IMMC 2015

I Abstract (Summary)

In the analysis of film scheduling, we based our two models on three instrumental factors: resource availability, time requirements, and dependency. Both models use a directed acyclic graph to represent the filmmaking process, requiring resources available for limited periods of time.

First, the Complete Search Model offers a comprehensive approach applicable in general cases, including special cases. A complete search algorithm is used to find an effective filming schedule, by repeatedly choosing plausible nodes and exploring the feasibility. If rejected, the program moves to the next possible node. Source Code (2011) is used as an example to offer insight on the algorithm, where significant resources, characters, sets and scenes are first identified, after which a filming schedule measured in time units is generated. The algorithm is then applied.

The Complete Search Model is advantageous due to its realistic considerations. Its practicality ensures that it is a working model in many cases. However, the time required to derive at an effective schedule based on this model depends on the limited time duration given for the film to be completed within.

The Rarity Model focuses on the fact that there may be limited resources during construction. Weightings are first assigned to every node in the graph, based on the range of time available for work, in time units, derived through an arbitrary assignment. This is followed by a renormalisation where the weights are adjusted accordingly, in relation to the node's position in the graph and dependency. From the set of all possible nodes, the rarity of each node is calculated, and the one with the highest rarity will be picked. The program is then run, as showcased in the example of Annabelle (2014).

The Rarity Model is advantageous as it considers the priority of events, assigning weights to determine the relative order. However, it is important to note that the program is unable to determine (or rather, will take an unrealistic amount of time) impossible cases, such as an impractical time duration.

Indisputably, unforeseen circumstances will occur throughout the course of filming, leading to delay in production time and affecting the filming schedule in return. Therefore, we have introduced modifications through three ways: adding additional nodes to the graph in case of refilming or extra takes, changing the resource availability table should there be a change in character or resource availability, and deleting completed nodes and any other edges connected to these nodes should there be unanticipated changes midway through the filming process.

In addition, constraints are determined based on the availability of resources during construction and filming, such as cast presence, sites, budget limitations, and sudden modifications to the original script. To do so, the constraints are ranked to identify what causes the most delay. After applying this to Annabelle using the Rarity Model, we obtained results to generalise patterns that indicate the most significant constraints for generic cases.

1. Restatement

1.1 Introduction

Oftentimes, filming schedules are instrumental in the development of movies, taking into consideration resource availability, time constraints, and dependency. Moreover, unforeseen circumstances such as a sudden change in weather that hinders work progress, or something as mundane as wardrobe malfunctions have the potential to affect the shooting schedule of the entire team. Therefore, it is of utmost importance that we develop a working model closely aligned to the availability of resources, while at the same time adhering to a flexibility should there be last minute changes. We also aim to identify and propose working solutions to cater to constraints causing considerable delays in filming.

1.2 Problem Outline

In this problem, we are required to develop a model which generates a reasonable and flexible schedule given the following types of constraints:

1. Resource availability (availability of stars, resources needed for construction or filming, manpower, sites, etc. under the umbrella of resources);
2. Time (amount of time required for construction or filming); and
3. Dependency (there exist scenes required to be shot in a sequential order, after certain preparations such as constructions).

The model should also be able to accommodate unpredictable changes during the filming such as delay or addition of scenes, change in the availability of resources.

1.3 Assumptions

The models take into consideration the following assumptions:

1. Duration of filming takes into account time taken for travel, break time, multiple takes, makeup, setting up, wardrobe fitting, extras, props, etc.
2. The work of construction of one set or filming of one scene has to be continuous. That is to say, one task that lasts for multiple days cannot be broken into separate parts.
3. There is only one director during the entire filming process, and the director must be present during the filming of every scene. However, the director does not necessarily have to be present during the constructions.
4. Availability dates of all resources throughout the entire duration of filming is known before the schedule is made.
5. Integer units for time are assumed.

2. Model

We have developed two models to generate the filming schedule, both using a directed acyclic graph model to represent the filmmaking process. For both models, filming of each scene requires certain types of resources that are only available during one or multiple periods of time.

2.1 Complete Search Model

2.1.1 Definition of Variables

e_i	last day for resource *i* to be available
i_r	total types of resources
j_e	total number of events
N_j	event (scene/construction)
p_k	number of resources of type *k* needed
r_k	resource of type *k*
s_i	first day for resource *i* to be available
t_j	time required for event *j* to be completed
T	current time
u	upper bound for total time required

In the first model, the filmmaking procedure is represented by a directed acyclic graph where each node represents an event (including the start, end, constructions and filming of scenes). The edges indicate the dependency of the events. For example, an edge pointing from node A to B indicates that B can only start after A is finished. For each node N_j, the event needs t_j units of times to be finished.

Resources and their availabilities are not represented in the graph; instead, they are recorded in a separate table. As there may be multiple available periods of time for certain resources (e.g. the casts), we denote the first available period of resource r_i as (s_i, e_i), the second period as (s'_i, e'_i), and so on.

2.1.2 Assumptions

Availability dates for all the resources, as well as the total timespan for the whole film production, are assumed to be reasonable such that small delays will not result in insufficient time for all the events to be finished.

2.1.3 Algorithm

A complete search algorithm is used here to find an effective filming schedule:

1. An upper bound u is determined based on the film studio's estimation.
2. Create a list *L* containing the start time of each node (if it is known). This list will be empty at the beginning. Create a process list *P* consisting of all the ongoing events (at $T = 0$, *P* is empty). Set $T = 1$ and start the algorithm.
3. Check if any event in *P* is ending at time *T*. Mark all events in *P* that end at the current time, *T*, as finished and remove these from *P*.
4. Compare the current time *T* with the upper bound
    a. If $T \leq u$, continue to step 4;

b. Else
   i. if any of the nodes have not been marked as finished, return to the previous step where a node is chosen. Remove the start times of that node from L. The node chosen previously should not be chosen again at the same time unit. Another node should be chosen, or if no other nodes can be chosen, the algorithm will move on to the next time unit, $T=T+1$, without choosing any of the nodes.
   ii. if all nodes are marked, the algorithm ends. The contents of list L will be printed out.

5. Check the possibility to choose node N_j, $j = 1, 2, ..., j_e$ at this unit of time. It is possible to choose node N_j only when these criteria are fulfilled
   a. the parent node(s) of N_j is completed or N_j has no incoming edges;
   b. resources required for N_j are available. That is to say, for every resource required r_k, where p_k number of that resource is needed, check whether the amount of r_k available at time T is $\geq p_k$.
   c. N_j is not in P and has not been marked as finished.
6. From the set of all possible nodes $A = \{N_j, N_{j'}, ...\}$ identified in step 4, choose a node and update the availability of the remaining resources.
   a. If set A is nonempty, update L by setting the start time for the node as T and repeat step 5-6;
   b. else if set A is empty, move on to the next time unit i.e. $T = T + 1$ and repeat steps 3-6.

2.1.3.1 Source Code (2011)

Source Code (2011) [1] is a science-fiction movie that narrates the story of a soldier who wakes up in someone else's body and discovers that he is part of a government program to identify the person behind the train bombing incident.

To demonstrate an example of the algorithm, we investigate the process of filming Source Code (2011). Firstly, we develop a directed graph, where the movie is broken down into 9 scenes. Scenes requiring construction of set are highlighted in **blue**, while filming scenes are highlighted in **purple**. **Red** is used to label the number of days approximated for the filming to take place. Note that the train in the film is a set construction, so scenes filmed in the train does not have to take place in the train station.

