Concurrency Detection on Finish-to-Start Activity Precedence Networks

Rui Moutinho[1] and Anabela Tereso[1],

[1] Department of Production and Systems Engineering
Algoritmi Centre
University of Minho, 4800-058 Guimarães, Portugal
rumout@gmail.com, anabelat@dps.uminho.pt

Abstract. We explore the finish-to-start precedence relations of project activities used in scheduling problems. From these relations, we devise a method to identify groups of activities that could execute concurrently, i.e. activities in the same group can all execute in parallel.
The method derives a new set of relations to describe the concurrency. Then, it is represented by an undirected graph and the maximal cliques problem identifies the groups.
We provide a running example with a project from our previous studies in resource constrained project cost minimization together with an example application on the concurrency detection method: the evaluation of the resource stress.

Keywords: Project Scheduling; Finish-to-Start Precedence Relations; Concurrency; Maximal Clique

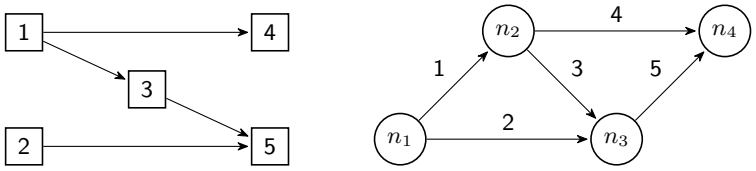
1 Introduction and Literature Review

Among the project scheduling problems, the precedence relation of the project activities is one of the most important and transversal aspects [4]. This paper addresses the specific case of the finish-to-start precedence of activities.

With the finish-to-start precedences, any activity must start only after all of its predecessors have finished. Often, the precedence relations are represented with a graph, either on an activity-on-node (AoN) network or on an activity-on-arc (AoA) network. Figure 1 shows such representations for the same relations, shown by enumeration, where each ordered pair (a, b) means b starts after a finishes.

Both representations are directed acyclic graphs. In the AoN network, each node represents an activity with arcs connecting a predecessor to some successors. In the example, an arc is connecting 1 to 3 meaning 3 starts after 1 has finished. Also, both 3 and 2 connect to 5. Hence, 5 requires 2 and 3 to finish so it can start. On the other hand, in the AoA network, the activities are coded in the arcs while the nodes represent a state (time instants). Any activity starting from a node requires the finish of any other which reaches that node. For example, activity 5 starts at node n_3 and activities 2 and 3 end at n_3. Therefore, activity 5

Fig. 1. Graphical representations of a network with finish-to-start precedence relations (*represented below*). *On the left*, the activity-on-node network and *on the right*, the activity-on-arc network from the relations.



$$\{(1,3),\ (1,4),\ (2,5),\ (3,5)\}$$

requires both 2 and 3 to complete before it could start. On both representations it is easy to identify the sets of initial and final activities. Initial activities do not require the completion of any other, thus they are the initial nodes, i.e. nodes with in-degree equal to zero (no arcs arriving at them). Analogously, in the AoA, the initial activities are the ones starting from the initial node. The dual reasoning applies to the final activities: in AoN, they are the nodes with zero out-degree (final nodes) while in AoA they are the ones ending at the final node.

We will explore the finish-to-start precedence relations of a project and provide a method to identify groups of activities that could execute in parallel by means of another relation and its representation as an undirected graph. The method relies on the maximal cliques of such graphs where a clique is any of its complete subgraphs (with every pair of vertices connected). A maximal clique is a clique that does not allow the inclusion of more graph vertices. The problem of determining the maximal cliques is known in the literature and is NP-complete. Despite executing in exponential time, the method provided by Bron et al. (1973) is still the reference, in general [1]. In its basic form, the method traverses recursively the graph and enumerates the maximal cliques. It can be improved by using a pivotal vertex and its effectiveness relates to the strategy of electing such vertex. The research in ongoing [8] and for some cases, such as sparse graphs, there are algorithms executing in polynomial time [3].

