Chapter 11: PERT for Project Planning and Scheduling

PERT, the Project Evaluation and Review Technique, is a network-based aid for planning and scheduling the many interrelated tasks in a large and complex project. It was developed during the design and construction of the Polaris submarine in the USA in the 1950s, which was one of the most complex tasks ever attempted at the time. Nowadays PERT techniques are routinely used in any large project such as software development, building construction, etc. Supporting software such as Microsoft Project, among others, is readily available. It may seem odd that PERT appears in a book on optimization, but it is frequently necessary to optimize time and resource constrained systems, and the basic ideas of PERT help to organize such an optimization.

PERT uses a network representation to capture the precedence or parallel relationships among the tasks in the project. As an example of a precedence relationship, the frame of a house must first be constructed before the roof can go on. On the other hand, some activities can happen in parallel: the electrical system can be installed by one crew at the same time as the plumbing system is installed by a second crew.

The PERT formalism has these elements and rules:

- Directed arcs represent *activities*, each of which has a specified *duration*. This is the "activity on arc" formalism; there is also a less-common "activity on node" formalism. Note that activities are considered to be uninterruptible once started.
- Nodes are *events* or points in time.
- The activities (arcs) leaving a node cannot begin until all of the activities (arcs) entering a node are completed. This is how precedence is shown. You can also think of the node as enforcing a rendezvous: no-one can leave until everyone has arrived.
- There is a single starting node which has only outflow arcs, and a single ending node that has only inflow arcs.
- There are no cycles in the network. You can see the difficulty here. If an outflow activity cannot begin until all of the inflow activities have been completed, a cycle means that the system can never get started!

Consider the example given in Figure 1. Perhaps the pouring of the concrete foundation (activity A-B), happens at the same time as the pre-assembly of the roof trusses (activity A-D). However, the finalization of the roof (activity D-E), cannot begin until both A-D and B-D (assembly of the house frame), are done. Of course B-D cannot start until the concrete foundation has been poured (A-B). All of this precedence and parallelism information is neatly captured in the PERT diagram.

There are two major questions about any project:

- What is the shortest time for completion of the project?

- Which activities *must* be completed on time in order for the project to finish in the shortest possible time? These activities constitute the *critical path* through the PERT diagram.

The process of finding the critical path answers the first question as well as the second. Of course we need to know how long each individual activity will take in order to answer these questions. This is why the arcs in Figure 1 are labelled with numbers: the numbers show the amount of time that each activity is expected to take (in days, let's say).

The critical path is of great interest to project managers. The activities on the critical path are the ones which absolutely must be done on time in order for the whole project to complete on time. If any of the activities on the critical path are late, then the entire project will finish late! For this reason, the critical path activities receive the greatest attention from management. The non-critical activities have some leeway to be late without affecting the overall project completion time.

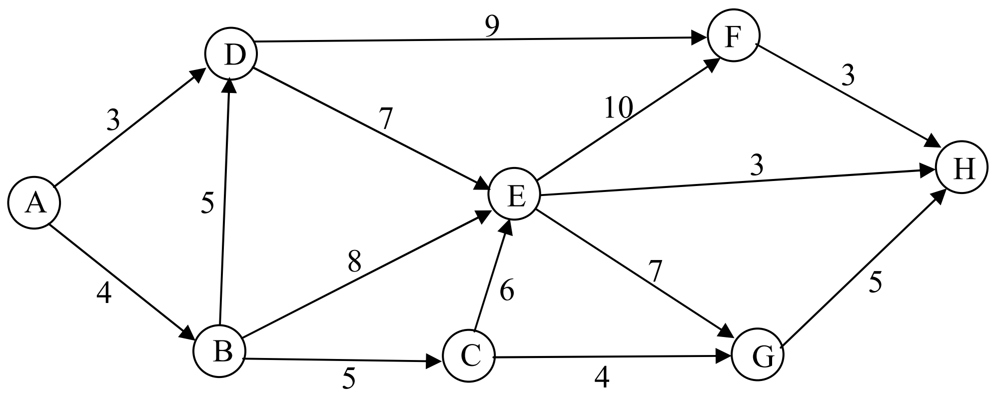

Figure 1: An example of a PERT diagram.

