Code Implementation Report

An exact method for scheduling of the alternative technologies in R&D projects

Mohammad Ranjbar | Morteza Davari

Sid Mahapatra (r0770996)

3rd September, 2020

Introduction

This report is based on implementing the algorithm in the paper of the same name as the title of this report. The design choices will be discussed prior to elucidating the intricacies of the C++ code that accompanies this report. Also note that the results displayed in : https://www.md6712.com/wp-content/uploads/2020/04/opt.txt , will henceforth be referred to as reported results and the entities that it encapsulates will be addressed with ‘reported’ as a prefix.

Design Choices

The paper proposes two variants of the depth-first Branch and Bound algorithm to solve the ATPSP problem, namely, Forward B&B and Forward-Backward B&B. The paper also proposes a preprocessing algorithm to reduce the size of the search space. The preprocessing algorithm divides the alternatives into favourable, unfavourable and undecided sets. The performance of the FB-B&B algorithm might get affected in a negative way if the number of favourable and unfavourable alternatives determined by this procedure is not noticeable and will probably result in an increase in run time without facilitating its ability to obtain an optimal concurrent schedule. Even with a large PTC, the average percent deviation from optimal solution is not enough to consider implementing the preprocessing procedure prior to a FB-B&B algorithm, atleast as far as the scope of this task is concerned. The paper states that the FB-B&B algorithm was tested without the preprocessing procedure as it has a negative impact on the total time. FB-B&B algorithm also requires a schedule deadline which wasn’t available in the testset that accompanies this work. Hence, FB-B&B gets ruled out and F-B&B is chosen as the one to be implemented. Also, note that the F-B&B assumes all alternatives to be favourable and avoids the deployment of a preprocessing algorithm.

The following section will first discuss the functions implemented in the code and then will describe the algorithm implementation followed by a conclusion based on a discussion of the performance.

C++ Code Features

1) Structure Nodes - stores duration, cost, PTS, Early Start time, Early Stop time, Late Start time, Late Stop time of each alternative
2) Topological Sort - Sorts the graph of alternatives in a linear order and is used to calculate and update ES, LS, EF, LF
3) Payoff - Given a (partial) schedule and the nodes structure, it calculates and returns the End-of-Project payoff which a product of the PTC and the sum of costs of all the alternatives in a schedule
4) NonUniqueFinish - Given a (partial) schedule and the nodes structure, it computes and returns a vector of times that are identical to the finish times of at-least two or more alternatives in a (partial) schedule
5) FinishBeforeAt - Given a (partial) schedule, a time t, and the nodes structure, it computes and returns a vector of alternatives of (partial) schedule, which are finished before or at time t
6) Finish Before - Same as the previous one, but considers the alternatives finished before time t but not at time t
7) FinishAt - Same as the previous one, but considers the alternatives finished at time t, but not before or after time t
8) UniqueFinish - Just the mirror of NonUniqueFinish. It returns a vector of times that are unique to the finish times of alternatives in a (partial) schedule
9) Enpv - This function represents the objective function and indicates the eNPV of a schedule. Given a (partial) schedule and the nodes structure, it computes and returns the eNPV of the schedule based on the non-linear integer programming model / equation as stated in section 3.1 of the paper
10) SA - Given a (partial) schedule, this function returns a list of the scheduled alternatives as a vector
11) RA - Given a (partial) schedule, this function returns a list of the non-scheduled / remaining alternatives as a vector
12) Intersect - A basic function that returns a vector of elements common to elements of two different vectors
13) UB - Given a (partial) schedule, nodes structure, 2D vector of successors and 2D vector of predecessors, this function first updates the ES,EF,LS,LF of the

remaining alternatives based on the start and finish times of scheduled alternatives and then computes the Upper Bound of the given schedule based on the equation stated in section 4.2 of the paper

14) EL - Returns a vector of Eligible alternatives based on the (partial) schedule and the predecessor matrix.
15) EST - Returns the earliest start time of a particular alternative in a (partial) schedule, by searching for the maximum finish time amongst the scheduled alternatives in its predecessor matrix
16) Possible_Start_Times - Returns the possible start times of a particular alternative (elem) based on the (partial) schedule, its EST, its EFT (EST + duration) and the nodes structure. It uses the following four pipelines
    a) Includes start times of scheduled alternatives if the start time is greater than EST
    b) Includes finish times of scheduled alternatives if the finish time is greater than EST
    c) Includes (start times of scheduled alternatives - Duration of elem), if the start time is greater than EFT
    d) Includes (finish times of scheduled alternatives - Duration of elem), if the finish time is greater than EFT
    e) Avoids duplicates in the vector and returns a sorted vector.