Resources	Sets (construction required)	Scenes

Sites	Cockpit	Interrogation
Train station	Train	Talking scene
Plaza	Car/ Van	Stevens unconscious
Characters	Rutledge's office	Chase Frost
Stevens		Christina chasing Stevens
Christina		Talking to Christina
Goodwin		Christina alone
Rutledge		Finding Frost
Derek Frost		Stevens and Christina walking to plaza
Supporting characters		

Table A: Source Code (2011) Compilation of Scenes

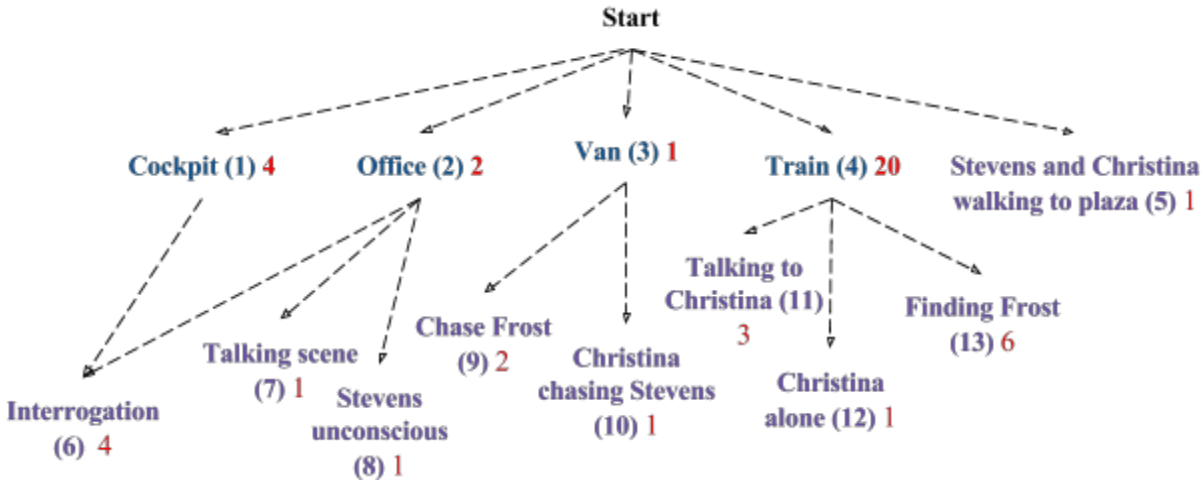

Figure B: Directed acyclic graph for Source Code

Note: Numbers in brackets indicate labels for the nodes. Numbers in red indicate time taken (in units) for event to be completed.

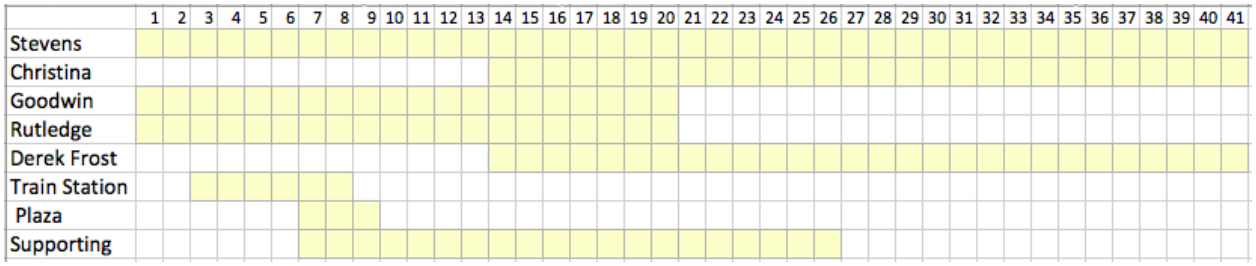

Figure C: Availability dates of resources for Source Code

Scene	**Characters Involved**
Interrogation	Stevens, Goodwin

Talking scene	Goodwin, Rutledge, Supporting characters
Stevens unconscious	Stevens
Chase Frost	Stevens, Frost
Christina chasing Stevens	Christina
Talking to Christina	Stevens, Christina, Supporting characters
Christina alone	Christina, Supporting characters
Finding Frost	Stevens
Stevens and Christina walking to plaza	Stevens, Christina, Supporting characters

Table D: Resources required for each scene for Source Code

Table E shows the simulation results of the algorithm.

T	Possible events	*P* before action	Events finished	Action taken	*P* after action
1	1,2,3,4	-	-	Pick 1,2,3 and 4	1,2,3,4
2	-	1,2,4	3	-	1,2,4
3	8	1,4	2,3	Pick 8	1,4,8
4	-	1,4	2,3,8	-	1,4
5	6	4	1,2,3,8	Pick 6	4,6
6	-	4,6	1,2,3,8	-	4,6
7	-	4,6	1,2,3,8	-	4,6
8	-	4,6	1,2,3,8	-	4,6
9	7	4	1,2,3,6,8	Pick 7	4,7
10	-	4	1,2,3,6,7,8	-	4
11	-	4	1,2,3,6,7,8	-	4
12	-	4	1,2,3,6,7,8	-	4
13	-	4	1,2,3,6,7,8	-	4
14	9,10,5	4	1,2,3,6,7,8	Pick 9	4,9
15	-	4,9	1,2,3,6,7,8	-	4,9

16	10, 5	4	1,2,3,6,7,8,9	Pick 10	4,10
17	5	4	1,2,3,6,7,8,9,10	Pick 5	4,14
18	-	4	1,2,3,5,6,7,8,9,10	-	4
19	-	4	1,2,3,5,6,7,8,9,10	-	4
20	-	4	1,2,3,5,6,7,8,9,10	-	4
21	11, 12, 13	-	1,2,3,4,5,6,7,8,9,10	Pick 11	11
22	-	11	1,2,3,4,5,6,7,8,9,10	-	11
23	-	11	1,2,3,4,5,6,7,8,9,10	-	11
24	12, 13	-	1,2,3,4,5,6,7,8,9,10,11	Pick 12	12
25	13	-	1,2,3,4,5,6,7,8,9,10,11,12	Pick 13	13
26	-	13	1,2,3,4,5,6,7,8,9,10,11,12	-	13
27	-	13	1,2,3,4,5,6,7,8,9,10,11,12	-	13
28	-	13	1,2,3,4,5,6,7,8,9,10,11,12	-	13
29	-	13	1,2,3,4,5,6,7,8,9,10,11,12	-	13
30	-	13	1,2,3,4,5,6,7,8,9,10,11,12	-	13
31	-	-	1,2,3,4,5,6,7,8,9,10,11,12,13	Terminate algorithm	-

Table E: Simulation of Algorithm

From the simulation we get the filming schedule that ends at $T = 31$. In this example, we assume that resources required during construction of sets are unlimited, so it will be possible for the constructions to start simultaneously from the first day of our schedule. This assumption was made to simplify the example due to the page constraints. **However, this model also work for examples that consists of limited resources required during the construction.** This is because in either case, the algorithm will check for the availability of resources at each unit of time, so the model will be able to solve more generic cases.

2.1.4 Strengths and Limitations of Model

The major strength for this Complete Search Model is that this model makes few assumptions, so generally it is applicable in most real-life cases given the data required.

The limitation of this model is that it takes relatively long time to make the schedule. This is because that this model works by randomly choosing one branch to start and backtracking whenever the time exceeds the deadline. This means that potentially the algorithm will have to check every possible way

to traverse the branch if there is only one schedule that completes everything within the timespan. However, the probability for this to happen in real-life cases is so low that it is insignificant, since the time given must be long enough to allow for flexible schedules.

Also, the algorithm terminates once a feasible schedule is found, thus it may not always yield the one that finishes within the shortest amount of time.

2.2 Rarity Model

2.2.1 Definition of Variables

C_i	start time of event *i*
d	duration of resource *i* to be available
e_i	last day for resource *i* to be available
e_j	last day for all resources needed by event *j* to be available
i_r	total types of resources
j_T	total number of nodes at time *T*
N_j	event (scene/construction)
p_k	number of resources of type *k* needed
r_k	resource of type *k*
R	rarity of the node
s_i	first day for resource *i* to be available
s_j	first day for all resources needed by event *j* to be available
t_j	time required for event *j* to be completed
T	current time
u	upper bound for total time required

Each node N_j is assigned a time period from s_j to e_j, during which all the resources needed to complete the event will be available. The length of the period will be represented by duration *d*, and at any time *T*, duration d_{jT} is calculated by

$$d_{jT} = e_j - T,$$

Using this duration and the longest possible time for all events to be finished, we calculate the rarity of each node, which indicates the difficulty to schedule the event. This helps us to decide on which node to choose if multiple nodes are available at the same time, and reduces the frequency for backtracking. At any time T, rarity R is a weight assigned to each node calculated by

$$R = j_T - (T - 1) - d_{jT},$$

where $(j_T - T)$ estimates the total number of unchosen nodes in the graph at time T assuming all events to be completed one by one. It acts as an upper bound of the total amount of time required for the filmmaking to be finished.

2.2.2 Assumptions

1. When the schedule is being made, it is assumed that there will be no change in availability of resources throughout the movie filming. For example, if the total amount of resource A is 4 during the first available period, the total amount is assumed to be unchanged for the rest periods whenever they are available. In the case when there *is* a change due to unforeseen circumstances, the algorithm will have to be rerun, this time considering the new availability.
2. Availability dates for all the resources, as well as the total timespan for the whole film production, are assumed to be reasonable such that small delays will not result in insufficient time for all the events to be finished.

