

American University in Cairo

## AUC Knowledge Fountain

---

Theses and Dissertations

Student Research

---

2-1-2016

### Cash flow optimization for construction engineering portfolios

Gasser Galal Ali

Follow this and additional works at: <https://fount.aucegypt.edu/etds>

---

#### Recommended Citation

##### APA Citation

Ali, G. (2016). *Cash flow optimization for construction engineering portfolios* [Master's Thesis, the American University in Cairo]. AUC Knowledge Fountain.

<https://fount.aucegypt.edu/etds/589>

##### MLA Citation

Ali, Gasser Galal. *Cash flow optimization for construction engineering portfolios*. 2016. American University in Cairo, Master's Thesis. *AUC Knowledge Fountain*.

<https://fount.aucegypt.edu/etds/589>

This Master's Thesis is brought to you for free and open access by the Student Research at AUC Knowledge Fountain. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact [thesisadmin@aucegypt.edu](mailto:thesisadmin@aucegypt.edu).

The American University in Cairo  
School of Sciences and Engineering  
Department of Construction Engineering

# CASH FLOW OPTIMIZATION FOR CONSTRUCTION ENGINEERING PORTFOLIOS

A thesis submitted to the School of Sciences and Engineering in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN CONSTRUCTION ENGINEERING

To The

Construction Engineering Department

By

GASSER GALAL ALI

BACHELOR OF SCIENCE IN CONSTRUCTION ENGINEERING

UNDER THE SUPERVISION OF

DR. A. SAMER EZELDIN

CHAIRMAN AND PROFESSOR

CONSTRUCTION ENGINEERING DEPARTMENT

THE AMERICAN UNIVERSITY IN CAIRO, EGYPT

DECEMBER 2016

---

# Abstract

One of the main issues in construction projects is finance; proper cash-flow management is necessary to insure that a construction project finishes within time, on budget, and yielding a satisfying profit. Poor financial management might put the contractor, or the owner, in a situation where they are unable to finance the project due to insufficient liquidity, or where they are engaged in excessive loans to finance the project, decreasing the profit, and even creating unsettled debts. Engagement with a portfolio of large construction projects, like infrastructure projects, makes attention to finance more critical, due to large budgets and long project durations, which also requires attention to the time value of money when the project spans over many years and the work environment has a high inflation rate.

This thesis aims at the analysis and optimization of the cash-flow request for large engineering portfolios from the contractor's point of view. A computational model, with a friendly user interface, was created to achieve that. The user is able to create a portfolio of projects, and create activities in them with different relationship types, lags, constraints, and costs, as similar to commercial scheduling software. Parameters necessary for the remuneration are also considered, which include the down payment percentage, duration between invoices, duration for payment, retention percentage, etc. The model takes into consideration the time value of money, calculated with an interest rate assigned to the projects by the user; this could be the inflation rate or the (Minimum Attractive Rate of Return) MARR of the contractor. Optimization is done with the objective of maximizing the Net Present Value (NPV) for the projects as a whole, discounted at the start of the portfolio. The variables for the optimization are lags that are assigned for each activity, which, after rescheduling, delays the activities after their early start with the value of those lags, and thus creates a modified cash flow for the project. Optimization of those variables, within scheduling constraints results in a near-optimum NPV. Verification of the model was done using sets of portfolios, and the validation was done using an actual construction portfolio from real life. The results were satisfactory and matched initial expectations. The NPV was successfully optimized to a near optimum. A sensitivity analysis of the model was conducted and it showed that the model behaves as expected for different inputs. A time test was performed, taking into consideration the effect of the size and complexity of a portfolio on the calculation time for the model, and it showed

---

that the speed was satisfactory, though it should be improved. Overall, the conclusion is that the model delivers its goal of maximizing the Net Present Value of a large portfolio as a whole.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Problem Statement . . . . .	6
1.3 Objective . . . . .	6
1.4 Scope of Work . . . . .	6
1.5 Research Methodology . . . . .	7
1.6 Detailed Outline . . . . .	7
<b>2 Literature Review</b>	<b>9</b>
2.1 Project Portfolio management . . . . .	9
2.1.1 Project and Portfolio Planning Cycle: Project-based Management for the Multi-project Challenge . . . . .	9
2.1.2 Multi-criteria Financial Portfolio Risk Management for Interna- tional Projects . . . . .	11
2.1.3 Risk Management Applied to Projects, Programs, and Portfolios . .	12
2.2 Cash Flow Analysis . . . . .	12
2.2.1 Profit Measures for Construction Projects . . . . .	13
2.2.2 Systems Analysis of Project Cash Flow Management Strategies . .	14
2.2.3 Analyzing the Impact of Negative Cash Flow on Construction Per- formance in The Dubai Area . . . . .	14
2.2.4 Financial Management of the Construction Projects: A Proposed Cash Flow Analysis Model at Project Portfolio Level . . . . .	15
2.3 Optimization and Resource/Finance Based Scheduling . . . . .	17
2.3.1 Optimization of Resource Allocation and Leveling Using Genetic Algorithms . . . . .	18
2.3.2 Expanding Finance-Based Scheduling to Devise Overall-Optimized Project Schedules . . . . .	20
2.3.3 Heuristic Method for Multi-Project Finance-Based Scheduling . . .	20

2.3.4	Scheduling Resource-Constrained Projects with Ant Colony Optimization Artificial Agents . . . . .	20
2.3.5	Multi-objective Optimization of Resource Leveling and Allocation during Construction Scheduling . . . . .	21
2.3.6	Multi-objective Evolutionary Finance-Based Scheduling: Entire Projects' Portfolio and Individual Projects within a Portfolio . . . . .	22
2.3.7	Improved Genetic Algorithm Finance-Based Scheduling . . . . .	23
2.3.8	Fast and Near-Optimum Schedule Optimization for Large-Scale Projects	24
2.3.9	Enhanced Trade-off of Construction Projects: Finance-Resource-Profit . . . . .	24
2.3.10	Finance-based Scheduling using meta-heuristics: discrete versus continuous optimization problems . . . . .	24
2.4	Outcomes From Literature Review . . . . .	27
<b>3</b>	<b>Model Development</b>	<b>29</b>
3.1	Assumptions . . . . .	29
3.2	Model Inputs and Outputs . . . . .	30
3.3	Input from Primavera . . . . .	31
3.4	Programming Language and Packages Used . . . . .	32
3.5	Database . . . . .	33
3.6	Scheduling Calculations . . . . .	35
3.7	Cash Flow Calculation . . . . .	40
3.8	Time Value of Money Calculations . . . . .	41
3.9	Optimization . . . . .	42
3.10	Graphical User Interface (GUI) . . . . .	44
<b>4</b>	<b>Results and Discussion</b>	<b>55</b>
4.1	Verification . . . . .	55
4.1.1	Verification Method . . . . .	55
4.1.2	Verification Results . . . . .	56
4.1.3	Verification Discussion . . . . .	56
4.2	Sensitivity Analysis . . . . .	65
4.2.1	Sensitivity Analysis Method . . . . .	65
4.2.2	Sensitivity Analysis Results . . . . .	65
4.2.3	Sensitivity Analysis Discussion . . . . .	65
4.3	NPV Improvement Test . . . . .	68
4.3.1	NPV Testing Method . . . . .	68
4.3.2	NPV Testing Results . . . . .	68
4.3.3	NPV Testing Discussion . . . . .	68
4.4	CPU Time Test . . . . .	69

---

4.4.1	CPU Time Test Method . . . . .	69
4.4.2	CPU Time Test Results . . . . .	69
4.4.3	CPU Time Test Discussion . . . . .	70
4.5	Validation . . . . .	71
4.5.1	Validation Method . . . . .	71
4.5.2	Validation Results . . . . .	71
4.5.3	Validation Discussion . . . . .	72
4.6	Validation with Updated Schedule . . . . .	81
4.6.1	Validation Method . . . . .	81
4.6.2	Validation Results and Discussion . . . . .	81
<b>5</b>	<b>Conclusion and Recommendations</b>	<b>83</b>
5.1	Conclusion . . . . .	83
5.1.1	Model and GUI . . . . .	83
5.1.2	CPU Time . . . . .	84
5.1.3	Verification . . . . .	84
5.1.4	Validation . . . . .	84
5.1.5	Optimization Algorithm . . . . .	84
5.1.6	Sensitivity Analysis . . . . .	85
5.2	Limitations . . . . .	85
5.3	Recommendations . . . . .	86
	<b>Appendices</b>	<b>91</b>
	<b>A Python Code</b>	<b>93</b>





# List of Figures

1.1	Relationship between the owner, contractor, and engineer in a traditional delivery method. . . . .	2
1.2	Cash-in Cash-out distribution. . . . .	2
1.3	Typical payment method in construction projects. . . . .	3
1.4	Example of a time-line showing cash flow in a construction project. . . . .	4
1.5	Typical Cumulative Cash-In Cash-Out Curves for a Construction Project. . . . .	4
1.6	Flow Chart for Items included in The Price . . . . .	5
2.1	Multiple Project Planning Phase as shown by Platje et al For A Research and Development Programme(Platje, Seidel, and Wadman, 1994) . . . . .	10
2.2	System dynamics model for project cash flow management (Cui, Hastak, and Halpin, 2010). . . . .	15
2.3	Gantt chart of the portfolio studied by Purnus and Bodea (Purnus and Constanta-Nicoleta, 2015) . . . . .	16
2.4	cash flow of the portfolio(Purnus and Constanta-Nicoleta, 2015) . . . . .	17
2.5	Finance of the portfolio (Purnus and Constanta-Nicoleta, 2015) . . . . .	17
2.6	Genetic Algorithm levelling algorithm as proposed by Hegazy (Hegazy, 1999) . . . . .	19
2.7	Optimization model done by Jun et al (Jun and El-Rayes, 2011) . . . . .	21
2.8	Computational flow for the strength Pareto evolutionary algorithm(Abido and Elazouni, 2011b) . . . . .	23
2.9	Flow chart of the chromosome-repairing GA (Alghazi, Elazouni, and Selim, 2013) . . . . .	26
3.1	Flowchart for the Scheduling Front-Run . . . . .	38
3.2	Flowchart for the Scheduling Back-Run . . . . .	39
3.3	Flowchart for The Cashflow calculation . . . . .	40
3.4	Example of an optimization trial . . . . .	42
3.5	Flowchart of the optimization process . . . . .	43
3.6	Graphical User Interface (GUI) on start-up . . . . .	44
3.7	GUI: File Menu . . . . .	45
3.8	GUI: Create New Project Window . . . . .	46

3.9	GUI: Create New Activity Window . . . . .	46
3.10	GUI: Create New Activity Window . . . . .	47
3.11	GUI: Create New Relationship Window . . . . .	47
3.12	GUI: Portfolio Menu . . . . .	48
3.13	GUI: Projects Menu . . . . .	48
3.14	GUI: Activities Menu . . . . .	49
3.15	GUI: Calculations Menu . . . . .	49
3.16	GUI: Plot Menu . . . . .	50
3.17	GUI: Activities Table . . . . .	50
3.18	GUI: Portfolio Table . . . . .	51
3.19	GUI: Projects Table . . . . .	51
3.20	GUI: Gantt Chart . . . . .	52
3.21	GUI: Overdraft Plot . . . . .	52
3.22	GUI: Optimized Overdraft Plot . . . . .	53
3.23	GUI: Optimized Gantt Chart . . . . .	53
4.1	Summary of the five portfolios used and their project Gantt charts . . . . .	58
4.2	Summary of the five portfolios used and their project Gantt charts . . . . .	59
4.3	Gantt charts for the verification projects . . . . .	60
4.4	Optimized Gantt charts for the verification projects . . . . .	61
4.5	Optimization trials for each on the 5 portfolios . . . . .	62
4.6	Optimized Cash Flow for the Portfolios . . . . .	63
4.7	Optimized Overdraft for the Portfolios . . . . .	64
4.8	Cost Sensitivity Analysis . . . . .	66
4.9	Interest Rate Sensitivity Analysis . . . . .	66
4.10	Overlay of The Sensitivity Analysis Results for Interest Rate and Cost . . . . .	67
4.11	Overlay of The Sensitivity Analysis Results for Interest Rate and Cost in percentage increase . . . . .	67
4.12	Histogram of Improvement in NPV for the trials. . . . .	69
4.13	CPU Time Vs. Number of Activities + Number of Relationships . . . . .	71
4.14	Summary of the Portfolio used for validation . . . . .	73
4.15	Portfolio Gantt Chart . . . . .	74
4.16	Portfolio Gantt Chart . . . . .	75
4.17	Optimization trials for the validation . . . . .	76
4.18	Optimized Cash Flow . . . . .	77
4.19	Optimized Overdraft . . . . .	78
4.20	Optimized Cash Flow for the validation Portfolio . . . . .	79
4.21	Optimized Overdraft for the validation Portfolio . . . . .	80
4.22	Portfolio Gantt Chart . . . . .	82

# List of Tables

2.1	Projects and portfolio contract price as studied by Purnus and Bodea (Purnus and Constanta-Nicoleta, 2015) . . . . .	16
4.1	Correlations for CPU time tests . . . . .	70
4.2	Projects used for the validation . . . . .	72



# List of Abbreviations

**ES** Early Start

**EF** Early Finish

**LS** Late Start

**LF** Late Finish

**OS** Optimized Start

**OF** Optimized Finish

**TF** Total Float

**FF** Free Float

**PV** Present Value

**FV** Future Value

**NPV** Net Present Value

*i* Interest Rate

**IRR** Internal Rate of Return

**MARR** Minimum Attractive Rate of Return



# Chapter 1

## Introduction

This chapter will provide a background on the topic of cash flow analysis, then it will provide the problem statement, the scope of work, the methodology followed, and finally detailed outline of the thesis.

### 1.1 Background

Just as other businesses operating in any field, a contracting company has to make profit, which means that it has to have strategic goals that are reasonable in light of future risks and resource constraints. A construction project is an investment; the contractor is paying the expenses for the construction and receiving the revenues in form of invoices from the owner, which means that the contractor will typically be financing the project in some durations, as an overdraft. Revenues are received for monthly invoices issued by the contractor. The full revenue, including profit or loss, is finalized with the final payment from the owner at the end of the project, or , in case of disputes, after the dispute resolution.

Figure 1.1 shows the relationship between major stakeholders in a construction project in a traditional delivery method, where the owner enters into contractual agreement with the contractor, and the engineer (the consultant), separately. There is a non contractual relationship between the contractor and the engineer, because the engineer supervises and inspects the work, and also approves drawings, materials, and invoices.

During the project, the contractor pays the expenses of the construction work, which is the Cash-out from the point of view of the contractor, and, as shown in Figure 1.2, the contractor receives payments for the work done for each invoice, which is typically issued monthly. These payments are the Cash-in from the point of view of the contractor. The amount of payments is calculated as the direct cost multiplied by a mark-up. The calculation of the price can have many forms and calculation methods. The general idea, however, is that the price of a product should include the cost of the product, plus an amount for profit, plus overhead or indirect cost which is the cost of doing business, plus



an amount added for risk. This can be summed in what is shown in Figure 1.2, such as  $Price = Direct_{Cost} + Profit + Contingency + Indirect_{Cost\ and\ Overhead}$ .

Payments are received once the invoices issued by the contractor are approved by the

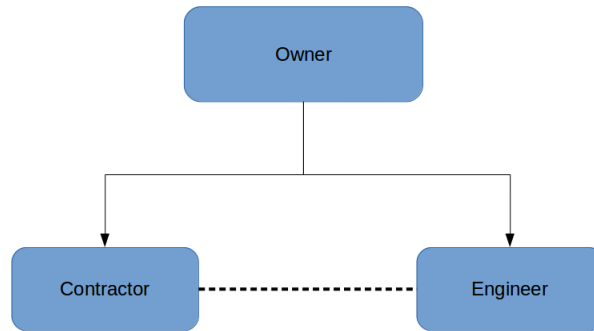


Figure 1.1: Relationship between the owner, contractor, and engineer in a traditional delivery method.

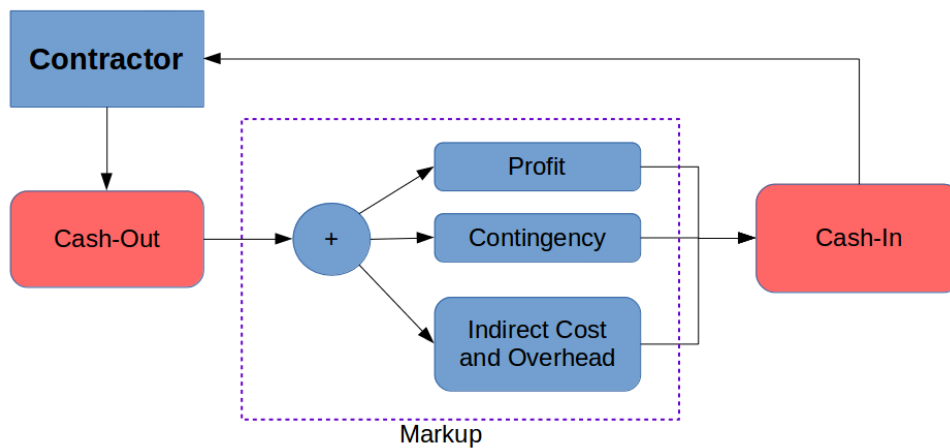


Figure 1.2: Cash-in Cash-out distribution.

engineer, according to the time bars shown in Figure 1.3 which shows the typical general case in a project. This process should be agreed and written in the contract between the employer and the contractor, as well as the time interval between invoices, allowed time for the engineer to approve, and the deadline for the engineer to pay. This whole process can be more generalized, as shown in Figure 1.4, where the downpayment and the retention (if applicable in a project) are included. Due to the nature of the cash flow in construction projects, there is a delay between the cash out for the contractor, where payments are made by the contractor for the work being done, and the actual receipt of payment as per the submitted invoice for that work, which is the cash in. This duration includes the time for approval of the invoice by the engineer, plus the duration until the owner sends actual payment. This raises problems concerning liquidity and profitability because the contractor’s cash flow will most probably be in the red for some durations during the project. To answer this issue, analysis of the cash-in cash-out curves

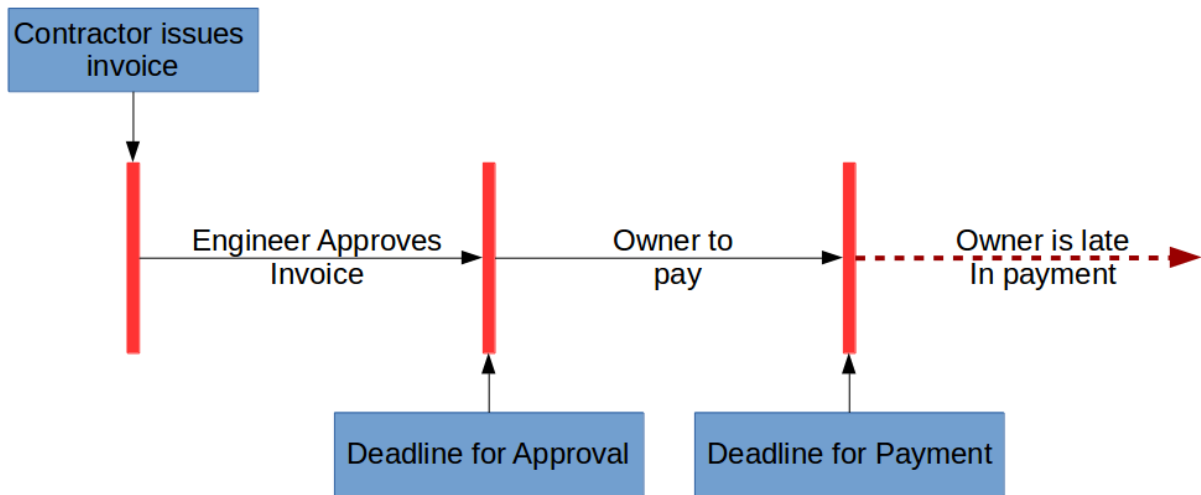


Figure 1.3: Typical payment method in construction projects.

is required. An example of these curves for a construction project Shown in Figure 1.5, is . The cash-out is typically an S-shaped curve, and it accounts for the cumulative direct costs up to a certain point in time. The direct costs mentioned include material, labor and equipment costs. Therefore, the cumulative cash-out curve at the end of the project equals the total cost of the project from the point of view of the contractor. The cash-in curve is a stepped curve where each rise or step in the curve means the contractor has received payment from the owner. The first step will occur at the start of the project if there is a down-payment. After that, each step means a payment of an invoice, then , at the end of the project the final payment including the retention if applicable. At the end of the project, the cumulative cash-in should equal the contract price. As shown in Figure 1.6, the total cost accounts for the direct and indirect costs. The former was explained earlier as the expenses for labor, material, and equipment. While the indirect cost is any expense indirectly related to a certain activity but relevant to the site, like generators or equipment or fuel, and also the overhead of the company, where it might include rent and expenses for an office or headquarters.

The previously mentioned mark-up percentage is a factor that accounts for the profit and risk, and may in some cases consider indirect costs. When choosing the mark-up, which is done during tendering, attention should be given to the companies **Minimum Attractive Rate of Return (MARR)**, project risks, inflation, currency, finance, ...etc.(Peterson, 2009)

Further analysis of the cash flow curves by calculating the difference between the cash-out and the cash-in yields the overdraft, which indicates the finance of the project. In other words, if the cumulative cash-out is higher than the cumulative cash-in at some point in time, it means that the contractor has financed more cash into the project than the cash received from invoices and down-payment. The opposite case, where the cumulative

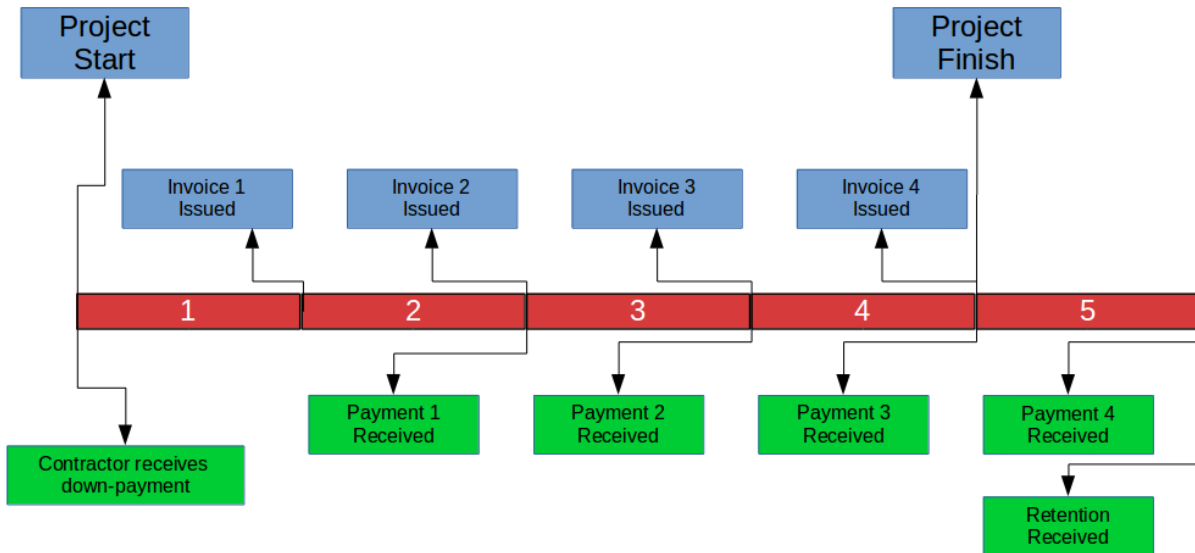


Figure 1.4: Example of a time-line showing cash flow in a construction project.

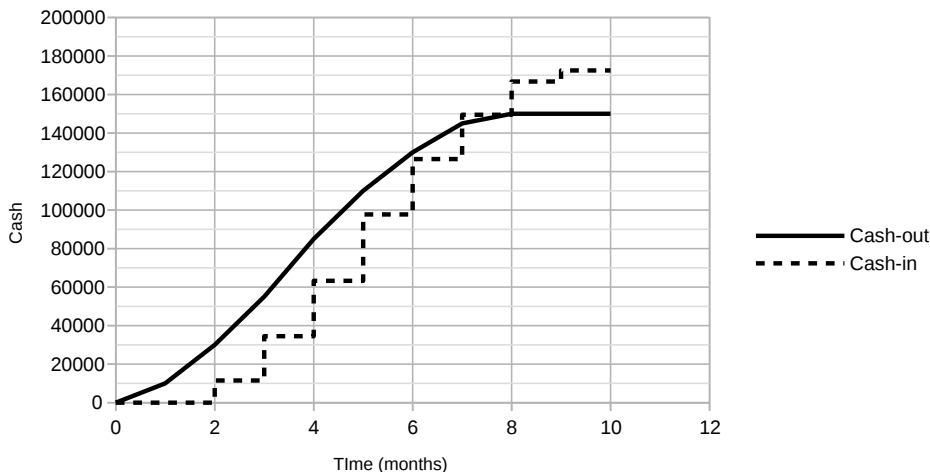


Figure 1.5: Typical Cumulative Cash-In Cash-Out Curves for a Construction Project.

cash-in of higher than the cumulative cash-out, means that the contractor has received more money that the cost incurred, which should be the case at the end of the project, provided that the project is profitable.

This sums up the cash flow analysis of a construction project. But, of course, a contracting company has more than one project in progress or under analysis for possible future bidding. This introduces the concept of **Project Portfolio Management (PPM)**. PPM is the centralized management of the enterprise’s company for a group of projects, this ensures better resource and risk allocation between projects. As analysis at the project-level may not correctly reflect the risks at the enterprise-level, a multiple projects approach, however, would be more fit. When analysing the cash flow for a portfolio as a whole, there can be further detailed analysis of the company’s profitability, liquidity, and expected risks, which ensures better decisions and strategy by the contractor. (Purnus

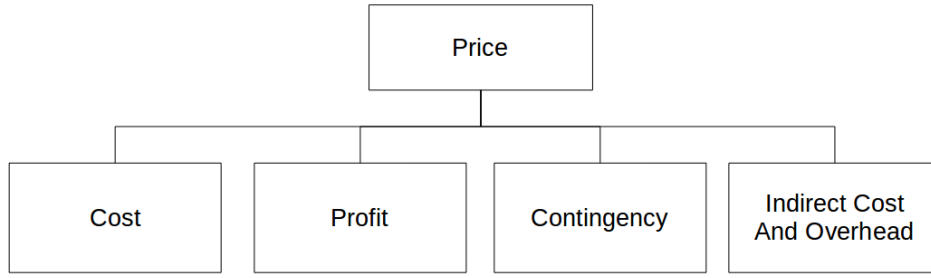


Figure 1.6: Flow Chart for Items included in The Price

and Constanta-Nicoleta, 2015) (Pinto, 2010).

## 1.2 Problem Statement

The Contractor needs to calculate and analyze the cash flow at the portfolio level. The analysis at a portfolio level is needed because it aims at the success of the company's profile as a whole, while analysis at the project level would aim at the success of each individual project separately, which may not result in the company's goals as a whole. This is especially important when resources are shared between projects and limited. Decisions based on a portfolio level assessment may, for example, result in a low profit for a project deliberately, or even a loss, in order to maximize the benefits from another project. Such analysis should provide information on the overdraft, liquidity needed, and profitability at the enterprise level to be able to balance the available resources and cash between multiple projects. This analysis needs to account for inflation and time value of money for proper prediction of the future cash flow needs. Therefore, there is a need for a computational model that can provide such analysis as well as optimize the cash flow request for a portfolio of construction projects.

## 1.3 Objective

This thesis aims at the analysis and optimization of the cash-flow request for large engineering portfolios from the contractor's point of view. A computational model, with a friendly user interface, was created to achieve that. The objective of the optimization is to maximize the Net Present Value of the cash flow from the point of view of a contractor.

## 1.4 Scope of Work

The scope of work of this thesis is as follows:

- Develop a computational model for the analysis and optimization of cash flow for construction engineering portfolios. The model needs to account for:
  - Interest Effect
  - The time value of money
  - Interaction with Oracle Primavera
- Develop a friendly graphical user interface for the model
- Verification the model using sets of randomly generated projects
- Validation the model using an actual real-life portfolio

## 1.5 Research Methodology

This thesis has the following research methodology:

**Step 1: Model Development:** The model was developed in Python, and it includes a friendly user interface.

**Step 2: Verification:** Verification was done to ensure that the model performs correctly

**Step 3: Sensitivity Analysis:** A sensitivity analysis was done to analyze the effect of different parameters on the final results. This was done to ensure that the model performs correctly as well.

**Step 4: CPU Time Test:** A test on the CPU time needed to solve portfolios of different sizes was done to measure the relation between the CPU time and the complexity of projects, and to ensure that the model performs within a satisfactory time.

**Step 5: Validation:** A validation was done using a very large and real construction portfolio. This was done to ensure that the model performs correctly within a real-life work-flow. Another validation was also done on an updated project to test the use of the model for controlling the cash flow of projects.

## 1.6 Detailed Outline

The synopsis of this work is as follows:

**Chapter 1 Introduction** This is the introduction, which is the current chapter, has introduced a background summary of the field targeted. A problem statement and a scope of work has been declared as well.

**Chapter 2 Literature Review** This chapter will cover a number of previous research works in the fields of portfolios, financial analysis, time-cost trade-off, and resource-based and financial-based scheduling.

**Chapter 3 Model Development** This chapter shows the development of the model.

**Chapter 4 Results and Discussion** This chapter shows the results and discussion of the results of the model. This includes the verification, validation, sensitivity analysis, and CPU time analysis.

**Chapter 5 Conclusion and Recommendations** This chapter concludes the thesis, discusses the main outcomes, and provides some recommendations for future research.



# Chapter 2

## Literature Review

The literature review will attempt to cover a range of previous research in the fields of project portfolio management, and cash-flow and resources analysis and optimization.

### 2.1 Project Portfolio management

There is a number of research in the field of construction portfolios including: (Platje, Seidel, and Wadman, 1994) where the concept of portfolio management was introduced and a practical framework was created; (Han et al., 2004) which focused on the financial risk management for international portfolios and highlighted its significance to the success of a contractor on a corporate level, which was also discussed by (Sanchez et al., 2009), where a research gap in that area, in comparison with project-level risk management, was highlighted; (Purnus and Constanta-Nicoleta, 2015) presented a complete case study for cash flow analysis for a portfolio. The studies range between general studies, financial analysis, risk analysis, project selection, and others. This section will attempt to cover a selection of them.

#### 2.1.1 Project and Portfolio Planning Cycle: Project-based Management for the Multi-project Challenge

(Platje, Seidel, and Wadman, 1994) published a paper regarding the challenge of multi-project management. The research is somewhat inclined towards Research and Development projects, but the concepts are also applicable in the construction industry. The authors present an implementation of the traditional Plan-Do-Check-Action management cycle in the multiple-projects environment, and a case study on an research and development programme in a company, which has the cycle shown Figure 2.1. The cycle is based on three parties in the organization, which is shown in Figure 2.1. Those are:



**Project Leaders - Project Managers** who are responsible for realizing the project goals and resource allocation.

**Department Heads - Resource Managers** who are responsible for efficiency and effectiveness of resources use, as well as quality control.

**Management - Programme Directors** who are responsible for setting and realizing of overall programme goals

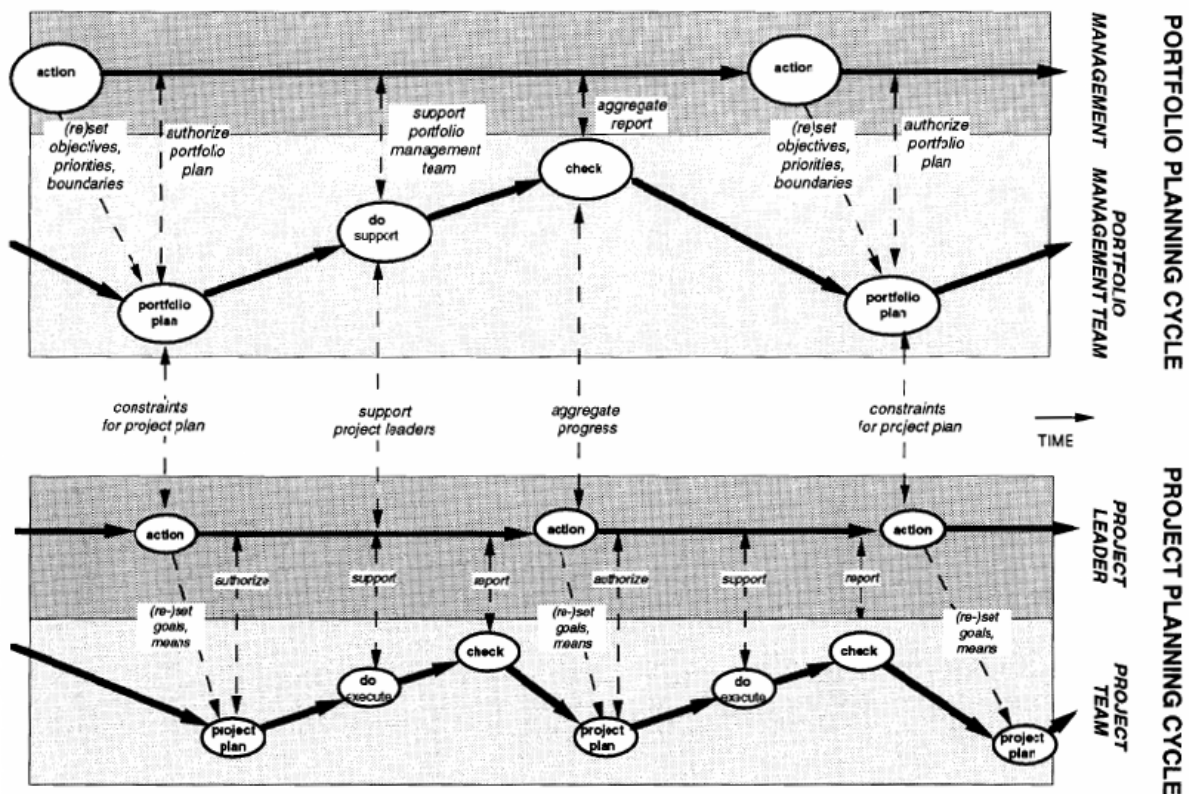


Figure 2.1: Multiple Project Planning Phase as shown by Platje et al For A Research and Development Programme(Platje, Seidel, and Wadman, 1994)

The Operation Breakdown Structure (OBS) and the Project Breakdown Structure (PBS) are therefore interlinked. The cycle is therefore as follows:

**Action** The management sets the priorities.

**Plan** The team develops a plan in an iterative process between management, project leaders, and department heads, as well as the projects’ sponsors - owners.