The following steps find the critical path and calculate other useful information about the project.

Step 1. Make a forward pass through the diagram, calculating the *earliest time* (T_E) for each event (node). In other words, what is the earliest time at which all of the activities entering a node will have finished? To find T_E, look at all of the activities which enter a node. T_E is the <u>latest</u> of the arrival times for entering arcs, i.e. T_E = max [(T_E of node at tail of arc) + (arc duration)] over all of the entering arcs. By definition, T_E of the starting node is zero.

Step 2. Make a backward pass through the diagram, calculating the *latest time* (T_L) for each event (node). In other words, what is the latest time that the outflow activities can begin without causing a late arrival at the next node for one of those activities? To find T_L, look at all of the activities which exit a node. T_L is the <u>earliest</u> of the leaving times for the exiting arcs, i.e. T_L = min [(T_L of node at head of arc) − (arc duration)] over all of the exiting arcs. By definition, the T_L of the ending node equals it's T_E.

Step 3. Calculate the *node slack time* (S_N) for each node (event). This is the amount of time by which an event could be adjusted later than its T_E without causing problems downstream. $S_N = T_L - T_E$ for each node.

Step 4. Calculate the *total arc slack time* (S_A) for each arc (activity). This is the amount of time by which an activity could be adjusted later than the T_E of the node at its tail with causing problems later. S_A = (T_L of node at arc tail) − (T_E of node at arc head) − (arc duration).

Step 5. The critical path connects the nodes at which $S_N = 0$ via the arcs at which $S_A = 0$.

It should be no surprise that the critical path connects the nodes and arcs which have no slack. If there is slack, then the activity does not need to be done on time, which is exactly the opposite definition of the critical path!

As an example, let's find the critical path for the PERT diagram in Figure 1. Note that there is an implied order in which the T_E's can be calculated in Step 1. For example, the T_E of node D cannot be found until the T_E of node B is known. The starting node in Figure 1 is node A, and by definition the T_E of the starting node is 0. To calculate T_E at a node, we need to know the T_E of the node at the tail of every entering arc, so we can next only calculate the T_E of node B. This is simple since there is only one inflow arc, from node A, so T_E(B) = T_E(A) + (duration of A-B) = 0 + 4 = 4. The complete set of T_E calculations follows:

T_E(A)	= Starting node	= 0
T_E(B)	= T_E(A)+(duration A-B)	= 0+4 = 4
T_E(D)	= max{T_E(A)+(duration A-D), T_E(B)+(duration B-D)}	= max{0+4, 4+5} = 9
T_E(C)	= T_E(B)+(duration B-C)	= 4+5 = 9
T_E(E)	= max{T_E(D)+(duration D-E), T_E(B)+(duration B-E), T_E(C)+(duration C-E)}	= max{9+7, 4+8, 9+6} = 16
T_E(F)	= max{T_E(D)+(duration D-F), T_E(E)+(duration E-F)}	= max{9+9, 16+10} = 26
T_E(G)	= max{T_E(E)+(duration E-G), T_E(C)+(duration C-G)}	= max{16+7, 9+4} = 23
T_E(H)	= max{T_E(F)+(duration F-H), T_E(E)+(duration E-H), T_E(G)+(duration G-H)}	= max{26+3, 16+3, 23+5} = 29

The shortest time in which the project can be completed is now known: it is the same as the T_E of the ending node, node H, i.e. 29 days. But we still need to complete the remaining 4 steps of the algorithm to positively identify the critical path.

The backwards pass in Step 2 begins with the ending node H. By definition, the T_L of the ending node is equal to its T_E so T_E(H) = 29. This makes sense: otherwise the whole project would be pushed later! As for the forward pass, there is an implied order in which the node T_L values can be found, determined by the outflow arcs for which T_L is known. The complete set of calculations follows.