2.2.3 Building the Model

2.2.3.1 Weighting and Renormalisation

In this process, the availability of all resources required to pick a node will be considered as a group, and the period of time when all resources required are available for this node will be where the available time for every resource intersects. The filmmaking procedure is represented by a directed acyclic graph where each node represents a time unit, and events that require multiple time units to be completed are split up into branches with multiple nodes. The edges indicate the dependency between the events. For example, an edge pointing from node A to B indicates that B can only start after A is finished. Each node is assigned a weight according to the period of time during which all resources required to complete the event represented by the note are available.

An example below demonstrates how the graph is formed. Nodes in light blue {A, B, I} denote construction vertices, while nodes in purple {C, D, E, F, G, H, I, J, K} denote filming vertices. The root node represents the start of the filmmaking process. For any node Nj, the starting and ending dates of d_j is denoted by s_j and e_j respectively. For example, for node A, $s_A = 1$ and $e_A = 6$.

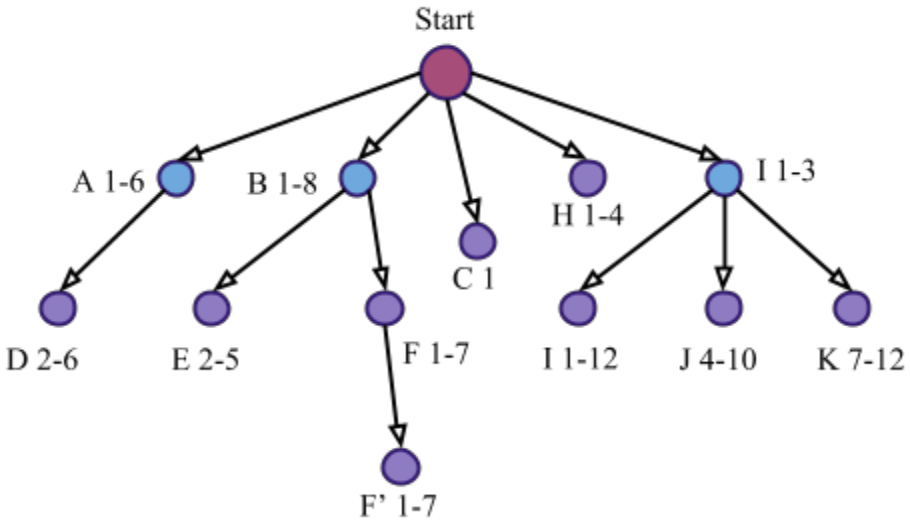

Figure F: Original Graph

Next, we use the following process to renormalise the weighting for every node:

1. For every branch in the tree, adjust the starting and ending dates of each node using the following rules. For any node N_i, $i \in 1, 2, ..., n$:

$$S_i = max\{S_{i-1} + 1,\ S_i\},\ i \neq 1,$$
$$E_i = min\{E_{i+1} - 1,\ E_i\},\ i \neq n.$$

   a. Note that the order of the operations are opposite i.e. operations for S_i start from the first node of the branch, while that for E_i start from the last node and go backwards.

   b. Consider the above example, A(1-6) is adjusted to A(1-5) as D ends on latest T= 6 and it takes 1 unit to complete D after A has been completed; B(1-8) is adjusted to B(1-4); and similarly, F(1-7) is adjusted to F(2-6), F'(1-7) to F'(3-7) accordingly.

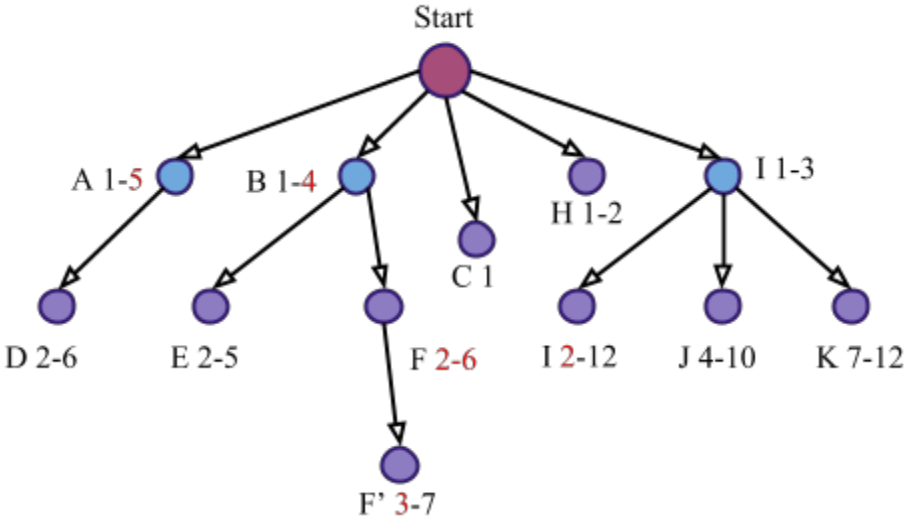

Figure G: Graph after Renormalisation

2. For every node N_j that have multiple (more than one) sub-branches, denote its daughter nodes as $\{v_1, v_2, ..., v_n\}$. Further adjust the ending date of N_j using the following rules:
    a. For all i such that $\exists\ e_j + i = e_{v_n}$ for some v_n: consider every v_n that satisfies $e_j + 1 \le e_{v_n} \le e_j + i$ as k, denote the total number of them for each value of i as $|v_i|$,
    b. Check for all possible value of i,
        i. If $|v_i| \ge i + 1$, record $l_i = |v_i| - i$ into a priority-queue $\{l_i\}$;
        ii. Choose the maximum l_i;
        iii. Adjust the ending date for N_j such that $e_j = e_j - l_i$.

At any unit of time, rarity of each node will be calculated based on the ending time obtained from the normalised graph and the current time.

$\forall N_i$ and N_j such that i and j are the same type of event and $C_i \le C_j < C_i + t_i$, where C_v denotes the current start time of vertex v, the events will clash. Then, for $E_j \le E_i$, set $C_i = C_j + t_j$, to resolve the conflict.

2.2.3.2 Resource availability

While several events may have the same start time s_j where all resources needed for that event are available, multiple events may not be able to start simultaneously, so it is necessary to check whether there are sufficient resources for constructions to be carried out. To represent the availability of each resource needed, each node representing the event is turned into a pie chart. Each sector of the pie chart, given a different colour, represents one type of resource needed for the event to be carried out. A number is also assigned to each sector, representing the quantity of that particular type of resources that is needed. It is only possible for two constructions to happen simultaneously if for two sectors having the same colour on the pie charts (for both events), the sum of the two numbers on those two sectors does not exceed the total amount of (that type of) resource available.

2.2.4 Scheduling by Priority

We will start with a specific example, after which the general algorithm will be presented.

2.2.4.1 Annabelle (2014)

Annabelle (2014) is an American supernatural psychological horror film [2] directed by John R. Leonetii. Against a budget of $6.5 million, it grossed over $225 million, making it one of the highest grossing horror films of all time. It is both a prequel and sequel to *The Conjuring*, inspired by a story of a doll named Annabelle and narrating the story of how a pregnant Mia is haunted by the doll in several occasions.