**Do** The team members execute the plan.

**Check** The team members report to the management for monitoring.

**Action** The management takes corrective actions and update as required.

This multi-project approach has the benefit of better resource allocation between projects, and aims towards organizational goals as a whole, instead of project constrained success. However, communication is more complicated. Communication and delegation should be properly and clearly planned.

### 2.1.2 Multi-criteria Financial Portfolio Risk Management for International Projects

In a paper by (Han et al., 2004), the authors studied the portfolio financial risk assessment for international projects. The goal was to introduce a framework of project-selection for multinational contractors, integrating the risks at the project level and the corporate levels. The authors note that a profit-oriented goal at the project-level does not reflect the overall risks at the corporate level, and goals of the company. The risks in a portfolio are distributed, reflecting the state of mind of *"not keeping all of your eggs in one basket"*. The return on the portfolio is a weighted average of the return on the individual projects. The authors use the Net Present Value (NPV) to reflect the portfolio's expected return, where the expected return is a three-point approximation of the worst, normal, and best expected NPV. The paper uses the Value at Risk (VaR), which is the worst expected loss of the portfolio within a given confidence interval, in an attempt to capture the risk. The paper introduces a decision model for portfolio selection for international contractors, incorporating three parts; financial risk analysis for cash flow analysis and estimating multi-criteria values such as NPV, Var, and efficiency (ROI), part2 to evaluate and integrate these values, and part 3 for the selection of the optimum portfolio. A case study was done on a list of 7 projects in 7 different countries, and a set of 5 possible projects resulted. In summary, the authors conclude that; the NPV, ROI, and VaR can reflect the benefits and risks of a portfolio; a higher profit ratio does not always guarantee a higher NPV; The NPV is essential and lowered the deviation and the VaR; A company can make a more inclusive decision based of the selection within a portfolio as a whole rather than selection of projects on individual basis. The authors note the limitation of this research is that it is applicable to large international contractors, application to medium to small contractors is recommended for future research. Another recommendation is to research into incorporating the risks at the project and the corporate level in a sequential manner, and the take into consideration current risks to incorporate a contingency against total risk exposure.

### 2.1.3 Risk Management Applied to Projects, Programs, and Portfolios

(Sanchez et al., 2009) did a thorough literature review paper on risk management at three levels; Project, Program, and Portfolio. The authors state the risk assessment at those levels are interdependent and should be co-ordinated. However, in practice, project risk management has been linked to the individual project level with less attention to the other levels, which doesn't reflect the strategic goals of the company. The authors show that, despite large literature, there is a gap between risk management applied to project level, and the organizational level. The authors expose some area of open research gaps; there is a need to implement continuous control and monitoring, this is needed for all three levels. Another gap in all levels taking into account vulnerabilities. Some other areas for portfolio and program are adapted from the project level analysis, but research written specifically for these upper levels is not complete. It should be noted however that a all-around generic solution may not be satisfactory, as each level's needs and criteria is different. Overall, the authors point at several open research areas are the program and portfolio risk management.

## 2.2 Cash Flow Analysis

This section shall cover some of the research in the field of financial analysis of construction projects. There are many research works in that field; to count a few: (Au and Hendrickson, 1985) which introduces cash-flow analysis and profit calculations for construction projects; (Kaka and Price, 1993) which focused in the modeling and prediction of the cost curves for contractors, which was also studied by (Hwee and Tiong, 2002) in combination with risk analysis using a number factors that affect the cash flow; cash flow forecasting for contractors was also analyzed by (Park, Han, and Russell, 2005); (Odeyinka and Kaka, 2005) evaluated the contractor's satisfaction with payment terms, and their impact on the construction cash flow by conducting surveys; (Khosrowshahi, 2007) continued the research into cash flow forecasting by implementing a decision making model for construction cash flow management on the corporate level; (Gorog, 2009) presented a comprehensive and copyrighted model for the analysis and control of cash flows for construction project, to be used by contractors; (Cui, Hastak, and Halpin, 2010) presented a system dynamics model for the project cash flow management, and analyzing different financial strategies. (Jiang, Issa, and Malek, 2011) presented a Pareto optimality multi-objective model, for the analysis of cash flows and financial strategies, to be used as a decision making tool; (Kishore, Abraham, and Sinfield, 2011) used fuzzy logic systems for cash flow analysis, for portfolios; (Lee, Lim, and Arditi, 2012) presented a stochastic financing analysis for construction projects, where simulation of projects is done in

Matlab using stochastic schedules, to handle uncertainties in activity durations and costs, which was also done by (Maravas and Pantouvakis, 2012); (Huang et al., 2013) produced a decision making system for financial prequalification of contractors using simulation; (Zayed and Liu, 2014) studied the complexity of financial management of construction projects and created a list of the most relevant financial parameters; finally, (Purnus and Constanta-Nicoleta, 2015) presented a complete insight into cash flow analysis, which proved to be an excellent reference. This section will attempt to cover a number of them,

### 2.2.1 Profit Measures for Construction Projects

A paper by (Au and Hendrickson, 1985) proposed cash flow analysis and profit measurement methods for construction projects. This paper was published in 1985, so these methods are relevantly old and proven. Those are the calculation include the cash in which is the receipts received by the contractor, the cash out which is the expenses spent by the contractor on the construction works, and the difference between them which is the overdraft. The author proposes calculations for to account for the time value of time, and the cost of finance as shown in the two following equations:

$$NPV_{t=0} = \sum_{t=0}^n A_t(1+i)^{-t} \quad (2.1)$$

$$NFV_{t=n} = \sum_{t=0}^n A_t(1+i)^t \quad (2.2)$$

where  $NPV$  and  $NFV$  are the Net Present Value and Net Future Value, respectively,  $A_t$  is the net cash flow for time period  $t$ , and  $i$  can be set as the Minimum Attractive Rate of Return (MARR) for the company.

Furthermore, the Internal Rate of Return ( $IRR$ ) can be calculated by letting  $NPV = 0$  or  $NFV = 0$  and calculating the  $i$  which becomes the IRR. However the author advises against using the MIRR as an indication of profitability, because the fact that almost all construction project are heavily dependant of borrowed resources, the MIRR would be therefore misleading.

The author then presents calculations for overdraft finance, loan interests, and inflation. Stoppage of work is also considered. The author's conclusions can be summed up that: The IRR is not a correct profit measure, the gross profit as measured by the residual net cash flow at the end of the profit does not take into account the project's finance, long-term loans may be a better finance decision than overdraft in long large-scale profits, and finally sharing of financial risks should be shared by the owner and the contractor may be less costly to the owner.

### 2.2.2 Systems Analysis of Project Cash Flow Management Strategies

A system dynamics approach for cash flow analysis of construction projects was proposed by (Cui, Hastak, and Halpin, 2010). A diagram of this system is shown in Figure 2.2. System dynamics is an approach to model complex systems, focusing on system behaviour over time. It has been used to model social, economic, and environmental systems. The model presented by the authors was tested on a case study, which was a storage house.

System dynamics proved useful in modelling the dynamic nature of the finance in construction projects. The model of a "cash balance module", a "material disbursement module", and a "project operation module". The "cash balance module" is the outer frame and is connected to the other modules. It includes cash flow from operating and financing activities for the period of the project construction. The "material disbursement module" includes cash with respect to material invoices, payments, etc. The "project operation module" handles rework, errors, changes in scope, etc. Other modules are included to handle labour payments, subcontractors payment.

The model can be used to perform what-if analysis using different cash flow management strategies: Front-end loading strategies include billing of mobilization costs, unbalanced pricing by overpricing activities done earlier in the project and under pricing later activities (which is generally unacceptable unless the risk is minor on the employer), and finally billing of materials prior to their installation (stored on site, in accordance with contract). Back-end loading strategies include trade credit, where the contractor receives material from suppliers and pays for them later after a grace period, and subcontracting, where the contractor assigns part of the work to sub-contractors but pays for them later (according to the invoices between them) and may even pay the retainage to the subcontractors when retainage is received from the employer.

A setback of the model, according to the authors, is its uniqueness for different projects, requiring some modification to the equations used. Also, a software package, VESIM DSS version 5.5, was used, so some changes in the software parameters are needed as well. The author recommends an unbounded software package to for better further research into the financial impacts of different cash strategies. (Cui, Hastak, and Halpin, 2010)

### 2.2.3 Analyzing the Impact of Negative Cash Flow on Construction Performance in The Dubai Area

(Al-Jabouri, Al-Aomar, and Bahri, 2012) presented a study into the patterns and effect of negative cash flow on construction project in the Dubai Area. The study was done on



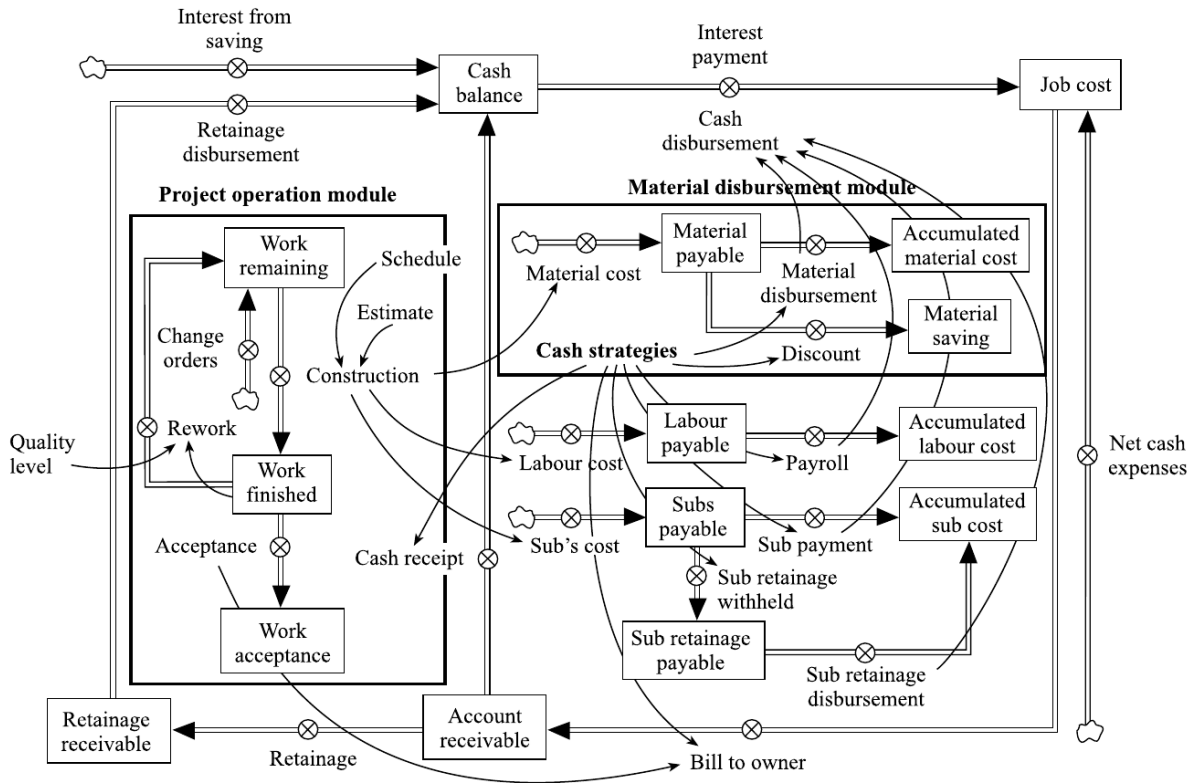


Figure 2.2: System dynamics model for project cash flow management (Cui, Hastak, and Halpin, 2010).

40 ongoing projects in the Dubai Area, and 4 of them were thoroughly studied. The analysis was for the Cash disbursements, cash receipts, and accumulated cash flow. It was found that there was a negative cash flow for 30 to 70% of the project duration in the projects studied, and the shortage values ranged between 2 to 4 times the monthly expenses. The author mentions that some contractors are able to reduce the extent of negative cash flow by rescheduling based on cash flow constraint. The author recommends attention to negative cash flow, cooperation between the contractor, employer, and other project stakeholders. The author also recommends more practical research using actual data to better understand the impact of cash flows.

## 2.2.4 Financial Management of the Construction Projects: A Proposed Cash Flow Analysis Model at Project Portfolio Level

Purnus and Bodea (Purnus and Constanta-Nicoleta, 2015) have presented a complete cash flow analysis as a case study on 5 projects as shown in Table 2.1. The projects have different start dates as well, as shown in Figure 2.3. The cash flow was calculated and is shown in Figure 2.4. The projects of 5 infrastructure projects awarded during 2013 and

Table 2.1: Projects and portfolio contract price as studied by Purnus and Bodea (Purnus and Constanta-Nicoleta, 2015)

Project	Dura- tion	Contract Price (Euro)	Project Type	Contract
1	21 months	15,518,964	Waste Water Plant	FIDIC 1999 Yellow Book
2	14 months	7,027,800	Waste Water Plant	FIDIC 1999 Yellow Book
3	24 months	5,527,942	Waste Water Plant	FIDIC 1999 Yellow Book
4	14 months	11,687,742	Rehabilitation of a water supply and waste water network	FIDIC 1999 Red Book
5	11 months	7,475,872	Rehabilitation of a road	FIDIC 1999 Red Book
Port- folio	36 months	47,238,320	-	-

2014 to a middle-sized construction company. Projects Their contract conditions were based on FIDIC 1999 conditions of contract for buildings and engineering work designed by the employer (Red Book) and FIDIC 1999 Conditions of Contract for Plant and Design-Build for Electrical and Mechanical Plant (Yellow Book). Due to the overlapping of the projects, the works done during October 2014 through August 2015 are over 2,000,000 Euros, with a peak of 5,626,187 Euros in July 20. Figure 2.4 shows cumulative cash flow of the portfolio. This is the combination of cash-in and cash-out where the negative values indicate the overdraft expected on part of the contractor, and the positive values indicate the profit. Figure 2.5 shows a cash flow combining finance, income, costs and return of finance after running multiple scenarios. The goal is to keep that cash flow positive at all time. The paper highlights the necessity of a detailed cash flow analysis on the portfolio level, and recommends probabilistic analysis and risk management.

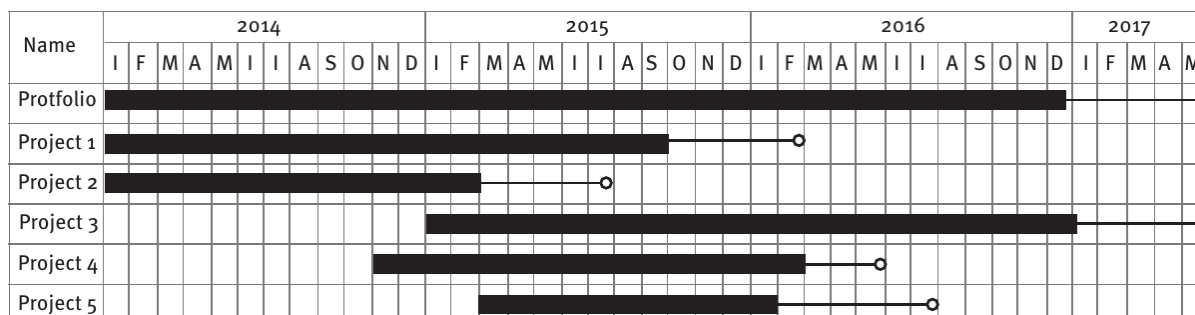


Figure 2.3: Gantt chart of the portfolio studied by Purnus and Bodea (Purnus and Constanta-Nicoleta, 2015)



Figure 2.4: cash flow of the portfolio(Purnus and Constanta-Nicoleta, 2015)

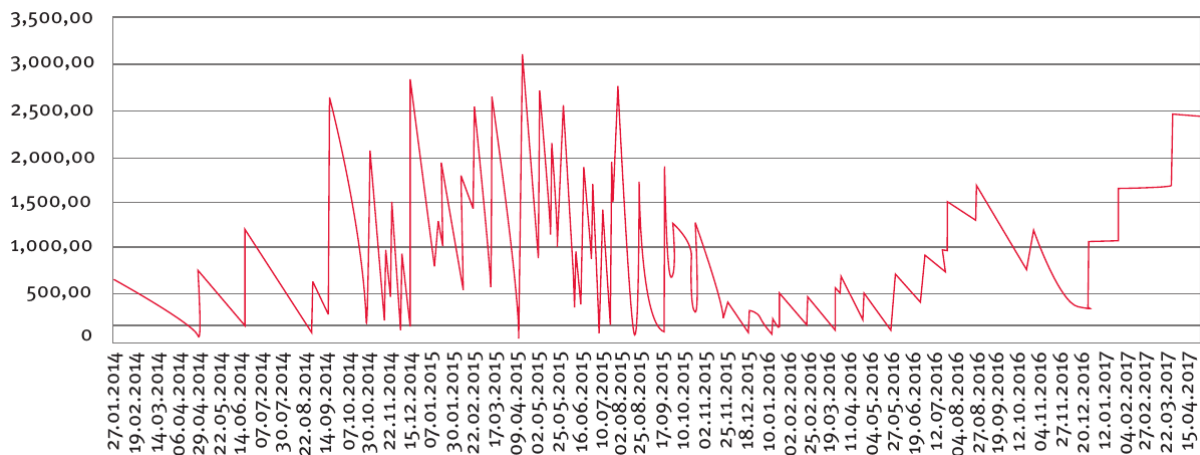


Figure 2.5: Finance of the portfolio (Purnus and Constanta-Nicoleta, 2015)

## 2.3 Optimization and Resource/Finance Based Scheduling

In continuity of the previous section, many researchers studied the optimization of resource constrained construction projects, or resource-constrained scheduling, or time cost trade-off. There are many techniques, methods, and optimization algorithms in this area. This section will attempt to cover a few. To name some research works in this area; (Li, 1996) is one of the oldest papers to handle the optimization problem for construction schedules; (Hegazy, 1999) introduced the optimization of resource allocation and leveling using genetic algorithms; (El-Rayes and Moselhi, 2001) used dynamic programming formula-



tion to optimum resource usage; (Elazouni and Metwally, 2007) used genetic algorithms for a time-cost trade-off, (Liu and Wang, 2008) created a model for resource-constrained scheduling, time-cost trade-off for non-serial repetitive projects was optimized using genetic algorithms and dynamic programming by (Ezeldin and Soliman, 2009), (Liu and Wang, 2009) studied profit optimization for linear projects; (Elazouni, 2009); (El-Rayes and Jun, 2009) presented a heuristic method for multi-project finance based scheduling; (Christodoulou, 2010) presented a new approach for resource-constrained scheduling using Ant Colony Artificial Agents; (Jun and El-Rayes, 2011) presented a multi-objective model for resource leveling and allocation; (Lucko, 2011) used singularity functions for resource optimization; (Abido and Elazouni, 2011b) presented a heuristic for multi project finance-based scheduling; (Abido and Elazouni, 2011b) used a strength Pareto evolutionary algorithm for creating optimum finance-based schedules; (Lucko, 2013) presented a decision making model using singularity functions and genetic algorithms for financial decision making, based on the time value of money;; (Alghazi, Elazouni, and Selim, 2013) presented a continuity into finance-based scheduling using genetic algorithms; (Li and Li, 2013) used self-adaptive ant colony optimization for time-cost optimization; (Menesi, Galzarpoor, and Hegazy, 2013) used constrained programming for large scale projects; (Tang, Liu, and Sun, 2014) continued research into linear scheduling method using constrained programming; (Elazouni and Abido, 2014) presented a strength Pareto evolutionary algorithm for the optimization of finance requirements, resource leveling and profit; another paper by (Elazouni, Alghazi, and Selim, 2015) presented meta-heuristics for finance-based scheduling; (Su and Lucko, 2015) used singularity functions for optimum present value scheduling; (Kim, Walewski, and Cho, 2016) used a modified niched pareto genetic algorithm for scheduling; finally, (Elbeltagi et al., 2016) used particle swarm for multi objective schedule optimization.

### **2.3.1 Optimization of Resource Allocation and Leveling Using Genetic Algorithms**

Hegazy (Hegazy, 1999) presented a paper in 1999 regarding an algorithm for resource allocation inside a MS Project™. The method relies on the fact that a user can already input "priorities" for activities in MS Project™, those can be from lowest to highest, and are used by the program to prioritize the levelling of resources in a heuristic method. The algorithm proposed in the paper is a genetic algorithm written in Visual Basic for Applications (VBA), which is built in the program, to optimize those priorities in order to get the optimum objective result, which can be combination of minimum project duration, minimum resource fluctuation, and minimum utilization period of resources. The algorithm starts by initiating the schedule, setting the priority to lowest for all activities, then

looping on the activities by setting the priority to highest and calculating the objective functions for each. The genetic algorithm is shown in Figure 2.6. The algorithm proposed has the advantage of being an add-on to a popular commercial software already used extensively in the construction industry. However, the processing time was quite high, as the author reported that four experiments took 50 to 120 minutes, but it should be noted that it was done on a Pentium 233 MMX Computer. Finally, the author recommends the application of a similar method using a more efficient programming language.

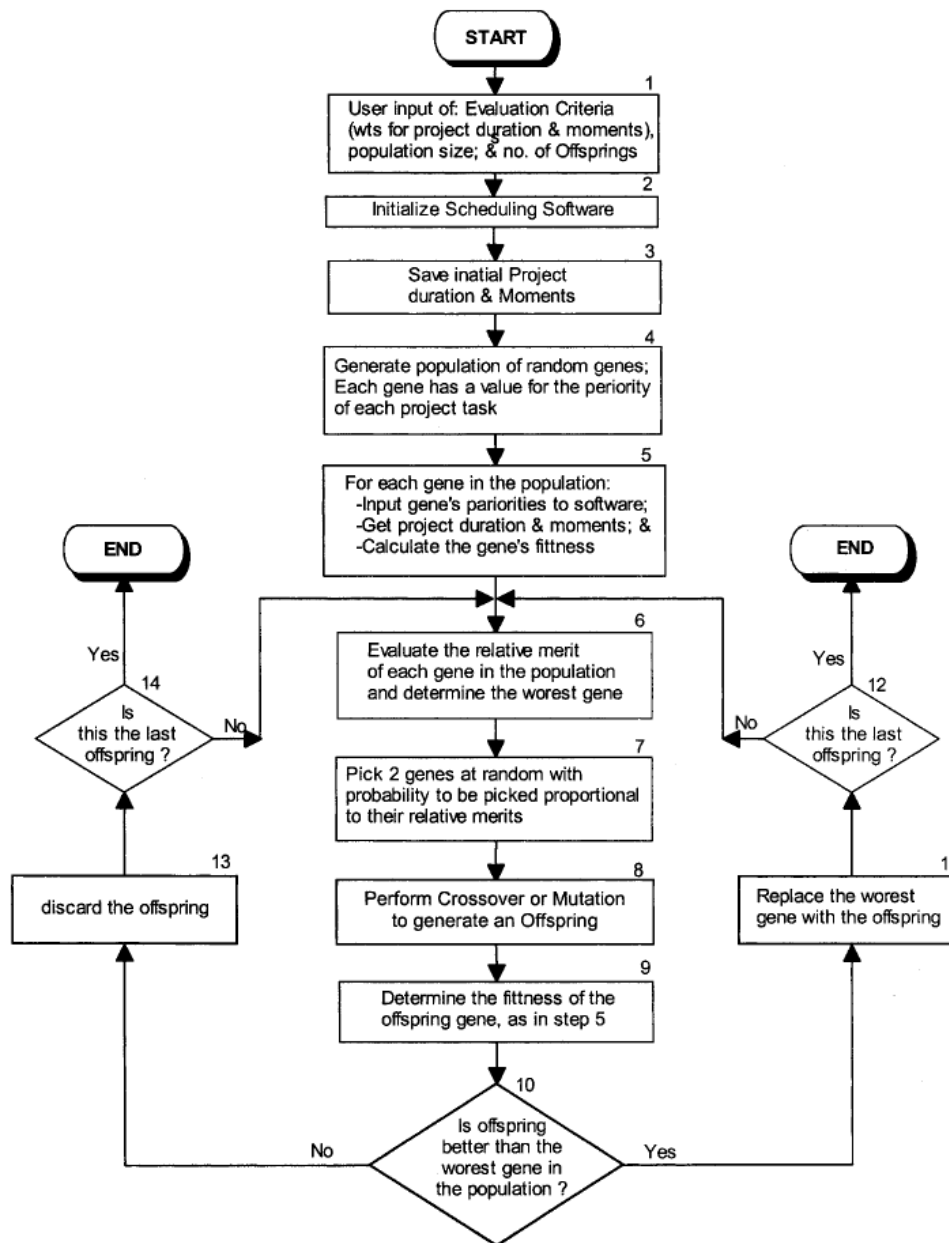


Figure 2.6: Genetic Algorithm levelling algorithm as proposed by Hegazy (Hegazy, 1999)

### **2.3.2 Expanding Finance-Based Scheduling to Devise Overall-Optimized Project Schedules**

Technical notes by (Elazouni and Metwally, 2007) presented the implementation of a model for finance based scheduling model implemented in Visual Basic. Time-cost trade-off (TCT) is done, due to the fact that finance based scheduling results in longer schedules than unconstrained ones. So the work included TCT analysis, resource allocation, and resource levelling, achieved through Genetic Algorithms. The model was tested a small 5 activities project.

### **2.3.3 Heuristic Method for Multi-Project Finance-Based Scheduling**

In another paper by (Elazouni, 2009), a heuristic method scheduling multiple project subject to cash constraints. The proposed heuristic method starts by determining the cash available to schedule activities during a given period; identifies all possible schedules; determines cash requirements and the impact of project completion, selects the best schedule; updates the cash flow; proceeds to the next periods, one period at a time till all activities are scheduled. The method was validated by comparing with previous results solved by the author using integer programming, and the solutions were very comparable. The author claims that the advantage of this heuristic method is its flexibility, and ability to schedule practical-size projects.

### **2.3.4 Scheduling Resource-Constrained Projects with Ant Colony Optimization Artificial Agents**

Research into scheduling resource-constrained projects using Ant Colony Optimization (ACO) was done by (Christodoulou, 2010). ACO is a population-based artificial agent which is inspired by the collective behavior of ants as they optimize their path between their nest and their food. Ants, in real life, leave a trail of pheromones on their path, and this trail steers the succeeding ants in the direction of the stronger pheromone concentrations, so each ant has a higher probability of following the path chosen by the majority of the preceding ants. The ACO method is applied on a resource constrained network, the effects of resource availability on the critical path and project completion time is examined. The search for the shortest path, as usual for ACO, is substituted with the search for the longest path, which is the Critical Path for the construction schedule, according to the Critical Path (CPM) method. This is done by treating the duration as negative numbers within the ACO. The method is tested on a small project of 17 activities, accuracy

of 100% for the unconstrained project and a 97% accuracy for the resource constrained project. The author claims that the ACO method, though iterative, is more suitable in parallel computing due to its branching nature. Testing into large projects with more than 1000 activities is in progress.

### 2.3.5 Multi-objective Optimization of Resource Leveling and Allocation during Construction Scheduling

(Jun and El-Rayes, 2011) proposed a model for resource optimization implemented into MS Project™ as an extension written in the programming language C#.net. A summary of the optimization model is shown in Figure 2.7. The model can have one of 2 metrics as objectives: Release and Rehire (RRH), or Resource Idle Days (RID). The decision variables are the Priority Value ( $P_n$ ) and Start Day ( $S_n$ ), the former is used to define the scheduling sequence of each activity while the latter is used to shift the activity. Each of those variables, for every activity  $n$  is used as a chromosome for the genetic algorithm. An example run was done using the data tested for validation by Hegazy (Hegazy, 1999) as described in a previous section.

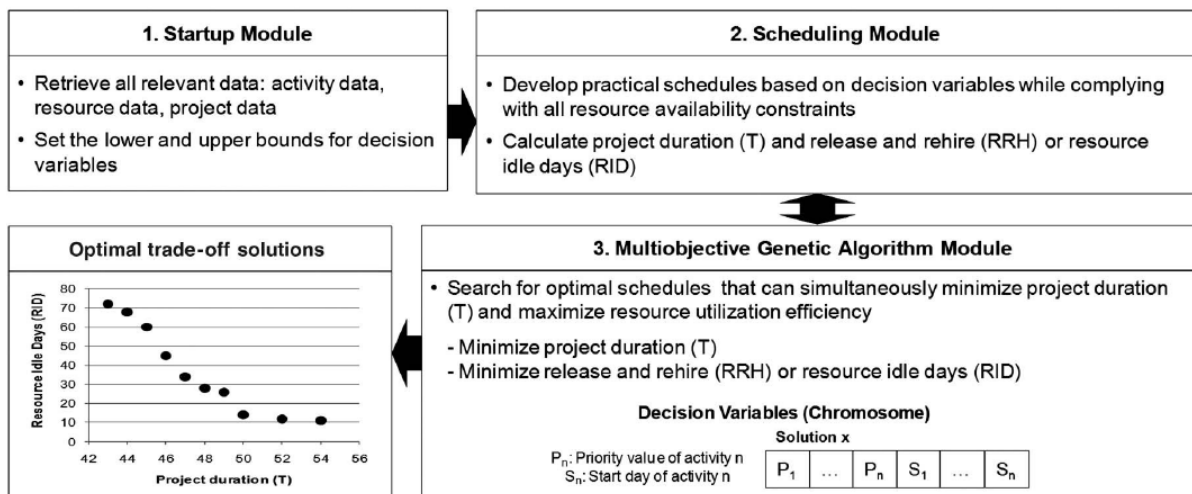


Figure 2.7: Optimization model done by Jun et al (Jun and El-Rayes, 2011)

### 2.3.6 Multi-objective Evolutionary Finance-Based Scheduling: Entire Projects' Portfolio and Individual Projects within a Portfolio

Two papers by the same authors presented a multi-objective scheduling model for portfolios and individual projects within a portfolio (Abido and Elazouni, 2011a) (Abido and Elazouni, 2011b). The authors proposed a multi-objective evolutionary scheduling model using a strength Pareto evolutionary algorithm shown in Figure 2.8 and fuzzy logic, and applied on 5 projects consisted of 25, 30, 225, 240, and 260 activities each. The decision variables are the start times of the projects' activities. The formulation of the multiple objectives include maximizing the profit, and minimizing the duration, financing cost, and credit.

The algorithm works as follows:

1. Generate an initial population into an empty external Pareto-optimal set.
2. Update the external Pareto-optimal set as follows:
  - (a) Search the population for the non-dominated solutions and copy them to the external Pareto set
  - (b) Search the external Pareto set for the non-dominated solutions and remove all dominated solutions from the set
  - (c) Reduce the set by means of clustering in case the number of the solutions externally stored in the Pareto set exceeds a pre-specified maximum size
3. calculate the fitness values of solutions in both external Pareto set and the population as follows:
  - (a) Assign the strength  $s$  for each solution in the external set. The strength is proportional to the number of solutions covered by that solution.
  - (b) The fitness of each solution in the population is the sum of the strengths of all external Pareto solutions which dominate that solution. A small positive number is added to the resulting sum to guarantee that Pareto solutions are most likely to be selected by the mating pool.
4. Select two solutions at random out of the combined population and external set solutions, compare their fitness, select the better one, and copy it to the mating pool.
5. Generate a random number between 0 and 1 and compare it with the preset crossover probability,  $P_c$ . If  $r$  is less than  $P_c$ , then carry out the crossover operator. Repeat for mutation operator.

6. Check for stopping criteria to terminate otherwise copy new population to old population and go to Step 2. In this study, the search will be stopped if the generation counter exceeds its maximum number.

