Table 5. Benchmark results

Net	n	T	cL	DP Time Matlab $k_2 = 3$	DP Time Java *dual thread* $k_2 = 3$	DP Time Java *cluster* $k_2 = 3$	DP Time Java *cluster* $k_2 = 5$
1	3	16	2	0.094s	0.015s	⋆	⋆
2	5	120	8	1.188s	0.140s	⋆	⋆
3	7	66	5	4.078s	0.188s	⋆	⋆
4	9	105	4	5m 39s	5.218s	⋆	⋆
5	11	28	8	10m 08s	22.422s	⋆	⋆
6	11	65	5	1h 03m 06s	2m 33s	29s	9m 11s
7	12	47	4	7h 43m 09s	19m 10s	2m 14s	1h 48m 06s
8	14	37	3	38h 45m †	1h 36m 17s	9m 45s	25h 31m 20s
9	14	188	6	441h 30m †	18h 16m 56s	1h 54m 12s	314h 46m †
10	17	49	7	117h 40m †	4h 52m 23s	29m 26s	135h 13m †
11	18	110	10	5285h †	218h 50m †	24h 19m 27s	11174h †

⋆ Same results as with the single machine tests. The network is too small for a cluster;
† Estimated.

Looking at the DP processing time results of networks 1 to 7 on a single machine, especially for those that need more than a second to be processed in our Java application, we can say that our Java implementation is about 25 times faster. Fig. 5 shows graphically the gain of Java *vs.* Matlab, for the example networks. This improvement is due to the search method referred in Section 3.3, the use of two threads (approximately duplicates the speed) and other code optimizations.

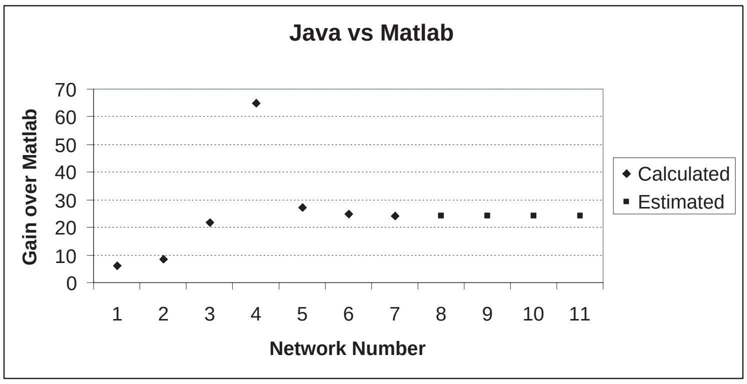

Figure 5. Gain from Java over Matlab

For the cluster version, we gave special consideration to the application performance when all the machines in our cluster were equal, just for testing purposes. In a real case, we can use all the machines to which we have access to create a cluster as large as possible. There is no need for the machines to be similar. We can even use computers with different OSs and different architectures on the same cluster.

4.2. Accuracy of the achieved solutions

Note that Table 5 does not contain the exact optimal solution to the problems studied, and the solution achieved could be different for the same network depending on the k_2 value. If we choose a larger k_2, we would get a better solution but the DP processing time will be drastically increased (see Table 5). Table 6 shows the solutions for the different k_2 values. For network 6, for example, a 1800% increase of the CPU time only resulted in a 2% improvement of the Best Cost Value.

5. Conclusions and future research

This implementation in Java is much faster than the previous one in Matlab but it is not fast enough unless one has many computers available to make a large cluster.

Even for a single machine version, there is a considerable improvement, compared with the Matlab implementation. In the first place, Java is a compiled

Table 6. Different solutions for the problem depending on k_2

Net	k$_2$	DP Best Cost	Set of best allocations to the first UDC and $\mathcal{F}$;
			$\{x_1, x_2, x_3, x_5, x_6, x_8, x_9, x_{10}\}$
6	3	272.298	$\{1.25, 0.5, 1.5, 0.5, 1.0, 1.0, 1.5, 1.0\}$
	5	266.498	$\{1.5, 0.5, 1.25, 0.5, 1.25, 1.0, 1.5, 1.0\}$
			$\{x_1, x_2, x_3, x_4, x_6, x_7, x_8, x_{10}, x_{12}\}$
7	3	182.914	$\{1.25, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.5\}$
	5	172.251	$\{1.5, 1.0, 0.75, 0.5, 0.75, .75, 0.5, 0.75, 1.5\}$
			$\{x_1, x_2, x_3, x_4, x_6, x_7, x_8, x_9, x_{10}, x_{12}, x_{14}\}$
8	3	120.335	$\{1.5, 1.5, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0\}$
	5	114.163	$\{1.5, 1.25, 1.25, 0.5, 0.75, 0.5, 0.75, 1.0, 0.75, 1.0, 0.75\}$