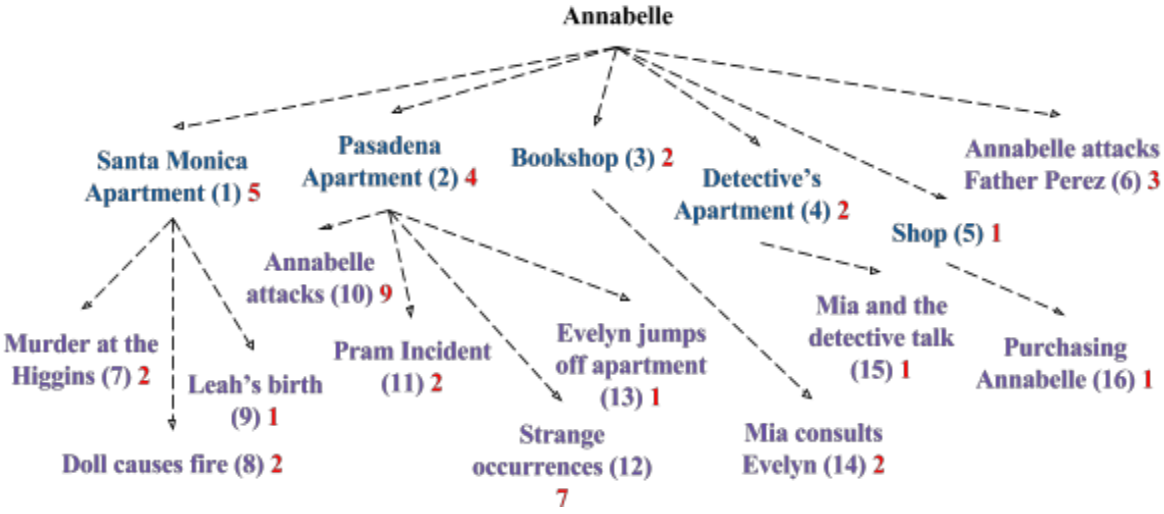

Figure H: Directed acyclic graph for Annabelle

Resources	Sets (construction required)	Scenes
Sites	Santa Monica Apartment	Murder at the Higgins
Church	Pasadena Apartment	Doll causes fire
Characters	Bookstore	Mia and the detective talk
Mia Form	Detective's Apartment	Leah's birth
John Form	Shop	Strange Occurrences
Father Perez		Mia consults Evelyn
Evelyn		Pram Incident
Detective Clarkin		Annabelle attacks Father Perez
Supporting characters		Annabelle attacks
Constructions		Evelyn jumps off apartment
4 Book Props		Purchasing Annabelle
20 Toys		
6 Dolls		
Annabelle		
55 Set Construction Helpers		

Table I: Annabelle (2014) Compilation of Scenes

If certain scenes are only to be filmed at particular periods in the day, for example, murder scenes at night, in order to take this into account, we assign another restriction to the availability of resource in place of that, by imposing that the resource is only available on days in multiples of k, where $k \in \mathbb{R}$.

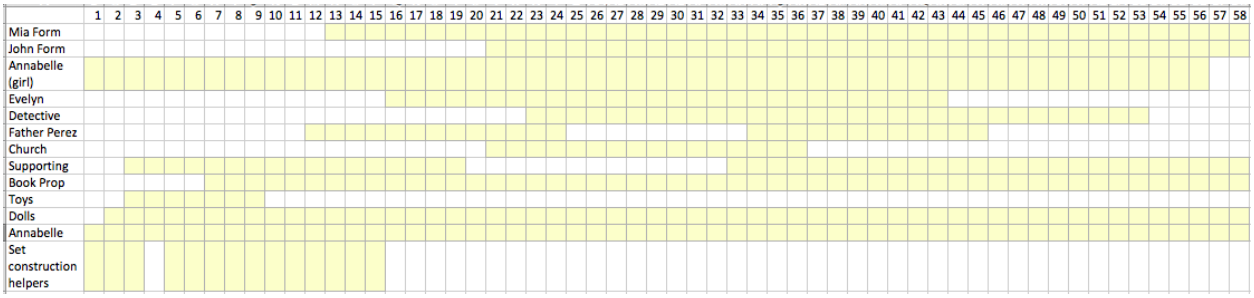

Figure J: Availability Dates of Resources for Annabelle

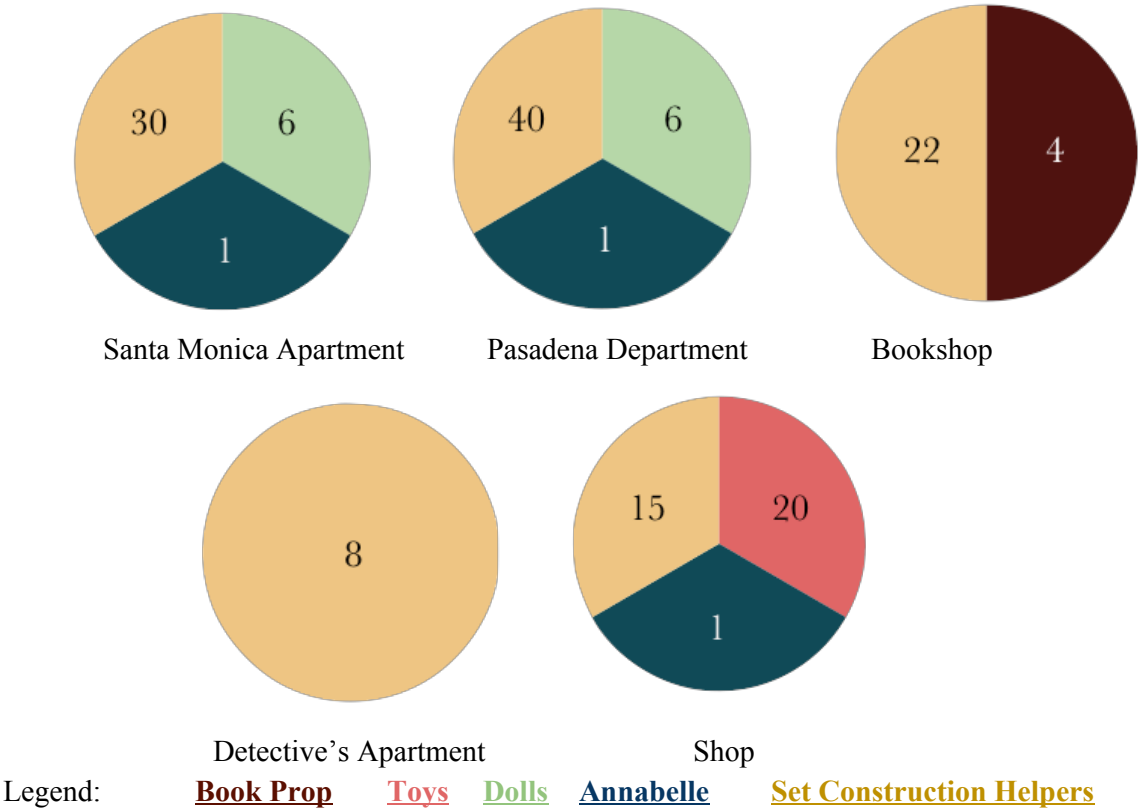

Figure K: Pie charts for constructions (numbers represent amount of resource needed)

Time Units	**Constructions**	**Resources required**	**Availability** (Units __ to __)
5	Santa Monica Apartment	Set construction helpers (30), dolls (6), Annabelle (1)	2 to 3 (N.A.), 5 to 15
4	Pasadena Department	Set construction helpers (40), dolls (6), Annabelle (1)	2 to 3 (N.A.), 5 to 15
2	Bookshop	Set construction helpers (22), books (4)	7 to 15
2	Detective's Apartment	Set construction helpers (8)	1 to 3, 5 to 15

1	Shop	Set construction helpers (15), toys (20), Annabelle (1)	3, 5 to 9

Figure L: Resources required for each construction

Note that it is not possible to finish the first two events from 2-3 as the time units required to finish the construction exceeds the limit.

Time Units	**Scenes**	**Characters Involved**	**Availability** (Units __ to __)
2	Murder at the Higgins	Mia, John, Annabelle, Supporting characters	33 to 56
2	Doll causes fire	Mia, Annabelle	13 to 56
1	Mia and the detective talk	Mia, John, Detective	23 to 53
1	Leah's birth	Mia, John, Supporting characters	33 to 58
12	Strange occurrences	Mia, Annabelle	13 to 56
2	Mia consults Evelyn	Mia, Evelyn	16 to 43
2	Pram incident	Mia	13 to 58
3	Annabelle attacks Father Perez	Annabelle, Father Perez	12 to 24, 34 to 45
9	Annabelle attacks	Mia, Annabelle	13 to 56
1	Evelyn jumps off apartment	Mia, Annabelle, Evelyn	16 to 43
1	Purchasing Annabelle	Annabelle, Supporting characters	3 to 19 or 33 to 58

Figure M: Resources required for each scene for Annabelle

Based on the data provided above, a directed graph is constructed as below that shows the filmmaking process of Annabelle after normalisation. Nodes in light blue denote construction vertices, while nodes in purple denote filming vertices. If an event has multiple periods of time where the resources it requires are available, there will be two branches representing the same event.