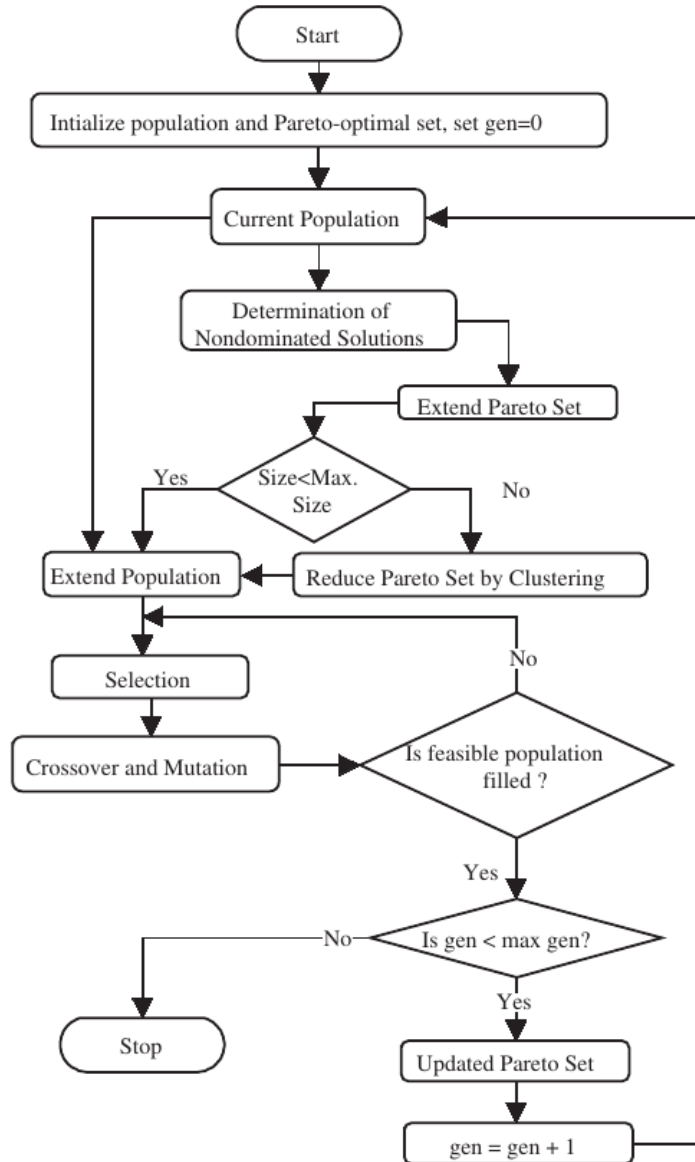


Figure 2.8: Computational flow for the strength Pareto evolutionary algorithm (Abido and Elazouni, 2011b)

### 2.3.7 Improved Genetic Algorithm Finance-Based Scheduling

Alghazi et al (Alghazi, Elazouni, and Selim, 2013) proposed a Genetic Algorithm (GA), coded in Matlab<sup>TM</sup>. The objective is to tackle the problem of infeasible chromosomes in resource levelling using GA. The chromosomes are assigned as the start of each activity in a project, and infeasible chromosomes occur when a chromosome, representing the start of an activity, creates a conflict with the logical relationships between activities or

when the resource constraint is not met. The authors presented a chromosome-repairing GA and stated that it outperformed replaced-chromosome GAs with limited computational effort. The results were verified using a 10 cash-constrained 30-activity problems. The flowchart of the chromosome-repairing GA is shown in Figure 2.9.

### **2.3.8 Fast and Near-Optimum Schedule Optimization for Large-Scale Projects**

Menesi et al. (Menesi, Galzarpoor, and Hegazy, 2013) presented a Constrained Programming (CP) Model in an attempt to reach optimum results for large projects quickly. The authors argue that focus on optimization of large scale projects (more than 1,000 activities) is lacking in research, though most construction projects, in reality, have large schedules. The model proposed was implemented in *IBM ILOG CPLEX Optimization Studio*, and produced near-optimum solutions for 1,000 and 2,000 activities projects in minutes, performing better than meta-heuristic models such as Genetic Algorithms. The authors also challenge other researches to improve upon the results with 1 percent deviation for projects consisting of 1,000 activities or more, on a personal computer.

### **2.3.9 Enhanced Trade-off of Construction Projects: Finance-Resource-Profit**

Another paper by (Elazouni and Abido, 2014), where the Trade-off between finance requirements, resource leveling, and anticipated profit are optimized. A Strength Pareto evolutionary algorithm (SPEA) is implemented for the trade-off, by solving a network of nine multi-mode activities and obtain the associated Pareto-optimal front, which comprised fifty solutions, in order to help the decision maker take the best balance. In addition, a fuzzy logic algorithm was implemented to compare the balance between those results. The author recommends research into involving large-sized practical projects within a portfolio.

### **2.3.10 Finance-based Scheduling using meta-heuristics: discrete versus continuous optimization problems**

(Elazouni, Alghazi, and Selim, 2015) compared the performance of genetic algorithms (GA), simulated annealing (SA) and shuffled frog-leaping algorithm (SFLA) in solving discrete and continuous variable optimization problems of finance-based scheduling. This

was tested on projects of 30, 120, and 210 activities. SA outperformed the SFLA and GA in terms of quality of results and computational cost with small networks of 20 activities, and resulted in the shorted durations for larger networks of 120 and 210 activities. The author recommends further researchers to use finance-based scheduling, due to its discrete or continuous nature, to use it as a test bed for testing the performance of new developments of meta-heuristics.



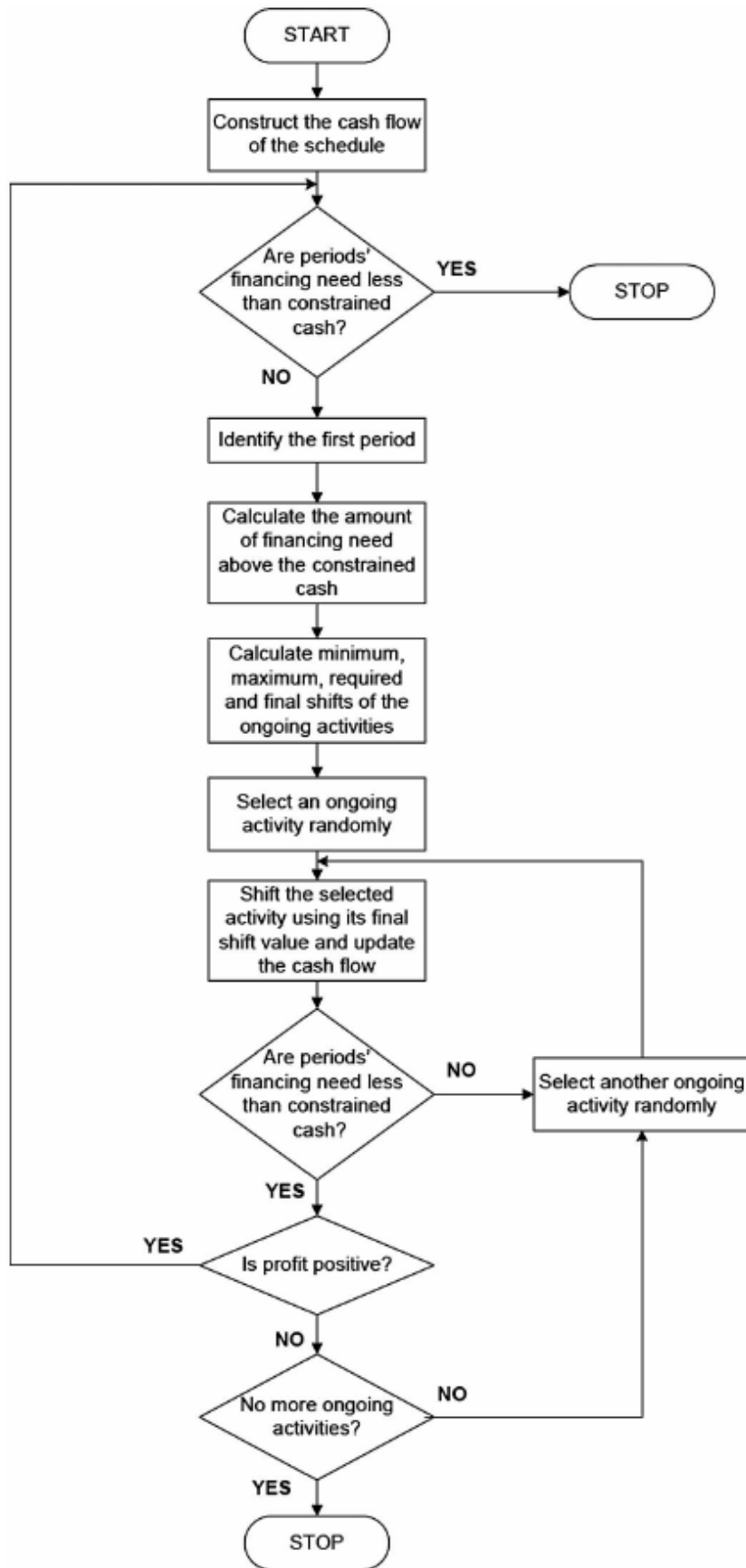


Figure 2.9: Flow chart of the chromosome-repairing GA (Alghazi, Elazouni, and Selim, 2013)

## 2.4 Outcomes From Literature Review

After conducting the literature review, it was found that the financial analysis on the portfolio and corporate levels is less tackled by research than analysis on the project level. It was also agreed among researchers that portfolio level analysis is more indicative on the success on the corporate level, as it includes multiple projects as a whole, rather than single projects, which is the case in any construction company because most finance and resources are shared between projects. It was also found that the time value of money has a great effect on cash flows, and the two most used parameters to indicate the profit from a project under that methodology is the Net Present Value, and the Internal Rate of Return, however, it was found that the Net Present Value is more appropriate. Regarding the complexity and size of the projects used as case studies in literature, most of them were small schedules with a limited number of activities, few papers handled large projects with up to a thousand activities, which may be impractical in real life because large projects, and when handled as portfolios, have much higher numbers of activities; huge schedules are unavoidable when handling large portfolios. Regarding optimization, there are many studies into different optimization techniques and algorithms. The most significant one to this thesis in the method used by (Hegazy, 1999), where lags were added before each activity to allow the model to delay each of them, and optimization was done resource allocation and leveling; the same concept was adopted in this thesis.



# Chapter 3

## Model Development

This Chapter covers the complete model development. This includes the inputs and outputs. The programming language used, which is Python, is described. The calculations and process are explained for the scheduling, cash flow analysis, time value of money, and optimization. Finally, the development of the Graphical User Interface (GUI) is described. The entire Python Code used is available in Appendix A.

### 3.1 Assumptions

As expected in any model development, some assumptions must be made. Those are the following:

- The cost of each activity was assumed to be uniformly distributed along each activity's duration, in contrast real life cases where the cost can be front allocated, or back allocated, or have any other distribution.
- The costs and expenses that are delayed after an activity or before it, such as in the case of paying for a supplier after a duration of time from an activity, or before the activity was neglected. Though they could be added in the model as separate activities that have delays between them.
- Payment of invoices, retention, and down-payments was assumed to be always on time, neither late nor early than the contractual time bars. Delays are completely out of scope.
- The retention was assumed to be paid completely after the Defects Liability Period. In other situation it could be paid in half at construction completion and half after the defects liability period.

## 3.2 Model Inputs and Outputs

The user is able to input project parameters for the projects, activities, and the relationships between the activities. The inputs are as follows:

- Projects (The interface is shown in Figure 3.8):

**Project ID** A unique id for each project

**Project Name** The name or description of the project

**Start** The start date of the project

**Interest** The interest percentage used, this can be the Minimum Attractive Rate of Return (MARR) for the company and should at least account for the expected Inflation.

**Mark-up** The mark-up percentage for the project. This should account for profit and contingency.

**Down-payment** The down-payment percentage for the project

**Invoice Interval** The interval between issuing of invoices. This is typically set as monthly.

**Payment Period** The time period in days between issuing an invoice and receiving the payment for that invoice.

**Retention** The retention percentage for the project. This amount is deducted from the invoices and received [by the contractor at the end of the project.

**Retention Period** The time period in days between the end of a project and the receipt of the retention payment.

- Activities (The interface is shown in Figure 3.10):

**Project ID** The ID of the project containing the activity. This id should match the id provided for a project.

**Activity ID** A unique ID for the activity. The ID should be unique for each activity within the same project.

**Activity Name** A name, WBS name, or description for the activity

**Duration** The duration in days for the activity

**Cost** The direct cost for the activity

- Relationships (The interface is shown in Figure 3.11):

**Project ID** The project ID for the project containing the predecessor and successor activities

**Activity1 ID** The ID of the predecessor activity

**Activity2 ID** The ID of the successor activities

**Type** The type of the relationship. This can be Finish-to-Start, Finish-to-Finish, Start-to-Start, or Start-to-Finish.

For the outputs, the model was built with a Graphical User Interface (GUI), which will be discussed thoroughly in a later section. The GUI allows the user to create the elements of the portfolio with the inputs just mentioned. It also allows the user to preview tables containing all fields for the elements, whether the portfolio, the projects, the activities, or the relationships. The GUI can also preview Gantt Charts, cash flow plots, overdrat plots, for the non-optimized and the the optimized portfolio, with discounted values or non-discounted values.

In addition, the program can output tables for the portfolio elements including the portfolio, projects, activities, relationships, cash flow, and trial calculations. The data is exported in comma separated values (csv) formats and Excel spreadsheet format. The complete log is exported in a text file. The plots and gantt charts in every mentioned form is exported in PDF or SVG files, for the pupose of previewing or compiling in a report, such as this thesis.

### 3.3 Input from Primavera

Projects can be imported from Otacle's Primavera. It should be noted that Primavera is not friendly to add-ins and mods. Another competitor, MS Project, for example, is more modifiable through the availability of developer tools in Visual Basic for Applications (VBA) within MS Project and other MS Office tools. However, Primavera is and has been more dominant in Egypt, so this thesis required the use of Primavera due to the actual work environment. The original projects used for the validation in this thesis were done in Oracle Primavera. To import the projects from Primavera into the model, a workaround is needed; the user has to export the projects from Primaverain spreadsheet xls format, but first the export options must be edited by the user to add the primary constraint, primary constraint date, original duration, Budgeted total cost, and the calendar name to the exported spreadsheet. To import into the mmodel, an algorithm was coded to import the projects from those xls spreadsheets.

## 3.4 Programming Language and Packages Used

The programming language used in this work is Python. Python is a relatively new programming language. It is a free and open source high-level scripting language. It's high-level, dynamic, allows for procedural and object-oriented programming among other paradigms. It has a community based development environment which resulted in a vast library of third party packages (Foundation, 2016). Though execution of python code is normally slower than other counterparts like C++ or JAVA, it is however known to be relatively easier, more readable, and faster for prototyping. It was ranked as fourth in the "Top 10 languages in 2015" listing by IEEE (IEEE, 2015). This programming language was chosen in this work due to its faster prototyping process because relatively simple and readable. This allowed for better experimentation during building the model with ease and wasting less time. In other words, It is faster to code in Python in comparison with other languages. The only disadvantage is that Python, due to the fact that it's a high-level language, is normally slower, in means of execution time, than otehr languages like C or C++ for example, which are lower level and "closer to the hardware". Fortunately, most of the critical packages in Python are coded and optimized in C to lower that effect. It should be noted that the "slower time" discussed here is more relevant to real time systems and computationally demanding softwares, which isn't too much of a nuisance within the scope of this thesis. The entire Python Code used is available in Appendix A. Python has a very good standard library with an excellent documentation and friendly community of developers. There are a lot of packages built for Python spanning over a lot of useful functions. Several packages built for the Python environment were used in this thesis. All of them are open source and easily installed. The packages used outside of the Python standard library or otherwise notable are listed below, according to their functions:

**Database Management** "sqlite3" was used for the database. It is part of the standard library, requires instructions syntax similar to MySQL. It has less capabilities than some other databases but none of those capabilities were required for the purpose of this work. It is also file-based as opposed to a server database, which limits to only one connection per database, but allows for higher read-write speeds.

**Graphical User Interface** "tkinter" was used because it's already part of the standard library, as well as simple and good enough for prototyping

**Plotting** "matplotlib" was used for plotting high quality svg files. It is a well known plotting library in the scientific community and has an excellent range of capabilities

**Other external packages** "xlsxwriter" and "xlrd" are 2 packages that are not included in the standard library. They were used for reading and writing to excel files. This

is needed to import excel files exported from primavera, and the standard library can only manipulate csv files.

### 3.5 Database

A relational database was used to store and handle data. The database used is Sqlite3, which is an open source file based database system, readily available in the Python standard library. Being connected to a single file on the hardisk, unlike MySQL which is a server, it is faster but allows for one connection at a time. The tables and column fields are listed below. The column fields can be considered as the variables used in the calculation, and many of them are the model inputs.

A complete list is as follows:

- |                              |                                 |                                   |
|------------------------------|---------------------------------|-----------------------------------|
| 1. trials                    | (j) invoiceinterval<br>(INT)    | (x) npvopt (REAL)                 |
| (a) trialid (INT)            | (k) paymentperiod<br>(INT)      | (y) maxoverdraftdiscopt<br>(REAL) |
| (b) initialnpv<br>(FLOAT)    | (l) downpayment<br>(REAL)       | (z) minoverdraftdiscopt<br>(REAL) |
| (c) trialnpv<br>(FLOAT)      | (m) cost (REAL)                 | 3. activities                     |
| (d) bestnpv<br>(FLOAT)       | (n) price (REAL)                | (a) projectid<br>(TEXT)           |
| 2. projects                  | (o) totalactivities<br>(INT)    | (b) activityid<br>(TEXT)          |
| (a) projectid<br>(TEXT)      | (p) criticalactivities<br>(INT) | (c) activityname<br>(TEXT)        |
| (b) projectname<br>(TEXT)    | (q) cashinpv (REAL)             | (d) duration (INT)                |
| (c) start (NUM)              | (r) cashoutpv<br>(REAL)         | (e) cost (REAL)                   |
| (d) finish (NUM)             | (s) npv (REAL)                  | (f) es (INT)                      |
| (e) duration (INT)           | (t) maxoverdraftdisc<br>(REAL)  | (g) ef (INT)                      |
| (f) interest (REAL)          | (u) minoverdraftdisc<br>(REAL)  | (h) ls (INT)                      |
| (g) markup (REAL)            | (v) cashinpvopt<br>(REAL)       | (i) lf (INT)                      |
| (h) retentionperiod<br>(INT) | (w) cashoutpvopt<br>(REAL)      | (j) ff (INT)                      |
| (i) retention<br>(REAL)      |                                 | (k) tf (INT)                      |
|                              |                                 | (l) lag (INT)                     |
|                              |                                 | (m) os (INT)                      |
|                              |                                 | (n) of (INT)                      |



4. relationships
- (a) projectid (TEXT)
  - (b) activity1id (TEXT)
  - (c) activity2id (TEXT)
  - (d) type (TEXT)
5. cashflow
- (a) date (INT)
  - (b) projectid (TEXT)
  - (c) cashout (REAL)
  - (d) invoice (REAL)
  - (e) cashin (REAL)
  - (f) cashoutcum (REAL)
  - (g) cashincum (REAL)
  - (h) overdraft (REAL)
  - (i) cashoutdisc (REAL)
  - (j) cashindisc (REAL)
  - (k) cashoutcumdisc (REAL)
  - (l) cashincumdisc (REAL)
  - (m) overdraftdisc (REAL)
6. cashflowall
- (a) date (INT)
- (b) projectid (TEXT)
- (c) cashout (REAL)
- (d) invoice (REAL)
- (e) cashin (REAL)
- (f) cashoutcum (REAL)
- (g) cashincum (REAL)
- (h) overdraft (REAL)
- (i) cashoutdisc (REAL)
- (j) cashindisc (REAL)
- (k) cashoutcumdisc (REAL)
- (l) cashincumdisc (REAL)
- (m) overdraftdisc (REAL)
7. cashflowopt
- (a) date (INT)
  - (b) projectid (TEXT)
  - (c) cashout (REAL)
  - (d) invoice (REAL)
  - (e) cashin (REAL)
  - (f) cashoutcum (REAL)
  - (g) cashincum (REAL)
  - (h) overdraft (REAL)
8. cashflowallopt
- (a) date (INT)
  - (b) projectid (TEXT)
  - (c) cashout (REAL)
  - (d) invoice (REAL)
  - (e) cashin (REAL)
  - (f) cashoutcum (REAL)
  - (g) cashincum (REAL)
  - (h) overdraft (REAL)
  - (i) cashoutdisc (REAL)
  - (j) cashindisc (REAL)
  - (k) cashoutcumdisc (REAL)
  - (l) cashincumdisc (REAL)
  - (m) overdraftdisc (REAL)
9. portfolio

(a) portfolioid (TEXT)	(m) minoverdraftdisc (REAL)	(d) type (TEXT)
(b) start (NUM)	(n) cashinpvopt (REAL)	(e) activity1es (INT)
(c) finish (NUM)	(o) cashoutpvopt (REAL)	(f) activity1ef (INT)
(d) duration (INT)	(p) npvopt (REAL)	(g) activity1ls (INT)
(e) numberofprojects (INT)	(q) maxoverdraftdiscopt (REAL)	(h) activity1lf (INT)
(f) numberofactivities (INT)	(r) minoverdraftdiscopt (REAL)	(i) activity1os (INT)
(g) cost (REAL)		(j) activity1of (INT)
(h) price (REAL)	10. big	(k) activity1duration (INT)
(i) cashinpv (REAL)	(a) projectid (TEXT)	(l) activity2es (INT)
(j) cashoutpv (REAL)	(b) activity1id (TEXT)	(m) activity2ef (INT)
(k) npv (REAL)	(c) activity2id (TEXT)	(n) activity2ls (INT)
(l) maxoverdraftdisc (REAL)		(o) activity2lf (INT)
		(p) activity2os (INT)
		(q) activity2of (INT)
		(r) activity2duration (INT)

### 3.6 Scheduling Calculations

The scheduling calculations follow a simple Critical Path Method (CPM) technique. The calculations are done in two steps where one is a forward run and the other is a backward run. The forward run's goal is to set the Early Start (ES) and Early Finish (EF) of each activity in the schedule. A flow chart of the front-run is shown, with some simplification, in Figure 3.1. The explanation of the part where an activity itself is calculated is shown in Equation 3.1. A summary of the forward run is executed roughly as follows:

1. Clear **all** previous data
2. For each project:
3. ES **for** activities with no predecessors = Project Start
4. EF **for** activities with no predecessors = ES + duration
5. While there are unscheduled activities:
6. acts = activities with at least one calculated predecessor
7. For each **in** acts:
8. If **all** predecessors are calculated:
9. **if** relationship **type** = FS:

10.  $ES = \max( EF_{predecessor}, constraint )$
11. **if** relationship **type** = SS:
12.  $ES = \max( ES_{predecessor}, constraint )$
13. **if** relationship **type** = FF:
14.  $ES = \max( EF_{predecessor} - duration, constraint )$
15. **if** relationship **type** = SF:
16.  $ES = \max( ES_{predecessor} - duration, constraint )$
17.  $EF = ES + duration$
18. Set Project Finish =  $\max(EF)$

In explanation of the preceding pseudo code and Figure 3.1, which provide a very rough summary of the forward run phase, first, the old calculations, if available, are deleted. Then a loop is started for each project on its own, which was found to be the better in computational effort than scheduling the portfolio as a bulk. Activities with no preceding activities are set at the project start. Then a list of activities with at least one calculated predecessor is retrieved from the database, then each one in that list is neglected if one or more of its predecessors is not calculated. This was done to get a balance between the speed of the database system to retrieve a simple query vs. its slowness to retrieve multiple sub queries, and the aforementioned power vs. slowness of Python. Lines 9 to 16 are a very logical set of instructions; an activity once its predecessors are known, and its time constraint is already set in the database (Start on or before a date, or finish on or after a date, etc), has its ES set according to the relationship type, which can be Finish to Start, Start to Start, Start to Finish, or Finish to Finish. These logical relationships are shown in Equation 3.1. And Finally the EF is set as the sum of the start and the activity's duration, and the project finish time is set.

$$ES_{activity} = \text{MAX OF} \begin{cases} EF_{predecessor} & : \text{ where relationship type is FS} \\ ES_{predecessor} & : \text{ where relationship type is SS} \\ EF_{predecessor} - DUR_{activity} & : \text{ where relationship type is FF} \\ ES_{predecessor} - DUR_{activity} & : \text{ where relationship type is SF} \end{cases} \quad (3.1)$$

The next run is the backward run, and its goal is to set the Free Floats (FF) and the Total Floats (TF) for the activities. The TF is essential to the upcoming optimization phase. The backward run is very similar in nature to the Front Run. A flowchart of that process is shown in Figure 3.2. The part where an activity is calculated is shown, with some simplification, in Figure 3.2. A rough summary of the backward-run process is shown in the following pseudo-code:

1. For each project:
2. **LF for** activities with no successor = Project Finish
3. **LS for** activities with no successor = LF - duration

4. While there are unscheduled activities :
5.     acts = **list** of activities with at least one calculated successor
6.     For each **in** acts :
7.         If **all** successor are calculated :
8.             **if** relationship **type** = FS :
9.                 LF = **min**(LSsuccessor , constraint)
10.             **if** relationship **type** = SS :
11.                 LF = **min**(LSsuccessor + duration , constraint)
12.             **if** relationship **type** = FF :
13.                 LF = **min**(LFsuccessor , constraint)
14.             **if** relationship **type** = SF :
15.                 LF = **min**(LFsuccessor + duration , constraint)
16.             LS = LF - duration
17.             TF = LS - ES

To explain Figure 3.2 and the previous pseudo-code. The backward run is very similar to the forward run. First the activities that have no successors can be calculated, as their  $LF = EF = Project_{finish}$ . The calculations are then looped on each project, and on each activity. In comparison with the forward run, the difference is that the ES is replaced by the LF, and it is set as the minimum of the successors LS or LF, according to the relationship type. The calculations according to logical relationships are different and are shown in Equation ??.

$$LF_{activity} = \text{MIN OF} \begin{cases} LS_{successor} & : \text{ where relationship type is FS} \\ LS_{successor} + DUR_{activity} & : \text{ where relationship type is SS} \\ LF_{successor} & : \text{ where relationship type is FF} \\ LF_{successor} + DUR_{activity} & : \text{ where relationship type is SF} \end{cases} \quad (3.2)$$

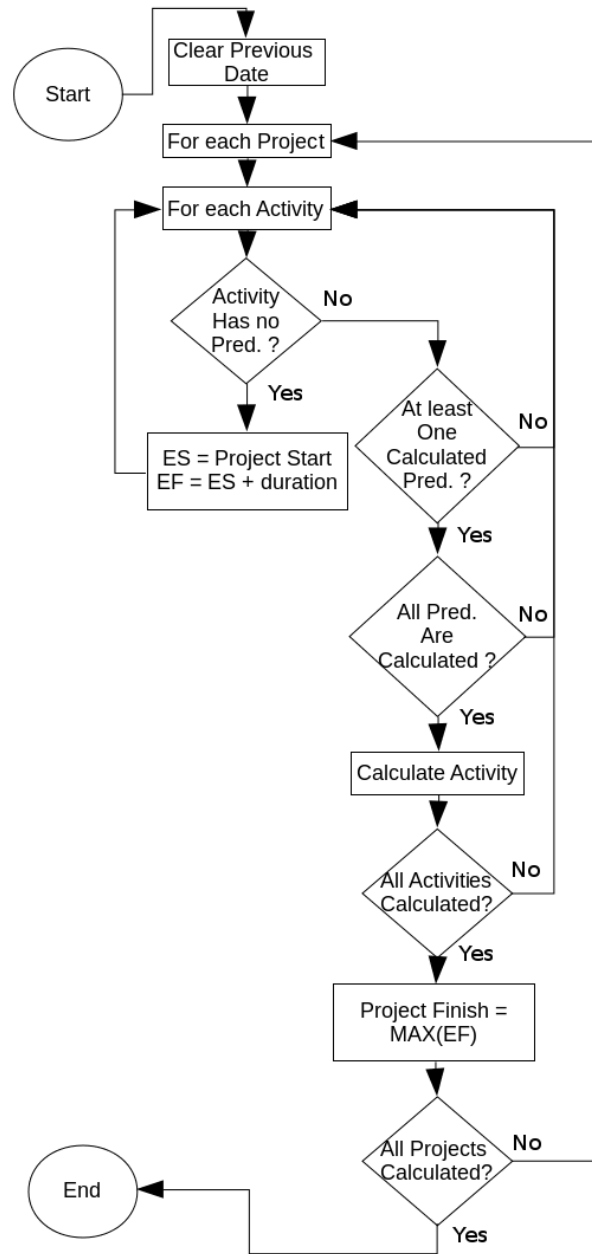


Figure 3.1: Flowchart for the Scheduling Front-Run

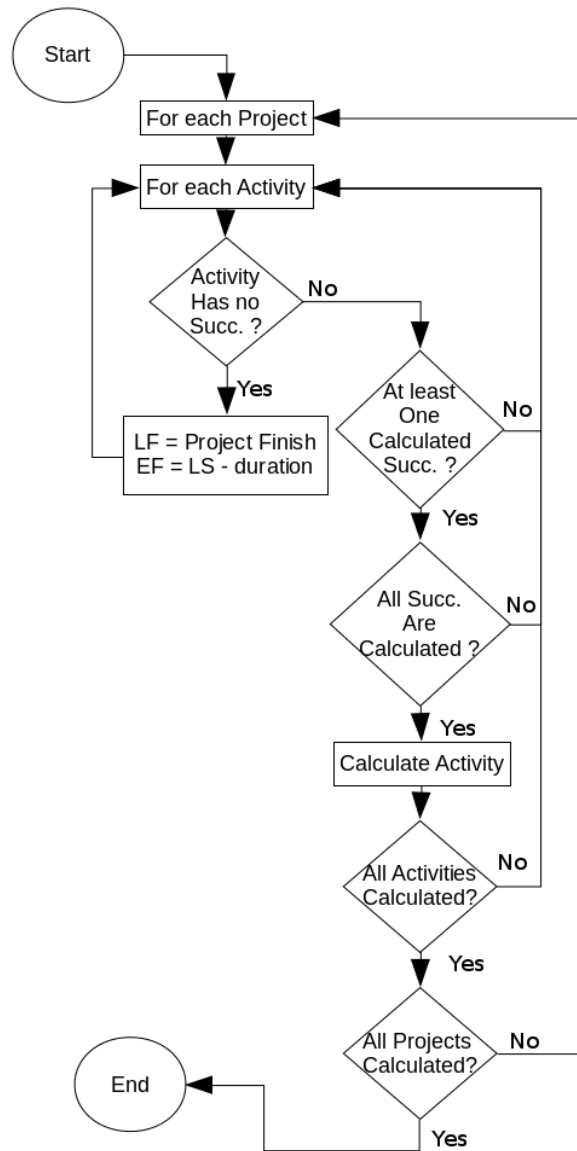


Figure 3.2: Flowchart for the Scheduling Back-Run

### 3.7 Cash Flow Calculation

Once the schedule has been calculated, the cash flow can be easily calculated. A flowchart of the process is shown in Figure 3.3, and a pseudo-code summarizing the process is as follows:

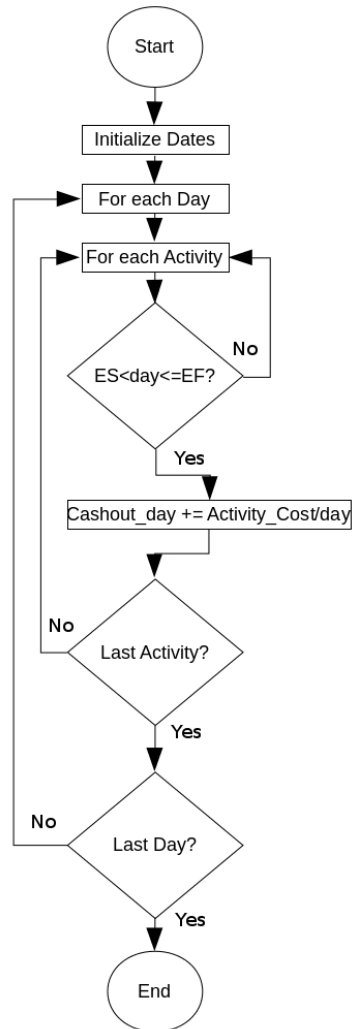


Figure 3.3: Flowchart for The Cashflow calculation

1. Portfolio finish =  $\max(\text{project finish} + \text{retention period})$
2. For each day **in range**(Portfolio start , Portfolio finish):
3. For each activity:
4. IF (activity ES < day <= activity EF):
5. cashout **for** this day += activity cost per day

So, first the range of days is established, which starts at the start of the portfolio and ends at the finish of the last project plus its retention period. Then a loop is done for each day in that range, and each activity, to sum the cost per day. Next, to calculate the cash in, the cash out is summed monthly then assigned as a bulk minus retention and

down payment, plus the markup, on the day of actual payment. The sum of the cash in is calculated as shown in Equation 3.3.

$$\begin{aligned} CashIn_{PaymentDay} = & (InvoiceSum * Markup) \\ & - (Invoicesum/TotalPrice * DownpaymentSum) \\ & - (Invoicesum * TotalPrice/RetentionSum) \end{aligned} \quad (3.3)$$

Where:

$$PaymentDay = EndOfinvoiceinterval + PaymentPeriod \quad (3.4)$$

$$DownPaymentSum = TotalProjectPrice * Downpayment\% \quad (3.5)$$

$$RetentionSum = TotalProjectPrice * Retention\% \quad (3.6)$$

The calculation of the payments follows the agreement that the down payment and retention values are deducted from the invoices by by a weighted average for each invoice. Next the down payment with a value as shown in Equation 3.5 is added to on the day of the start of the project, and the retention with a sum as calculated in Equation 3.6 is added at day when the retention is due for payment. The cash in and the cash out is now calculated. Next, the cash in cumulative and the cash out cumulative are calculated. The overdraft is calculated as the difference between them. Simply:

$$CashInCumulative = \sum_{PortfolioStart}^{PortfolioFinish} (CashIn_{day}) \quad (3.7)$$

$$CashOutCumulative = \sum_{PortfolioStart}^{PortfolioFinish} (CashOut_{day}) \quad (3.8)$$

$$Overdraft = CashInCumulative - CashOutCumulative \quad (3.9)$$

### 3.8 Time Value of Money Calculations

The calculations of the Present Value (PV) and the Net Present Value (NPV) is straightforward. Generally, the PV is calculated as shown in Equation 3.10. The PV in the model is calculated according to Equation 3.12, which was gotten from Equation 3.11. It should be noted that the PV is calculated at the start of the portfolio, and that the interest rate is yearly. The idea is that cash loses value with time, meaning that a sum or money has a different value depending of the time it is calculated, whether due to investment, or inflation. In the case of a contractor, the value of getting a sum of money soon, is higher that getting that same amount of money later, for example 1000 pounds having a value, or a buying power, now, that is higher than it will have in the future. This is the time



value of money. The final number that measures the value of the portfolio from that point of view, is the NPV, and is shown in Equation 3.13. The NPV is calculated as the sum of the discounted overdraft for the whole portfolio, and  $i$  is the yearly interest, which is the inflation rate of the Minimum Attractive Rate of Return (MARR) of the company.

$$PV = \sum \frac{Cost}{(1 + Interest)^n} \tag{3.10}$$

$$FV = PV * (1 + \frac{i}{365})^{(Day - PortfolioStart)} \tag{3.11}$$

$$PV = \frac{FV}{(1 + \frac{i}{365})^{(Day - PortfolioStart)}} \tag{3.12}$$

$$NPV = \sum_{PortfolioStart}^{PortfolioFinish} (PV(Overdraft)_{day}) \tag{3.13}$$

### 3.9 Optimization

Optimization is done by first assigning lags to activities. The lags are a duration inserted to delay each activity for a number of days. The lags are assigned such as:

$$0 \leq Lag_i \leq TF_i \tag{3.14}$$

It should be noted that each activity can be delayed within its total float (TF). Since critical activities have a TF of 0 days, it will always be assigned a Lag of 0 days, which retains its critical state. This can be visualized as shown in Figure 3.4 where activities B and D where assigned Lags, while Activities A, C, and F are critical activities and were assigned a Lag of 0 days. Activity E became a critical activity and was assigned a Lag of 0 days as well. The previous part allowed for the creation of a new schedule to be used

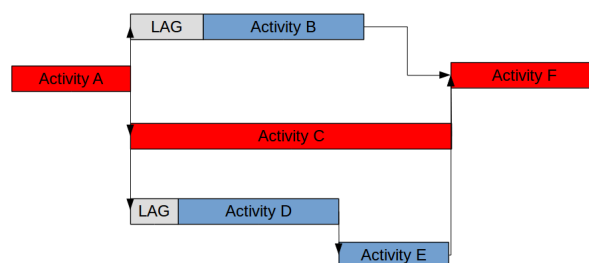


Figure 3.4: Example of an optimization trial

as a trial. The schedule then undergoes a front calculation to calculate the OS of each activity, then a the cash flow is calculated using OS and OF instead of the early starts (ES) and early finishes (EF) which was previously done to the normal schedule.