The presented process for determining concurrent activities, as permissible by the precedence relations, was devised in order to provide a tool to construct additional experiments on a previous study on minimization of total cost in projects with multimodal activities and multiple resources constrained in maximum availability [5]. In this problem, the activities may use any amount of resource in a real interval ranging from minimum to maximum demands. Additionally, the total available quantity of resource is limited and renewable during the project execution. The initial experiments suggested the behavior of our algorithms might be different when applied on projects without the availability restriction. Moreover, the effect of such constraint should be the subject of subsequent work. In this regard, the concurrency detection provides both an hint

for the overall resource demand and the stress the availability constraint may impose.

2 Concurrency Detection Method

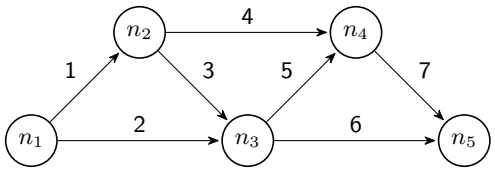
We start by denoting the finish-to-start relation with the symbol $<$, where $a < b$ reads b *must start after* a. It is natural to assume such relation as transitive, i.e.:

$$a < b \wedge b < c \Rightarrow a < c \ . \tag{1}$$

However, such property tends to be addressed implicitly as the relation usually is represented graphically.

Take the example of a project with seven activities where the finish-to-start relation is represented in Fig. 2 using an AoA network accompanied by its matrix form $\boldsymbol{M}_<$. The rows and columns of the matrix correspond to the activities and its entries, addressed by $(row \times column)$, are evaluated as:

$$\boldsymbol{M}_<(r,c) = \begin{cases} 1 & r < c \\ 0 & \neg(r < c) \end{cases} \ . \tag{2}$$



$\boldsymbol{M}_< =$

	1	2	3	4	5	6	7
1	0	0	1	1	0	0	0
2	0	0	0	0	1	1	0
3	0	0	0	0	1	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Fig. 2. Example AoA network (*on the left*) representing the finish-to-start precedence relations of seven activities described in the matrix (*on the right*).

It is easy to realize that activity 5 has to start only after activity 1 finishes, despite $1 < 5$ not being explicit. In fact, $1 < 3$ and $3 < 5$ from which $1 < 5$ is inferred.

In order to clarify the transitivity, we define the following relation.

Definition 1 (Transitive Finish-to-Start Relation).

$$a \ll b \Leftrightarrow a < b \vee \exists c : (a < c \wedge c \ll b) \ . \tag{3}$$

When $a \ll b$ then activity a *must* occur (and finish) before activity b starts, either this being an immediate successor or not. Naturally, if there are two activities with neither taking precedence over the other, then they may execute concurrently, as far as the precedence relation applies.

Definition 2 (Parallel Relation). *Given the transitive finish-to-start relation* $\ll$*, we define a* parallel relation*, denoted by* $\shortparallel$ *such:*

$$a \shortparallel b \Leftrightarrow (a \neq b \wedge \neg(a \ll b) \wedge \neg(b \ll a)) \quad . \tag{4}$$

Continuing with the example in Fig. 2, we evaluate the two matrices shown in Fig. 3: the transitive version $\boldsymbol{M}_{\ll}$ and the parallel relation $\boldsymbol{M}_{\shortparallel}$. These matrices are obtained similarly to the $\boldsymbol{M}_{<}$:

$$\boldsymbol{M}_{\ll}(r,c) = \begin{cases} 1 & r \ll c \\ 0 & \neg(r \ll c) \end{cases} \qquad \boldsymbol{M}_{\shortparallel}(r,c) = \begin{cases} 1 & r \shortparallel c \\ 0 & \neg(r \shortparallel c) \end{cases} \quad . \tag{5}$$

$\boldsymbol{M}_{\ll} =$

	1	2	3	4	5	6	7
1	0	0	1	1	1	1	1
2	0	0	0	0	1	1	1
3	0	0	0	0	1	1	1
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

$\boldsymbol{M}_{\shortparallel} =$

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	1	1	0
5	0	0	0	1	0	1	0
6	0	0	0	1	1	0	1
7	0	0	0	0	0	1	0

Fig. 3. Matrices for relations $\ll$ (*on the left*) and $\shortparallel$ (*on the right*) from the network represented in Fig. 2.