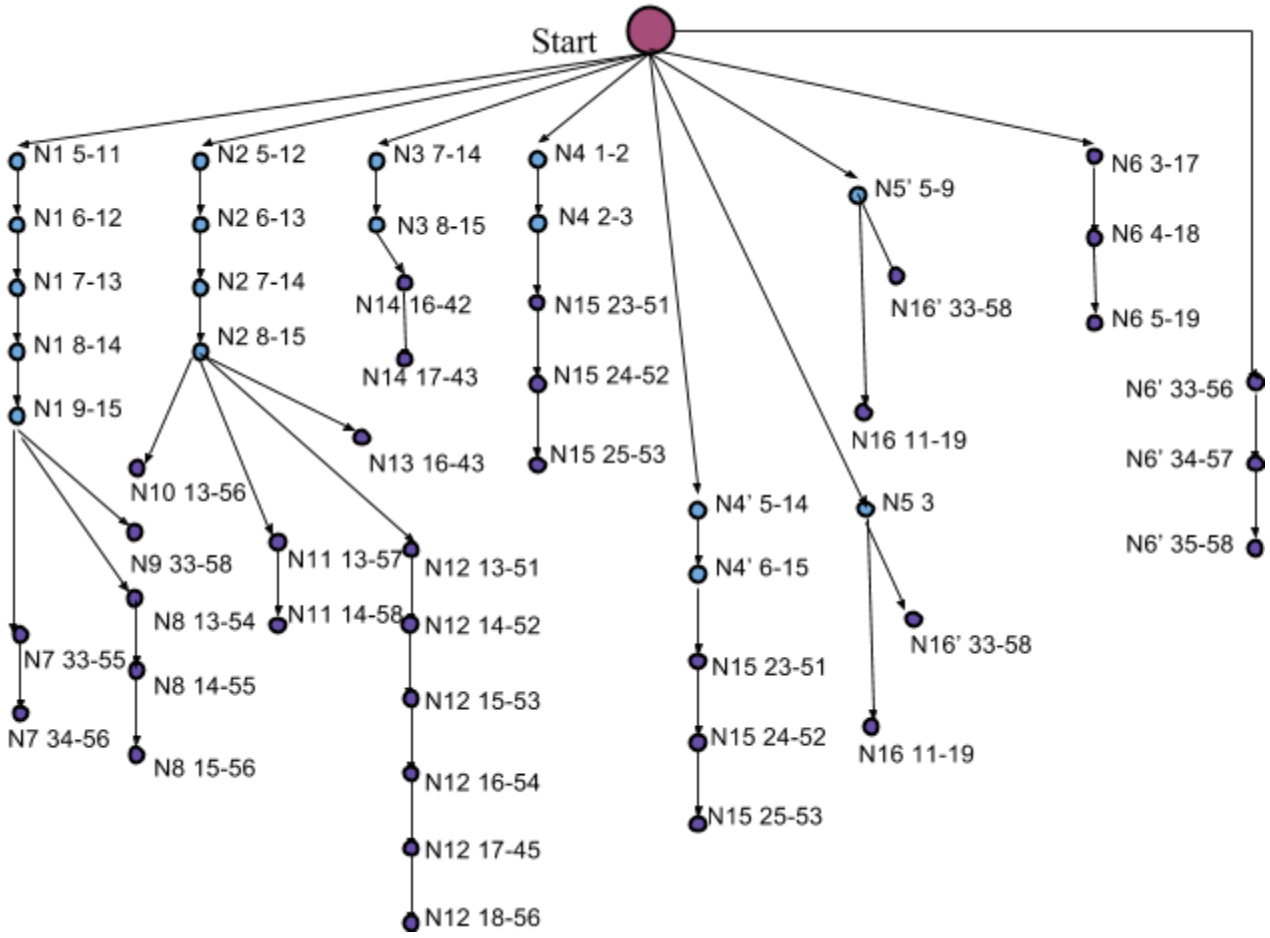

Figure N: Graph for Annabelle (after renormalisation)

2.2.4.2 Algorithm

Next, we use the algorithm shown below to attain a schedule based on the graph obtained from 2.2.4.1.

1. Run steps 1-4 as in the complete search algorithm.
2. Check the possibility to choose node N_j, $j = 1, 2, ..., j_e$ at this unit of time. It is possible to choose node N_j only when these criteria are fulfilled
    a. the parent node(s) of N_j is completed or N_j has no incoming edges;
    b. resources required for N_j are available. The current time T is between s_j and e_j. For every resource required r_k, where p_k number of that resource is needed, check whether the amount of r_k available at time T is $\geq p_k$.
    c. N_j is not in P and has not been marked as finished.
    d. if N_j is a node with multiple possible durations, N_j **had not been chosen in any of the previous possible durations.**
3. From the set of all possible nodes $A = \{N_j, N_{j'}, ...\}$ that can be chosen, **calculate the rarity of each node in set A and pick the node with the highest rarity. If this node has multiple**

possible durations, run the sub-algorithm in step 4. Update the availability of the remaining resources.

a. If set A is nonempty, update L by setting the start time for the node as T and repeat step 2-3;
b. else if set A is empty, move on to the next time unit i.e. T=T+1.

4. For nodes that have **multiple possible durations**, run the following sub-algorithm:
   a. Let the ith duration of the node be the interval of $s_i - e_i$. Start the below steps for $i = 1$, and repeat for $i + 1$:
      i. Consider the set of collection of integers $J = \{j\}$ where $j \geq 1$ such that $\exists\ m \in G \backslash F$, $e_1 + j = e_m$ where F is the set of daughter vertices of the node and the node itself.
      ii. Define $U_j = \{u\}\ \forall\ j \in J$ where $u \in G \backslash F$ satisfies the inequality $e_1 + 1 \leq e_u \leq e_1 + j$.
      iii. If $\exists\ j$ (satisfying the conditions illustrated in the previous step) such that $|U_j| \geq j + 1$, remove this interval from the graph.

For the above example of Annabelle (2014), we get the following result:

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Action taken
1	-	-	-	**1**	-	-	-	-	-	-	-	-	-	-	-	-	Pick 4
2	-	-	-	*P*	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	Done	**1**	-	-	-	-	-	-	-	-	-	-	-	Pick 5
4	-	-	-	Done	Done	-	-	-	-	-	-	-	-	-	-	-	-
5	**1**	2	-	Done	Done	-	-	-	-	-	-	-	-	-	-	-	Pick 1
6	*P*	-	-	Done	Done	-	-	-	-	-	-	-	-	-	-	**1**	Pick 16
7	*P*	-	**1**	Done	Done	-	-	-	-	-	-	-	-	-	-	Done	Pick 3
8	*P*	-	*P*	Done	Done	-	-	-	-	-	-	-	-	-	-	Done	-
9	*P*	-	Done	Done	Done	-	-	-	-	-	-	-	-	-	-	Done	-
10	Done	**1**	Done	Done	Done	-	-	-	-	-	-	-	-	-	-	Done	Pick 2
11	Done	*P*	Done	Done	Done	-	-	-	-	-	-	-	-	-	-	Done	-
12	Done	*P*	Done	Done	Done	**1**	-	-	-	-	-	-	-	-	-	Done	Pick 6
13	Done	*P*	Done	Done	Done	*P*	-	-	-	-	-	-	-	-	-	Done	-
14	Done	Done	Done	Done	Done	*P*	-	-	-	-	-	-	-	-	-	Done	-
15	Done	Done	Done	Done	Done	Done	-	3	-	**1**	3	2	-	-	-	Done	Pick 10
16	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
17	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
18	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
19	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
20	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-

21	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
22	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
23	Done	Done	Done	Done	Done	Done	-	-	-	*P*	-	-	-	-	-	Done	-
24	Done	Done	Done	Done	Done	Done	-	2	-	Done	2	**1**	3	2	3	Done	Pick 12
25	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	*P*	-	-	-	Done	-
26	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	*P*	-	-	-	Done	-
27	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	*P*	-	-	-	Done	-
28	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	*P*	-	-	-	Done	-
29	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	*P*	-	-	-	Done	-
30	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	*P*	-	-	-	Done	-
31	Done	Done	Done	Done	Done	Done	-	1	-	Done	1	Done	2	**1**	2	Done	Pick 14
32	Done	Done	Done	Done	Done	Done	-	-	-	Done	-	Done	-	*P*	-	Done	-
33	Done	Done	Done	Done	Done	Done	1	1	2	Done	**1**	Done	2	Done	2	Done	Pick 11
34	Done	Done	Done	Done	Done	Done	-	-	-	Done	*P*	Done	-	Done	-	Done	-
35	Done	Done	Done	Done	Done	Done	1	**1**	2	Done	Done	Done	2	Done	2	Done	Pick 8
36	Done	Done	Done	Done	Done	Done	-	*P*	-	Done	Done	Done	-	Done	-	Done	-
37	Done	Done	Done	Done	Done	Done	**1**	Done	2	Done	Done	Done	2	Done	2	Done	Pick 7
38	Done	Done	Done	Done	Done	Done	*P*	Done	-	Done	Done	Done	-	Done	-	Done	-
39	Done	Done	Done	Done	Done	Done	Done	Done	1	Done	Done	Done	1	Done	**1**	Done	Pick 15
40	Done	Done	Done	Done	Done	Done	Done	Done	1	Done	Done	Done	**1**	Done	Done	Done	Pick 13
41	Done	Done	Done	Done	Done	Done	Done	Done	**1**	Done	Done	Done	Done	Done	Done	Done	Pick 9
42	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Done	Terminate algorithm

Table O: Simulation of Algorithm. Note that numbers indicate the ranking of nodes by rarity (not the actual value of the rarity of the node). Bold numbers indicate that the event has been picked for that particular time unit.