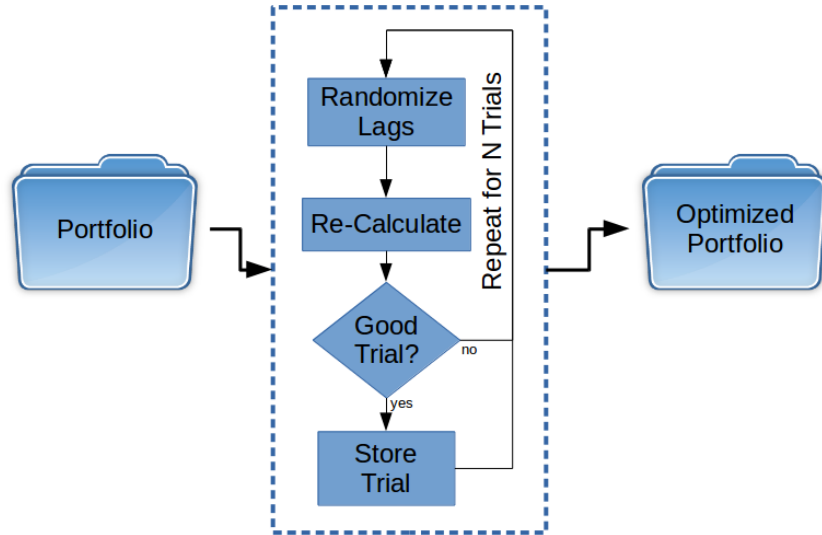


Figure 3.5: Flowchart of the optimization process

The previous part allowed for the creation of an new schedule to be used as a trial. The schedule then undergoes a front calculation to calculate the new OS for each activity. This is dependant on the relationships between activities as follows:

$$OS_{activity} = \text{MAX OF} \begin{cases} ES_{activity} + Lag_{activity} & : \text{ where relationship type is FS} \\ OF_{predecessor} & : \text{ where relationship type is SS} \\ OS_{predecessor} & : \text{ where relationship type is FF} \\ OF_{predecessor} - DUR_{activity} & : \text{ where relationship type is SF} \\ OS_{predecessor} - DUR_{activity} & : \text{ where relationship type is SF} \end{cases} \quad (3.15)$$

What follows is the cash flow calculation just as done previously in the normal cash flow analysis but using the OS and OF instead of the ES and EF. A new Net Present Value (NPV) is calculated for the trial, then it is compared with the highest NPV reached in a previous trial or the initial NPV of the un-optimized schedule if no previous trial was done. If the NPV is a new highest, the trial is stored in the schedule and a new trial begins. To sum up, the steps are as follows:

**Step 1:** If not previously done, the portfolio is calculated for scheduling and cash-flow.

**Step 2:** The lags are initiated as per Equation 3.14

**step 3:** The OS and OF of each activity is calculated as per Equation 3.15

**Step 4:** The cash-flow is calculated using OS and OF

**Step 5:** Compare new NPV with last best NPV or initial portfolio NPV if this is the first trial. If current trial is a new optimum: store it, otherwise: discard it.

**Step 6:** Proceed to Step 2 again if number of trials done is less than the targeted number of trials. Otherwise, finish.

### 3.10 Graphical User Interface (GUI)

A GUI was developed, as specified in the Methodology, using a package called "Tkinter" from the Python standard library. It can be used to create new projects and activities, delete them if necessary, display tables containing them, and it can display plots for the Gantt charts and the cash flow. A screen shot of the GUI on startup is shown in Figure 3.6. The main tool-bar in the top area of the window has seven menus.

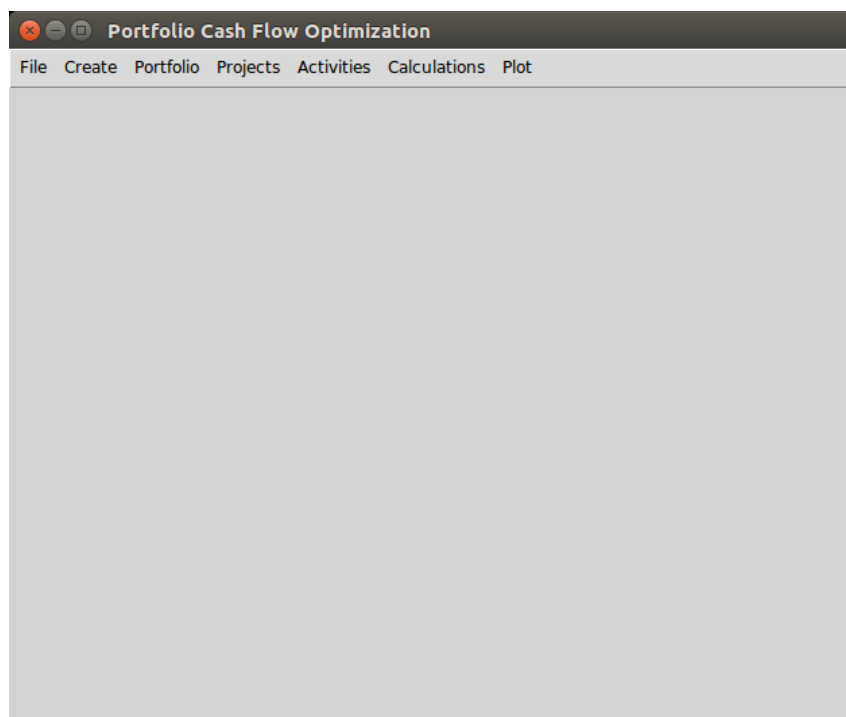


Figure 3.6: Graphical User Interface (GUI) on start-up

The first menu, as shown in Figure 3.7, allows the user to: clear all data; create a new random portfolio, for testing or used as a demo; import validation portfolio, which is a large portfolio used for the validation of the model; "Database Info" will display information about the database, number of projects and activities and relationships, and other useful information; clean database is self explanatory, it will delete create a new empty database, "Export" will export spreadsheets, csv files, plots in PDF format, and logs in txt format for the portfolio and the calculations; "Verify" and "Validate" buttons are used to automate the verification and validation process by importing, calculating, optimizing, and exporting.

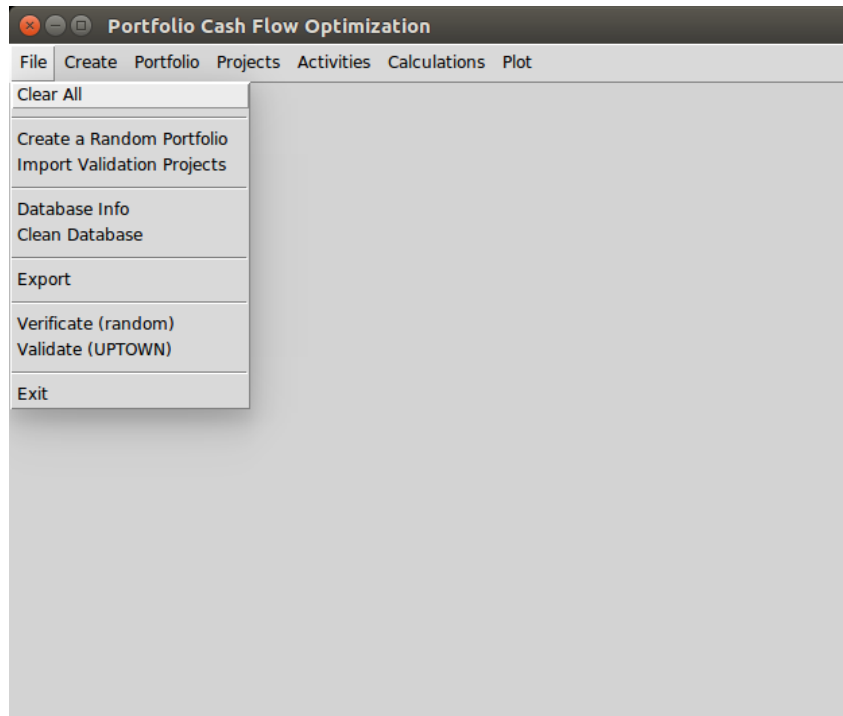


Figure 3.7: GUI: File Menu

The "Create" menu allows for the creation of new projects, activities, or relationships, as shown in Figure 3.8. Each button will show its respective item creation window. The window for the creation of a new project is shown in Figure 3.9, and it requires the project id, name, start, interest, markup %, downpayment %, Invoice interval in days (the time duration between invoices), payment period, retention %, and the retention duration. The window for a new activity is shown in Figure 3.10 and it requires the project for the activity, the activity ID, name, duration in days, and the cost. Finally, the window for a new relationship is shown in Figure 3.11 and it required the project id, the preceding activity id, the successive activity id, and the relationship type, which can be FS, SS, SF, or SS.

Figures 3.13 and 3.14 show the menus that enable the user to see a table of the portfolio, activities, or the relationships. Each one shows its respective table that lists the parameters for each item, these include the inputs and outputs. Figure 3.15 shows the "Calculations" menu, which executes the calculation or the optimization. The calculation must be done for the portfolio before the optimization, in case the portfolio wasn't calculated before, otherwise the optimization will fail to run. Finally, Figure 3.16 shows the plots menu, which enables the user to see many plots for the portfolio, which includes the Gantt charts, cash flows, and overdrafts, optimized or not optimized, as well as discounted to their Present Value, or not discounted.

Examples of the previously mentioned tables are shown in Figures 3.17, 3.18, and 3.19. First, Figure 3.17 shows the table for the activities, which includes all activities in the

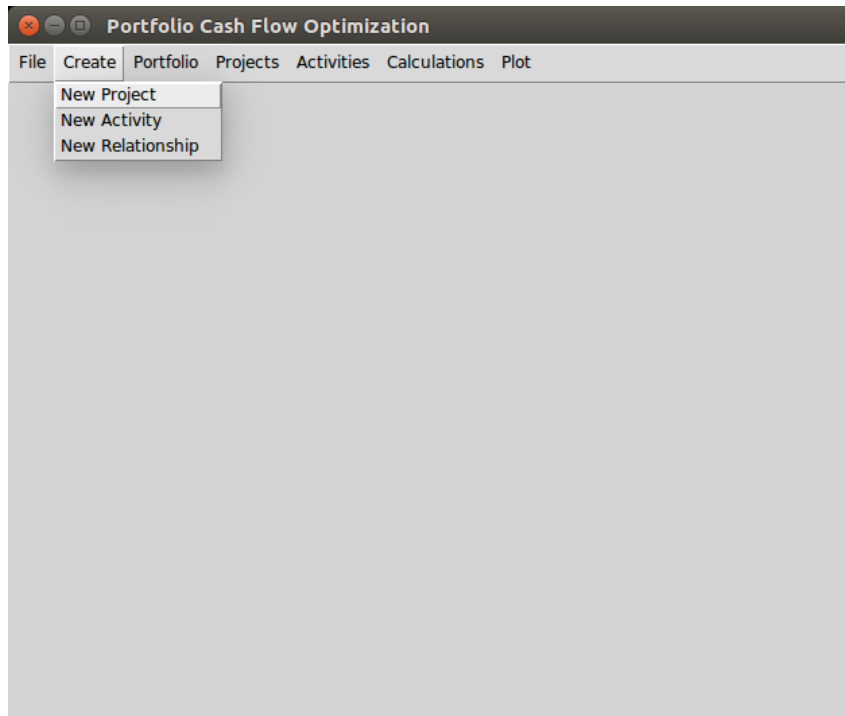


Figure 3.8: GUI: Create New Project Window

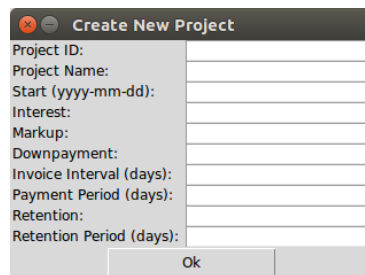


Figure 3.9: GUI: Create New Activity Window

portfolio. All parameters and properties for the activities are shown in that table, including the IDs, names, durations, CPM calculations, lags from the optimization algorithm, and others. Similarly, Figures 3.17 and 3.18 show the tables for the portfolio and the projects, respectively. Again, the tables include all properties for all items. The tables shown in the figures can be scrolled vertically and horizontally to see the remaining items and fields. Also, the user is able to delete selected items

Figure 3.20 shows the Gantt chart for all activities in the portfolio at their earliest start state. Activities in red are critical activities. Green activities are non-critical and their total float is marked in a thin blue bar. Arrows mark the relationships between the activities, where the head of the arrow points to the successor. The location of the arrow on each activity depends on the type of the relationships, so for example a Finish to Start will have an arrow from the end of the predecessor to the start of the successor, and other relationship types have similar logic.

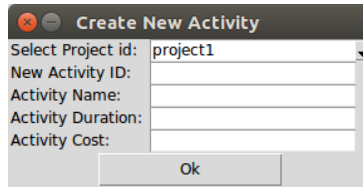


Figure 3.10: GUI: Create New Activity Window

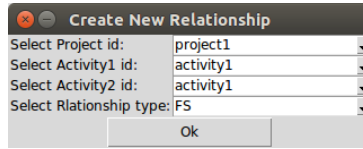


Figure 3.11: GUI: Create New Relationship Window

Figures 3.21 and 3.22, show plots of the overdraft vs. time. The first shows the plot for a random portfolio, while the other shows an overlay of the optimized overdraft on the non-optimized for the same portfolio. Finally, Figure 3.23, shows an optimized Gantt Chart; the thin grey bars span from the Early Start to the Late Finish of each activity. The activity bars are marked in green or red depending on their criticality. This visualization ensures that the user can easily understand the effect of the optimization on the schedule.

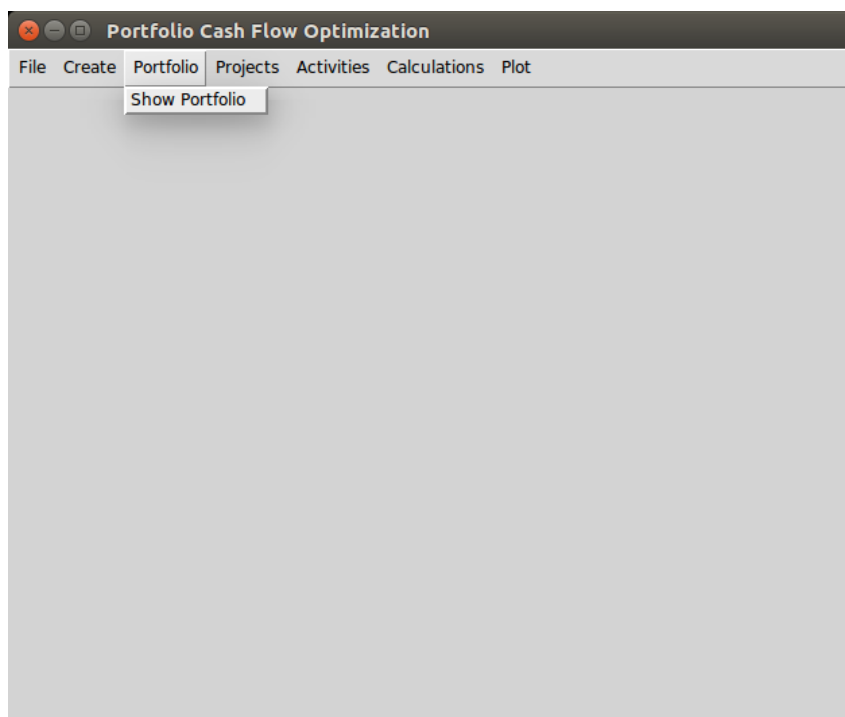


Figure 3.12: GUI: Portfolio Menu

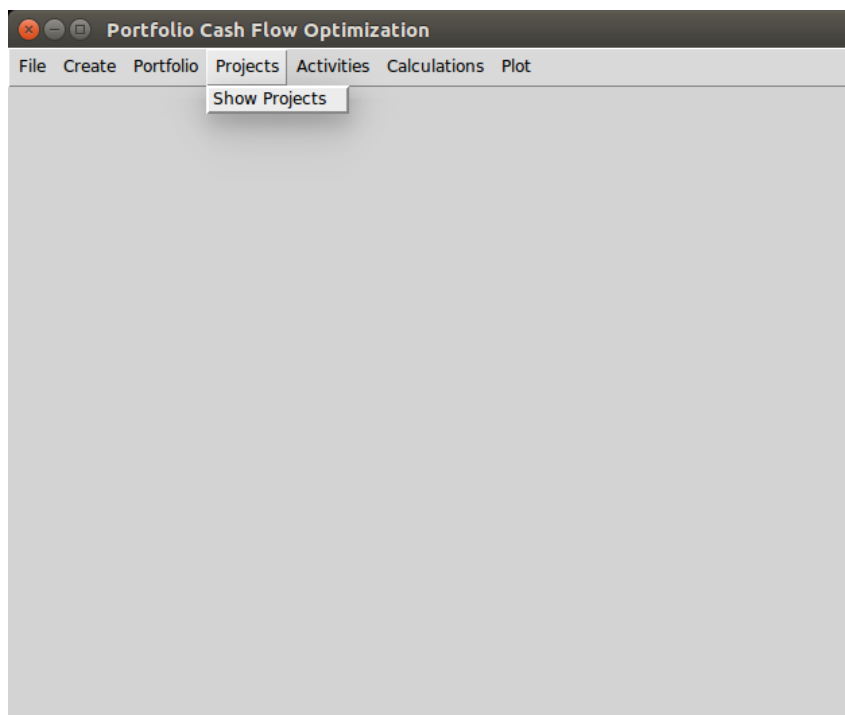


Figure 3.13: GUI: Projects Menu

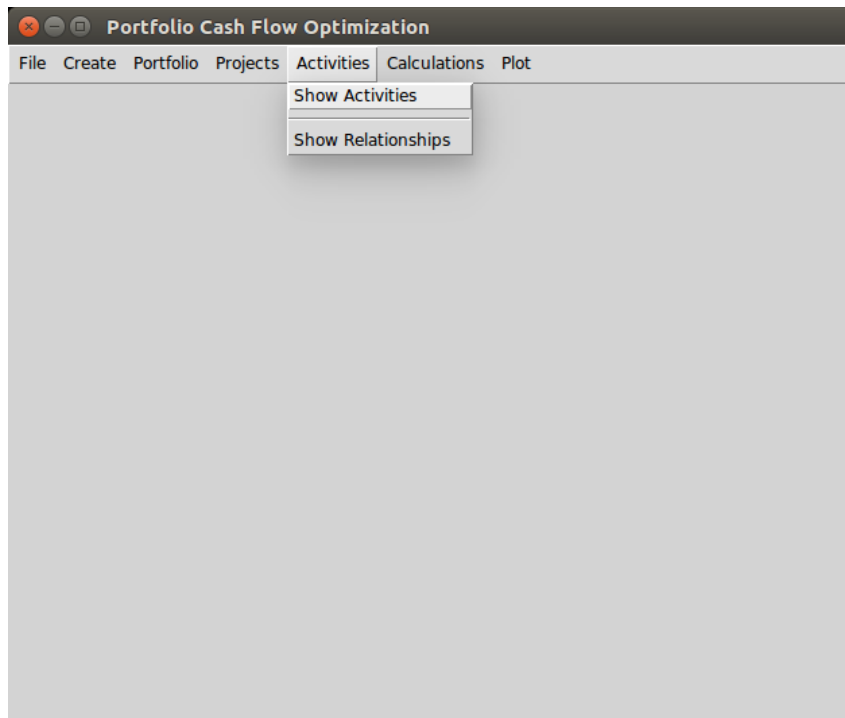


Figure 3.14: GUI: Activities Menu

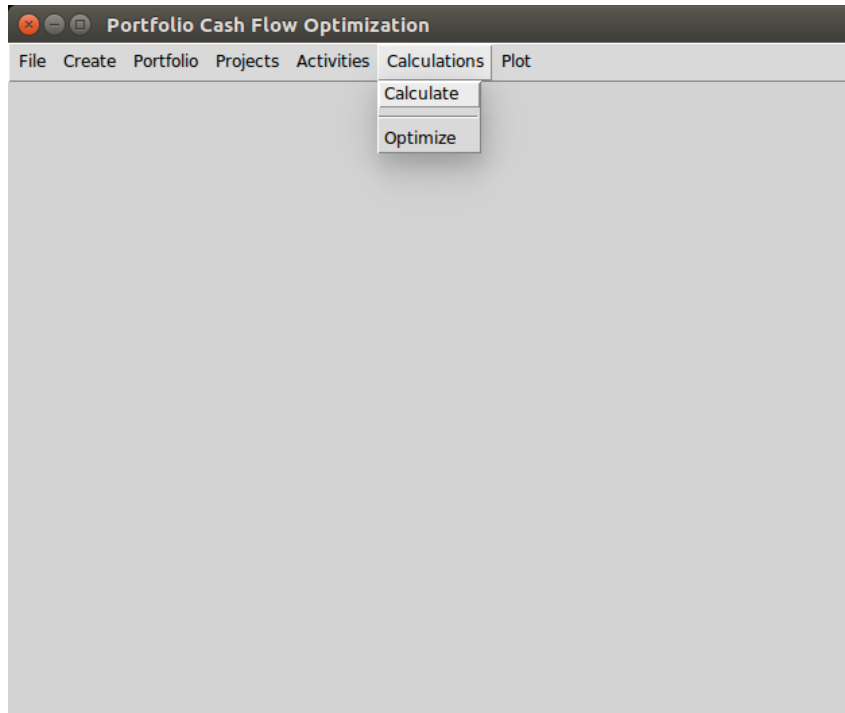


Figure 3.15: GUI: Calculations Menu



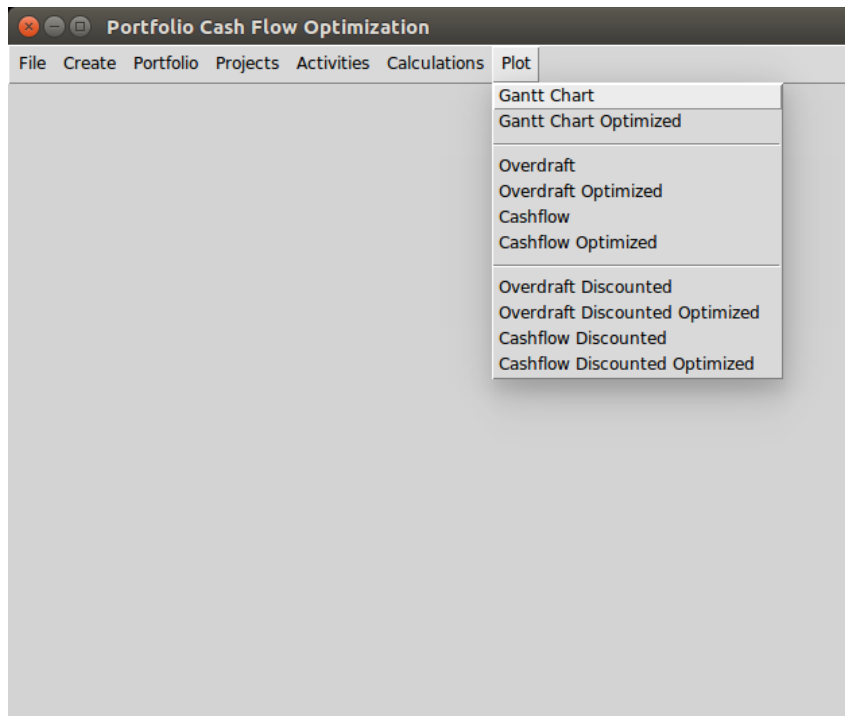
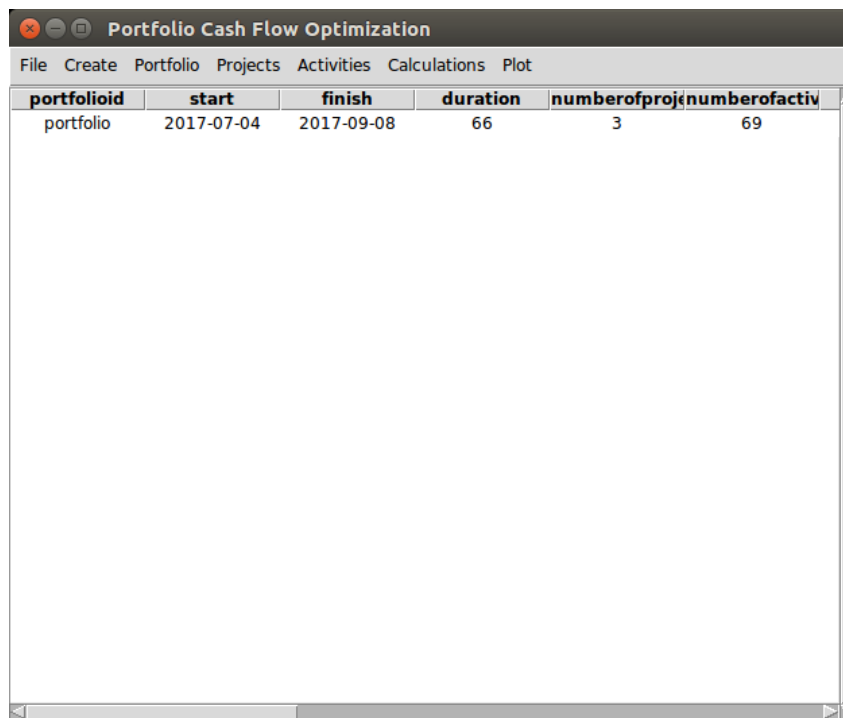


Figure 3.16: GUI: Plot Menu

projectid	activityid	activityname	duration	cost	es
project1	activity1	activity1	14	2.0	2017-07-04
project1	activity2	activity2	11	5.0	2017-07-04
project1	activity3	activity3	14	5.0	2017-07-15
project1	activity4	activity4	18	4.0	2017-07-11
project1	activity5	activity5	19	6.0	2017-07-29
project1	activity6	activity6	19	6.0	2017-07-15
project1	activity7	activity7	19	3.0	2017-07-04
project1	activity8	activity8	12	8.0	2017-07-29
project1	activity9	activity9	14	9.0	2017-07-04
project1	activity10	activity10	10	6.0	2017-07-15
project1	activity11	activity11	16	4.0	2017-07-29
project1	activity12	activity12	14	9.0	2017-07-11
project1	activity13	activity13	15	1.0	2017-07-30
project1	activity14	activity14	11	6.0	2017-07-23
project1	activity15	activity15	12	1.0	2017-07-17
project1	activity16	activity16	12	8.0	2017-07-18
project1	activity17	activity17	13	1.0	2017-07-23
project1	activity18	activity18	20	7.0	2017-07-23
project1	activity19	activity19	17	4.0	2017-07-04
project1	activity20	activity20	18	2.0	2017-07-30
project2	activity1	activity1	14	4.0	2017-09-04

Figure 3.17: GUI: Activities Table

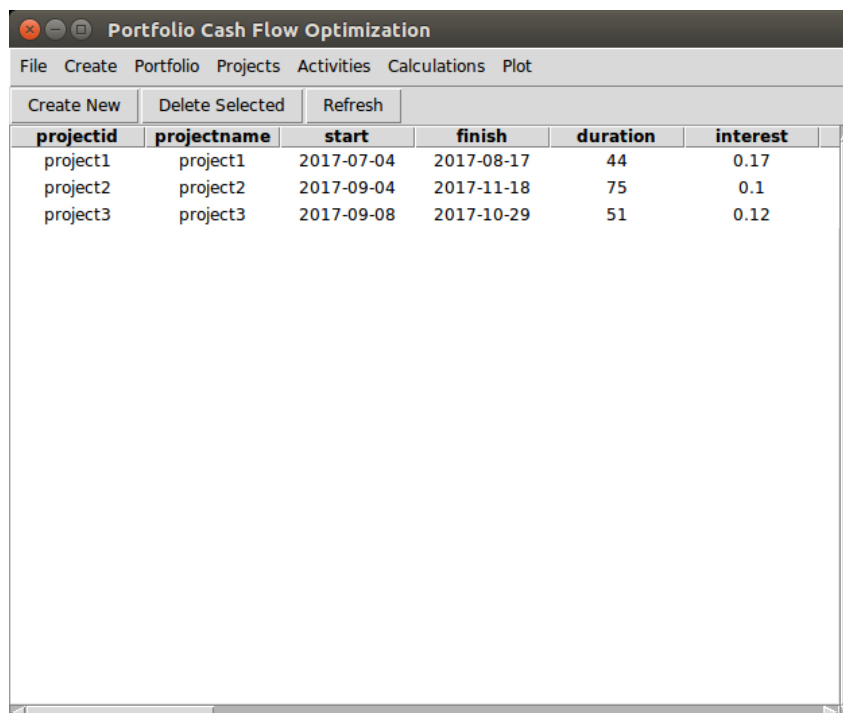


Portfolio Cash Flow Optimization

File Create Portfolio Projects Activities Calculations Plot

portfolioid	start	finish	duration	numberofproj	numberofactiv
portfolio	2017-07-04	2017-09-08	66	3	69

Figure 3.18: GUI: Portfolio Table



Portfolio Cash Flow Optimization

File Create Portfolio Projects Activities Calculations Plot

Create New Delete Selected Refresh

projectid	projectname	start	finish	duration	interest
project1	project1	2017-07-04	2017-08-17	44	0.17
project2	project2	2017-09-04	2017-11-18	75	0.1
project3	project3	2017-09-08	2017-10-29	51	0.12