T_L(H)	= ending node = T_E(H)	= 29

$T_L(F)$	$= T_L(H) - (\text{duration F-H})$	$= 29-3 = 26$
$T_L(G)$	$= T_L(H) - (\text{duration G-H})$	$= 29-5 = 24$
$T_L(E)$	$= \min\{T_L(F) - (\text{duration E-F}),\ T_L(H) - (\text{duration E-H}),\ T_L(G) - (\text{duration E-G})\}$	$= \min\{26-10,\ 29-3,\ 24-7\} = 16$
$T_L(C)$	$= \min\{T_L(E) - (\text{duration C-E}),\ T_L(G) - (\text{duration C-G})\}$	$= \min\{16-6, 24-4\} = 10$
$T_L(D)$	$= \min\{T_L(F) - (\text{duration D-F}),\ T_L(E) - (\text{duration D-E})\}$	$= \min\{26-9, 16-7\} = 9$
$T_L(B)$	$= \min\{T_L(D) - (\text{duration B-D}),\ T_L(E) - (\text{duration B-E}),\ T_L(C) - (\text{duration B-C})\}$	$= \min\{9-5,\ 16-8,\ 10-5\} = 4$
$T_L(A)$	$= \min\{T_L(D) - (\text{duration A-D}),\ T_L(B) - (\text{duration A-B})\}$	$= \min\{9-3, 4-4\} = 0$

It's no surprise that T_L of the starting node is 0. If it wasn't it would mean that we could start the whole project late and yet still finish on time!

In Step 3 we find the *node slack time* (S_N) for each node (event) as shown below:

Node	A	B	C	D	E	F	G	H
S_N	0	0	1	0	0	0	1	0

This small PERT diagram is quite tight: only two of the nodes have nonzero slack. Larger diagrams with many parallel activities often have much more slack in the nodes.

In Step 4 we find *total arc slack time* (S_A) for each arc (activity) as shown below.

Arc	AB	AD	BC	BD	BE	CE	CG	DE	DF	EF	EG	EH	FH	GH
S_A	0	6	1	0	4	1	11	0	8	0	1	10	0	1

The arcs have quite a bit more slack time than the nodes in this small example. Later we will see how this slack can be put to good use in adjusting resource demands.

Finally, in Step 5, we find the critical path by linking the nodes having no slack via the arcs having no slack. Figure 2 shows the critical path for our example PERT diagram. The nodes and arcs having no slack are shown in boldface. If you've been watching closely, you might have noticed that the critical path through the PERT diagram is actually the *longest* path through the network. If you only needed the critical path and its length, it's easy to convert Dijkstra's shortest route algorithm to a longest route algorithm to find it.

Sometimes a situation arises in which one activity must precede two *different* events. How can this happen when a single arc can terminate only at a single event node? The solution lies in the use of *dummy arcs* which have a duration of zero. Dummy arcs are normally shown as dashed lines, as in the diagram fragment in Figure 3, in which activity A-B is the immediate predecessor of both event C and event D.

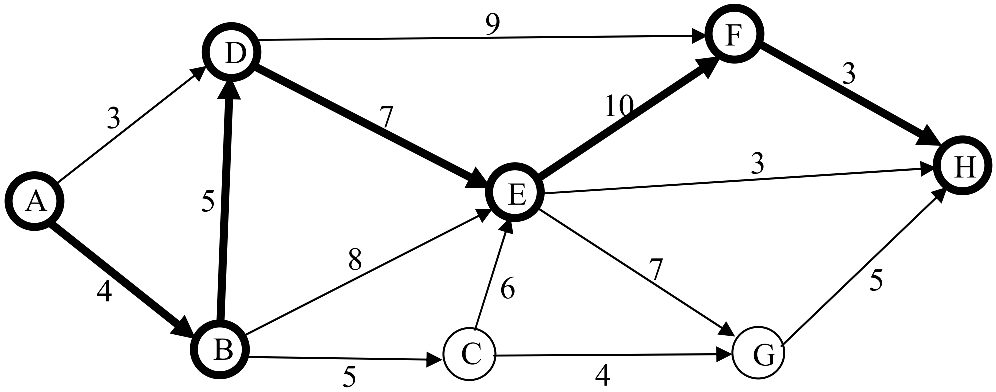

Figure 2: The critical path.