language while Matlab is interpreted. But this represents no relevant improvement, considering that Java is a much more complex and powerful language than Matlab and so having more objects created in its environment. The main reason has to do with the changing of the search method, using an hash table instead of an array. This represents up to 90% saving in running time. Finally, the possibility to use hiperthreading and clusters gives the Java implementation a theoretically infinite possibility to improve performance, depending on the size of the cluster available. But it is a fact that, with the resources of a lab, it was possible to solve a problem (network 11 in Table 5) in less than 25 hours, that would take more than seven months, for the Matlab implementation.

In order to use a cluster on an everyday basis using computers with other light tasks running (such as office applications), a simpler client program, invisible to the common user, should be written. Besides that, the chunk (set of resource combinations) size should be dynamically adapted to each client based on his processing speed and round trip time ('network distance').

The code could be improved by using a small k_2 and successively finer mesh. Consider network number 6 and its lower and upper bounds on the resource allocation (0.5 and 1.5 respectively for all activities). If $k_2 = 3$ we use both extremes (0.5 and 1.5) and a third point in the middle (1.0); if $k_2 = 5$, we use the two boundary values plus three more points in the middle (0.75, 1.0, 1.25). The idea is to use $k_2 = 3$ and when this first iteration finishes we will get

$$\{x_1, x_2, x_3, x_5, x_6, x_8, x_9, x_{10}\} = \{1.5, 0.5, 1.5, 0.5, 1.0, 1.0, 1.5, 1.0\}.$$

Then we need to refine the upper and lower bounds on the resource allocation before the next iteration, reducing the range between the lower and the upper bound to one half and centering the interval on the results of the previous iteration, never forgetting the original limits. In this case we would get the pairs

(1.25, 1.5), (0.5, 0.75), (1.25, 1.5), (0.5, 0.75), (0.75, 1.25), (0.75, 1.25), (1.25, 1.5) and (0.75, 1.25) for activities x_1, x_2, x_3, x_5, x_6, x_8, x_9 and x_{10} respectively. Theoretically, we should get the same or a better result as running this application once with $k_2 = 5$, taking twice the time needed to run the application with $k_2 = 3$. Looking to the cluster times to this network (see Table 5), within less than a minute, we would reach the same result that now takes more than nine minutes.

We have only used networks with *Activity on Arc* (AoA) representation and never with *Activity on Node* (AoN) representation of activities. In the future it could be useful to implement an algorithm to convert either mode of representation into the other. If we can convert networks from AoN to AoA easily (that is, in polynomial time), then we can use this application to solve the problem.

References

Tereso, A.P., Araújo, M.M. and Elmaghraby, S.E. (2004) Adaptive Resource Allocation in Multimodal Activity Networks. *International Journal of Production Economics* **92**, 1-10.

Tereso, A.P. and Araújo, M.M. (2003) Experimental Results of an Adaptive Resource Allocation Technique to Stochastic Multimodal Projects. *International Conference on Industrial Engineering and Production Management* (*IEPM'03*), Porto, Portugal, 26-28 May 2003.

Demeulemeester, E.L. and Herroelen, W.S. (2002) *Project Scheduling: A Research Handbook.* Kluwer Academic Publishers, Boston.

Neumann, K., Schwindt, C. and Zimmermann, J. (2002) *Project Scheduling with Time Windows and Scarce Resources.* Springer-Verlag, Berlin.