Figure 3.19: GUI: Projects Table

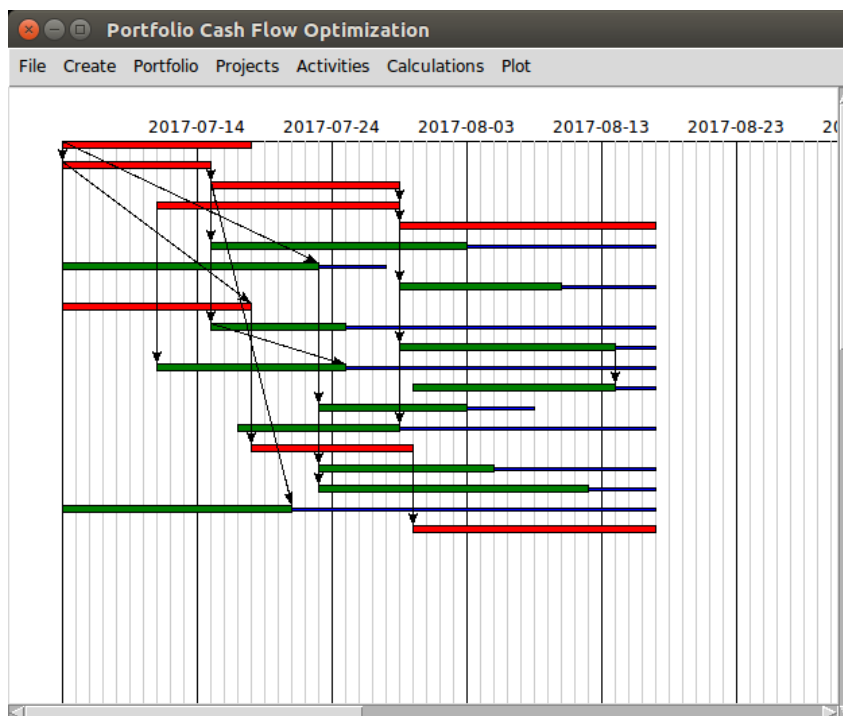


Figure 3.20: GUI: Gantt Chart

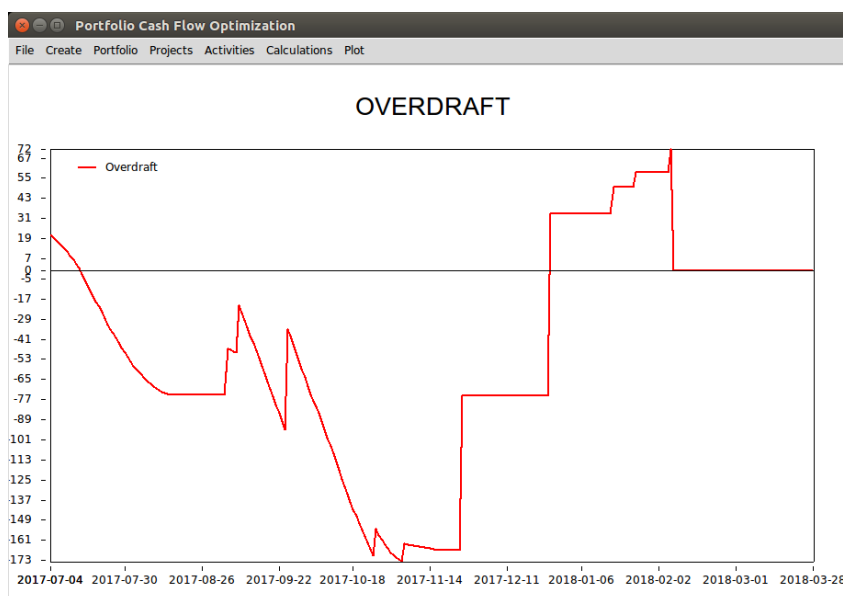


Figure 3.21: GUI: Overdraft Plot

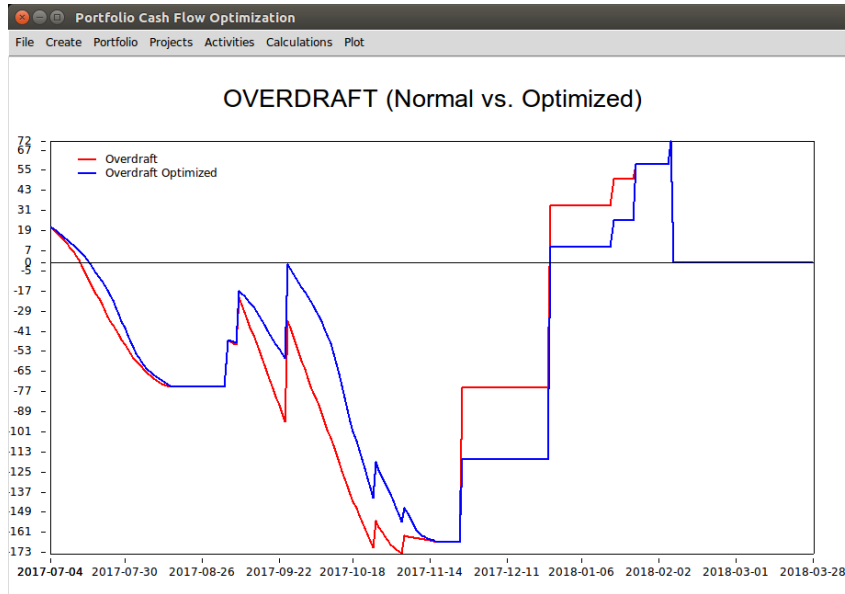


Figure 3.22: GUI: Optimized Overdraft Plot

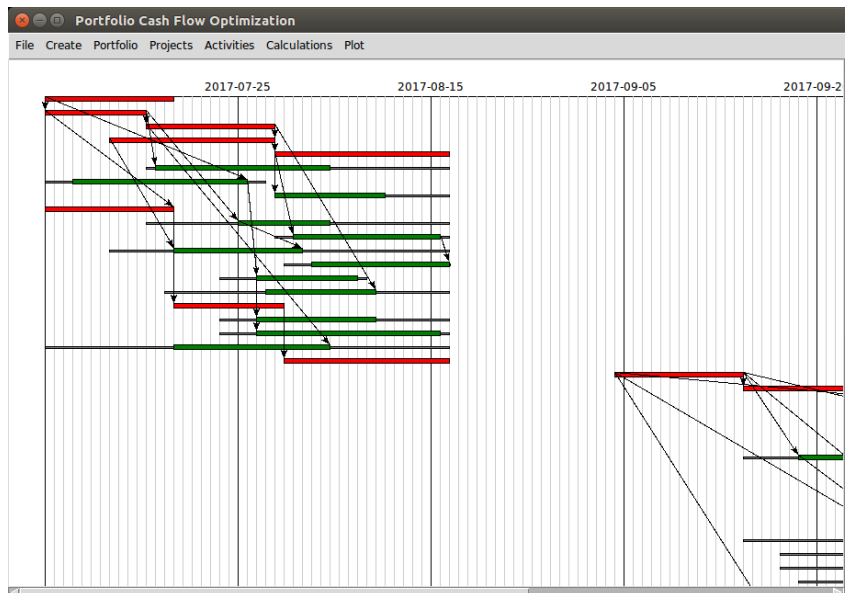


Figure 3.23: GUI: Optimized Gantt Chart



# Chapter 4

## Results and Discussion

This chapter will cover the results of the model. This includes the verification which was done using randomized sets of portfolios, the sensitivity analysis using the interest rate and the cost as the parameters under study, a CPU time test. Finally, the results of the validation, which was done using a real and very large portfolio, are described and discussed. The entire Python Code used is available in Appendix A.

### 4.1 Verification

#### 4.1.1 Verification Method

Verification was done using randomly generated sets of portfolios. An algorithm was written to generate random portfolios with random number of projects, activities, and all needed parameters. The randomized portfolios then undergo analysis and optimization. For the sake of verification, the portfolios generated had 3 projects each, where each project had a random number of activities between 25 to 30 activities. The start of each project was randomized for up to 300 days from the start of the portfolio. Each activity, except one activity had a random number of predecessors where the probability of having one relationship was 75% and the probability of having 2 relationships was 25%, while the relationship type was equally randomized. Other parameters for duration, costs, and financial parameters were randomized as well. The stopping criteria is an improvement of 0.002% on a moving average of the last 3 best trials, or 20 trials with no improvement. Various other settings were tried as well, including the financial parameters to test the model, but they are fixed for the examples given in this section. The randomization of the parameters for the verification created random portfolios with different durations, number of activities, and relationship types, which tested the performance of the model thoroughly.

### 4.1.2 Verification Results

For the purpose of this thesis, five verification trials are presented. More were done to and they resulted in the same conclusion. Figure 4.1 shows the 5 portfolios used for the verification, and as mentioned earlier they are completely randomized. The Gantt charts of the portfolios are shown in Figure 4.2, showing the start and end of each project in each of the five portfolios. The activities contained in them as shown in Figure 4.3. As the portfolios are random, they have a random number of activities, and the criticality ratio is also variable. Figure 4.4 shows the Gantt chart of the projects, where each project has a random start dates, duration, and finish dates. The activities contained in the projects, as shown in Figure 4.3, are also randomized. So, generally this methodology allows for the rigor testing of the model under different conditions.

Moving fast forward to the optimization, then to the optimized cash flows. Figure 4.5 shows the optimization process in an informative plot where the trial NPV is plotted against the number of trials. It can be noted that the model converges in all cases. In some rare cases, the optimum NPV occurs when the activities have an early start state, therefore the model won't improve, otherwise the model converges. The optimized cash flow in shown in Figure 4.6. The optimized overdraft is shown in Figure 4.7.

### 4.1.3 Verification Discussion

The methodology of the verification allowed for the rigor testing of the model, by creating custom randomized portfolios to test different costs, interests, number of activities, different relationship types, etc. The random sets used are shown in Figure 4.1, and were successfully randomized; the number of activities are different and the number of critical activities are different for each project. As shown in Figure 4.3, the model successfully scheduled the activities in each project according to their assigned relationships, which are indicated by arrows, and as shown in Figure 4.2, the model succeeded in calculating the start and end of each portfolio according to the scheduling of the activities in them. Next, for the cash flow analysis, Figures 4.6 and 4.7 show that the calculation of the cash flow and overdraft, before optimization, was successful; the cash out has an shape similar to an S-curve, to some extent, which is typical to constriction project, and the end of the cash-out sums up to the total cost of the portfolio; the cash in has steps matching the down-payments, invoice payments, and retention receipts at the end of the projects, and the curve ends with a value equal to the total price of the portfolio; while the overdraft is correct as it matches the difference between the cash in and the cash out curves, with an end value that matches the profit from the portfolio.

In the same Figures (4.6 and 4.7), the discounted value, the Present Value (PV), of the cash flow curves and the overdraft curves are increasingly lower than the Future Value (FV) curves for each point in time as the time increases. This is due to the power of the

time value of money because a sum of money will have a lower value as time progresses. Finally, for the optimization, the trials are shown in Figure 4.5, and Net Present Value (NPV) which is the objective of the optimization, is converging to a maximized value in progression with the number of trials. The stopping criteria for the max number of trials, which was 20 trials, was the deciding factor in the sets under study. The optimized Gantt charts for the projects are shown in 4.4, where the activity Optimized Start (OS) was set to a value in their total float, and the relationships between them were also respected. The effect of the optimization is shown for the cash flow in Figure 4.6 and for the overdraft in Figure 4.7. The optimization seems to have generally modified the start of the activities in a way that would balance between receiving cash as early as possible, while at the same time reducing the peaks in the overdraft. So the NPV, as an indicator, may have solved multiple objectives. This seems logical because in real life, a contractor would rather receive cash early, for investment in other projects, and at the same time should attempt to reduce maximum overdraft to reduce the investment from the company's resources or external loans. In overall regarding the optimization, it is successfully converging and had positive effects on the cash flow of the portfolio. The outcomes of the verification are satisfying; the cash flows and the overdrafts have typical shapes for construction projects. Checks on the values were matching. The optimization process converged in all cases. The optimization seemed to find a balance between getting payments early for maximum time value of money, and getting a lower negative cash flow, as it is noted that the peaks in the cash flow are affected by the optimization.



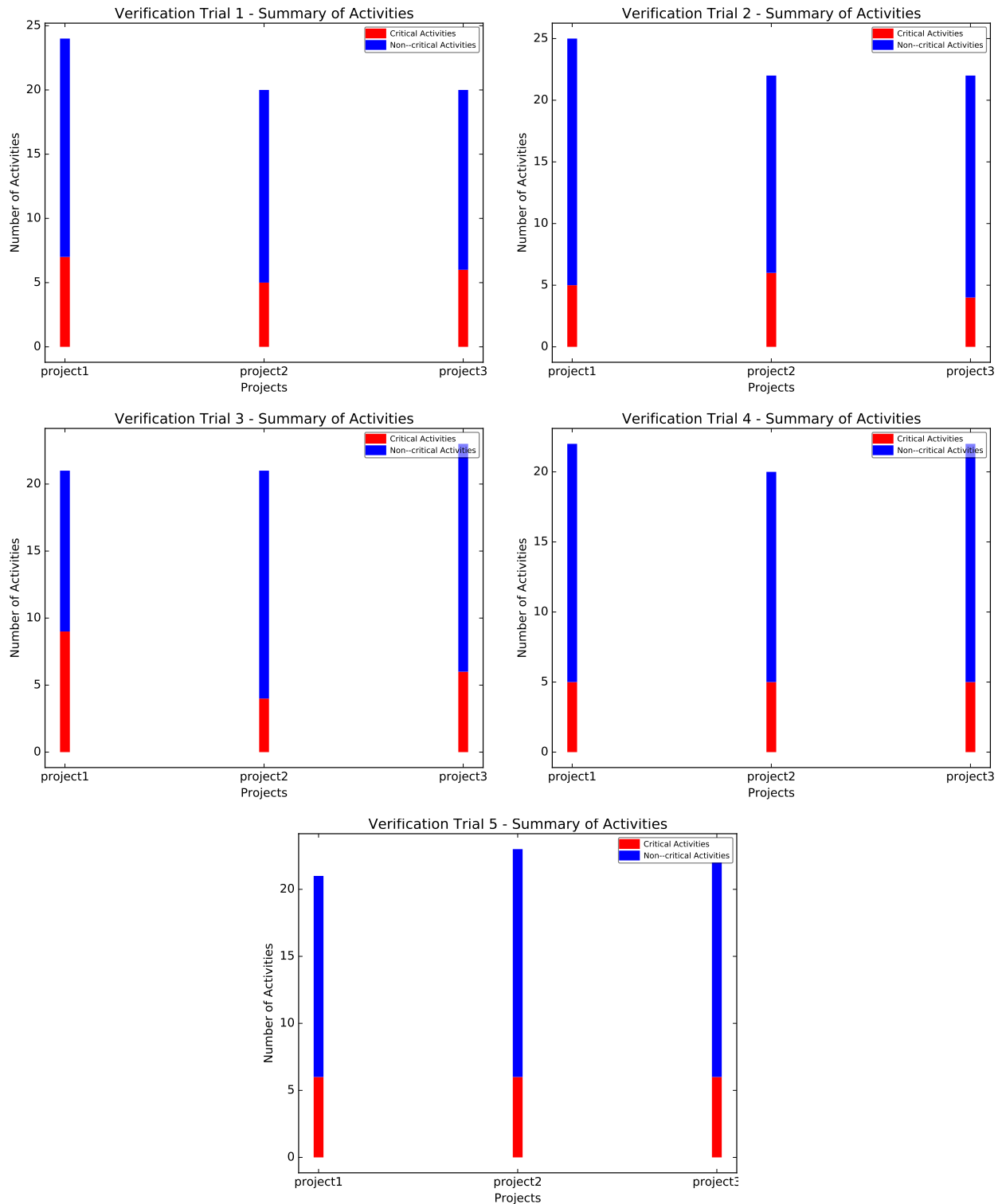


Figure 4.1: Summary of the five portfolios used and their project Gantt charts

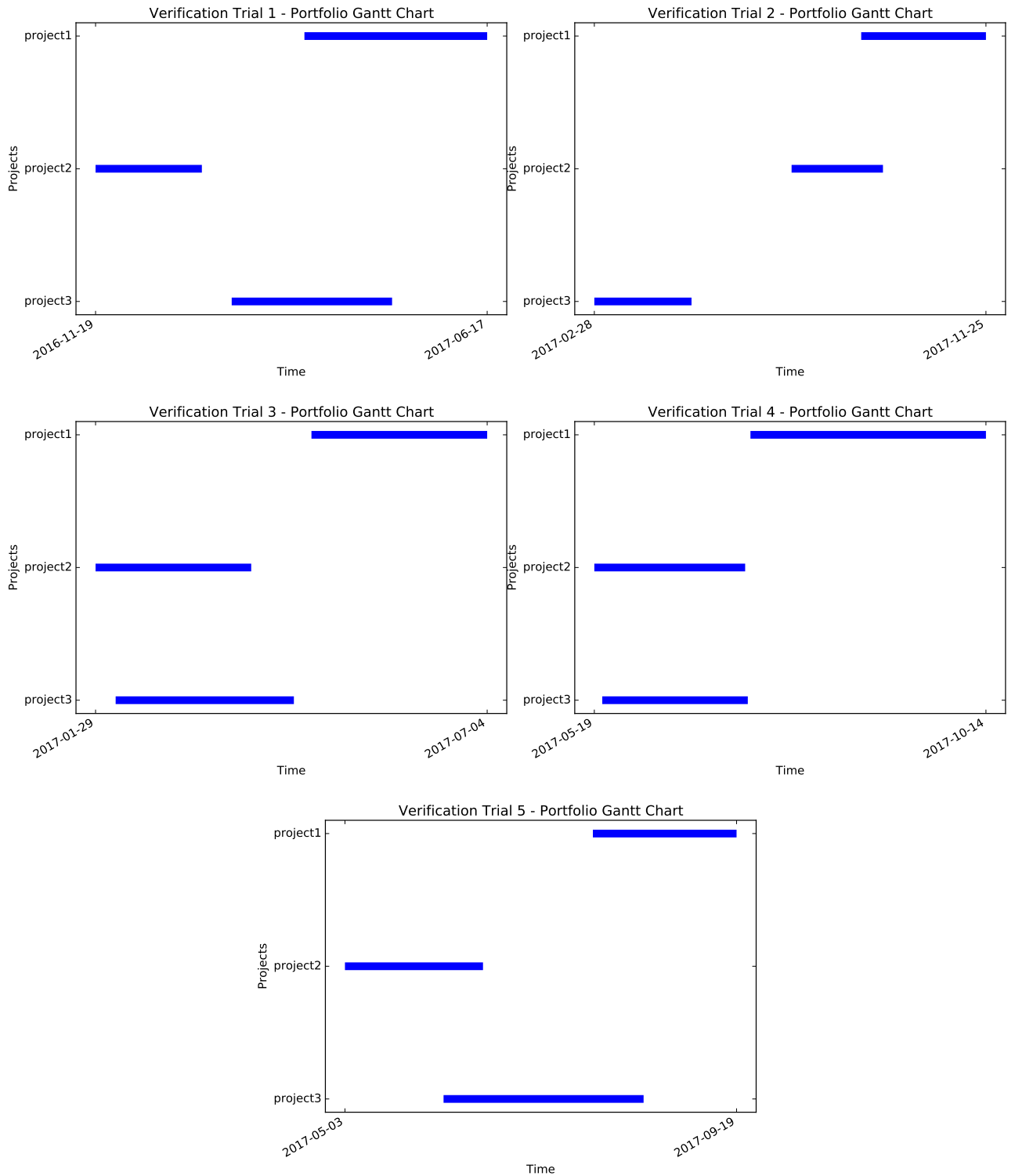


Figure 4.2: Summary of the five portfolios used and their project Gantt charts

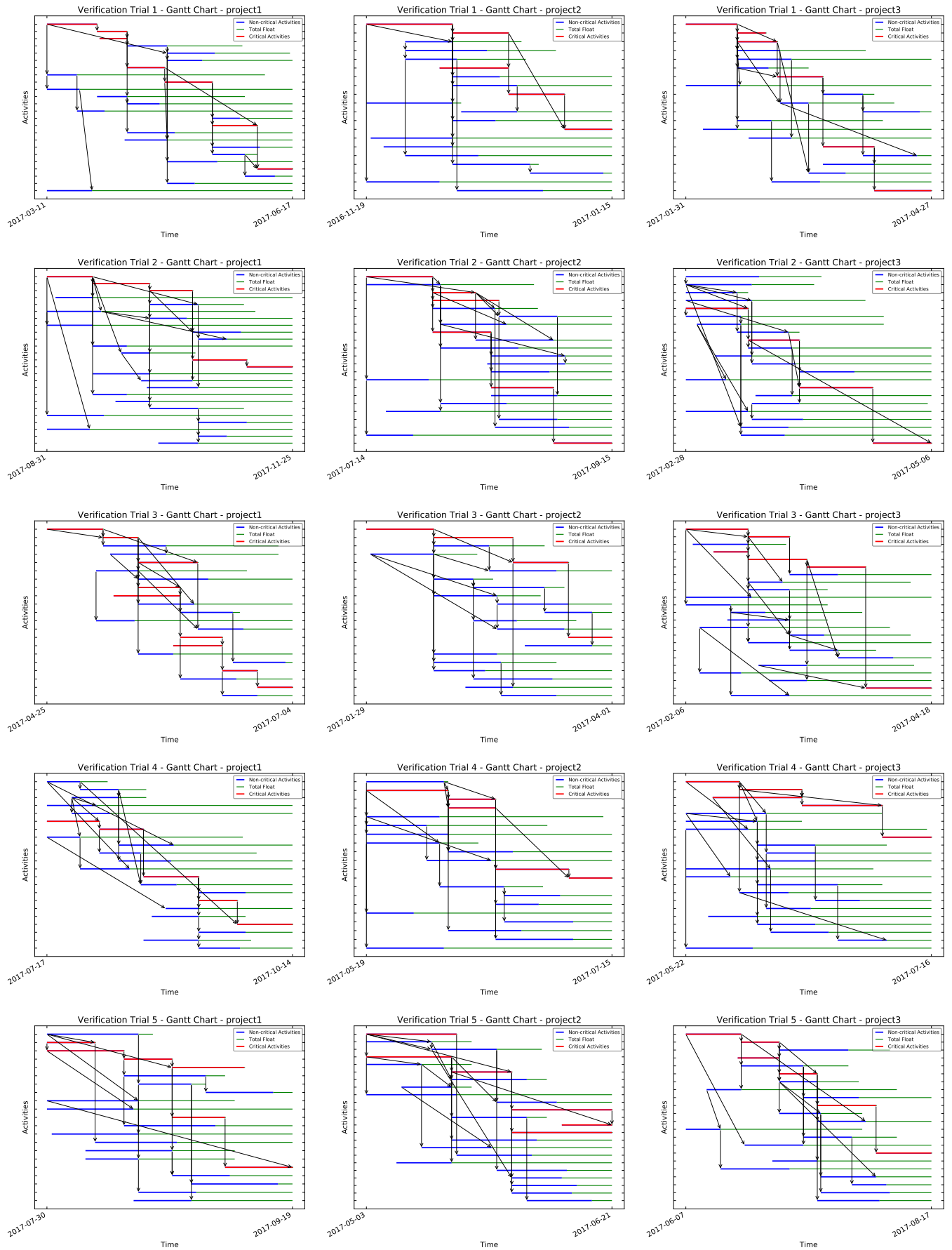


Figure 4.3: Gantt charts for the verification projects

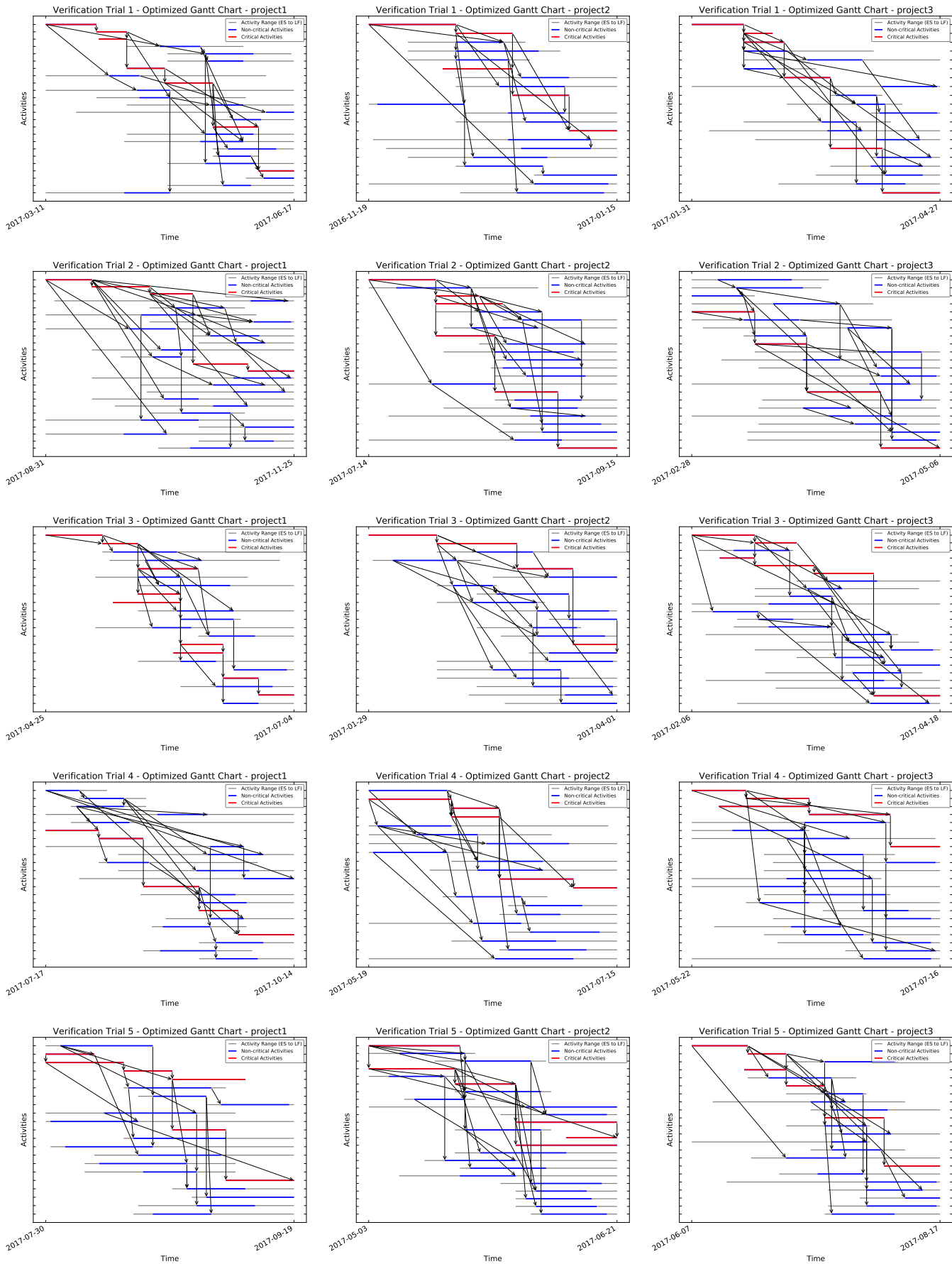


Figure 4.4: Optimized Gantt charts for the verification projects

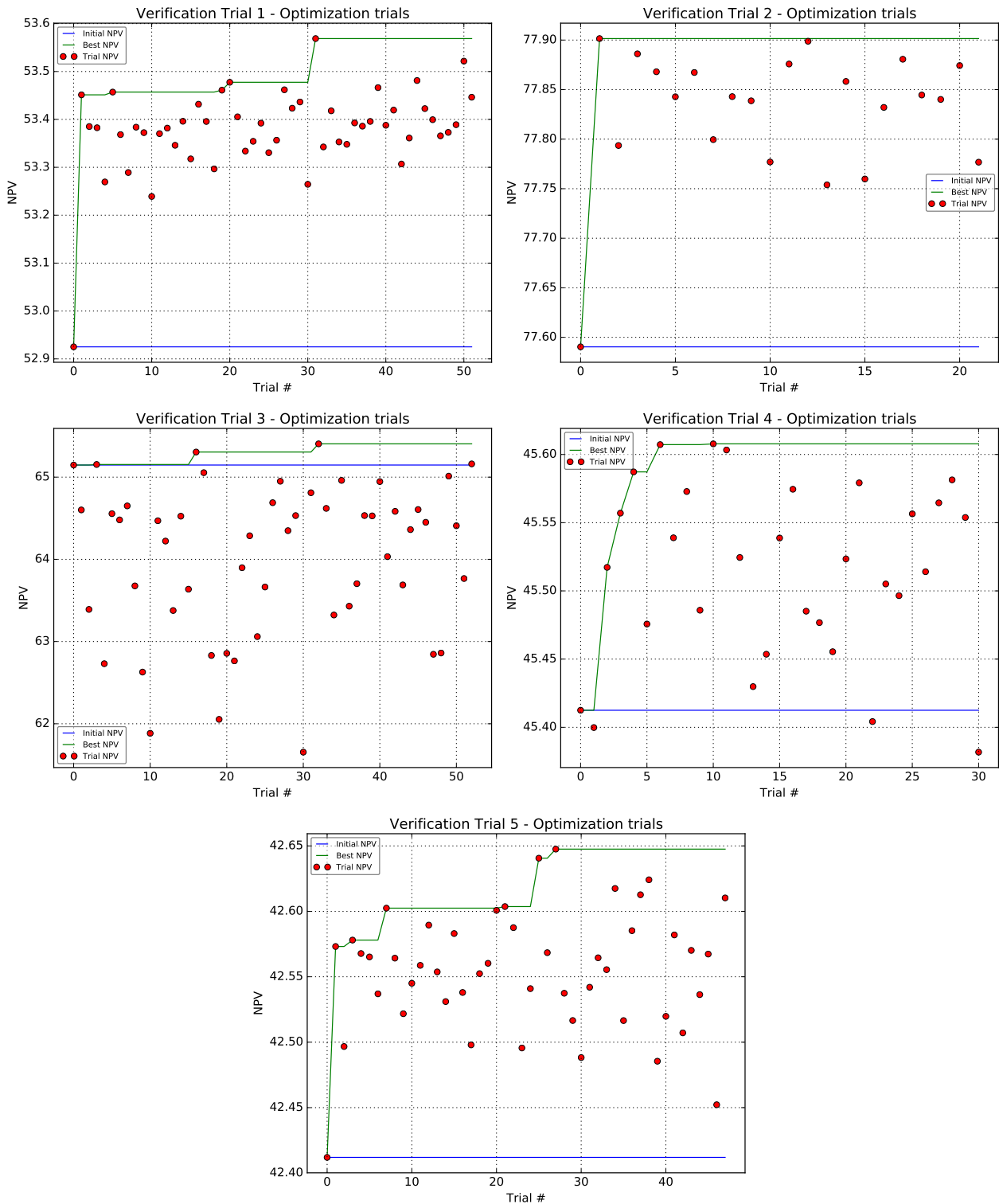


Figure 4.5: Optimization trials for each on the 5 portfolios

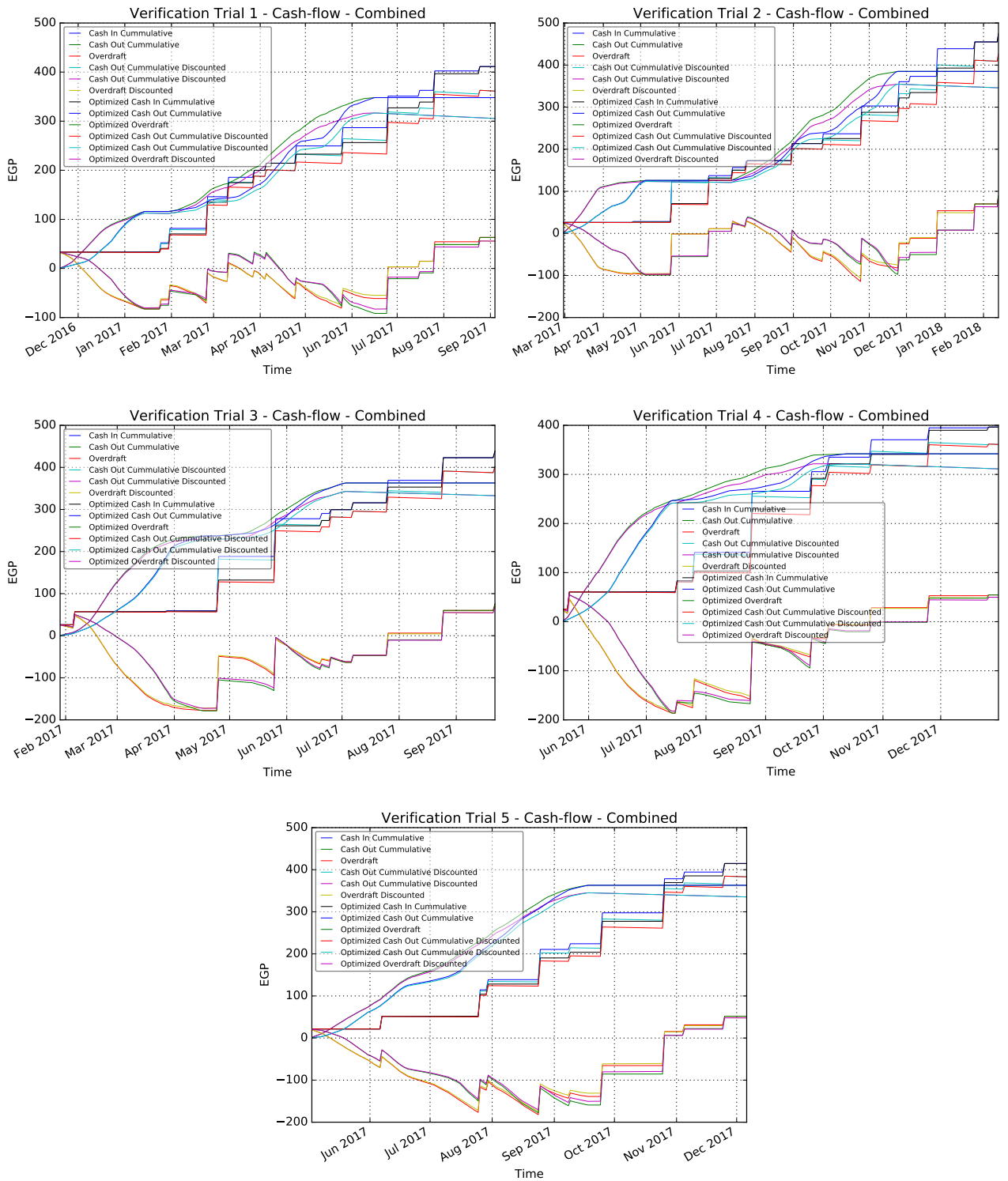


Figure 4.6: Optimized Cash Flow for the Portfolios

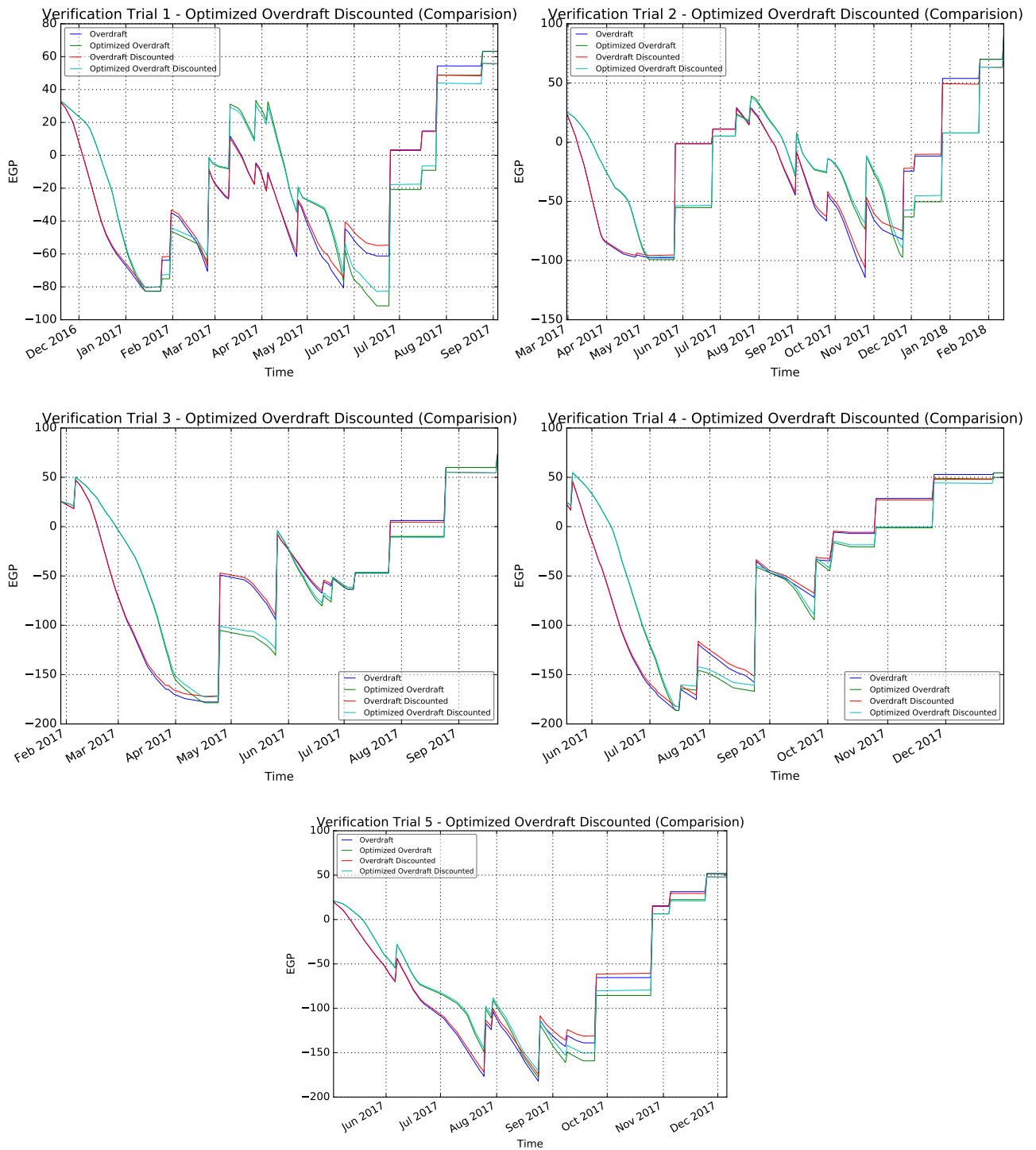


Figure 4.7: Optimized Overdraft for the Portfolios

## 4.2 Sensitivity Analysis

### 4.2.1 Sensitivity Analysis Method

A sensitivity analysis was conducted to ensure that the final main result, the Net Present Value (NPV), is calculated correctly according to other parameters. Two parameters were chosen, they are the interest rate and the cost, and their implication on the NPV for a chosen portfolio was tested. The interest was tested from 0 to 50 per cent, with increments of 2 per cent. This parameter was initialized for each project in the portfolio, and the NPV was calculated for each. While, for the sensitivity analysis of the the cost, the costs for the activities was incremented for up to 200 per cent of the original cost, with increments of 10 per cent. This increased the cost of the portfolio and the NPV was calculated as such.