The parallel relation is not transitive, in general. For example, from $\boldsymbol{M}_{\shortparallel}$ at Fig. 3, $1 \shortparallel 2$ and $2 \shortparallel 3$ but $\neg(1 \shortparallel 3)$ because $1 < 3$. Still, we want to find sets of activities that can be concurrent with each other.

Definition 3 (Parallel Set). *Any* non-empty *set of project activities, say* P*, respecting the conditions:*

1. $\forall\, a, b \in P,\ a = b \vee a \shortparallel b$;
2. $\forall x \notin P,\ \exists y \in P,\ \neg(x \shortparallel y)$.

The first condition for a parallel set states that every two activities from the same set are concurrent with each other; or are the same ($a = b$). In particular, the parallel relation applied only to such set would be transitive. The second condition enforces each parallel set as the largest set possible, satisfying the first. It is easy to realize that:

1. a set with a single activity means it is impossible for any other activity to be concurrent with it;
2. if a project has n activities all in sequence (one preceding the other), then there will be exactly n parallel sets, each with a single activity;
3. the dual of the previous observation means a parallel set will contain all the project activities if none precedes any other;

4. the set of the initial activities is a parallel set;
5. the set of the final activities is a parallel set.

All the observations, other than items 4 and 5, are straightforward. Still, the demonstration of these last ones is easy. Before providing the proof, it is useful to formalize the definitions of initial and final activity.

Definition 4 (Initial Activity). *Let A be the set of all project activities. $a \in A$ is an* initial activity *when*

$$\nexists x \in A,\, x \ll a \ . \tag{6}$$

Definition 5 (Final Activity). *Let A be the set of all project activities. $a \in A$ is a* final activity *when*

$$\nexists x \in A,\, a \ll x \ . \tag{7}$$

Proof (Observations 4 and 5). If the set of initial activities has only one activity, then it trivially satisfies the first condition in the definition of parallel set.

Now, consider the case when there are more than one initial activity. Suppose a, b any two of those activities. We want to prove $a \shortparallel b$, i.e.:

$$\neg(a \ll b) \wedge \neg(b \ll a)$$

If $a \ll b$ then, by (6), b would not be initial. This is absurd, thus $\neg(a \ll b)$. With similar reasoning, also $\neg(b \ll a)$. Hence, the first condition on the parallel set is satisfied.

In order to prove the second condition, we must prove that there is no non-initial activity x that could belong to the parallel set. If x is non-initial, then $\exists y \in A,\, y \ll x$. Since the relation $\ll$ is transitive, there exists at least one initial activity that precedes x. Thus, x is not parallel to all of the initial activities.

Therefore, the set of initial activities is a parallel set.

The above proof can be easily adapted to the set of final activities. Suppose any two final activities (the singular case is again, trivial) a, b. We prove that neither $a \ll b$ nor $b \ll a$ as their assumption falls in absurd from the hypothesis of both being final, referring (7). The second condition is also proved similarly to the initial activities case. In this case, any non-final activity would have at least one final being its successor. Thus, not able to belong to the same parallel set as all the final activities.

As a final remark, note the case where there are only initial activities. This is a degenerated case where all the activities are also final. Thus, the second condition in the definition is immediately satisfied. □

The set of all the parallel sets provides a complete profile on the possible concurrencies among project activities. The relation is symmetric, so finding any set with two activities is trivial. However, finding the largest possible set is more difficult as more activities are involved.

By representing the relation with an undirected graph, using the parallel relation matrix as its adjacency, the problem of detecting parallel sets translates as finding the maximal graph cliques. Given the set of project activities and the

$\boldsymbol{M}_{\shortparallel}$, we take the activities as the graph vertices and whenever $\boldsymbol{M}_{\shortparallel}(a, b) = 1$ an arc connecting a to b is established. The graph is made undirected because $\boldsymbol{M}_{\shortparallel}$ is symmetric.

Again, with our running example, the graph for the parallel relation is shown in Fig. 4. The two maximum cliques are marked, thus the finish-to-start relation from Fig. 2 allows the two largest parallel sets $\{2, 3, 4\}$ and $\{4, 5, 6\}$.