The determination of the critical path and hence of the duration of the entire project obviously depends very heavily on accurately assessing the duration of each individual activity. How is this done in practice? There are two main approaches: direct estimates, and the three-estimate method.

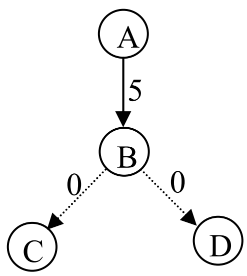

Figure 3: Dummy arcs.

In direct estimation, a single number is stated directly, perhaps based on long experience with similar projects. In the three-estimate approach, 3 estimates with specific properties are used in a weighted average. m is the most likely value, obtained in a manner similar to the direct estimate. a is an optimistic estimate, i.e. the time needed if everything goes just right (the weather is good, building materials arrive on time, crew is on time). Finally b is the pessimistic estimate, i.e. the time needed if everything goes wrong (it's raining, materials are late, crew books off sick, etc.). Given these three estimates, the final duration is set as $(a+4m+b)/6$.

Probabilistic PERT

Estimating is an inexact art, so we expect that our initial duration estimates have some error in them. What we would really like to know is how much this error is going to affect our estimate of the total project duration. Fortunately, with a few assumptions and very little extra work we can make some judgments about the likely amount of variation in the total project time. To do this we start with the three-estimate approach to estimating the activity durations. Next we make the following assumptions:

- The activity durations fit a Beta distribution.
- The range from a to b in the three-estimate approach covers 6 standard deviations.
- The activity durations are statistically independent.
- The critical path now means the path that has the longest *expected value* of total project time.
- The overall project duration has a normal distribution.

Given these assumptions, the expected value of each activity duration is given in exactly the same way as for the three-estimate approach: $(a+4m+b)/6$. The variance of each activity duration in this model is $[(b-a)/6]^2$.

Now the expected value of the total project duration is the sum of the expected activity durations along the critical path, which is found in the usual way. Finally the payoff: the variance of the total project duration is the sum of the variances of the activity durations for the activities in the critical path. This is very useful information for the managers: now they have some idea of how much the total project time might vary.

Resource Leveling

The method shown previously for determining the minimum amount of time to complete a project assumes that you have all the resources that you need. For example, it assumes that you have several crews to carry out activities simultaneously where the activities run in parallel in the PERT diagram. But suppose two activities should be done at the same time, but both require the use of the bulldozer, and you only have one bulldozer? If this happens the project may take longer to complete because the two activities must now be done in sequence, instead of in parallel. I say *may* take longer, because there may be slack in the two arcs which allows them to be shifted apart in time so that they no longer compete for the bulldozer, without lengthening the overall project time. As you see, the time and resources needed to complete a project interact in fundamental ways.

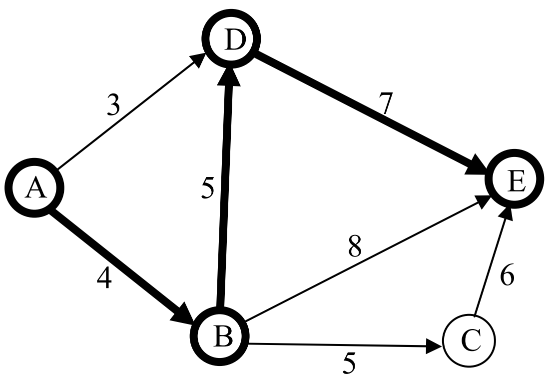

Figure 4: A small part of Figure 1.