### 4.2.2 Sensitivity Analysis Results

The results for the Interest Rate sensitivity analysis is shown in Figure 4.9, the plot shows a slight second degree curve. while for the sensitivity analysis for the cost which is shown in Figure 4.8, the plot resulted in a straight first degree line. An overlay of the sensitivity analysis for the interest rate and the cost combined is shown in 4.10. The same plot but with the NPV measured in percentage increase, for easier analysis, is shown in Figure 4.11.

### 4.2.3 Sensitivity Analysis Discussion

The charts obtained from the sensitivity analysis of the cost and interest rate against the NPV matches expectations perfectly. To begin with the sensitivity analysis for the interest rate, it was expected to be a curve, because ,as discussed before in Section 3.8, the NPV is calculated generally as shown in Equation 4.1. So, due to the fact that the interest is raised to the power of the time period, it has a curve. As for the sensitivity analysis for the cost, and again in accordance with Equation 4.1, the relationship between the cost and the NPV is linear, therefore the plot shows a straight line. This concludes that the model behaves correctly regarding these main parameters.

$$NPV = \sum \frac{Cost}{(1 + Interest)^n} \quad (4.1)$$



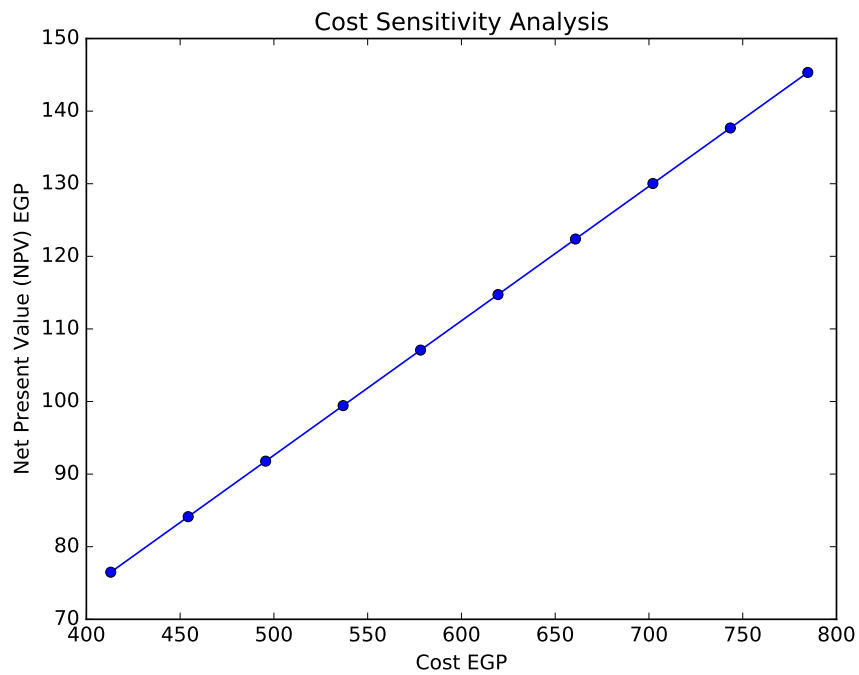


Figure 4.8: Cost Sensitivity Analysis

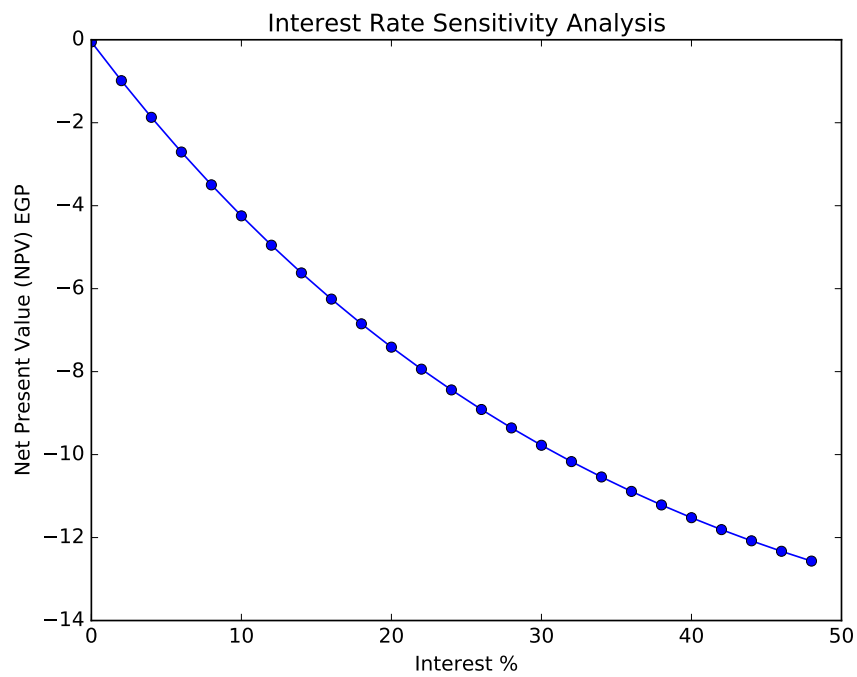


Figure 4.9: Interest Rate Sensitivity Analysis

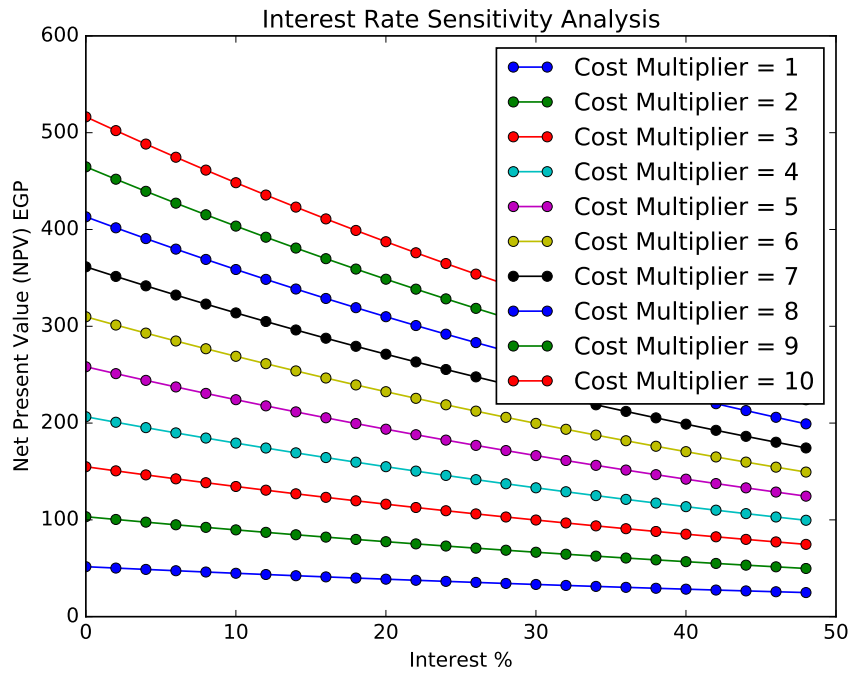


Figure 4.10: Overlay of The Sensitivity Analysis Results for Interest Rate and Cost

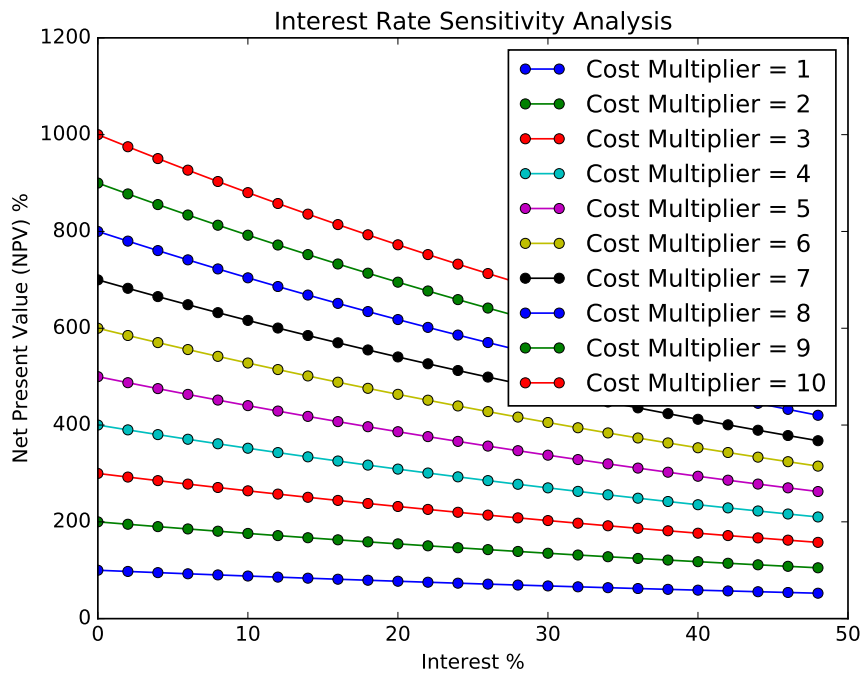


Figure 4.11: Overlay of The Sensitivity Analysis Results for Interest Rate and Cost in percentage increase

## 4.3 NPV Improvement Test

### 4.3.1 NPV Testing Method

This test was done to indicate the impact of the model on the improvement of the NPV. This was done by using the same methodology for the verification, but repeated on a number of trials to get different optimized NPVs. Portfolios were generated randomly with the following conditions: each portfolio had three projects, and each project had 20 to 25 activities. Each project's start date was set randomly for up to 300 days from the start of the first project. The interest, markup, and down-payment percentages were set as 10 to 20%, 15 to 25%, and 15 to 25%, respectively. The payment period and the retention period were set to 56 and 80 days, respectively. The test was done for 200 trials and the values were recorded.

### 4.3.2 NPV Testing Results

This results of the test are shown on Figure 4.12. The x-axis shows the improvement as  $NPV_{Optimized}/NPV_{Original}$ . It shows that most of the numbers lie between 0.5% to 1% improvement. For some projects, that value increased for up to 2.5%.

### 4.3.3 NPV Testing Discussion

The test showed that the improvement in the NPV that the model can achieve relies heavily on the nature of the project, this includes the number of activities, the relationships between them, and the available float, as well as the financial parameters for the projects. In some projects, the optimized NPV is the original NPV, which means that the early start and finish state of the activities is the optimum case and no improvement can be made. Generally, the percentage of improvement for the NPV is small, but for large projects it is significant as a sum of money.

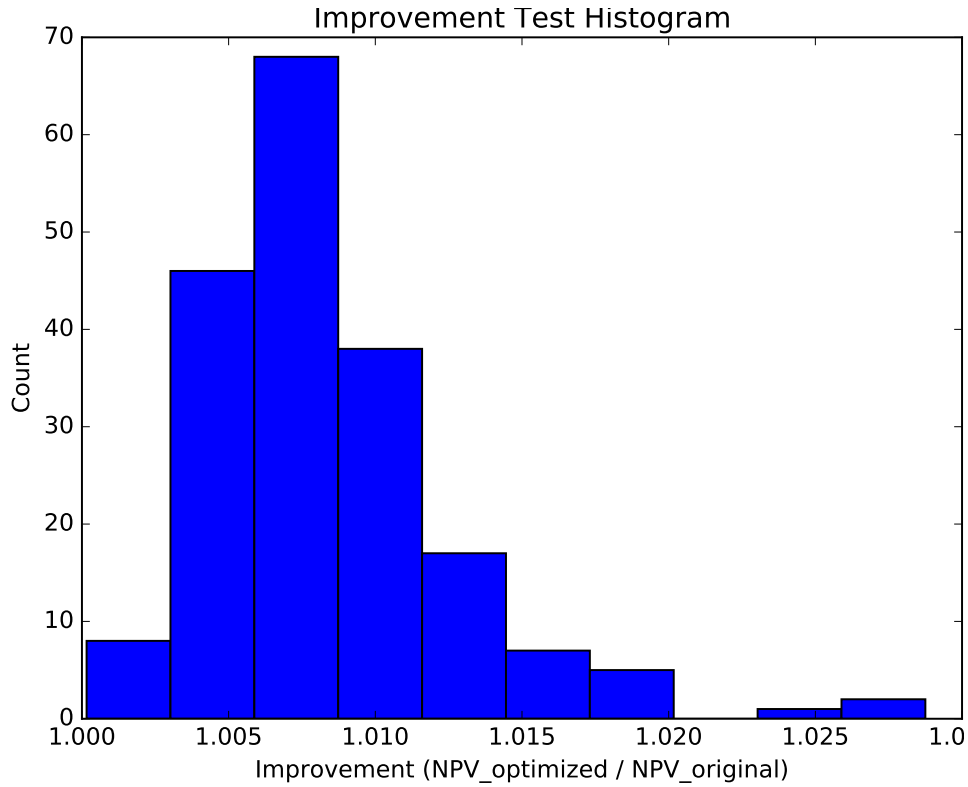


Figure 4.12: Histogram of Improvement in NPV for the trials.

## 4.4 CPU Time Test

### 4.4.1 CPU Time Test Method

A test for the CPU time was done to relate it to the size of the portfolios. Trials were done for random portfolios where each one had 3 projects that contained between 50 and 2000 activities. The stopping criteria was the same as the verification, and the randomization of the relationships was done in the same way as well. The time to optimize each project was recorded. In order to compare those time durations with the size of the projects, Correlation was done between time, number of activities, number of relationships, number of activities + number of relationships, and the number of activities \* number of relationships.

### 4.4.2 CPU Time Test Results

The correlation results are shown in Table 4.1. There is a fair and approximately equal correlation between time and the other variables. A plot between CPU Time Vs. Number of Activities + Number of Relationships is shown in Figure 4.13. There is a positive correlation between those variables, but the deviation increases as the number of activities and relationships increase.

Table 4.1: Correlations for CPU time tests

Correlation	Number of Activities	Number of Relationships	Number of Activities X Number of Relationships	Number of Activities + Number of Relationships	Time (secs)
Number of Activities	1	-	-	-	-
Number of Relationships	0.999	1	-	-	-
Number of Activities X Number of Relationships	0.987	0.987	1	-	-
Number of Activities + Number of Relationships	0.999	0.999	0.987	1	-
Time (secs)	0.656	0.658	0.652	0.657	1

#### 4.4.3 CPU Time Test Discussion

The results obtained from the CPU time test have an expected positive trend; as the number of activities and relationships increase, the complexity increases and the CPU time increases. The spread of the time as the complexity increases, however, is intriguing; it could be due to the random nature of the inputs, and/or the random nature of the solver. It is noted that in large projects, such as the one in the validation of this thesis, there may be multiple complicated relationships for activities, meaning that a single activity has a high number of relationships. This condition increases the computational effort in the model heavily. Overall, the CPU time obtained using this model is satisfactory,

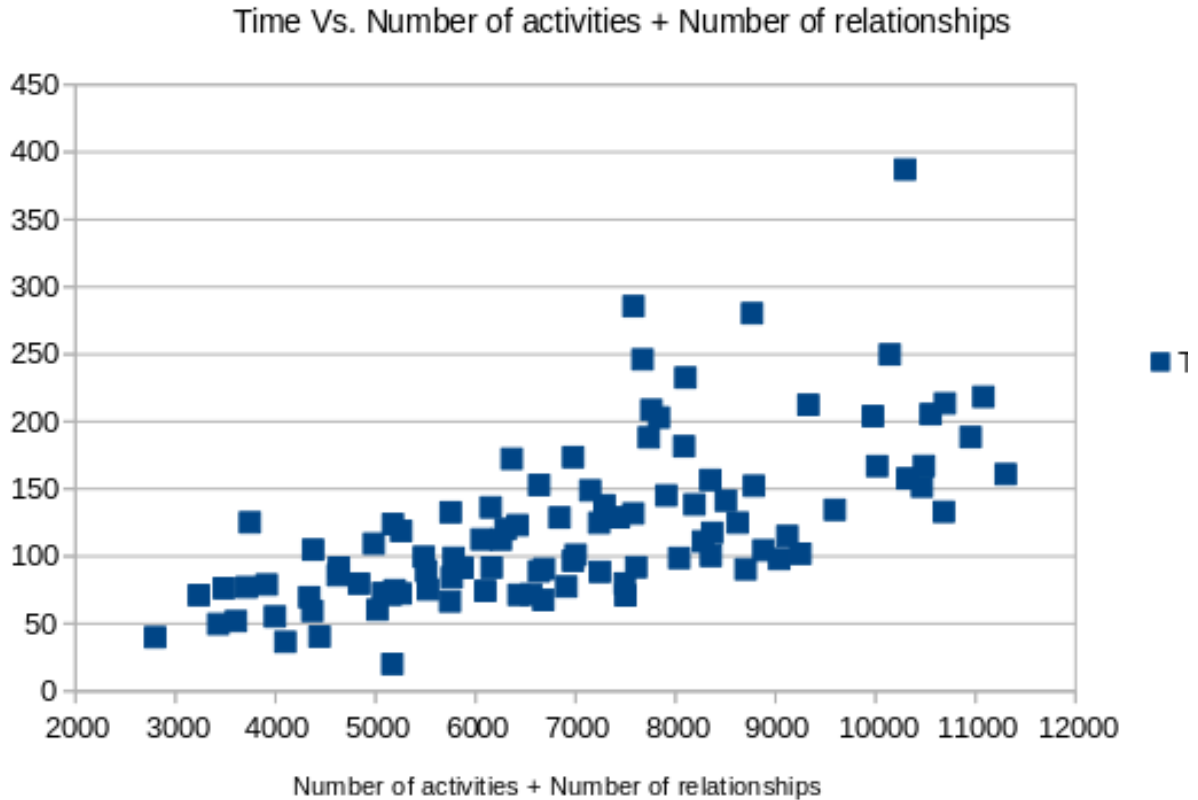


Figure 4.13: CPU Time Vs. Number of Activities + Number of Relationships

## 4.5 Validation

### 4.5.1 Validation Method

Validation was done a portfolio of projects, from actual projects by a contractor. General Information about the projects used are shown in Table 4.2 and Figure 4.15. The portfolio includes three residential projects in Cairo, under construction at the same construction company. Two of them are Villas and the third is apartment buildings. Further details are confidential as per the request of the company. The validation is test of a real and applicable situation. The portfolio used is a relatively very large one; The total number of activities is 28,994 activities, distributed as 6489, 8073, and 14432 activities for each of the projects. The total number of relationships is 69,717 relationship. The stopping criteria is an improvement of 0.002% on a moving average of the last 3 best trials, or 20 trials with no improvement.

### 4.5.2 Validation Results

The results for the validation are shows in Figures 4.17, 4.20, and 4.21. The initial NPV was 432,964,013. The Optimized NPV was 433,150,506. The improvement was 186,493,

Table 4.2: Projects used for the validation

-	Start	finish	Cost	Total Activities
Project 1	03/25/13	03/25/15	102,000,002.57	6,489
Project 2	01/01/14	02/19/16	128,190,586.00	8,073
Project 3	10/11/14	04/27/17	272,000,000.00	14,432

which is an 0.04% improvement from the initial NPV. This result was achieved in 4 hours and 39 minutes. The validation was redone with different stopping criteria, by increasing the max number of trials without improvement to 20 trials, but no significant improvement was achieved. All optimization Plots are shown in the following figures.

### 4.5.3 Validation Discussion

The schedule was calculated successfully, and the cash flow as well. The cash flow, as shown in Figure 4.16, has a typical shape of a cash flow for a construction project. The cash in has steps that follow the invoicing of the three projects. It should be noted, as mentioned before, that this portfolio is huge and is computationally intensive. Moving on to the optimization, the cash flow was optimized as shown in Figure 4.17. The number of trials is small, but a notable improvement in the NPV was achieved. The optimized cash flow is shown in Figure 4.20, and the optimized overdraft is shown in Figure 4.21. It is noted that there is a trend that favors early payment, but not excessively, which seems to be logical, as early payment would make benefit from a higher time value of money, but, on the other hand, increased cash out in respect to the cash in would result in a harmful and excessive negative cash flow. So, it seems that some sort of balance is being achieved. Overall, the main concern after finishing the validation is the long time spent for calculating the project, in specific in the scheduling process. This is the reason that made evolutionary algorithms unfavorable due to them required an initial population, which would in turn require extensive computational power and weeks of computer time. The use of an algorithm or a heuristic that doesn't necessarily be deterministic but would have a satisfactory accuracy would be valuable, especially if it allow for parallel computation.