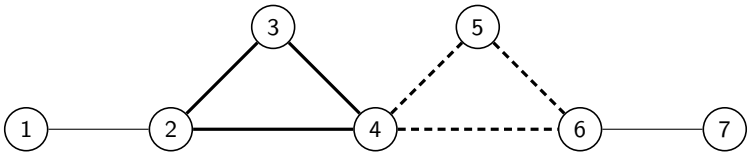


Fig. 4. Graph using $\boldsymbol{M}_{\shortparallel}$ from Fig. 3 as adjacency matrix. With *solid thick*, the arcs joining the vertices 2,3,4 representing the parallel set $\{2, 3, 4\}$. And the *dashed thick* arcs refer to the parallel set $\{4, 5, 6\}$.

3 Example of Application – Project Resource Availability Stress

In this section, we extend the running example to encompass one possible application for the concurrency detection. Consider a project whose activities follow the finish-to-start precedence relations from Fig. 2 and use a single resource constrained in its total availability to 8 units. Also, each activity accepts a range of resource requirements as presented in Tab. 1.

Table 1. Minimum and maximum resource demands in an example project with seven activities.

activity	1	2	3	4	5	6	7
minimum	2	1	1	1	2	2	2
maximum	7	5	4	4	3	5	3

An implicit rule is that all the activity demands should be lower or equal to the resource total availability. In such resource quantity constrained projects, some activities may have to compete for some resource amount with another when they execute concurrently because the total demand accumulates. Since the parallel sets can be identified, it is possible to evaluate minimum and maximum requirements per set.

Definition 6 (Parallel Set Requirements). *Let A be the set of project activities, $\mathbb{P} \subset \mathcal{P}(A)$ the set of all the supported parallel sets, ℓ_a and u_a the minimum*

and maximum demands of activity $a \in A$, respectively. The minimum and maximum requirements for each $p \in \mathbb{P}$, denoted by ℓ_p^* and u_p^*, respectively, are:

$$\ell_p^* = \sum_{a \in p} \ell_a \qquad u_p^* = \sum_{a \in p} u_a \ . \tag{8}$$

Using the maximum requirements of all the parallel sets, it is possible to negate the resource maximum availability constraint by, simply, setting a new maximum greater enough to encompass the worst scenario.

In our project example, from Fig. 4, it is easy to identify all the parallel sets: $\{1, 2\}$, $\{2, 3, 4\}$, $\{4, 5, 6\}$ and $\{6, 7\}$. Their requirements are shown in Tab. 2.

Table 2. Minimum and maximum requirements per parallel set.

set	$\{1, 2\}$	$\{2, 3, 4\}$	$\{4, 5, 6\}$	$\{6, 7\}$
minimum	3	3	5	4
maximum	12	13	12	8

In this case, by resetting the resource maximum to, at least, 13 would create a new project derived from the original but without the interference of the resource maximum availability constraint in the concurrency of activities. Incidently, only the $\{6, 7\}$ is able to execute (concurrently) within the resource maximum of 8 units.

In practice, however, one must consider the imposed constraint. In the example, for $\{1, 2\}$, if activity 1 uses 7 units, then activity 2 cannot use more than 1 unit without being forced to execute *after* activity 1. Such extreme cases may be easy to find. But, if activity 1 used slightly less, say 6 units, then activity 2 could choose to claim up to 2 units. Therefore, there is a compromise of one activity towards another when executing. In some cases, the concurrency is negated and in the others, there are several possibilities for the parallel execution within a confined range.

To consider all the cases, for each parallel set may become cumbersome in the generality of projects, particularly in those with many activities. So, instead of a deterministic analysis, we can evaluate the probability of each parallel set to disrupt the concurrency they describe due to their activities resource usage. We named it as *resource stress* (of a parallel set).

Definition 7 (Resource Stress). *Let p be a parallel set. The* resource stress *of p is the probability of the accumulated quantity x used by its activities being greater than the resource maximum availability K:*

$$\mathrm{P}(x > K),\ x \sim \mathrm{U}\left(\ell_p^*, u_p^*\right) \ . \tag{9}$$

The resource stress gives an hint about the impact of the resource maximum availability to the concurrency in a given parallel set. We assume the accumulated

usage as uniformly distributed due to its simplicity. Also, it is continuous which is required to our case studies where the activities can use any real amount of resource within a feasible interval ($\subset \mathbb{R}$).