A. Java Code

A.1. Problem Class

```
public class Problem {
...

public ResourceCombination solve() {
  /* Variable initialization */
  ...

  // Create RunDPS objects with null Combinations
  Runable runable1 = new RunDPS(null, this);
  Runable runable2 = new RunDPS(null, this);

  while (!presentFixedCombination.isTheLast()) {
    ((RunDPS) runable1).setCombination(presentFixedCombination);
    thread1 = new Thread(runable1);
    thread1.start();
    presentFixedCombination = presentFixedCombination.next();

    if (!presentFixedCombination.isTheLast()) {
      ((RunDPS) runable2).setCombination(presentFixedCombination);
      thread2 = new Thread(runable2);
      thread2.start();
      presentFixedCombination = presentFixedCombination.next();
    } else
      thread2 = null;

    try {
      thread1.join();
      ResourceCombination result1 =
         ((RunDPS) runable1).getResult();
      if (result1.getExpectedValue()
          < bestResult.getExpectedValue())) {
        bestResult = result1;
      }
      if (!presentFixedCombination.isTheLast()) {
        thread2.join();
        ResourceCombination result2 =
            ((RunDPS) runable2).getResultado();
        if (result2.getExpectedValue()
            < bestResult.getExpectedValue()) {
          bestResult = result2;
        }
      }
    }
    catch (InterruptedException ex) {
      ...
    }
  }
  return bestResult;
}

}
```

A.2. RunDPS Class

```
public class CorrerDPS implements Runnable {
  ...

public void run() {

  Vector udcs = this.problem.getUdcs();
  BestCombinations results;
  UniformelyDirectedCutSet presentUDC=((UniformelyDirectedCutSet)
      udcs.elementAt(0));
  UniformelyDirectedCutSet previousUDC;

  // DP 1st stage
  results =
    this.problem.dynamicProgrammingFirstStage(combination,
                                              presentUDC);

  // DP stage 2 to n
  int i;
  for (i = 1; i < udcs.size(); i++) {
    presentUDC = (UniformelyDirectedCutSet) udcs.elementAt(i);
    previousUDC = (UniformelyDirectedCutSet) udcs.elementAt(i-1);
    results = this.problem.dynamicProgramming(combination,
        presentUDC, previousUDC, results);
  }

  this.result = (ResourcesCombination)
              result.toVector().firstElement();
}

...
}
```

B. Example Networks

B.1. Network 1

The first network tested is represented in Fig. 6. It is a very simple network, with only three activities. The due date of this network, T, is 16 and the tardiness penalty, c_L, is two per unit time. The remaining parameters are represented in Table 7. These parameters are the origin and target node of each activity, the parameter (λ) of the exponential distribution, that represents the Work Content of each activity, and the minimal and maximal amount of resource to allocate to each activity ($\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$). The expected duration of activity 1 is

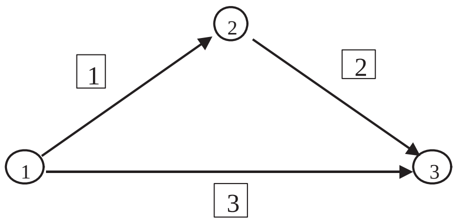

Figure 6. Network 1

$1/\lambda = 1/0.2 = 5$, and for activity 2 and 3, 10 and 14.29 respectively. In this way, the PERT expected duration for this network is 15. The due date of the project is selected to be a value above the PERT expected duration (approximately 5% more).

Table 7. Parameters for network 1

Activity	**1**	**2**	**3**
Origin	1	2	1
Target	2	3	3
λ	0.2	0.1	0.07
$\mathbf{x}_{\min}$	0.5	0.5	0.5
$\mathbf{x}_{\max}$	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_3^*\} = \{1.0, 1.0\}$, with an expected cost of 43.32.

B.2. Network 2

This network has five activities. The due date is $T = 120$ and the tardiness cost is $c_L = 8$. In Table 8 there are the remaining parameters. The PERT expected duration for this network is 115.

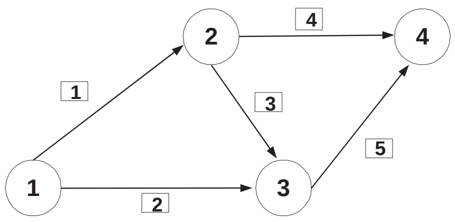

Figure 7. Network 2

Table 8. Parameters for network 2

Activity	**1**	**2**	**3**	**4**	**5**
Origin	1	1	2	2	3
Target	2	3	3	4	4
λ	0.02	0.03	0.04	0.024	0.025
$\mathbf{x}_{\min}$	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\max}$	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_4^*\} = \{1.0, 1.0, 1.5\}$ with an expected cost of 304.62.