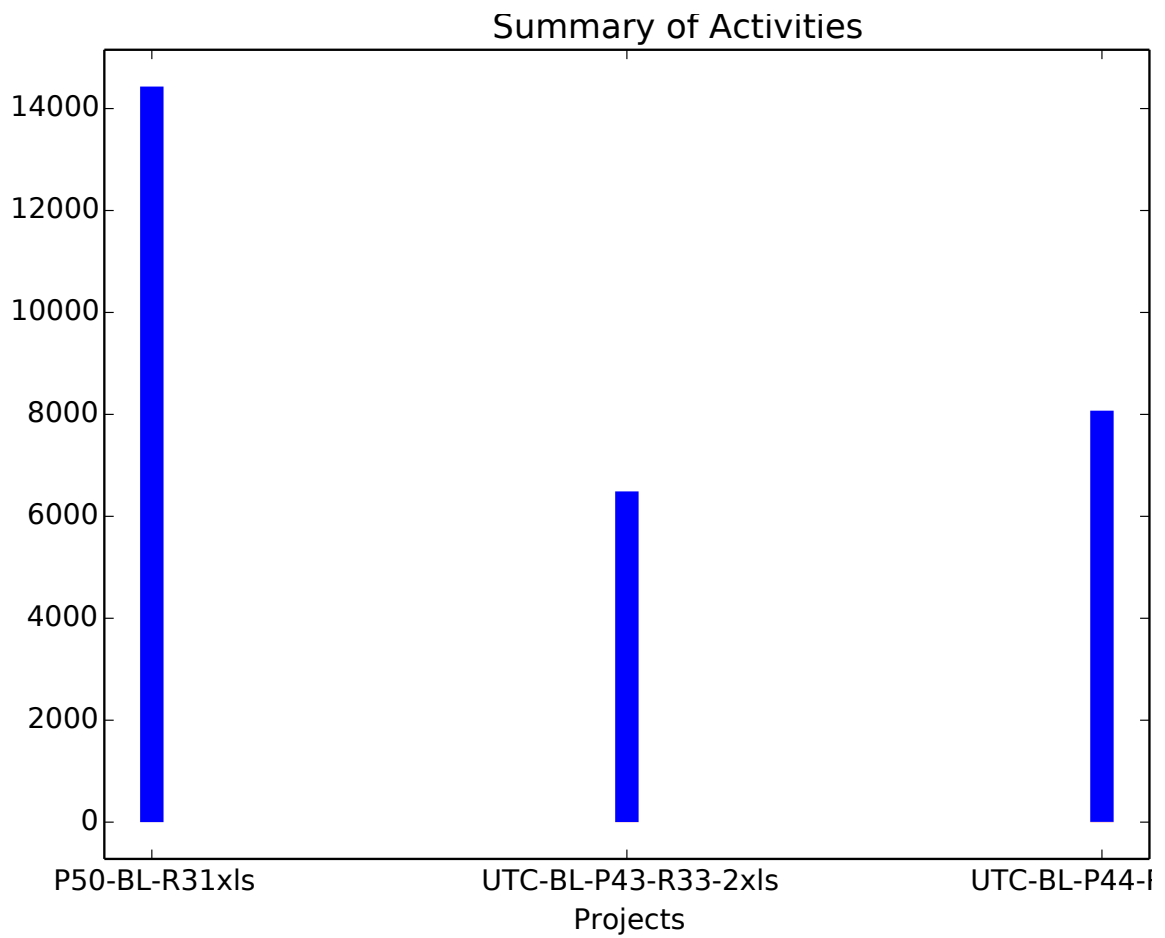


Figure 4.14: Summary of the Portfolio used for validation



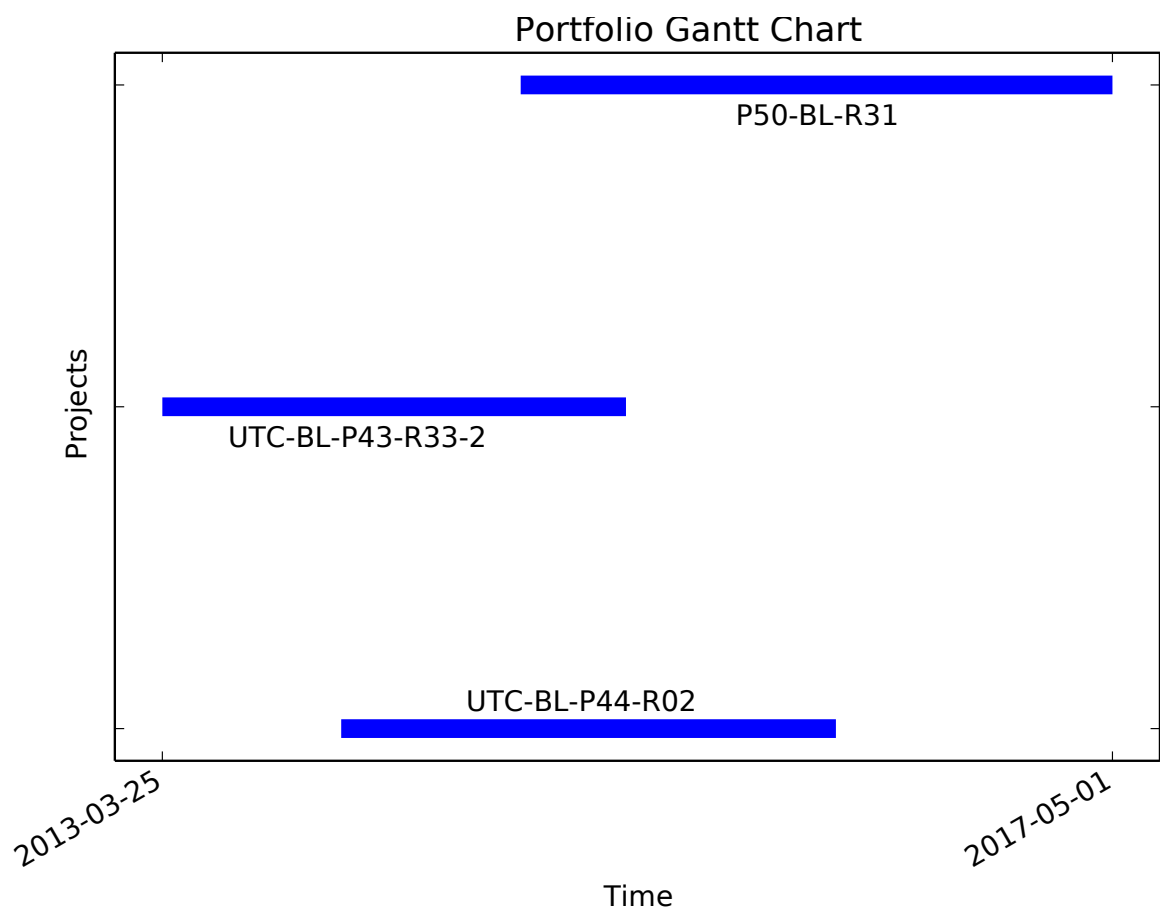


Figure 4.15: Portfolio Gantt Chart

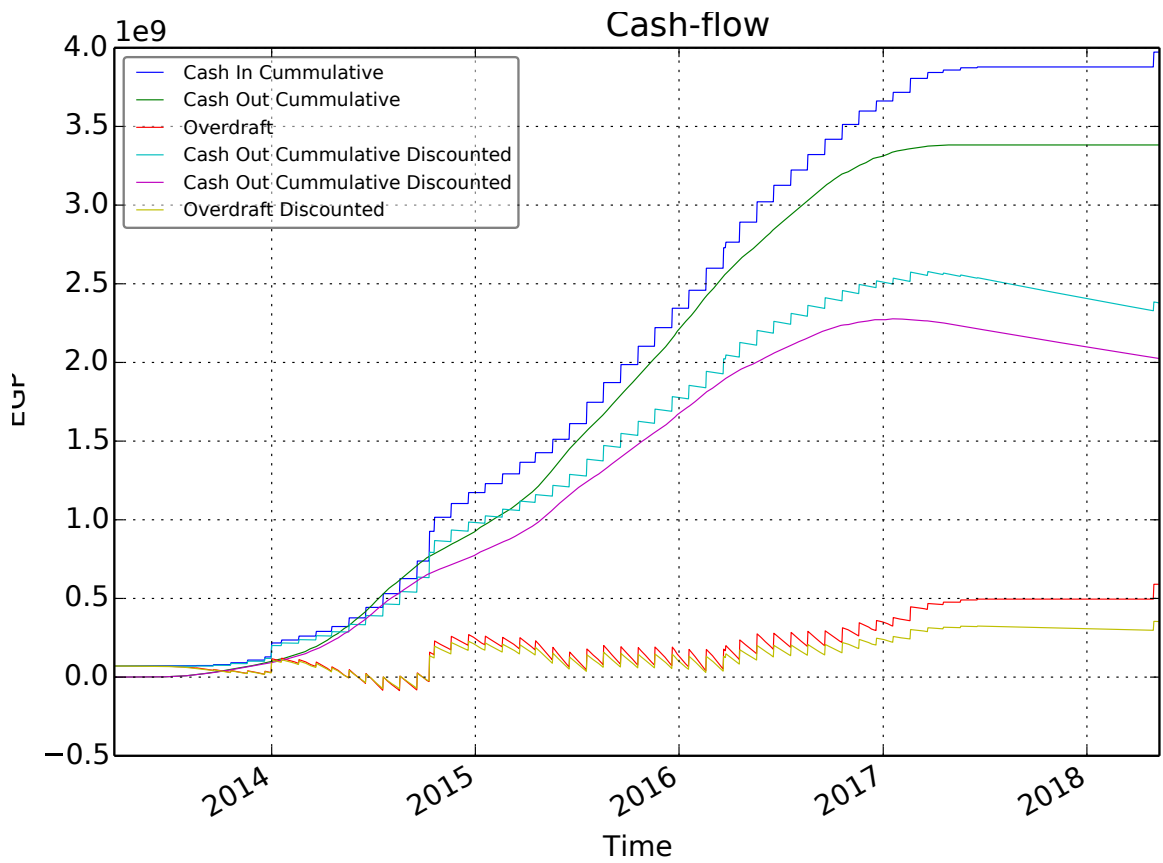


Figure 4.16: Portfolio Gantt Chart

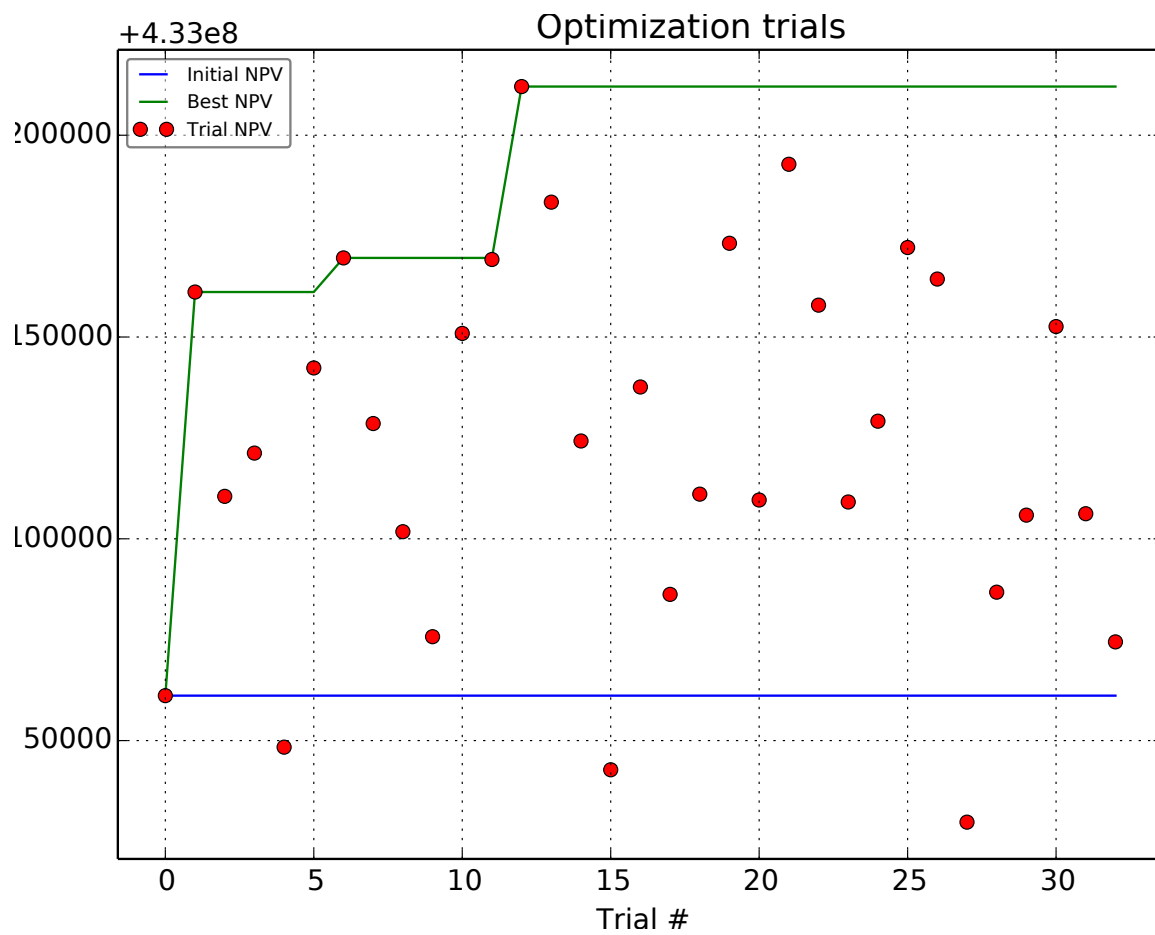


Figure 4.17: Optimization trials for the validation

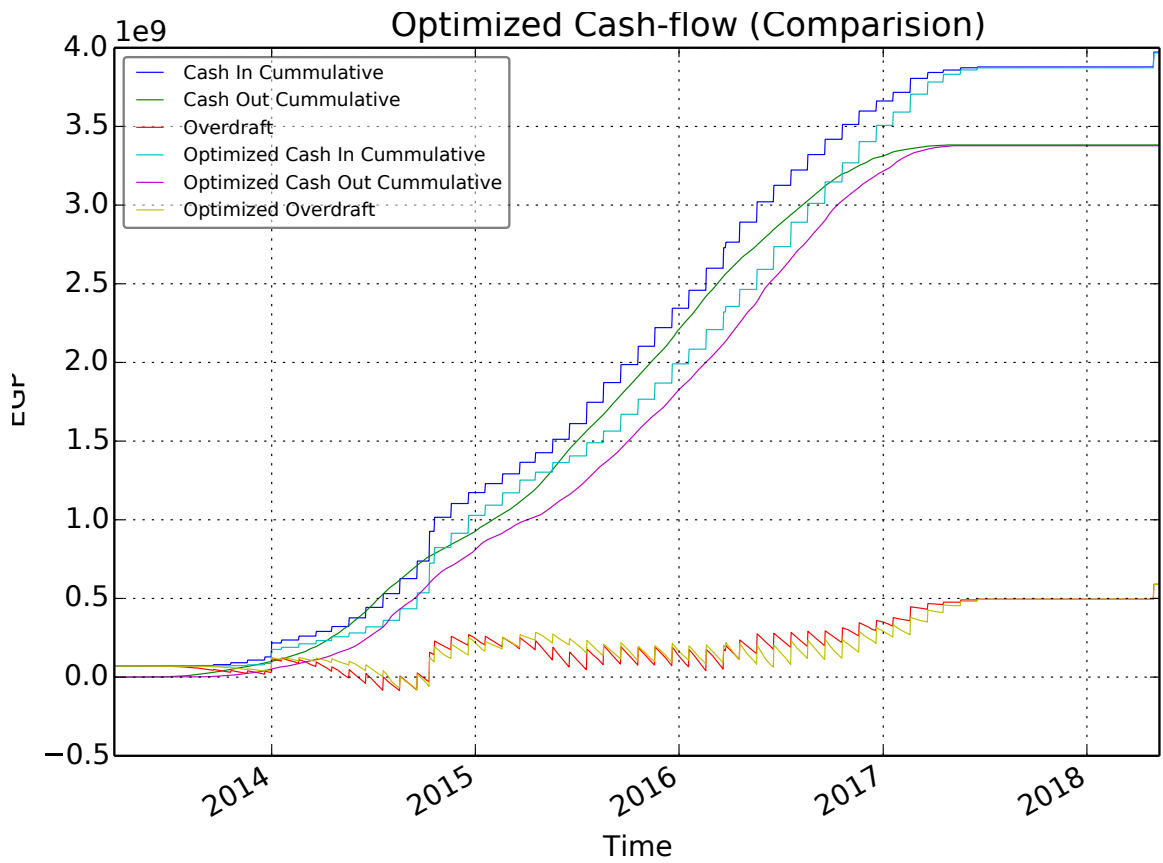


Figure 4.18: Optimized Cash Flow

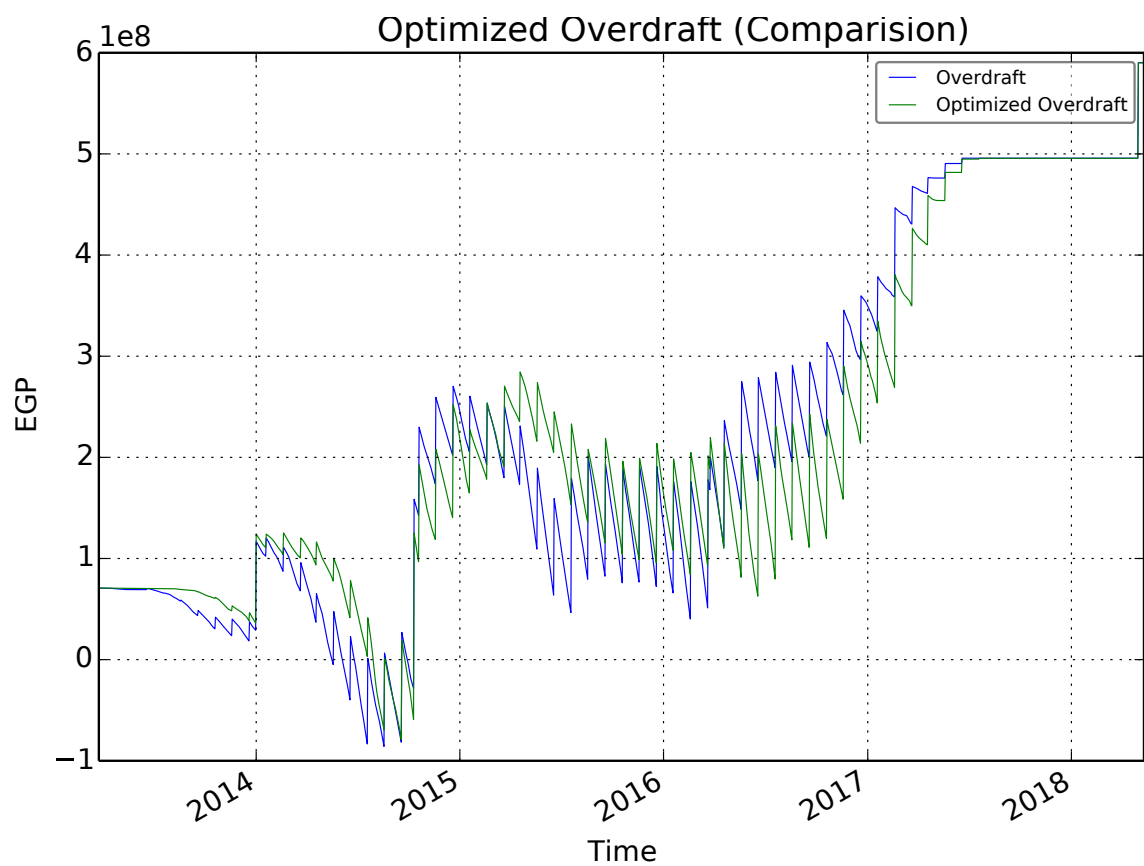


Figure 4.19: Optimized Overdraft

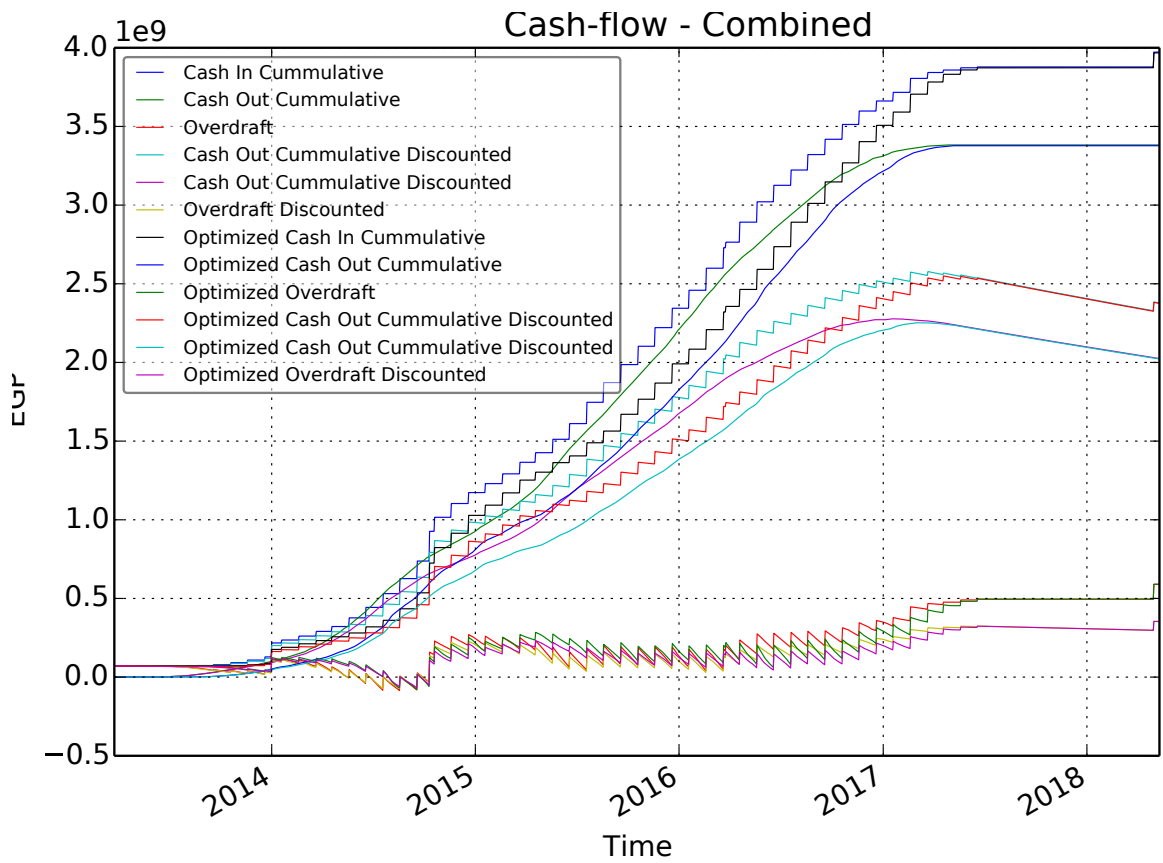


Figure 4.20: Optimized Cash Flow for the validation Portfolio

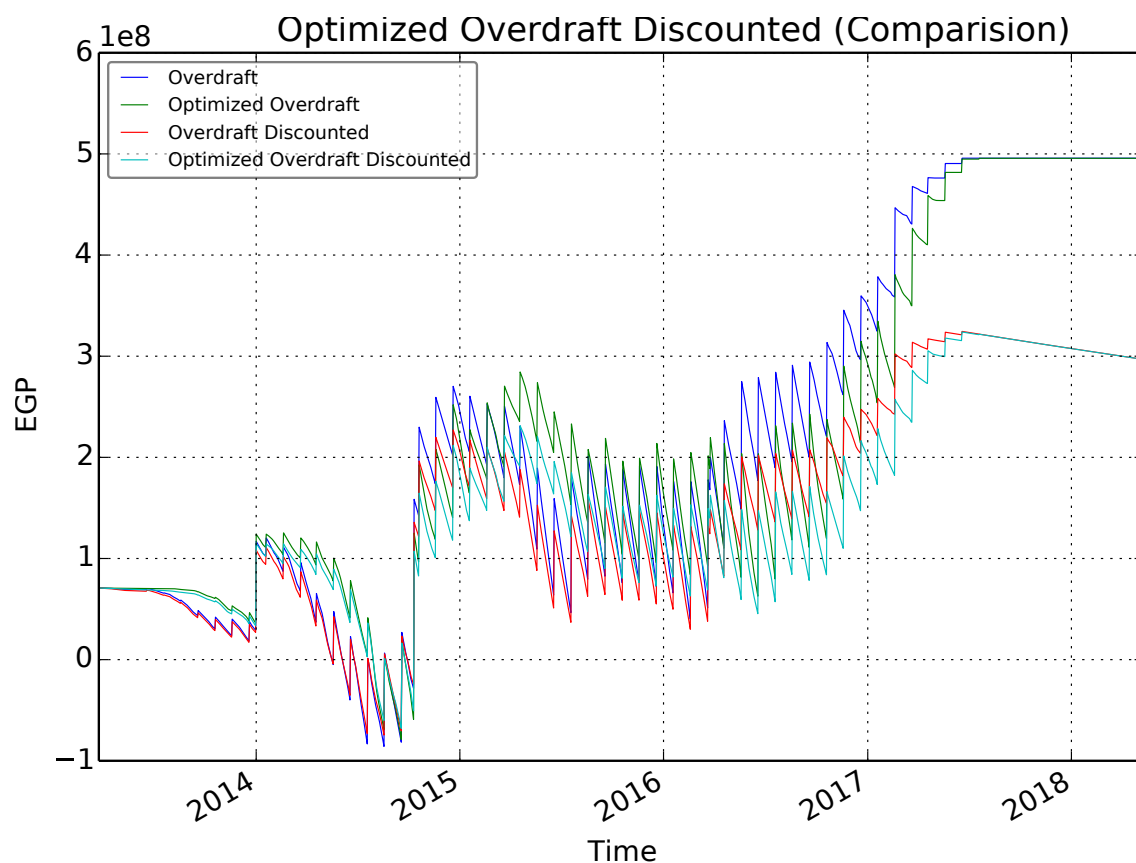


Figure 4.21: Optimized Overdraft for the validation Portfolio

## 4.6 Validation with Updated Schedule

### 4.6.1 Validation Method

Another validation was done for a project with an updated schedule. The model was executed for the updated schedule. The project is a landscape construction project in Cairo. The schedule has 477 activities. That project start was 2014-03-13 and the Finish was expected to be 2016-29-07, as the update date for the schedule was May 2016. The Baseline start and finish were dates were 2014-03-13 and 2014-11-13, respectively. So, currently time is at large. The % Schedule completion was 94.7% at the update time in May 2016, and the % schedule completed was 99.2%. The costs of the activities were changed for confidentiality as requested by the data provider. The Total Cost was 15,644,990 EGP and the Total price was 18,773,988 EGP.

### 4.6.2 Validation Results and Discussion

The cash flow was calculated for the updated schedule. The resulting cash flow is shown in Figure 4.22. The curves show an S curve trend. The NPV was found to be 2,570,178 EGP. It should be noted that the curve begins at a positive value that equals the downpayment value, and the curve extends till the receipt of the retention. Overall, this validation showed that the model can handle updated schedules. These can be utilized to calculate the actual NPV and Discounted values of the cash flow, which can be used to indicate the success (or failure) of a project during construction.



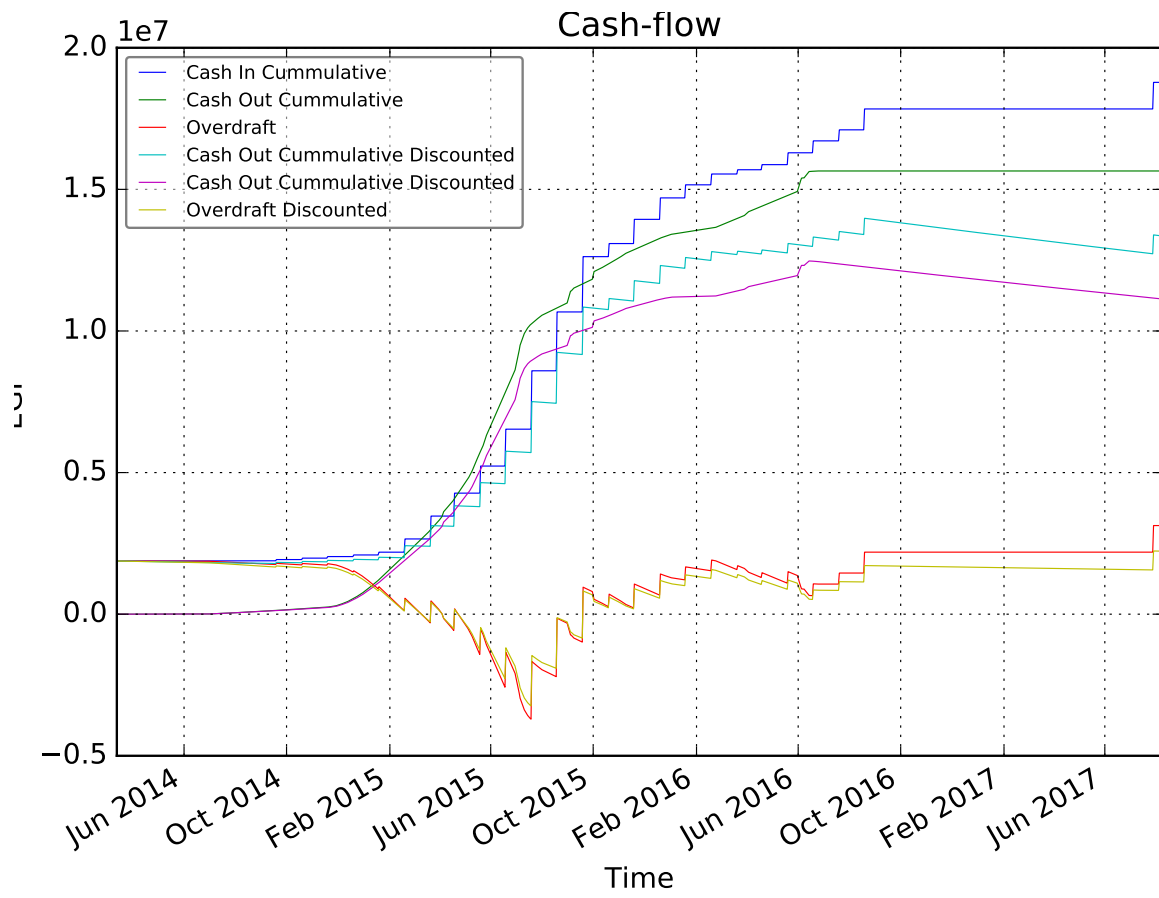


Figure 4.22: Portfolio Gantt Chart

# Chapter 5

## Conclusion and Recommendations

This chapter will conclude the thesis regarding the model and Graphical User Interface (GUI), the optimization, the results of the verification, validation, and CPU time. Finally, limitations and recommendations for the future research are advised.

### 5.1 Conclusion

Taking the point of view of the contractor in a construction project, the developed model and Graphical User Interface (GUI) can be used to perform analysis and optimization of the cash flow of a portfolio of construction project. The analysis includes the Cash In, Cash out, and the Overdraft, which are calculated according to the time schedule, the financial parameters and contractual time bars like the down-payment, retention, invoice interval,..etc. The time value is also taken into consideration as an interest rate, which can be the inflation rate or the Minimum Attractive Rate of Return (MARR) for the contractor. The optimization had the objective of reducing the Net Present Value (NPV) of the whole portfolio. This had the effect of increasing the profit of the contractor for all the projects as a whole, taking into effect the time value of money. Excessive overdraft is also reduced as an effect. The model achieved its targeted scope.

#### 5.1.1 Model and GUI

The scope was achieved by creating a model that can do the analysis and optimization of construction portfolios. Python proved to be a good choice for prototyping and fast implementation. The computational time wasn't greatly affected, since most of the packages used are coded in C. A friendly Graphical User Interface was also created. It allows the user to create a portfolio, projects in it, activities in the project, and relationships between the activities. The user can also modify financial parameters and contractual time bars.

### 5.1.2 CPU Time

A test for the CPU time was done and described in Section 4.4. There is a fair correlation between the CPU time and the number of projects of course. But it seems that the CPU time is greatly affected by the structure of the projects; projects where there are several complicated relationships between activities, especially where one activity has multiple relationships, seem to be more computationally costly, in addition to their large size. This was more apparent in the validation. Generally, the CPU time is satisfactory, for small and large projects.

### 5.1.3 Verification

The verification was described in Section 4.1.1. The trials were done for random projects to verify the results and effectiveness of the algorithm. The model converged in all cases. It should be noted that in some cases, the optimum NPV for the project would occur when all activities start as soon as possible, meaning the optimum start is the early start. It should also be noted that the user should not create relationships between activities that are cyclic, meaning that, for example, 2 activities cannot be the predecessors of each other, and the same applies to longer chains of activities. Otherwise the model will keep calculating in an endless loop.

### 5.1.4 Validation

The validation was done for a large portfolio of real projects from a single construction company. The portfolio had, approximately, 29 thousand activities with 70 thousand relationships between them. Further details were described in 4.5.1. The model converged in a relatively satisfactory time, compared to the size of the portfolio. It was noticed that the bottleneck for the model is the calculation of the activity start and ends. This due to the large number of activities and relationships, in addition to the fact that some activities had multiple relationships that connected to many activities and complicated the calculations.

### 5.1.5 Optimization Algorithm

The verification and the validation shows that the bottleneck was the calculation of the activities' start and finish dates, especially when the relationships connect too many activities, which complicates the computations and makes the whole process slower. Due to this issue and the very large number of variables, as shown in the validation, the use of evolutionary algorithms (EA) is unfeasible; the model would be unable to create a first population for the EA in a satisfactory time. The optimization technique used in this model is a form of Brute Forcing, as discussed in 3.9, and it proved to be satisfactory for

a large project, as shown in the validation, and also for smaller projects, as shown in the verification.

### 5.1.6 Sensitivity Analysis

A sensitivity analysis was performed for the model, taking into consideration the Interest Rate and Cost parameters' effect on the Net Present Value (NPV). The results showed consistency with the equations provided. The increase in the interest rate increased the NPV with a curved shape while the increase in the cost increased the NPV linearly. This behavior was consistent with the the given equations and the behavior of the time value of money.

## 5.2 Limitations

Due to assumptions that were utilized in the model development, the limitations are:

- The cost of each activity was assumed to be uniformly distributed along each activity's duration, in contrast real life cases where the cost can be front allocated, or back allocated, or have any other distribution. These options should be added to simulate real situations.
- The costs and expenses that are delayed after an activity or before it, such as in the case of paying for a supplier after a duration of time from an activity, or before the activity was neglected. Though they could be added in the model as separate activities that have delays between them.
- Payment of invoices, retention, and down-payments was assumed to be always on time, neither late nor early than the contractual time bars. Delays are completely out of scope. This limitation could be fixed by adding the the model the liabilities and delay penalties. This could result in situations where, after optimizations, delay damages will be paid, but the profit is higher.
- The retention was assumed to be paid completely after the Defects Liability Period. In some situations, however, it could be paid in half at construction completion and half after the defects liability period.
- Financial situations for loans, bonds, procurement agreements, and similar items were not considered, though they can be added as separate activities with their costs.
- Exhaustive numeration was used for the optimization, though it leads to a global near-optimum solution, it is slower and more computationally cumbersome than other higher-level methods, such as evolutionary algorithms.

- There is a bottleneck when calculating large schedules, due to their size and complexity, and it greatly affects optimization process as well, leading to long calculation time.
- The options included in the model for the payments, invoicing, advanced payment, and retention, are limited.
- The model doesn't do resource leveling.

### 5.3 Recommendations

Researchers in this topic are advised to notice the limitations of the model. The most important limitation is the bottleneck for the optimization of large projects in the proposed model, in the calculation of the start and finish times for activities, which increases the overall time the optimizations significantly because optimization trials require recalculation of the schedule. Practical schedules, specially for large construction portfolios, are expected to have thousands of activities, just as the one used for the validation, therefore a faster algorithm is needed, at least for the sake of optimization. This algorithm doesn't have to be deterministic or very accurate, but it needs to be accurate enough and much quicker in order to allow for faster optimization or the use of more complicated optimization algorithms, followed by an accurate calculation of the resulting model after optimization.

# Bibliography

- Abido, M. A. and Ashraf M. Elazouni (2011a). "Multiobjective Evolutionary Finance-Based Scheduling: Entire Projects' Portfolio". In: *Journal of Computing in Civil Engineering* 25.1, pp. 85–97.
- Abido, Mohammad and Ashraf Elazouni (2011b). "Multiobjective Evolutionary Finance-Based Scheduling: Individual Projects Within a Portfolio". In: *Automation in Construction* 20, pp. 755–766.
- Al-Jabouri, Khalil I., Raid Al-Aomar, and Mohammed E. Bahri (2012). "Analyzing The Impact of Negative Cash Flow on Construction Performance in the Dubai Area". In: *Journal of Management in Engineering* 28.4, pp. 382–390.
- Alghazi, Anas, Ashraf Elazouni, and Shokri Selim (2013). "Improved Genetic Algorithm for Finance-Based Scheduling". In: *Journal of Computing in Civil Engineering* 27.4, pp. 379–394.
- Au, Tung and Chris Hendrickson (1985). "Profit Measures for Construction Projects". In: *Journal of Construction Engineering and Management* 112, pp. 273–286.
- Christodoulou, Symeon (2010). "Scheduling Resource-Constrained Projects with Ant Colony Optimization Artificial Agents". In: *Journal of Computing in Civil Engineering* 24.1, pp. 45–55.
- Cui, Qingbin, Makarand Hastak, and Daniel Halpin (2010). "Systems analysis of project cash flow management strategies". In: *Construction Management and Economics* 28.4, pp. 361–376.
- El-Rayes, Khaled and Dho Heon Jun (2009). "Optimization Resource Leveling in Construction Projects". In: *Journal of Construction Engineering and Management* 23.3, pp. 1172–1180.
- El-Rayes, Khaled and Osama Moselhi (2001). "Optimization Resource Utilization for Repetitive Construction Projects". In: *Journal of Construction Engineering and Management*, pp. 18–27.
- Elazouni, Ashraf (2009). "Heuristic Method for Multi-project Finance-based Scheduling". In: *Construction Management and Economics* 27.2, pp. 199–211.
- Elazouni, Ashraf and M. A. Abido (2014). "Enhanced Trade-off of Construction Projects: Finance-Resource-Profit". In: *Journal of Construction Engineering and Management* 140.9, p. 04014043.

- Elazouni, Ashraf, Anas Alghazi, and Shokri Z. Selim (2015). "Finance-based Scheduling Using Meta-heuristics: Discrete versus continuous optimization problems". In: *Journal of Financial Management of Property and Construction* 20.1, pp. 85–104.
- Elazouni, Ashraf M. and Fikry G. Metwally (2007). "Expanding Finance-Based Scheduling to Devise Overall-Optimized Project Schedules". In: *Journal of Construction Engineering and Management* 133.1, pp. 86–90.
- Elbeltagi, Emad et al. (2016). "Overall Multiobjective Optimization of Construction Projects Scheduling Using Particle Swarm". In: *Journal of Financial Management of Property and Construction* 23.3, pp. 265–282.
- Ezeldin, A. Samer and Ahment Soliman (2009). "Hybrid Time-Cost Optimization of Non-serial Repetitive Construction Projects". In: *Journal of Construction Engineering and Management* 135.1, pp. 42–55.
- Foundation, Python Software (2016). *The Official Home of The Python Programming Language*. URL: <https://www.python.org/>.
- Gorog, Mihaly (2009). "A Comprehensive Model for Planning and Controlling Contractor Cash-flow". In: *International Journal of Project Management* 27, pp. 481–492.
- Han, Seung H. et al. (2004). "Multicriteria Financial Portfolio Risk Management for International Projects". In: *Journal of Construction Engineering and Management* 130.3, pp. 346–356.
- Hegazy, Tarek (1999). "Optimization of Resource Allocation And Leveling Using Genetic Algorithms". In: *Journal of Construction Engineering and Management* 125.3, pp. 167–175.
- Huang, Wen-Haw et al. (2013). "Contractor Financial Prequalification Using Simulation Method Based on Cash Flow Model". In: *Automation in Construction* 35, pp. 254–262.
- Hwee, Ng Ghim and Robert L. K. Tiong (2002). "Model on Cash Flow Forecasting and Risk Analysis for Contracting Firms". In: *Journal of Project Management* 20, pp. 351–363.
- IEEE (2015). *The 2015 Top Ten Programming Languages*. URL: <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>.
- Jiang, Aiyin, Raja R. A. Issa, and Maged Malek (2011). "Construction Project Cash Flow Planning Using the Pareto Optimality Efficiency Network Model". In: *Journal of Civil Engineering and Management* 17.4, pp. 510–519.
- Jun, Dhoo Heon and Khaled El-Rayes (2011). "Multiobjective Optimization of Resource Leveling and Allocation during Construction Scheduling". In: *Journal of Construction Engineering and Management* 137.12, pp. 1080–1088.
- Kaka, A. P. and A. D. F. Price (1993). "Modelling Standard Cost Commitment Curves for Contractors' Cash Flow Forecasting". In: *Construction Management and Economics* 11.4, pp. 271–283.

- Khosrowshahi, F. (2007). "A Decision Support Model for Construction Cash Flow Management". In: *Computer-Aided Civil and Infrastructure Engineering* 22, pp. 527–539.
- Kim, Kyungki, John Walewski, and Yong K. Cho (2016). "Multiobjective Construction Schedule Optimization Using modified Niche Pareto Genetic Algorithm". In: *Journal of Management in Engineering* 32.2, p. 04015038.
- Kishore, Varun, Dulcy M. Abraham, and Joseph V. Sinfield (2011). "Portfolio Cash Assessment Using Fuzzy Systems Theory". In: *Journal of Construction Engineering and Management* 137.5, pp. 333–343.
- Lee, Dong-Eun, Tae-Kyung Lim, and David Arditi (2012). "Stochastic Project Financing Analysis System for Construction". In: *Journal of Construction Engineering and Management* 138.3, pp. 376–389.
- Li, Huimin and Peng Li (2013). "Self-Adaptive Ant Colony Optimization for Construction Time-Cost Optimization". In: *Kybernetes* 24.8, pp. 1181–1194.
- Li, Shirong (1996). "New Approach for Optimization of Overall Construction Schedule". In: *Journal of Construction Engineering and Management* 122.1, pp. 7–13.
- Liu, Sh-Shun and Chang-Jung Wang (2009). "Two-Stage Profit Optimization Model for linear Scheduling Problems Considering Cash Flow". In: *Construction Management and Economics* 27.11, pp. 1023–1037.
- Liu, Shu-Shun and Chang-Jung Wang (2008). "Resource-Constrained Construction Project Scheduling Model for Profit Maximization Considering Cash Flow". In: *Automation in Construction* 17, pp. 966–974.
- Lucko, Gunnar (2011). "Integrating Efficient Resource Optimization and Linear Schedule Analysis with Singularity Functions". In: *Journal of Construction Engineering and Management* 137.1, pp. 45–55.
- Lucko, Gunnar (2013). "Supporting Financial Decision-Making Based on Time Value of Money with Singularity Functions in Cash Flow Models". In: *Construction Management and Economics* 31.3, pp. 238–253.
- Maravas, Alexander and John Paris Pantouvakis (2012). "Project Cash Flow Analysis in The Presence of Uncertainty in Activity Duration and Cost". In: *International Journal of Project Management* 30, pp. 374–384.
- Menesi, Wail, Behrooz Galzarpoor, and Tarek Hegazy (2013). "Fast and New-Optimum Schedule Optimization for Large-Scale Projects". In: *Journal of Construction Engineering and Management* 139.9, pp. 1117–1124.
- Odeyinka, Henry A. and Ammar Kaka (2005). "An Evaluation of Constructors' Satisfaction With Payment Terms Influencing Construction Cash Flow". In: *Journal of Financial Management of Property and Construction* 30.3, pp. 171–180.
- Park, Hyung K., Seung H. Han, and Jeffrey S. Russell (2005). "Cash Flow Forecasting Model for General Contractors Using Moving Weights of Cost Categories". In: *Journal of Management in Engineering* 21.4, pp. 164–172.