We will explore this concept of using the slack time in the activities to move resource demands around. To keep things manageable, we will use a small piece of the PERT diagram from Figure 1, as shown in Figure 4.

There are actually two kinds of slack associated with an arc, e.g. arc B-C in Figure 4. The *total arc slack*, as we have seem before, is given in this case by $T_L(C) - T_E(B) -$ (duration B-C) $= 10 - 4 - 5 = 1$. The *free arc slack* is given in this case by $T_E(C) - T_E(B) -$ (duration B-C) $= 9 - 4 - 5 = 0$.

The total arc slack is the maximum amount of slack time available. If you start an activity later than the T_E of the node at its tail, by an amount equal to the total arc slack, this pushes the node at the head of the arc to its latest time. This has consequences downstream: activities later than the head node event time may now not be able to use their total slack. It's always OK to use up the free arc slack because this has no downstream consequences. However if you wish to push an activity later by any amount of time between the free slack and the total slack, you must be careful that there are no negative downstream consequences. In other words, you can schedule actvity B-C to begin at any time between $T_E(B)$ and $T_E(B)$ + (free slack B-C) without fear. However if you want to schedule activity B-C to start sometime between $T_E(B)$ + (free slack B-C) and $T_E(B)$ + (total slack B-C), then there may be downstream consequences because node C will be pushed later than $T_E(C)$.

The table below summarizes the total and free slack for all of the non-critical activities. Why do we list only the non-critical activities? Because the slack, both total and free, is always zero for the critical activities, by definition.

Activity	*Total slack*	*Free slack*
A-D	6	6
B-E	4	4
B-C	1	0
C-E	1	1

Now suppose that each activity, including the critical activities, must be staffed by certain numbers of employees, as shown in the following table:

Activity	*Employees needed*
A-B	3
B-D	2
D-E	3
A-D	4
B-E	3
B-C	2
C-E	2

Now we will try a few simple scenarios to see the effect that using the slack time in different ways has on the total number of employees needed at any point in the project. Remember that each activity can be scheduled individually within the parameters of its free and total slack. However, to keep things simple, we will try these two scenarios: all activities scheduled as early

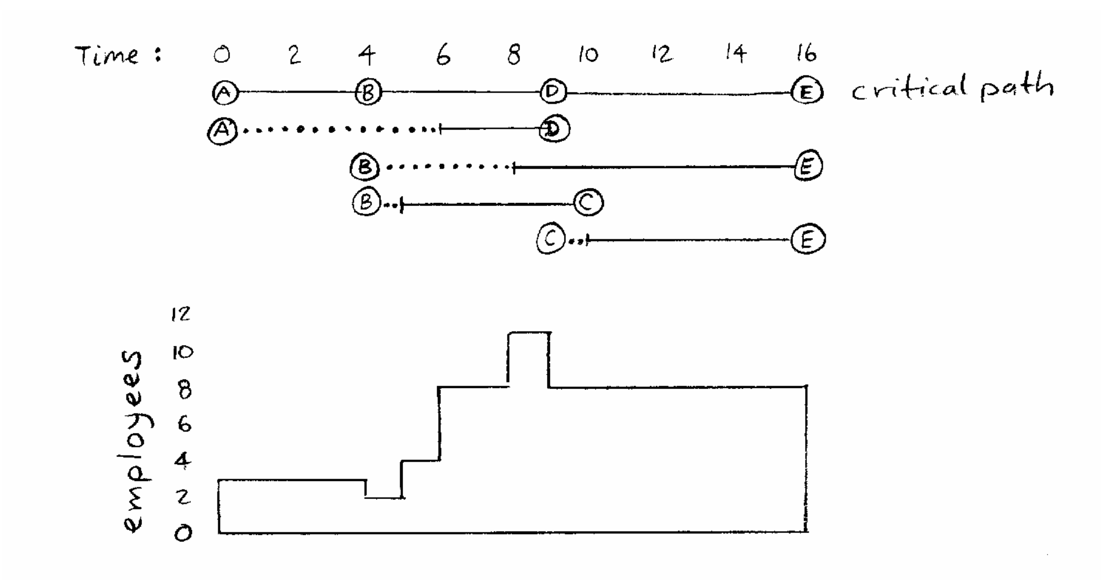

Figure 5: Every activity scheduled as *late* as possible.

as possible, and all activities scheduled as late as possible.