B.3. Network 3

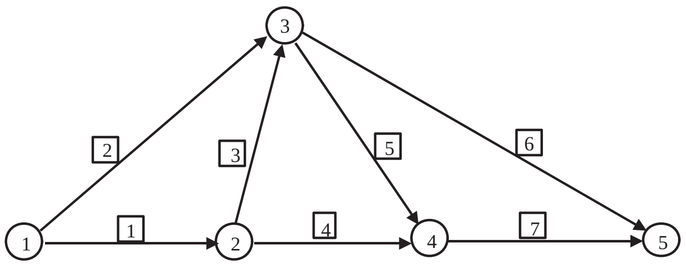

Figure 8. Network 3

This network has seven activities (see Fig. 8). The due date is $T = 66$ and the tardiness cost $c_L = 5$. The remaining parameters are given in Table 9. The PERT expected duration for this network is 62.9.

Table 9. Parameters for network 3

Activity	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Origin	1	1	2	2	3	3	4
Target	2	3	3	4	4	5	5
λ	0.08	0.06	0.09	0.05	0.07	0.03	0.04
$\mathbf{x}_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_4^*, x_6^*\} = \{1.25, 1.0, 1.5, 1.5\}$, with the expected cost of 193.99.

B.4. Network 4

This network has 9 activities. For this network, $T = 105$ and $c_L = 4$. See Table 10 for the remainig parameters. The PERT expected duration for this network is 100.

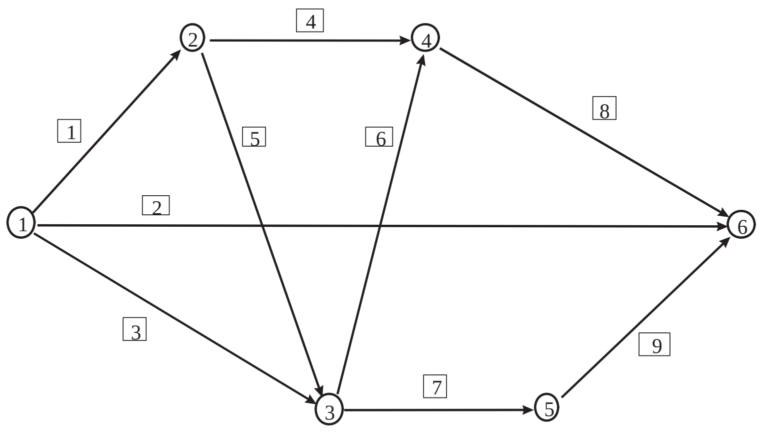

Figure 9. Network 4

Table 10. Parameters for network 4

Activity	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Origin	1	1	1	2	2	3	3	4	5
Target	2	6	3	4	3	4	5	6	6
λ	0.04	0.01	0.07	0.035	0.05	0.06	0.045	0.06	0.039
$\mathbf{x}_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_4^*, x_7^*, x_9^*\} = \{1.0, 1.5, 0.5, 1.0, 1.0, 1.0\}$, with the expected cost of 399.52.

B.5. Network 5

Network 5 (see Fig. 10) is of larger dimension (11 activities). For this network, $T = 28$ (due date) and $c_L = 8$ (tardiness cost). The remaining parameters are presented in Table 11. The PERT expected duration for this network is 26.67.

Table 11. Parameters for network 5

Activity	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Origin	1	1	1	2	3	2	3	4	3	5	4
Target	2	3	4	3	4	5	5	5	6	6	6
λ	0.1	0.09	0.4	0.2	0.3	0.08	0.4	0.2	0.1	0.3	0.3
$\mathbf{x}_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