From the simulation we get the filming schedule that ends at $T = 42$. This shows that the total deadline we have given based on the total number of nodes on the graph (i.e. 58) is relatively loose compared to the actual timespan. Therefore, it allows certain delays to take place, which will be further elaborated in section 3.

2.3 Strengths and Limitations

Compared to the Complete Search Model, it takes a shorter time to yield a schedule on average. This is because after ranking the possible events at every time unit, the programme will always pick the first one in the queue according to the difficulty of scheduling it. Therefore, it reduces the chances for backtracking and reduces time required to find a schedule.

For the limitations of this model, if the total time given for filming is too short, or the director made a mistake and it is not possible to complete filming (e.g. because certain resources are available for only a very short period of time), the models are unable (or take an unrealistically long time) to determine that no filming schedule can be produced.

Additionally, the algorithm terminates automatically once a feasible schedule is found, where a solution is possible for the schedule to be completed within a given timespan. However, it may not always yield the best and most efficient solution -- one that requires the minimum amount of time.

As a solution to this, it is always possible to reduce the total time limit to check for possible ways to yield a more efficient solution. Moreover, according to our assumption, as the reasonable amount of available dates for resources and casts ensures that certain delays are allowed, the possibility of certain days to not be available is low, to the point of insignificance. Furthermore, we have extended the availability dates of the resource to ensure that the chances of required resources being unavailable on any day is low.

3 Modifications

In the event of changes, the models above can be modified in the following ways:

1. If some scenes need to be refilmed, or extra scenes must be filmed: Additional nodes can be added to the graph.
2. If the availability of certain resources or actor changes: The resource availability table will be changed.
3. If the change occurs midway through the filming: All completed nodes (e.g. construction of sets) will be deleted from the graph, as well as any edges connected to any of these nodes, because the nodes are not affected by the changes and do not need to be re-completed.
    a. If a scene needs to be refilmed, a new node will be added to the graph, rather than "re-completing" the original node.)

The algorithm will be rerun with the changed constraints and a modified filming schedule (for the remaining scenes/events that have not be completed) will be produced.

4 Determination of Constraints

In reality, the process of film production can be influenced by unforeseen incidents, causing changes to the constraints. Based on our model, possible modifications include:

1. Availability of resources during the construction
2. Availability of resources during the filming
    a. Casts
        i. Fast & Furious stuntman Paul Walker [3] unfortunately passed away in November 2013, affecting the progress of Fast & Furious 7 (2015). Filming was delayed as directors and producers scrambled to find replacements; and

even so, training had to be conducted with Walker's brothers to ensure that they are well prepared to act out the scenes. Time was also taken to produce CGI.

b. Sites
    i. Even though PAs are employed to keep out the crowd during filming in New York, for example, pedestrians still walk into filming scene, disrupting filming and often causing delay and multiple retakes [4].
c. Budget
    i. For only USD 7000, director Carruth was tasked to make the best out of Primer (2004) with a heavily limited budget [5].
d. Other resources required during filming
    i. On Dec 2014, a cameraman died during the filming of Jackie Chan's movie because of a boat capsizing accident. Time was taken to find a replacement [6].

3. Sudden modifications of original script

Research is done to list out accidents that happened during filming in recent years, among which the most common are during film production -- the change in availability of dates of resources during filming. In general, it is easier to find a replacement for resources during construction, for instance manpower and construction materials, than during filming.

Next, we examined a specific example using the Rarity Model. In order to identify the most important constraints in the filmmaking, constraints are modified in a standardised way and delays are approximated and calculated. After which, we ranked the constraints and identified what causes the most delay. Based on the results obtained from the example, we then generalised patterns that indicate the most important constraints for more generic cases.

To compare between different constraints, we use the same amount of days for the delays in all constraints. The following indicates how change of constraints will be executed:

1. For every case that shows change in available dates of any resource required during filming, we delay both s_i and e_i of one resource by 5 units.
2. For time required to shoot each scene, we increase the timespan for filming each scene by 5 units respectively.

Resource	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Delay	+3	0	0	0	0	+1	0	-2	0	0	0	0	0	0

Table P: Delay in overall schedule for a delay of 5 units for each type of resource

From the results, we observed that resource 1 will cause the most delay if its availability was delayed. As most of the events are not scheduled on the first day that all the resources required are available, the delay in available dates does not affect the final timespan significantly. However, resource 8 results in shorter total amount of time because of the fact that some events that do not require resource 8 are

shifted earlier in the new schedule so that a few days when nothing could be done in the original schedule are also utilised.

Node	6	7	8	9	10	11	12	13	14	15	16
Delay	+5	+5	+5	+5	+5	+5	+5	+5	+5	+5	0

Table Q: Delay in overall schedule if the time taken for filming a certain scene is increased by 5 units

As explained previously, no two scenes can be filmed concurrently, so most of the time the increase in timespan for certain scenes to be filmed will directly result in everything happening afterwards to be shifted back.

Based on the examples above, we examined how the total duration of film production can be affected, and propose the following algorithms to determine how much delay each constraint will cause should any incident happens. The first algorithm calculates a lower bound of the overall delay to the production timespan of the film if the duration of filming of any scene requires an extension.

Define a block to be a filming event with length equal to the amount of time required for filming. Let the blocks be B_i where blocks are named in chronological order. Let S_{B_i} be the starting time of the filming event of B_i and similarly define E_{B_i} to be the ending time of the filming event of B_i.

Let $FC_{B_i} = S_{B_{i+1}} - E_{B_i}$ and $FC = \{FC_{B_i}\}_{i=1}$. Let ε be the amount of extension to a certain filming event duration.

1. If $\varepsilon > max\ FC$ then the overall production process will take at least $\varepsilon - max\ FC$ more number of days.
2. If $\varepsilon < max\ FC$, let $J = \{j\}$ be the set of indices j such that $\varepsilon < FC_{B_j}$. In addition, let L be the set of all vertices excluding the vertex set J and also $FC' = \{FC_{B_l}\}_{l \in L}$ Then:
    a. If the filming event that needs extension is block B_l where $l \notin J$ then the whole production will take at least $\varepsilon - max\ FC'$ more days.
    b. If the filming event that needs extension is block B_j where $j \in J$ then the whole production take 0 more days.

Using this algorithm, we can rank every scene according to the lower bound of total amount of delay, which indicates how much the delay of each scene can affect the holistic film production. In this way, we can identify the most important constraints.

The second algorithm ranks all the constraints according to the amount of delay will be caused for every resource (and thus, the overall delay caused) if its availability changes.

1. Define $K_j = \{k\}$ to be the set of numbers k where R_k is used in a block B_j (that is, a filming event, refer to the above algorithm). Run the following step for each resource R_i:

a. For each resource R_i consider the blocks denoted by the collection $B_{R_i} = \{B_i\}$ such that S_{R_i} (that is, the starting time of the resource R_i) is the maximum among all the $S_{R_k} \forall\ k \in K_i$.

b. Let the availability of the resource be changed to $S_{R_i} + \varepsilon$ to E_{R_i} (that is, the ending time of the resource R_i). Consider $B_p \in B_{R_i}$. Calculate the largest value of l_{p,R_i} such that the following inequality is satisfied:
$\varepsilon \geq \sum_{j=p}^{l_{p,R_i}} FC_{B_j}$ (where FC_{B_j} is defined as before)

c. Now, calculate $F_{R_i} = \sum_{p} l_{p,R_i}$ for all values of p such that $B_p \in B_{R_i}$.

2. Let $F = \{F_{R_i}\}$ where R_i ranges over all resources. The most essential resource (that will cause the longest delay) is resource R_{max} that satisfies $F_{R_{max}} = max\ F$. (Analogously, the qth most important resource would be resource R_q that satisfies F_{R_q} being the qth largest in the set F.)