Finalizing our example, Tab. 3 shows the resource stress evaluated per parallel set. From Tab. 2, $\ell^*_{\{1,2\}} = 3$ and $u^*_{\{1,2\}} = 12$ which evaluates the resource stress for $\{1, 2\}$ as:

$$\mathrm{P}(x > 8) = \frac{12 - 8}{12 - 3} = \frac{4}{9} \approx 0.444 \ . \tag{10}$$

Table 3. Resource stress per parallel set.

set	$\{1,2\}$	$\{2,3,4\}$	$\{4,5,6\}$	$\{6,7\}$
stress	0.444	0.5	0.571	0

We assess about 57% of maximum resource stress for the example project.

4 Conclusions and Future Research

Starting with the finish-to-start precedence relations on project activities, usually represented in AoN or AoA networks, we devise a method to identify groups of those activities which can execute concurrently – the parallel sets. All of these contain information about the concurrency potential induced by the precedence relations and can be a tool to further assess properties useful to the context of project management. In particular, we show how to evaluate the maximum resource stress a project could incur under resource maximum availability with multimodal activities. The resource stress definition assumes the accumulated activity usages to be uniformly distributed, however. This is simpler to calculate but may be too simplistic. A formal statistical analysis could establish a better distribution in order to provide greater quality of the evaluated stress values.

The concurrency detection method relies in the maximal cliques of the graph whose adjacency is the matrix of the parallel relations. The enumeration of all the maximal cliques of an undirected graph may be exponential in output size [2]. And this in independent of the efficiency of the algorithm that may be used. Therefore, one should accept that there can be cases where our concurrency detection method may produce a very large amount of parallel sets. One possibility is to filter some of the output by picking only the larger parallel sets. Alternatively, on much larger projects, one could take an approach based on just the maximum cliques, representing the largest parallel sets. However, and despite several research, the problem of finding the maximum clique is also NP-complete [6, 7]. Moreover, the usefulness of taking only the largest parallel set may not be enough to assess the intended properties.

Since the maximal clique problem is NP, future iterations on the presented method should exploit the fact that the sets of initial activities and of final activities are always two of the possible parallel sets. This may reduce the overall problem to a more treatable level for arbitrarily large projects. This is of particular interest if one could find similar relationships among other activities without difficulty.

References

1. Bron, C., Kerbosch, J.: Algorithm 457: Finding all cliques of an undirected graph. Commun. ACM 16(9), 575–577 (Sep 1973), http://doi.acm.org/10.1145/362342.362367
2. Cazals, F., Karande, C.: A note on the problem of reporting maximal cliques. Theoretical Computer Science 407(13), 564 – 568 (2008), http://www.sciencedirect.com/science/article/pii/S0304397508003903
3. Chang, L., Yu, J., Qin, L.: Fast maximal cliques enumeration in sparse graphs. Algorithmica 66(1), 173–186 (2013), http://dx.doi.org/10.1007/s00453-012-9632-8
4. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling - A Research Handbook, International Series in Operations Research & Management Science, vol. 49. Kluwer Academic Publishers (2002)
5. Moutinho, R., Tereso, A.: Scheduling of multimodal activities with multiple renewable and availability constrained resources under stochastic conditions. Procedia Technology 16(0), 1106 – 1115 (2014)
6. Pattabiraman, B., Patwary, M., Gebremedhin, A., Liao, W.k., Choudhary, A.: Fast algorithms for the maximum clique problem on massive sparse graphs. In: Bonato, A., Mitzenmacher, M., Praat, P. (eds.) Algorithms and Models for the Web Graph, Lecture Notes in Computer Science, vol. 8305, pp. 156–169. Springer International Publishing (2013)
7. Prosser, P.: Exact algorithms for maximum clique: A computational study. Algorithms 5(4), 545–587 (2012)
8. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science 363(1), 28 – 42 (2006), http://www.sciencedirect.com/science/article/pii/S0304397506003586, computing and Combinatorics 10th Annual International Conference on Computing and Combinatorics (COCOON 2004)