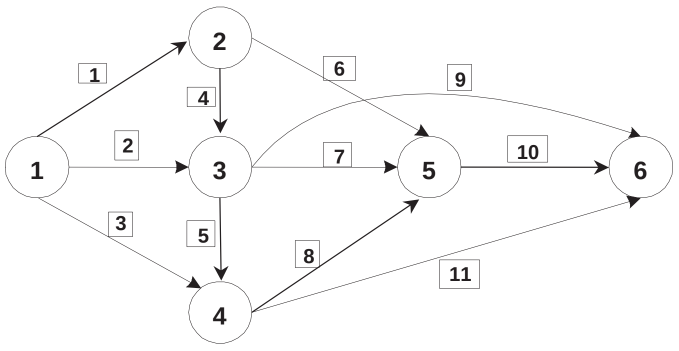

Figure 10. Network 5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_6^*, x_7^*, x_9^*, x_{11}^*\} = \{1.25, 1.0, 0.5, 1.0, 0.5, 1.5, 1.0\}$, with the expected cost of 130.23.

B.6. Network 6

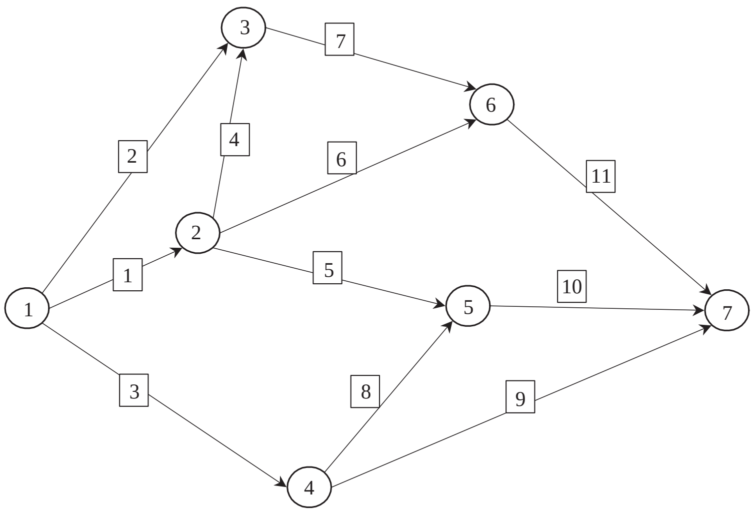

Figure 11. Network 6

This network has 11 activities. The due date is $T = 65$ and the cost of tardiness is $c_L = 5$. See Table 12 for the information. The PERT expected duration for this network is 62.08.

Table 12. Parameters for network 6

Activity	1	2	3	4	5	6	7	8	9	10	11
Origin	1	1	1	2	2	2	3	4	4	5	6
Target	2	3	4	3	5	6	6	5	7	7	7
λ	0.1	0.12	0.05	0.08	0.2	0.04	0.03	0.04	0.024	0.15	0.16
$x_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$x_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_5^*, x_6^*, x_8^*, x_9^*, x_{10}^*\} = \{1.25, 0.5, 1.5, 0.5, 1.0, 1.0, 1.5, 1.0\}$, with the expected cost of 272.30.

B.7. Network 7

Network 7 has one more activity than the last one (see Fig. 12), and different topology. The due date is $T = 47$ and the tardiness cost $c_L = 4$. The remaining parameters are presented in Table 13. The PERT expected duration for this network is 44.72.