Main and the Algorithm

The Main function computes the successor and predecessor matrix based on the inputs. It also computes the initial EF, LF, ES, LS of all alternatives based on the CPM method.

The algorithm implementation starts with initializing a vector called frontier and declaring another vector called explored. The frontier vector stores the open nodes of a tree while the explored vector stores the nodes that have already been explored / fathomed. Each node is represented by a vector of fixed length. The length is based on the number of alternatives in the project. An index of the vector corresponds to the alternative of the same number as the index, and the value stored in that index corresponds to the start times (or sometimes the finish times) of that particular alternative. The non-scheduled alternative gets assigned the value -1 in its corresponding index of the node vector. For example, the following node vector with 4 alternatives assuming the project has 4 alternatives, [0,2,1,-1] represents that Alt 1 has a start time of 0, Alt 2 has a start time of 2, Alt 3 has a start time of 1 and Alt 4 is unscheduled so far. This in turn helps the algorithm maintain Dominance rule 1. The frontier is initialized with the value 0 in its first index denoting start time of the dummy start activity, and -1 in the rest.

Step 0 : If Frontier is empty, report failure

Step 1: Pop the vector from the front of Frontier (as per the Depth First rules)

Step 2: If the vector exists in the Explored list, then go back to step 1. This check is performed to enforce Dominance rules 1 and 2

Step 3: Calculate UB of the vector

Step 4: If UB < LB (the best solution found so far), put vector in explored list and go to step 1

Step 5: If UB > best Upper Bound found so far, then make UB the best Upper Bound

Step 6: Calculate eNPV of the vector. If it is greater than LB, store it in LB

Step 7: Store the vector in the explored list

Step 8: Check if a solution has been found, if yes, then display it with its eNPV and UB

Step 9: If a solution has not been found then branch. Compute the list of Eligible alternatives based on the vector and the predecessor matrix

> 9.1: Pop the front element of the EL list
>
> 9.2: Find the Earliest start times (EST) and EFT of the alternatives(from 9.1) based on the vector and the predecessor matrix
>
> 9.3: Based on the vector, 9.2 and 9.1, find all the Possible Start Times of the alternative
>
> > 9.3.1: Pop the front element of the PST list
> >
> > 9.3.2: Store the front element (9.3.1) in the index of the vector corresponding to element in 9.1

```
                9.3.3: Push the modified vector to the front of frontier
                9.4.4: Repeat the steps till the PST list is empty
        9.4: Repeat the Steps till the EL list is empty
Step 10: Repeat the steps till the frontier is empty or a solution is found
```

Results and Discussion

The eNPV and UB of the optimal solution needs to be the same and it is the same. The eNPV and UB of the optimal solution in the reported results can be directly computed using the code, however they don't match with the ones in the reported results. The eNPV and UB of the optimal solution matches with the code computed eNPV and UB of the schedule in the reported results, but the schedule does not. To be precise, if the optimal schedule in the reported results displays a eNPV or UB of 'x', then upon feeding the same schedule into the code to compute its eNPV or UB, produces the value 'y'. However, the optimal schedule found by the code also has the value 'y' as its eNPV and UB. Hence, it's safe to say that the schedule displayed in the reported results and the schedule produced by the code are both optimal as they both have the same eNPV and UB and if one of them has been declared as optimal, the other has to be too. This is justifiable from the fact that section 3.2 in the paper states that according to the concurrency property, if all alternatives of a given schedule are concurrent, then the concurrency property is satisfied for the schedule and that there is one or more optimal concurrent schedule for each instance of the ATPSP. In other words, both the schedule from the reported results and the schedule from the code are both optimal and concurrent. F-B&B is quite quick in finding a concurrent solution, and as reported in the paper, it finds an optimal solution at least 55 out of 60 times. The only disadvantage being its inability to identify or process unfavourable alternatives, unlike FB-B&B as FB-B&B especially with a preprocessing procedure, has the ability to plan a schedule the F-B&B way for favourable alternatives and schedule the unfavorable alternatives starting from the deadline and working its way backwards (with a gap in between). As a conclusion, this code implementation uses F-B&B to successfully find an optimal schedule with the highest possible eNPV 57 out of 60 times wrt the test cases.