The scenario in which all activities are scheduled as late as possible is shown in Figure 5. The top part of Figure 5 is a Gantt chart, which shows the timing of each activity as a solid line. For example, activity A-D starts at time 6 and runs to time 9. The dashed extensions on the A-D line indicate the range of other possible starting times for A-D. The length of the dotted section is the amount of free slack available to the activity. There is no slack for the critical path activities.

The Gantt chart shows when each activity takes place, and we know how many employees are needed for each activity, so we can calculate the number of employees needed every day. For example, between time 8 and time 9 there are four activities taking place simultaneously: activity B-D (2 employees), activity A-D (4 employees), activity B-E (3 employees) and activity B-C (2 employees) for a total of 11 employees, the peak demand over the whole 16 day period. The demand for employees is charted in the bottom part of Figure 5.

The resource demands in Figure 5 are very uneven. Few employees are needed over the first 6 days, but then the demand shoots up quickly to a peak of 11 employees at time 8 to 9 before settling back to a need for 8 employees. This is a real problem if the company is operating with 8-person crews. For the first several days, most of the employees have nothing to do except drink coffee, while at time 8-9 the company will need to hire extra people or to pay overtime. For reasons such as these, managers usually want to level out the demand for resources, and this is referred to as *resource leveling*. What we'd like to do in Figure 5 is to somehow move some of the resource demand from the later part of the project to the earlier part of the project. This can be done by rescheduling some of the activities which have slack.

Figure 6 shows a second scenario, this time with all of the activities scheduled as early as possible. This scenario eliminates the peak of demand for employees that was seen in Figure 5. In fact, this is a schedule which can be accomplished without overtime using an 8-employee crew, so it is much preferred by management.

Note that many other scheduling scenarios are possible, using any combination of the possible starting times for the non-critical activities. For example, in Figure 6 we could shift activity C-E one day later in order to remove part of the 3-day period that requires 8 employees. Now it might be cheaper to run the project with a 7-employee crew and pay two days of overtime. Note that you must be careful with events such as C which have node slack. As you see in both Gantt charts, the ending C in activity B-C is offset from the starting C in activity C-E, indicating the possible ending and starting times of the two activities. However, if activity B-C uses its total slack, then C is pushed to its latest time, which means that activity C-E cannot start any earlier than that.

Common questions related to resource leveling include: Can I complete the project on time with a limited amount of resource (e.g. 8 employees)? What is the minimum amount of resource needed to complete this project on time? Given that I have only a set amount of resource, what is the minimum amount of time in which the project can be completed? Note that this last question takes the resource limit as fixed, and instead allows the project to take more time to complete. As an example, consider a project in which there is exactly one unit of resource (e.g. one employee) and each activity requires 1 employee. Now how long will it take the complete