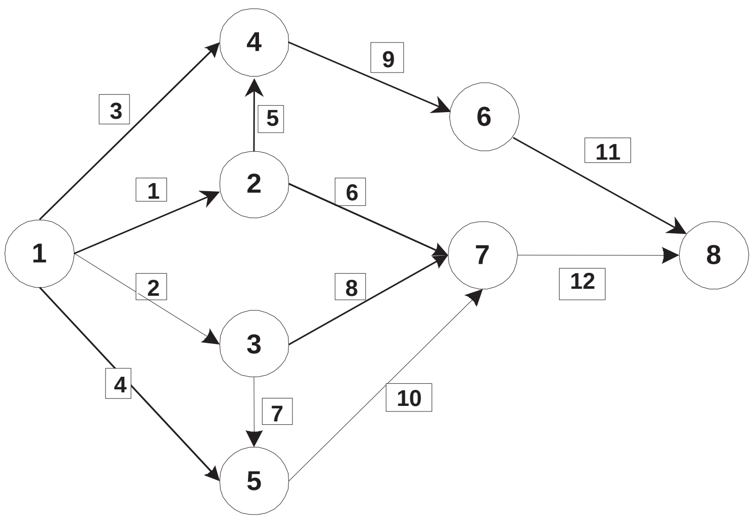

Figure 12. Network 7

Table 13. Parameters for network 7

Activity	1	2	3	4	5	6	7	8	9	10	11	12
Origin	1	1	1	1	2	2	3	3	4	5	6	7
Target	2	3	4	5	4	7	5	7	6	7	8	8
λ	0.1	0.09	0.08	0.1	0.09	0.08	0.1	0.09	0.08	0.1	0.09	0.1
$x_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$x_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_4^*, x_6^*, x_7^*, x_8^*, x_{10}^*, x_{12}^*\} = \{1.25, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.5\}$, with the expected cost of 182.91.

B.8. Network 8

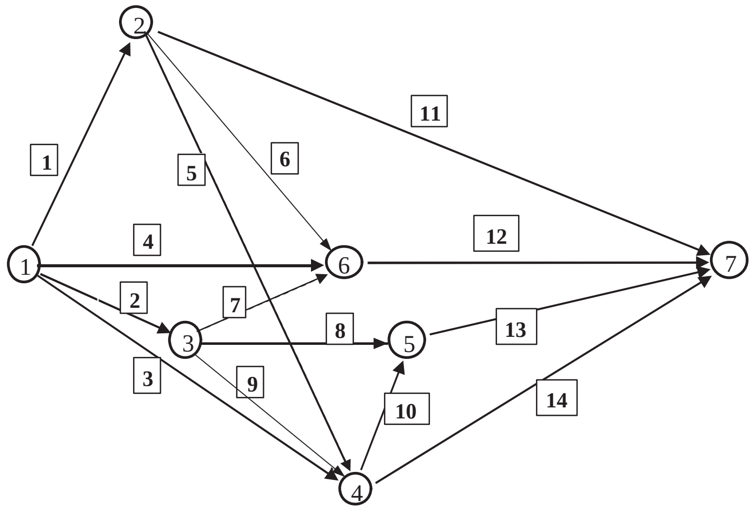

Figure 13. Network 8

This network has 14 activities. T is 37 and c_L is 3. The remaining parameters are presented in Table 14. The PERT expected duration for this network is 35.5.

Table 14. Parameters for network 8

Activity	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Origin	1	1	1	1	2	2	3	3	3	4	2	6	5	4
Target	2	3	4	6	4	6	6	5	4	5	7	7	7	7
λ	0.2	0.25	0.16	0.2	0.1	0.16	0.5	0.25	0.2	0.08	0.09	0.1	0.125	0.1
$x_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$x_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_4^*, x_6^*, x_7^*, x_8^*, x_9^*, x_{11}^*, x_{12}^*, x_{14}^*\} = \{1.5, 1.5, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0\}$, with the expected cost of 120.34.

B.9. Network 9

Network 9 has the same number of activities as the previous one (14 activities). Its due date is 188 and its tardiness cost is 6. The other parameters can be seen in Table 15. The PERT expected duration for this network is 178.57.

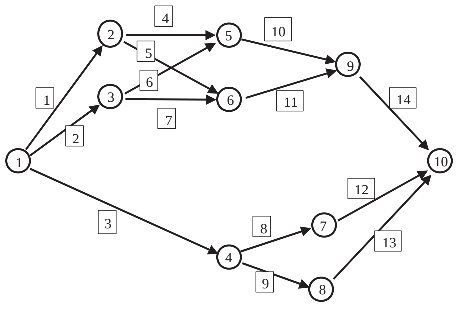

Figure 14. Network 9

Table 15. Parameters for network 9

Activity	1	2	3	4	5	6	7
Origin	1	1	1	2	2	3	3
Target	2	3	4	5	6	5	6
λ	0.02	0.03	0.04	0.025	0.035	0.045	0.05
$x_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$x_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5
Activity	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Origin	4	4	5	6	7	8	9
Target	7	8	9	9	10	10	10
λ	0.06	0.03	0.02	0.015	0.02	0.025	0.03
$x_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$x_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_5^*, x_6^*, x_7^*, x_8^*, x_9^*, x_{11}^*, x_{12}^*, x_{13}^*\} = \{1.0, 0.5, 1.5, 0.5, 0.5, 0.5, 0.5, 1.5, 0.5, 0.5, 1.5\}$, with the expected cost of 1275.97.