5 Conclusion

In conclusion, we have proposed two methods in the form of acyclic graphs to approach the task of film scheduling: first, the Complete Search Model, and second, the Rarity Model. In doing so, we employed graph theory to aid us in building the models to represent the film production process, after which we developed algorithms to find effective and flexible schedules.

Question 1 is sufficiently addressed in the report, with two models feasible for use and flexible in the cases of including extra time for shots and taking into consideration the various limitations imposed upon the models. For Question 2, the models are used for rescheduling should there be modifications. To address Question 3, we made use of the Rarity Model to arrive at a quantifiable delay caused by each constraint, and summarised notable points that may possible indicate the importance of the named constraints. Based on these observations, we are able to propose algorithms to verify the importance of two main categories of constraints: the availability of resources, and time required to shoot the scenes.

In closing, the rigorous algorithms and proposed models are not only feasible in the process of scheduling filmmaking, but also in, for instance, the management of shared timetables, such as for schools, or even the planning of events. The models may also be modified to introduce further restraints, or the interruption of tasks before completion, in addition to tracking the performance of these models to output the most efficient and time-conserving schedule.

Appendix

A1 Programme (Complete Search Model)

The following illustrates the programme written in C++ language:

```
//
//  main.cpp
//  movie
//
//  Created by Team 2015017 on 11/5/15.
//  Copyright (c) 2015 IMMC Team 2015017. All rights reserved.
//

#include <iostream>
#include <vector>
#include <queue>
using namespace std;
typedef pair<int, int> ii;
int t[105], bg[105];
bitset<105> hpn, fin;
vector<ii> res[105];
vector<int> d[105], fw[105], st[105], adjlist[105], p[105], start[105];
int n,e,r,tg;
bool fnd=false;
priority_queue<ii, vector<ii>, greater<ii> > vec;

void update(int i, int j, int amt, vector<int> &f) {
   for (int x=i; x<=j; x++) f[x]+=amt;
}

bool check(int i, int j, int amt, vector<int> &f) {
   for (int x=i; x<=j; x++) if (f[x]<amt) return false;
   return true;
}

void resolve(int ct, int j, vector<int> st1[], int bg1[], vector<int> fw1[], bitset<105> fin1, bitset<105>
hpn1, priority_queue<ii, vector<ii>, greater<ii> > vec1) {
   while (!vec1.empty() && vec1.top().first==ct) {
      int c=vec1.top().second;
      fin1[c]=true;
      vec1.pop();
   }
```

```
//check if valid schedule has already been found
bool b=true;
for (int i=1; i<=n; i++) {
   if (!fin1[i]) {b=false; break;}
}
if (b && ct<=tg) {
   fnd=true;
   for (int i=1; i<=tg; i++) {
      for (int k=0; k<st1[i].size(); k++) {
         start[i].push_back(st1[i][k]);
      }
   }
   return;
}
//save current information in case backtracking is needed
vector<int> stcpy[105], fwcpy[105];
int bgcpy[105];
bitset<105> hpncpy, fincpy;
for (int k=1; k<=tg; k++) {
   bgcpy[k]=bg1[k];
   for (int l=0; l<st1[k].size(); l++) {
      stcpy[k].push_back(st1[k][l]);
   }
   for (int l=0; l<fw1[k].size(); l++) {
      fwcpy[k].push_back(fw1[k][l]);
   }
   hpncpy[k]=hpn1[k];
   fincpy[k]=fin1[k];
}
priority_queue<ii, vector<ii>, less<ii> > rarity;
for (int i=j; i<=n; i++) {
   if (fnd) break;
   if (!hpn1[i]) {
      bool pos=true;
      //check resource availability and parents of the node
      for (int k=0; k<p[i].size(); k++) {
         if (!fin1[p[i][k]]) {pos=false; break;}
      }
      for (int k=0; k<res[i].size(); k++) {
         if (!check(ct,ct+t[i]-1,res[i][k].second,fw1[res[i][k].first])) {pos=false; break;}
      }
      if (pos && ct+t[i]<=tg) {
         priority_queue<ii, vector<ii>, greater<ii> > veccpy;
```

```
            vector<ii> add;
            while (!vec1.empty()) {
               add.push_back(vec1.top());
               vec1.pop();
            }
            for (int k=0; k<add.size(); k++) {
               vec1.push(add[k]);
               veccpy.push(add[k]);
            }
            vec1.push(make_pair(ct+t[i],i));
            //update resource availability
            for (int k=0; k<res[i].size(); k++) {
               update(ct,ct+t[i]-1,-res[i][k].second,fw1[res[i][k].first]);
            }
            //set start time of this node to current time
            st1[ct].push_back(i);
            bg1[i]=ct;
            hpn1[i]=true;
            //try to find schedule if this node is picked at the current time
            resolve(ct,i+1,st1,bg1,fw1,fin1,hpn1,vec1);
            for (int k=0; k<105; k++) {
               bg1[k]=bgcpy[k];
               st1[k].clear();
               fw1[k].clear();
               for (int l=0; l<stcpy[k].size(); l++) {
                  st1[k].push_back(stcpy[k][l]);
               }
               for (int l=0; l<fwcpy[k].size(); l++) {
                  fw1[k].push_back(fwcpy[k][l]);
               }
               hpn1[k]=hpncpy[k];
               fin1[k]=fincpy[k];
            }
            while (!vec1.empty()) {
               vec1.pop();
            }
            while (!veccpy.empty()) {
               vec1.push(veccpy.top());
               veccpy.pop();
            }
         }
      }
   }
```

```
    if (!fnd && ct<tg) {
        //if no other nodes can be picked, move on to the next time unit
        resolve(ct+1,1,st1,bg1,fw1,fin1,hpn1,vec1);
    }
}

int main() {
    //input
    scanf("%d%d%d%d", &n, &e, &r, &tg);
    hpn.reset();
    fin.reset();
    for (int i=1; i<=n; i++) {
        scanf("%d", &t[i]);
    }
    for (int i=0; i<e; i++) {
        int a,b;
        scanf("%d%d", &a, &b);
        adjlist[a].push_back(b);
        p[b].push_back(a);
    }
    for (int i=1; i<=r; i++) {
        int tp;
        scanf("%d", &tp);
        for (int j=0; j<tp; j++) {
            fw[i].assign(tg+5,0);
            int a,b,val;
            scanf("%d%d%d", &a, &b, &val);
            update(a,b,val,fw[i]);
        }
    }
    for (int i=1; i<=n; i++) {
        int types;
        scanf("%d", &types);
        for (int j=0; j<types; j++) {
            int a,b;
            scanf("%d%d", &a, &b);
            res[i].push_back(make_pair(a,b));
        }
    }
    resolve(1,1,st,bg,fw,fin,hpn,vec);
    if (fnd) {
        //print start times for each event if valid schedule is found
        for (int i=1; i<=tg; i++) {
```

```
            printf("At time %d: ", i);
            for (int j=0; j<start[i].size(); j++) {
                printf("%d, ", start[i][j]);
            }
            printf("\n");
        }
    }
    else {
        //print impossible if no valid schedule is found
        printf("Impossible\n");
    }
}
```