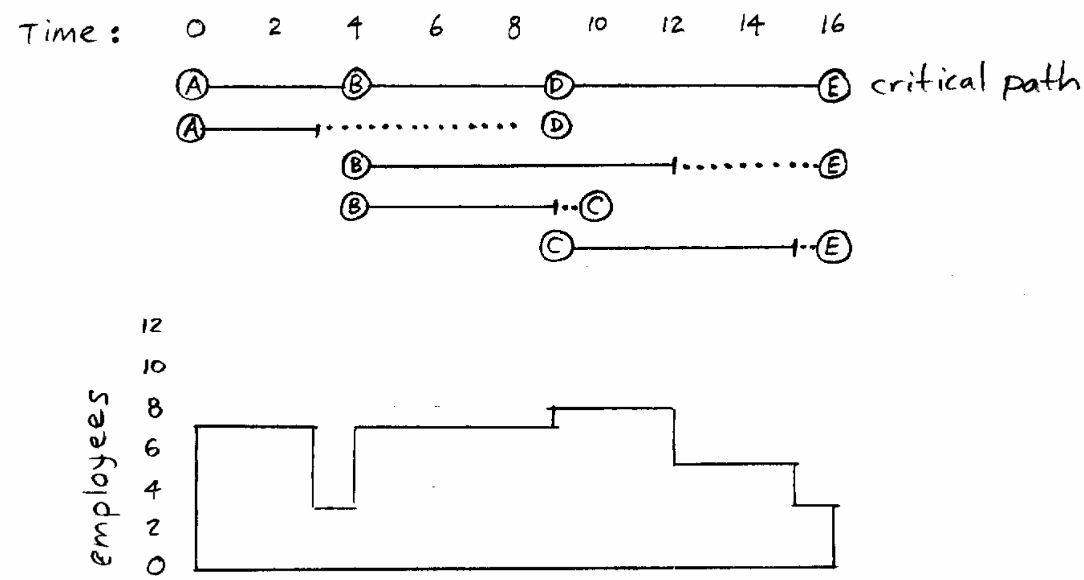

Figure 6: Every activity scheduled as *early* as possible.

the project? The answer is simple: since there is no possibility of doing any of the activities in parallel, the time needed to complete the project is the simple sum of *all* of the activity durations!

We have considered a simple case in which there is only a single type of resource (employees). In more complex projects there can be numerous resource types, e.g. bulldozers, graders, cement mixers, electricians, plumbers, dump trucks, front-end loaders, excavators, etc. Now you will have numerous interrelated resource leveling problems since a single activity will likely involve some subset of several of these resources.

Leveling resources to minimize the maximum amount of resource needed at any point in time is a very difficult combinatorially explosive problem, so it is solved via heuristics. Every commercial vendor of project planning software uses different proprietary heuristics, which means that the same resource leveling problem submitted to several different software packages will probably give you several different solutions. One heuristic I am aware of was developed by Pierre Paulin for the Ph.D. thesis in the Electronics Department here at Carleton University. His problem involved implementing a series of instructions on a chip which had a limited number of devices such as adders, buffers etc., so it was a classic multi-resource leveling problem. He developed the "force-directed" method which first works out the average rate of resource usage, and then conceptually adds "springs" to pull the valleys in the resource diagram up towards the average level and pull down the peaks. You then seek a kind of force balance between the "spring constants" which is achieved when the peaks and valleys are smoothed out as much as possible.

Time-Cost Tradeoffs

In real applications it is sometimes possible to reduce the amount of time that an individual activity takes by paying more, e.g. for overtime, for faster computers, for courier delivery, etc. If the project takes longer to complete than you have available, then you will have to spend some money to speed things up. But which activities should you speed up? Obviously those on the critical path, but if you speed up one activity on the critical path, the critical path itself may change to some other set of activities. And which activity on the critical path should you speed up, since some may be more expensive than others? And how much should you speed it up?

This is a complicated problem. You really can't solve it by looking at the activities one at a time – you need to consider them all simultaneously. Fortunately there is a nifty optimization formulation which addresses this issue. Here we will show a linear programming formulation, but the extension to nonlinear programming should be obvious.

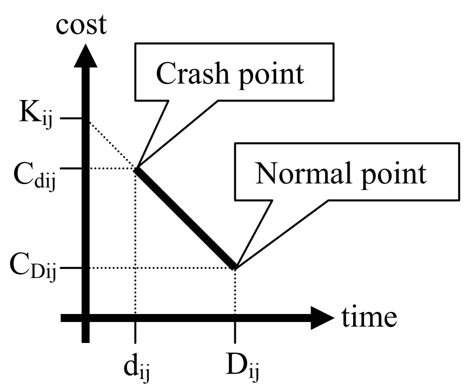

Figure 7: Time-cost tradeoff for activity ij.