B.10. Network 10

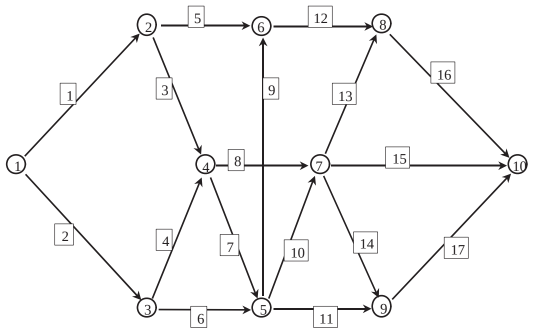

Figure 15. Network 10

Network 10 has 17 activities. For this network, $T = 49$ and $c_L = 7$. The remaining parameters are shown in Table 16. The PERT expected duration for this network is 44.98.

Table 16. Parameters for network 10

Activity	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Origin	1	1	2	3	2	3	4	4	5
Target	2	3	4	4	6	5	5	7	6
λ	0.167	0.1	0.2	0.1	0.25	0.2	0.1	0.333	0.333
$\mathbf{x}_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
Activity	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	
Origin	5	5	6	7	7	7	8	9	
Target	7	9	8	8	9	10	10	10	
λ	0.25	0.5	0.167	0.143	0.5	0.125	0.167	0.11	
$\mathbf{x}_{\min}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	
$\mathbf{x}_{\max}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	

The solution for this network was $\{x_1^*, x_2^*, x_4^*, x_5^*, x_6^*, x_8^*, x_{10}^*, x_{11}^*, x_{13}^*, x_{14}^*, x_{15}^*, x_{17}^*\} = \{1.25, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 1.5\}$, with the expected cost of 141.34.

B.11. Network 11

Network 11 can be seen in Fig. 16. It is composed of 18 activities. Here, $T = 110$ and $c_L = 10$. The remaining parameters are presented in Table 17. The PERT expected duration for this network is 106.11.

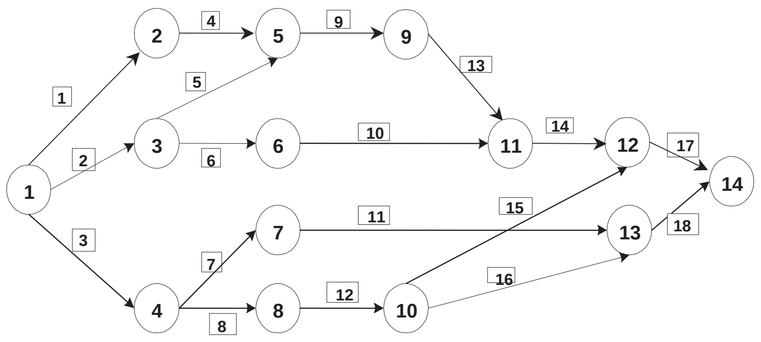

Figure 16. Network 11

Table 17. Parameters for network 11

Activity	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Origin	1	1	1	2	3	3	4	4	5
Target	2	3	4	5	5	6	7	8	9
λ	0.06	0.04	0.1	0.07	0.08	0.04	0.08	0.2	0.07
$\mathbf{x}_{\mathrm{min}}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\mathrm{max}}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
Activity	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Origin	6	7	8	9	11	10	10	12	13
Target	11	13	10	11	12	12	13	14	14
λ	0.05	0.08	0.07	0.09	0.09	0.05	0.09	0.04	0.06
$\mathbf{x}_{\mathrm{min}}$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$\mathbf{x}_{\mathrm{max}}$	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

The solution for this network was $\{x_1^*, x_2^*, x_3^*, x_5^*, x_6^*, x_7^*, x_8^*, x_{10}^*, x_{11}^*, x_{12}^*, x_{15}^*, x_{16}^*, x_{18}^*\} = \{1.25, 1.5, 1.5, 1.5, 1.0, 1.0, 1.0, 1.0, 1.5, 1.5, 1.0, 1.5, 1.0\}$, with the expected cost of 357.86.