A2 Programme (Rarity Model)

```
//
//  main.cpp
//  movie_rarity
//
//  Created by IMMC Team 2015017 on 12/5/15.
//  Copyright (c) 2015 IMMC Team 2015017. All rights reserved.
//

#include <iostream>
#include <vector>
#include <queue>
using namespace std;
typedef pair<int, int> ii;
int t[105], bg[105];
bitset<105> hpn, fin;
vector<ii> res[105];
ii el[105];
vector<int> d[105], fw[105], st[105], adjlist[105], p[105], start[105], splt;
int n,e,r,tg;
int curr_n;
bool fnd=false;
priority_queue<ii, vector<ii>, greater<ii> > vec;

void update(int i, int j, int amt, vector<int> &f) {
   for (int x=i; x<=j; x++) f[x]+=amt;
}

bool check(int i, int j, int amt, vector<int> &f) {
   for (int x=i; x<=j; x++) if (f[x]<amt) return false;
   return true;
}

void split(int v, int n_v) {
   for (int i=0; i<p[v].size(); i++) {
      adjlist[p[v][i]].push_back(n_v);
      p[n_v].push_back(p[v][i]);
   }
   for (int j=0; j<adjlist[v].size(); j++) {
      adjlist[n_v].push_back(adjlist[v][j]);
      p[adjlist[v][j]].push_back(n_v);
   }
```

```
    for (int i=0; i<res[v].size(); i++) {
        res[n_v].push_back(res[v][i]);
    }
    splt.push_back(v);
}

void renorm_st(int v) {
    for (int i=0; i<p[v].size(); i++) {
        if (el[v].first<el[p[v][i]].first+t[p[v][i]]) {
            el[v].first=el[p[v][i]].first+t[p[v][i]];
            for (int j=0; j<adjlist[v].size(); j++) {
                renorm_st(adjlist[v][j]);
            }
        }
    }
}

void renorm_end(int v) {
    for (int i=0; i<adjlist[v].size(); i++) {
        if (el[v].second+t[v]>el[adjlist[v][i]].second) {
            el[v].second=el[adjlist[v][i]].second-t[v];
            for (int j=0; j<p[v].size(); j++) {
                renorm_end(p[v][j]);
            }
        }
    }

}

ii esls(int v) {
    vector<ii> s;
    for (int i=1; i<=tg-t[v]; i++) {
        bool z=true;
        for (int k=0; k<res[v].size(); k++) {
            ii temp=res[v][k];
            z=check(i,i+t[v]-1,res[v][k].second,fw[res[v][k].first]);
            if (!z) break;
        }
        if (z) {
            //start time has been found
            int j=i+t[v];
            while (j<=tg) {
                bool avail=true;
```

```
            for (int k=0; k<res[v].size(); k++) {
               if (fw[res[v][k].first][j]<res[v][k].second) {
                  avail=false;
                  break;
               }
            }
            if (!avail) break;
            j++;
         }
         //end time has been found
         s.push_back(make_pair(i,min(j-t[v],tg-t[v])));
         i=j+1;
      }
   }
   for (int i=1; i<s.size(); i++) {
      //if there are multiple possible time periods, split the event into multiple nodes
      split(v,curr_n);
      el[curr_n].first=s[i].first;
      el[curr_n].second=s[i].second;
      curr_n++;
   }
   return s[0];
}

void resolve(int ct, vector<int> st1[], int bg1[], vector<int> fw1[], bitset<105> fin1, bitset<105> hpn1,
priority_queue<ii, vector<ii>, greater<ii> > vec1) {
   while (!vec1.empty() && vec1.top().first==ct) {
      int c=vec1.top().second;
      fin1[c]=true;
      vec1.pop();
   }
   //check if a valid schedule has already been found
   bool b=true;
   for (int i=1; i<=n; i++) {
      if (!fin1[i]) {b=false;}
   }
   if (b && ct<=tg) {
      fnd=true;
      for (int i=1; i<=tg; i++) {
         for (int k=0; k<st1[i].size(); k++) {
            start[i].push_back(st1[i][k]);
         }
      }
```

```
    return;
}
//save all the current information in case backtracking is needed
vector<int> stcpy[105], fwcpy[105];
int bgcpy[105];
bitset<105> hpncpy, fincpy;
for (int k=1; k<=tg; k++) {
    bgcpy[k]=bg1[k];
    for (int l=0; l<st1[k].size(); l++) {
        stcpy[k].push_back(st1[k][l]);
    }
    for (int l=0; l<fw1[k].size(); l++) {
        fwcpy[k].push_back(fw1[k][l]);
    }
    hpncpy[k]=hpn1[k];
    fincpy[k]=fin1[k];
}
//priority queue: sorts nodes according to rarity
priority_queue<ii, vector<ii>, less<ii> > rarity;
for (int i=1; i<curr_n; i++) {
    if (fnd) break;
    if (!hpn1[i] && el[i].first<=ct && el[i].second>=ct) {
        bool pos=true;
        //check resource availability and parents of node
        for (int k=0; k<p[i].size(); k++) {
            if (!fin1[p[i][k]]) {pos=false; break;}
        }
        for (int k=0; k<res[i].size(); k++) {
            if (!check(ct,ct+t[i]-1,res[i][k].second,fw1[res[i][k].first])) {pos=false; break;}
        }
        if (pos && ct+t[i]<=tg) {
            rarity.push(make_pair((tg+1-(el[i].second)),i));
        }
    }
}
while (!rarity.empty()) {
    int i=rarity.top().second;
    rarity.pop();
    priority_queue<ii, vector<ii>, greater<ii> > veccpy;
    vector<ii> add;
    while (!vec1.empty()) {
        add.push_back(vec1.top());
        vec1.pop();
```

```
        }
        for (int k=0; k<add.size(); k++) {
            vec1.push(add[k]);
            veccpy.push(add[k]);
        }
        vec1.push(make_pair(ct+t[i],i));
        for (int k=0; k<res[i].size(); k++) {
            //update resource availability
            update(ct,ct+t[i]-1,-res[i][k].second,fw1[res[i][k].first]);
        }
        //set start time of this event to current time
        st1[ct].push_back(i);
        bg1[i]=ct;
        hpn1[i]=true;
        for (int k=1; k<curr_n; k++) {
            if (k!=i && splt[k-1]==splt[i-1]) {
                //if a single event has been split into multiple nodes, set all the other nodes that are part of the
same event to "finished"
                hpn1[k]=true;
                fin1[k]=true;
            }
        }
        //try to schedule if the node with highest rarity is picked at this time
        resolve(ct,st1,bg1,fw1,fin1,hpn1,vec1);
        for (int k=0; k<105; k++) {
            bg1[k]=bgcpy[k];
            st1[k].clear();
            fw1[k].clear();
            for (int l=0; l<stcpy[k].size(); l++) {
                st1[k].push_back(stcpy[k][l]);
            }
            for (int l=0; l<fwcpy[k].size(); l++) {
                fw1[k].push_back(fwcpy[k][l]);
            }
            hpn1[k]=hpncpy[k];
            fin1[k]=fincpy[k];
        }
        while (!vec1.empty()) {
            vec1.pop();
        }
        while (!veccpy.empty()) {
            vec1.push(veccpy.top());
            veccpy.pop();
```

```
        }

    }
    if (!fnd && ct<tg) {
        //move on to the next time unit if no other nodes can be picked
        resolve(ct+1,st1,bg1,fw1,fin1,hpn1,vec1);
    }
}

int main() {
    //input
    scanf("%d%d%d%d", &n, &e, &r, &tg);
    curr_n=n+1;
    hpn.reset();
    fin.reset();
    for (int i=1; i<=n; i++) {
        scanf("%d", &t[i]);
        splt.push_back(i);
    }
    for (int i=0; i<e; i++) {
        int a,b;
        scanf("%d%d", &a, &b);
        adjlist[a].push_back(b);
        p[b].push_back(a);
    }
    for (int i=1; i<=r; i++) {
        int tp;
        scanf("%d", &tp);
        fw[i].assign(tg+5,0);
        for (int j=0; j<tp; j++) {
            int a,b,val;
            scanf("%d%d%d", &a, &b, &val);
            update(a,min(b,tg),val,fw[i]);
        }
    }
    for (int i=1; i<=n; i++) {
        int types;
        scanf("%d", &types);
        for (int j=0; j<types; j++) {
            int a,b;
            scanf("%d%d", &a, &b);
            res[i].push_back(make_pair(a,b));
        }
```

```
        //find start and end times
        el[i]=esls(i);
    }
    for (int i=1; i<curr_n; i++) {
        printf("%d %d %d\n", i, el[i].first, el[i].second);
    }
    for (int i=1; i<curr_n; i++) {
        //renormalisation
        renorm_st(i);
        renorm_end(i);
    }
    resolve(1,st,bg,fw,fin,hpn,vec);
    if (fnd) {
        //print start times for each node if a valid schedule is found
        for (int i=1; i<=tg; i++) {
            printf("At time %d: ", i);
            for (int j=0; j<start[i].size(); j++) {
                printf("%d, ", start[i][j]);
            }
            printf("\n");
        }
    }
    else {
        //print impossible if no valid schedule found
        printf("Impossible\n");
    }
}
```

Bibliography

[1] IMDb, Source Code (2011), April 2011.

[2] IMDb, Annabelle (2014), October 2014.

[3] Upadhyaya R., International Business Times, Fast and Furious 8: Will Cody Replace Paul Walker? Dwayne Johnson will return; Vin Diesel shares plot updates, April 2015.

[4] Grant D., Observer.com, Ruining the Shot: Would You Walk Through a Production Filming On The Street?, April 2013.

[5] Josie, Urban Times, 10 Low-Budget Films That Made It Big, August 2013.

[6] Carballo C., Dailymail MailOnline, Jackie Chan's Cameraman Chan Kwok-Hung, 51, drowns in Hong Kong while filming the action-comedy Skiptrace that also stars Johnny Knoxville, December 2014.