The fundamental element in the linear programming formulation is a simple model showing how time and money trade off for each activity, as shown in Figure 7. In Figure 7, the "normal point" shows the length of time (D_{ij}, large D for large duration) and cost (C_{Dij}) when activity *ij* is run in the normal manner. The "crash point" shows the length of time (d_{ij}, small d for small duration) and cost (C_{dij}) when the activity is "crashed", or sped up to the maximum extent possible. Note that the cost for the activity goes up as the duration goes down – money is exchanged for time. We assume that any amount of speedup between D_{ij} and d_{ij} is possible. K_{ij} is just the value on the cost axis reached by extending the time-cost tradeoff line for the activity; it makes the formulation a little simpler.

Now every activity can have a duration somewhere d_{ij} and D_{ij}. The unknown duration for activity ij is represented by the variable x_{ij}. We also define the constant $C_{ij} = (C_{dij} - C_{Dij})/(D_{ij} - d_{ij})$, which is just the negative of the slope of the time-cost tradeoff line. C_{ij} allows us to compactly express the cost of any intermediate value on the time-cost tradeoff line as $K_{ij} - C_{ij}x_{ij}$, where $d_{ij} \leq x_{ij} \leq D_{ij}$.

The objective is to minimize the total costs of speeding up activities while meeting a specified deadline, i.e. to minimize $\sum_{ij}(K_{ij} - C_{ij}x_{ij})$. However we can always drop constants which appear in objective functions, so it can be rewritten as *minimize* $\sum_{ij}(-C_{ij}x_{ij})$, which is the same as *maximize* $\sum_{ij}C_{ij}x_{ij}$.

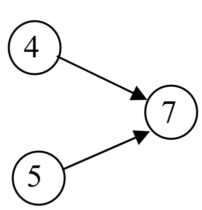

Figure 8: Capturing precedence.

We also need to capture the precedence relationships in the PERT diagram. This seems difficult because we don't know the event times, let alone the durations of the activities. We define the variables y_k to represent the unknown event times. Now how do we capture the precedence relationships? The main idea is to make sure that no activity leaving an event node begins before all of the activities entering the event node have terminated. For example, consider Figure 8. How do we make sure that y_7, the event time for node 7, is later than the latest arrival time of activities 4-7 and 5-7? The behaviour we want is $y_7 = \max\{y_4+x_{47}, y_5+x_{57}\}$. This is easily achieved by using two inequalities: y_4+x_{47}

$\leq y_7$ and $y_5+x_{57} \leq y_7$.

Now we are almost ready to set up the entire linear program. For the starting event, define $y_1 = 0$ and for the ending event, define $y_n \leq T$. Note that T, the project deadline, should be less than the shortest project time if every activity is run in normal time, otherwise the solution is obvious: just run every activity in normal time.

Now we can summarize the entire formulation:

maximize $\sum_{ij} C_{ij} x_{ij}$

subject to: $d_{ij} \leq x_{ij} \leq D_{ij}$ for all activities (arcs)

$y_i + x_{ij} - y_j \leq 0$ for all events (nodes)

$y_1 = 0$

$y_n \leq T$

$x_{ij}, y_k \geq 0$

The minimum cost for speeding up the project is not given directly by the optimum value of the objective function, but it is easy to calculate it once the x_{ij} values are known.

As you can see, the same general formulation can be used when the time-cost tradeoff curve is nonlinear, with the exception that the objective function will not be linear. I have used a nonlinear version of this formulation in a transistor-sizing optimization. The idea is to keep the signal propagation time below a certain limit. The propagation time is given by the critical path through the transistor network. However the propagation time through an individual transistor depends on the sizes of the transistor and its neighbours. "Cost" in this instance is transistor size, where larger transistors are faster. The idea is to make sure that the signal propagates in a time below the stated limit while keeping the total transistor area as small as possible. The objective function in this case was nonlinear, as were the constraints defining the x_{ij}.