- Peterson, Steven J. (2009). *Construction Accounting and Financial Management*. Second. Pearson.
- Pinto, Jeffrey K. (2010). *Project Management: Achieving Competitive Advantage*. Second. Prentice Hall.
- Platje, Adri, Herald Seidel, and Spike Wadman (1994). "Project and Portfolio Planning Cycle". In: *International Journal of Project Management* 7.12, pp. 100–106.
- Purnus, Augustin and Bodea Constanta-Nicoleta (2015). "Financial Management of the Construction Projects: A proposed Cash Flow Analysis Model at Project Portfolio Level". In: *Organization, Technology and Management in Construction* 7.1, pp. 1217–1227. URL: [http://www.grad.hr/otmcj/clanci/vol%207\\_1/OTMC\\_6.pdf](http://www.grad.hr/otmcj/clanci/vol%207_1/OTMC_6.pdf) (visited on 01/28/2016).
- Sanchez, Hynuk et al. (2009). "Risk Management Applied to Projects, Programs, and Portfolios". In: *International Journal of Managing Projects in Business* 2.1, pp. 14–35.
- Su, Yi and Gunnar Lucko (2015). "Optimum Present Value Scheduling Based on Synthetic Cash Flow Model with Singularity Functions". In: *Journal of Construction Engineering and Management* 141.11, p. 04015036.
- Tang, Yuanjie, Rengkui Liu, and Quanxin Sun (2014). "Two-Stage Scheduling Model for Resource Leveling of Linear Projects". In: *Journal of Construction Engineering and Management* 140.7, p. 04014022.
- Zayed, Tarek and Yaqiong Liu (2014). "Cash Flow Modeling for Construction Projects". In: *Engineering, Construction, and Architectural Management* 21.2, pp. 170–189.

# Appendices



# Appendix A

## Python Code

```

1  '''
2  Cash Flow Optimmmization for Construction Engineering Portfolios
3  Author: Gasser Galal Ali
4  This code was developped for the purpose of the
5  fullfilment of the requirements of the thesis for the degree
6  of Master of Science in Construction Engineering at The
7  American University in Cairo.
8  This code may not be fully or partially used without written
9  approval of the author
10 '''
11
12
13 import sqlite3, re, os, datetime, functools, math, random, sys, webbrowser, csv
14 import xlswriter, xlrd
15 import matplotlib.pyplot as plt
16 import tkinter as tk
17 import tkinter.ttk as ttk
18
19 def log(text):
20     global log file name
21     print(text)
22     f = open(log file name, 'a+')
23     f.write(text + '\n')
24     f.close()
25
26 def pause():
27     input("Paused. Press Enter to resume.")
28     print("Resuming...")
29
30 def adddays(date, days, calendar):
31     # Function to add or subtract days from a date with days off included, the
32     # daysoff should be a tuple of 0 to 6 where 0 is Monday
33     condition = True
34     counter = 0
35     output = date
36     if calendar == None or '7d' in calendar.lower() or '7 d' in calendar.lower():
37         listofdaysoff = ()
38     elif '6d' in calendar.lower() or '6 d' in calendar.lower():
39         listofdaysoff = (4,)
40     elif '5d' in calendar.lower() or '5 D' in calendar.lower():
41         listofdaysoff = (4,5)
42     else:
43         print(' [!] unrecognized calendar : ' + calendar)
44         listofdaysoff = ()
45     while condition and days != 0:
46         if days > 0:
47             output += datetime.timedelta(1)
48         else:
49             output += datetime.timedelta(-1)
50         if output.weekday() not in listofdaysoff:
51             counter += 1
52         if counter == abs(days):
53             condition = False
54     return output
55
56 def new database(): # Deploys a new database file. DELETES OLD FILE IF FOUND
57     log('Deploying Database')
58     global conn
59     conn.commit()
60     conn.close()
61     if os.path.exists(database file name):
62         os.remove(database file name)
63         log(" - Removed old file")
64     conn = sqlite3.connect(database file name)
65     conn.execute("PRAGMA default cache size = 500000;")
66     conn.commit()
67     conn.execute("CREATE TABLE projects (projectid TEXT UNIQUE NOT NULL, projectname
68     TEXT, start DATE, finish DATE, duration INT, interest FLOAT, markup FLOAT,

```

- 1 -

```

retentionperiod INT, retention FLOAT, invoiceinterval INT, paymentperiod INT,
downpayment FLOAT, cost FLOAT, price FLOAT, totalactivities INT,
criticalactivities INT, cashinpv FLOAT, cashoutpv FLOAT, npv FLOAT,
maxoverdraftdisc FLOAT, minoverdraftdisc FLOAT, cashinpvopt FLOAT, cashoutpvopt
67 conn.execute("CREATE INDEX projectsindex ON projects (projectid);")
68 conn.execute("CREATE TABLE activities (projectid TEXT, activityid TEXT,
activityname TEXT, duration INT, cost FLOAT, es INT, ef INT, ls INT, lf INT, ff
INT, tf INT, laq INT, os INT, of INT, primaryconstraint TEXT,
primaryconstraintdate DATE, calendar TEXT);")
69 conn.execute("CREATE INDEX activitiesindex ON activities (activityid);")
70 conn.execute("CREATE TABLE relationships (projectid TEXT, activitylid TEXT,
activity2id TEXT, type TEXT, rlag INT);")
71 conn.execute("CREATE INDEX relationshipsindex ON relationships
(projectid,activitylid,activity2id,type);")
72 conn.execute("CREATE TABLE cashflow (date INT, projectid TEXT, cashout FLOAT,
invoice FLOAT, cashin FLOAT, cashoutcum FLOAT, cashincum FLOAT, overdraft FLOAT,
cashoutdisc FLOAT, cashindisc FLOAT, cashoutcumdisc FLOAT, cashincumdisc FLOAT,
overdraftdisc FLOAT)")
73 conn.execute("CREATE INDEX cashflowindex ON cashflow (date, projectid);")
74 conn.execute("CREATE TABLE cashflowall (date INT, projectid TEXT, cashout FLOAT,
invoice FLOAT, cashin FLOAT, cashoutcum FLOAT, cashincum FLOAT, overdraft FLOAT,
cashoutdisc FLOAT, cashindisc FLOAT, cashoutcumdisc FLOAT, cashincumdisc FLOAT,
overdraftdisc FLOAT)")
75 conn.execute("CREATE INDEX cashflowallindex ON cashflowall (date, projectid);")
76 conn.execute("CREATE TABLE cashflowopt (date INT, projectid TEXT, cashout FLOAT,
invoice FLOAT, cashin FLOAT, cashoutcum FLOAT, cashincum FLOAT, overdraft FLOAT,
cashoutdisc FLOAT, cashindisc FLOAT, cashoutcumdisc FLOAT, cashincumdisc FLOAT,
overdraftdisc FLOAT)")
77 conn.execute("CREATE INDEX cashflowoptindex ON cashflowopt(date, projectid);")
78 conn.execute("CREATE TABLE cashflowallopt (date INT, projectid TEXT, cashout
FLOAT, invoice FLOAT, cashin FLOAT, cashoutcum FLOAT, cashincum FLOAT, overdraft
FLOAT, cashoutdisc FLOAT, cashindisc FLOAT, cashoutcumdisc FLOAT, cashincumdisc
FLOAT, overdraftdisc FLOAT)")
79 conn.execute("CREATE INDEX cashflowoptallindex ON cashflowallopt (date,
projectid);")
80 conn.execute("CREATE TABLE portfolio (portfolioid TEXT UNIQUE NOT NULL, start
DATE, finish DATE, duration INT, numberofprojects INT, numberofactivities INT,
cost FLOAT, price FLOAT, cashinpv FLOAT, cashoutpv FLOAT, npv FLOAT,
maxoverdraftdisc FLOAT, minoverdraftdisc FLOAT, cashinpvopt FLOAT, cashoutpvopt
81 conn.execute("INSERT INTO portfolio (portfolioid) VALUES ('portfolio');")
82 conn.execute("CREATE TABLE trials (trialid INT, initialnpv FLOAT, trialnpv
FLOAT, bestnpv FLOAT)")
83 conn.execute("CREATE INDEX trialindex ON trials(trialid);")
84 conn.execute("CREATE view big AS SELECT relationships.*, activities1.es AS
activityles, activities1.ef AS activitylef, activities1.ls AS activitylls,
activities1.lf AS activityllf, activities1.os AS activitylos, activities1.of AS
activitylof, activities1.duration AS activitylduration, activities2.es AS
activity2es, activities2.ef AS activity2ef, activities2.ls AS activity2ls,
activities2.lf AS activity2lf, activities2.os AS activity2os, activities2.of AS
activity2of, activities2.duration AS activity2duration FROM relationships INNER
JOIN activities AS activities1 ON relationships.projectid =
activities1.projectid AND relationships.activitylid = activities1.activityid
INNER JOIN activities AS activities2 ON relationships.projectid =
activities2.projectid AND relationships.activity2id = activities2.activityid;")
85 conn.commit()
86 log(' - Done')
87
88 def print table(name): # Prints a table from the database, input "all" for all tables
89 if name in ['all','ALL','ALL']:
90     tables = [a for a in get("Select name FROM sqlite master WHERE type =
'table';")]
91 else:
92     tables = [name]
93 for table in tables:
94     heads = []
95     for a in conn.execute("PRAGMA table_info(%s);" %table):

```

```

96         heads.append(a[1])
97     log('\n\nTABLE: %s' % name)
98     log(heads)
99     for c in conn.execute("select * from %s;" %table):
100         log(c)
101     log('==== end of table =====')
102
103 def database info(): # prints some database info
104     log("Number of projects: %s project"%conn.execute("SELECT COUNT(*) FROM
projects;").fetchall()[0][0])
105     log("Number of activities: %s activity"%conn.execute("SELECT COUNT(*) FROM
activities;").fetchall()[0][0])
106     log("Number of relationships: %s relationship"%conn.execute("SELECT COUNT(*)
FROM relationships;").fetchall()[0][0])
107     log("Distinct relationships: %s"%[a[0] for a in conn.execute("SELECT DISTINCT
type FROM relationships;").fetchall()])
108     projects = [a[0] for a in conn.execute("SELECT projectid FROM projects;").
fetchall()]
109     for projectid in projects:
110         number of activities = conn.execute("SELECT COUNT(*) FROM activities WHERE
projectid = ?;",(projectid,)).fetchall()[0][0]
111         number of critical activities = conn.execute("SELECT COUNT(*) FROM
activities WHERE projectid = ? AND tf = 0;",(projectid,)).fetchall()[0][0]
112         log(' - +projectid + ' -> ' + str(number of activities) + ' activity -> ' +
str(number of critical activities) + ' critical activity')
113
114 def project create(projectid,projectname,start,interest,markup,downpayment,
invoiceinterval,paymentperiod,retention,retentionperiod): # Create a new Project
115     conn.execute("INSERT INTO projects \
116
117         (projectid,projectname,start,interest,markup,downpayment,invoiceinterval,paymentpe
riod,retention,retentionperiod) VALUES
118         ('%s','%s','%s','%s','%s','%s','%s','%s','%s','%s')"%(projectid,projectname,start
,interest,markup,downpayment,invoiceinterval,paymentperiod,retention,
retentionperiod))
119
120 def activity create(projectid,activityid,activityname,duration,cost): # Create a new
Activity in a Project
121     conn.execute("INSERT INTO activities
122         (projectid,activityid,activityname,duration,cost) VALUES
123         ('%s','%s','%s','%s','%s')"%(projectid,activityid,activityname,duration,cost))
124
125 def relationship create(projectid,activitylid,activity2id,relationship type): #
Create a new Relationship between 2 Activities
126     conn.execute("INSERT INTO relationships (projectid,activitylid,activity2id,type)
VALUES (?,?,,?);", (projectid,activitylid,activity2id,relationship type))
127
128 def create a portfolio(): # Creates a sample portfolio for testing
129     log('Creating a random Portfolio')
130     number of projects = 3
131     min number of activities = 20
132     max number of activities = 25
133     for p in range(1,number of projects+1):
134         number of activities = random.randint(min number of activities,
135         max number of activities)
136         projectid = 'project' + str(p)
137         projectname = projectid
138         start = (datetime.date.today() + datetime.timedelta(days = random.randint(10,
300))).isoformat()
139         interest = random.randint(10,20)/ 100
140         markup = random.randint(15,25)/100
141         downpayment = random.randint(15,25)/100
142         invoiceinterval = 'monthly'
143         paymentperiod = 56
144         retention = 0.1
145         retentionperiod = 80
146         conn.execute("INSERT INTO projects
147         (projectid,projectname,start,interest,markup,downpayment,invoiceinterval,payme

```

```

ntperiod,retention,retentionperiod) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (projectid
, projectname,start,interest,markup,downpayment,invoiceinterval,paymentperiod
,retention,retentionperiod))
142 for a in range(1,number of activities+1):
143     projectid = projectid
144     activityid = 'activity' + str(a)
145     activityname = activityid
146     duration = random.randint(10,20)
147     cost = random.randint(1,10)
148     conn.execute("INSERT INTO activities
(projectid,activityid,activityname,duration,cost) VALUES (?, ?, ?, ?, ?)", (
projectid,activityid,activityname,duration,cost))
149     if a > 1:
150         for i in range([1,1,1,2][random.randint(0,3)]): # number of
relationships for each activity
151             for r in [random.randint(1,a-1)]:
152                 projectid = projectid
153                 activitylid = 'activity' + str(r)
154                 activity2id = activityid
155                 #- relationship type =
[ 'fs', 'sf', 'ss', 'ff' ][random.randint(0,3)]
156                 relationship type = [ 'fs', 'fs', 'fs', 'sf', 'ss', 'ff' ][random.
randint(0,5)]
157                 conn.execute("INSERT INTO relationships
(projectid,activitylid,activity2id,type) VALUES (?, ?, ?, ?)", (
projectid,activitylid,activity2id,relationship type))
158     conn.commit()
159     log(' - Done. ')
160
161 def create a portfolio2(number of projects,min number of activities,
max number of activities): # Creates a sample portfolio for testing
162     log('Creating a random Portfolio')
163     for p in range(1,number of projects+1):
164         number of activities = random.randint(min number of activities,
max number of activities)
165         projectid = 'project' + str(p)
166         projectname = projectid
167         start = (datetime.date.today() + datetime.timedelta(days = random.randint(10,
300))).isoformat()
168         interest = random.randint(10,20)/ 100
169         markup = random.randint(15,25)/100
170         downpayment = random.randint(15,25)/100
171         invoiceinterval = 'monthly'
172         paymentperiod = 56
173         retention = 0.1
174         retentionperiod = 80
175         conn.execute("INSERT INTO projects
(projectid,projectname,start,interest,markup,downpayment,invoiceinterval,payme
ntperiod,retention,retentionperiod) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (projectid
, projectname,start,interest,markup,downpayment,invoiceinterval,paymentperiod
,retention,retentionperiod))
176     for a in range(1,number of activities+1):
177         projectid = projectid
178         activityid = 'activity' + str(a)
179         activityname = activityid
180         duration = random.randint(10,20)
181         cost = random.randint(1,10)
182         conn.execute("INSERT INTO activities
(projectid,activityid,activityname,duration,cost) VALUES (?, ?, ?, ?, ?)", (
projectid,activityid,activityname,duration,cost))
183     if a > 1:
184         for i in range([1,1,1,2][random.randint(0,3)]): # number of
relationships for each activity random between 1 and 2
185             for r in [random.randint(1,a-1)]:
186                 projectid = projectid
187                 activitylid = 'activity' + str(r)
188                 activity2id = activityid
189                 #- relationship_type =

```



```

190         ['fs','sf','ss','ff'][random.randint(0,3)]
191         relationship type = ['fs','fs','fs','sf','ss','ff'][random.
192         randint(0,5)]
193         conn.execute("INSERT INTO relationships
194         (projectid,activitylid,activity2id,type) VALUES (?,?,,?)", (
195         projectid,activitylid,activity2id,relationship type)
196     conn.commit()
197     log(' - Done.')
198
199 def clean database(): # clean redundant elements
200     print('Cleaning Database')
201     global conn
202     conn.execute('DELETE FROM activities WHERE projectid NOT IN (SELECT projectid
203     FROM projects);')
204     conn.execute('DELETE FROM relationships WHERE projectid NOT IN (SELECT projectid
205     FROM projects);')
206     conn.execute('DELETE FROM relationships WHERE activitylid NOT IN (SELECT
207     activitylid FROM activities WHERE activities.projectid =
208     relationships.projectid);')
209     conn.execute('DELETE FROM relationships WHERE activity2id NOT IN (SELECT
210     activitylid FROM activities WHERE activities.projectid =
211     relationships.projectid);')
212     conn.commit()
213     print(' - Done')
214
215 def import uptown projects():
216     # importing uptown cairo files
217     log('Importing UPTOWN projects...')
218     global conn
219     # Find files
220     path = './projectsfromprimavera/'
221     files = []
222     for a in os.listdir(path):
223         if 'xl' in a:
224             files.append(path+a)
225     # Cycle through the files
226     for file in files:
227         # Create New Project
228         projectid = file.replace(path,'')
229         projectid = projectid.replace('.', '')
230         projectname = projectid
231         start = 0
232         interest = 0.1
233         markup = 0.2
234         downpayment = 0.1
235         invoiceinterval = 'monthly'
236         paymentperiod = 50
237         retention = 0.05
238         retentionperiod = 365
239         conn.execute("INSERT INTO projects
240         (projectid,projectname,start,interest,markup,downpayment,invoiceinterval,payme
241         ntperiod,retention,retentionperiod) VALUES (?,,?,,?,,?,,?,,?,,?);", (
242         projectid,projectname,start,interest,markup,downpayment,invoiceinterval,
243         paymentperiod,retention,retentionperiod))
244     # open the workbook
245     wb = xlrd.open workbook(file)
246     sheet names = wb.sheet names()
247     needed sheets = ['TASK', 'TASKPRED']
248     #open the task sheet
249     sheet = wb.sheet by name("TASK")
250     # get the indexes for needed rows
251     r = sheet.row values(1)
252     activityidindex = r.index('Activity ID')
253     activitynameindex = r.index('Activity Name')
254     startindex = r.index('(*)Start')
255     endindex = r.index('(*)Finish')
256     durationindex = r.index('Original Duration(h)')
257     costindex = r.index('(*)Budgeted Total Cost($)')

```

- 5 -

```

244 primaryconstraintindex = r.index('Primary Constraint')
245 primaryconstraintdateindex = r.index('Primary Constraint Date')
246 calendarindex = r.index('Calendar Name')
247
248 for i in range(2,sheet.nrows): # Loop on each row to get each activity
249     r = sheet.row values(i)
250     activityid = r[activityidindex]
251     activityname = r[activitynameindex]
252     duration = r[durationindex]
253     calendar = r[calendarindex]
254     if calendar == '':
255         calendar = None
256     cost = float(r[costindex]) / (random.randint(15, 20) / 100)
257
258     # handle the primary constraint
259     if r[primaryconstraintindex] == '': # find out if there is a primary
        constraint
260         primaryconstraint = None
261         primaryconstraintdate = None;
262     else:
263         primaryconstraint = r[primaryconstraintindex]
264         constdate = [int(a) for a in re.split("[: /]", r[
            primaryconstraintdateindex])[0:3]] # this syntax is used to break
            the dates
265         primaryconstraintdate = datetime.date(constdate[2],constdate[0],
            constdate[1])
266
267     if r[startindex] == '':
268         start = [int(a) for a in re.split("[: /]", r[endindex])[0:3]] # this
            syntax is used to break the dates
269         es = datetime.date(start[2],start[0],start[1])
270         ef = es
271     elif r[endindex] == '':
272         start = [int(a) for a in re.split("[: /]", r[startindex])[0:3]] #
            this syntax is used to break the dates
273         es = datetime.date(start[2],start[0],start[1])
274         ef = es
275     else:
276         start = [int(a) for a in re.split("[: /]", r[startindex])[0:3]] #
            this syntax is used to break the dates
277         end = [int(a) for a in re.split("[: /]", r[endindex])[0:3]] # this
            syntax is used to break the dates
278         es = datetime.date(start[2],start[0],start[1])
279         ef = datetime.date(end[2],end[0],end[1])
280
281     conn.execute("INSERT INTO activities
        (projectid,activityid,activityname,duration,cost,es,ef,primaryconstraint,
        primaryconstraintdate,calendar) VALUES (?,?,?,?,?,?,?,?,?,?);", (projectid
        ,activityid,activityname,duration,cost,es,ef,primaryconstraint,
        primaryconstraintdate,calendar))
282
283     # update the projects with the new start
284     conn.execute("UPDATE projects SET start = (SELECT DATE(MIN(JULIANDAY(es)))
        FROM activities WHERE projects.projectid=activities.projectid and
        activities.es IS NOT NULL);")
285     # open the relationships sheet
286     sheet = wb.sheet by name('TASKPRED')
287     r = sheet.row values(1)
288     activity1index = r.index('Predecessor')
289     activity2index = r.index('Successor')
290     relationshiptypeindex = r.index('Relationship Type')
291     rlagindex = r.index('Lag(h)')
292     for i in range(2,sheet.nrows):
293         r = sheet.row values(i)
294         activity1id = r[activity1index]
295         activity2id = r[activity2index]
296         relationship type = r[relationshiptypeindex]
297         rlag = r[rlagindex]

```

```

298         conn.execute("INSERT INTO relationships
299             (projectid,activitylid,activity2id,type, rlag) VALUES (?,?,?,?);", (
300             projectid,activitylid,activity2id,relationship type,rlag))
301     conn.commit()
302     log('Done.')
303
304     def pv(interest, days): # Function to calculate the present value inside SQLite
305         return math.pow(1+interest/365,days)
306
307     def parse date(date isoformat): # parse a date formatted as an iso format string
308         'yyyy-mm-dd' into a date object
309         try:
310             year = int(date isoformat.split("-")[0])
311         except:
312             log(' ! error in year in "%s"%date isoformat)
313             return 'null'
314         try:
315             month = int(date isoformat.split("-")[1])
316         except:
317             log(' ! error in year in "%s"%date isoformat)
318             return 'null'
319         try:
320             day = int(date isoformat.split("-")[2])
321         except:
322             log(' ! error in year in "%s"%date isoformat)
323             return 'null'
324         return datetime.date(year,month,day)
325
326     def calculate(scope): # calculate schedule and cashflow, the scope can be "normal"
327         or "opt"
328         log("SCHEDULING STARTED")
329         starttime = datetime.datetime.now()
330         if scope in ['normal']:
331             cond = ''
332         elif scope in ['opt']:
333             cond = 'opt'
334         else:
335             log(' [!] Error in parameter for calculate function')
336         if cond == '':
337             conn.execute("UPDATE activities SET es = NULL, ef = NULL, ls = NULL, lf =
338             NULL, ff = NULL, tf = NULL, os = NULL, of = NULL, lag = NULL;") # clear
339             previous results
340             conn.execute("Update projects set finish = NULL, duration = NULL;")
341             conn.execute("Update portfolio set start = NULL, finish = NULL, duration =
342             NULL;")
343             projects = [a[0] for a in conn.execute("SELECT projectid FROM projects").fetchall
344             ())
345
346         if cond == '': # FRONT AND BACK CALCULATION for the early start and finish
347             for projectid in projects: # loop for each project NOTE: Foor some reason,
348             it may better to do it this way
349                 log(' > %s Project %s/%s with %s activity'%(datetime.datetime.now() -
350                 starttime,projects.index(projectid) + 1, len(projects),conn.execute(
351                 "SELECT COUNT(*) FROM activities WHERE projectid = ?",(projectid,)).
352                 fetchall()[0][0]))
353                 conn.execute("UPDATE activities SET es = (SELECT start FROM projects
354                 WHERE projectid = ?) WHERE projectid = ? AND activityid NOT IN (SELECT
355                 activity2id FROM relationships WHERE relationships.projectid = ?);",(
356                 projectid,projectid,projectid))
357                 conn.execute("UPDATE activities SET ef = DATE(JULIANDAY(es) + duration)
358                 WHERE projectid = ? AND es IS NOT NULL;",(projectid,))
359                 while conn.execute("SELECT COUNT(*) FROM activities WHERE projectid = ?
360                 AND es IS NULL;",(projectid,)).fetchall()[0][0] > 0: # loop while there
361                 are unscheduled activities
362                     log(' + %s New front - Remaining activities = %s activity'%(
363                     datetime.datetime.now() - starttime,conn.execute("SELECT COUNT(*)
364                     FROM activities WHERE projectid = ? AND es IS NULL;",(projectid,)).
365                     fetchall()[0][0]))

```

- 7 -

```

345 acts = [a[0] for a in conn.execute("SELECT DISTINCT(activity2id)
FROM big WHERE projectid = ? AND activity2es IS NULL AND activityles
IS NOT NULL;",(projectid,)).fetchall()]
346 log("    -> Focusing on %s activity"%len(acts))
347 count = 0
348 for activityid in acts: # loop on each unscheduled activity
349     d = conn.execute('SELECT activityles,activitylef,
activity2duration,type, rlag FROM big WHERE projectid = ? AND
activity2id = ?;',(projectid,activityid)).fetchall()
350     if None not in[a[0] for a in d]: # check if all needed data in
there
351         es1l = [a[0] for a in d]
352         ef1l = [a[1] for a in d]
353         dur2l = [a[2] for a in d]
354         rtypel = [a[3] for a in d]
355         rlags = [a[4] for a in d]
356         project start = parse date(conn.execute("SELECT start FROM
projects WHERE projects.projectid = ?",(projectid,)).fetchall
()[0][0])
357         if None not in es1l + ef1l + dur2l + rtypel and d != []: #
If this is true, then the activity can be scheduled because
all its predecessors are set
358             es1l = [parse date(a) for a in es1l]
359             ef1l = [parse date(a) for a in ef1l]
360             possible es2 = [project start]
361             for es1,ef1,dur2,rtype,rlag in zip(es1l,ef1l,dur2l,rtypel
,rlags):
362                 if rlag == None:
363                     rlag = 0
364                 if rtype in ['fs','FS','fS','Fs']:
365                     possible es2.append(ef1 + datetime.timedelta(rlag))
366                 if rtype in ['ss','SS','sS','Ss']:
367                     possible es2.append(es1 + datetime.timedelta(rlag))
368                 if rtype in ['ff','FF','fF','Ff']:
369                     possible es2.append(ef1 - datetime.timedelta(dur2
) + datetime.timedelta(rlag))
370                 if rtype in ['sf','SF','sF','Sf']:
371                     possible es2.append(es1 - datetime.timedelta(dur2
) + datetime.timedelta(rlag))
372                 es2 = max(possible es2) # get the max of the
possible es2
373
374 # compare if there is a constraint of the activity
375 primaryconstraint, primaryconstraintdate,duration,
calendar = conn.execute("SELECT primaryconstraint,
primaryconstraintdate,duration,calendar FROM activities
WHERE projectid = ? AND activityid = ?",(projectid,
activityid)).fetchall()[0]
376 if primaryconstraint != None:
377     primaryconstraintdate = parse date(
primaryconstraintdate)
378     if primaryconstraint == "Finish On or Before" and es2
+ datetime.timedelta(duration) >
primaryconstraintdate:
379         es2 = primaryconstraintdate - datetime.timedelta(
duration)
380     elif primaryconstraint == "Start On or After" and es2
< primaryconstraintdate:
381         es2 = primaryconstraintdate
382     ef2 = adddays(es2,duration,calendar)
383
384 #add the new calculated early start
385 conn.execute("UPDATE activities SET es = ? WHERE
projectid = ? AND activityid = ?;",(es2.isoformat(),
projectid,activityid))
386 conn.execute("UPDATE activities SET ef = ? WHERE
projectid = ? AND activityid = ?;",(ef2.isoformat(),
projectid,activityid))

```

```

387         count += 1
388     log("        -> set %s activity "%count)
389     # write the new project finish dates
390     log(" + %s Writing project finish dates"%(datetime.datetime.now() -
391         starttime,))
392     conn.execute("UPDATE projects SET finish = (SELECT
393         DATE(MAX(JULIANDAY(ef))) FROM activities WHERE activities.projectid = ?)
394         WHERE projectid = ?;",(projectid,projectid))
395     conn.execute("UPDATE projects SET duration = JULIANDAY(finish) -
396         JULIANDAY(start) WHERE projectid = ?;",(projectid,))
397     # write the new values in the portfolio
398     conn.execute("UPDATE portfolio SET start = (SELECT
399         DATE(MIN(JULIANDAY(start))) FROM projects);")
400     conn.execute("UPDATE portfolio SET finish = (SELECT
401         DATE(MAX(JULIANDAY(start))) FROM projects);")
402     conn.execute("UPDATE portfolio SET duration = JULIANDAY(finish) -
403         JULIANDAY(start);")
404     conn.execute("UPDATE portfolio SET numberofprojects = (SELECT COUNT(*)
405         from projects);")
406     conn.execute("UPDATE portfolio SET numberofactivities = (SELECT COUNT(*)
407         from activities);")
408     # Back calculations
409     conn.execute("UPDATE activities SET lf = (SELECT finish FROM projects
410         WHERE projectid = ?) WHERE projectid = ? AND activityid NOT IN (SELECT
411         activitylid FROM relationships WHERE relationships.projectid = ?);",(
412         projectid,projectid,projectid))
413     conn.execute("UPDATE activities SET ls = DATE(JULIANDAY(lf) - duration)
414         WHERE projectid = ? AND lf IS NOT NULL;",(projectid,))
415     while conn.execute("SELECT COUNT(*) FROM activities WHERE projectid = ?
416         AND ls IS NULL;",(projectid,)).fetchall()[0][0] > 0: # loop while there
417         are unscheduled activities
418         log(' + %s New back "%(datetime.datetime.now() - starttime,))
419         acts = [a[0] for a in conn.execute("SELECT DISTINCT(activitylid)
420         FROM big WHERE projectid = ? AND activity1ls IS NULL AND activity2ls
421         IS NOT NULL;",(projectid,)).fetchall()]
422         log("        -> Focusing on %s activity"%len(acts))
423         count = 0
424         for activityid in acts: # loop on each unscheduled activity
425             d = conn.execute("SELECT activity2ls,activity2lf,
426                 activitylduration,type, rlag FROM big WHERE projectid = ? AND
427                 activitylid = ?;",(projectid,activityid)).fetchall()
428             if None not in [a[0] for a in d]:
429                 ls2l = [a[0] for a in d]
430                 lf2l = [a[1] for a in d]
431                 dur1l = [a[2] for a in d]
432                 rtype1 = [a[3] for a in d]
433                 rlags = [a[4] for a in d]
434                 project finish = parse date(conn.execute("SELECT finish FROM
435                 projects WHERE projects.projectid = ?",(projectid,)).fetchall
436                 ())[0][0])
437                 if None not in ls2l + lf2l and d != []: # If this is true,
438                     then the activity can be scheduled because all its
439                     predecessors are set
440                     ls2l = [parse date(a) for a in ls2l]
441                     lf2l = [parse date(a) for a in lf2l]
442                     possible lf1 = [project finish]
443                     for ls2,lf2,dur1,rtype,rlag in zip(ls2l,lf2l,dur1l,rtype1
444                     ,rlags):
445                         if rlag == None:
446                             rlag = 0
447                         if rtype in ['fs','FS','fS','Fs']:
448                             possible lf1.append(ls2 + datetime.timedelta(rlag))
449                         if rtype in ['ss','SS','sS','Ss']:
450                             possible lf1.append(ls2 + datetime.timedelta(dur1
451                             ) + datetime.timedelta(rlag))
452                         if rtype in ['ff','FF','fF','Ff']:
453                             possible lf1.append(lf2 + datetime.timedelta(rlag))
454                         if rtype in ['sf','SF','sF','Sf']:

```

```

430         possible lf1.append(lf2 + datetime.timedelta(durl
431         ) + datetime.timedelta(rlag))
432         lf1 = min(possible lf1)
433
434         # compare if there is a constraint of the activity
435         primaryconstraint, primaryconstraintdate,duration,
436         calendar = conn.execute("SELECT primaryconstraint,
437         primaryconstraintdate,duration,calendar FROM activities
438         WHERE projectid = ? AND activityid = ?",(projectid,
439         activityid)).fetchall()[0]
440         if primaryconstraint != None:
441             primaryconstraintdate = parse date(
442             primaryconstraintdate)
443             if primaryconstraint == "Finish On or Before" and lf1
444             > primaryconstraintdate:
445                 lf1 = primaryconstraintdate
446             elif primaryconstraint == "Start On or After" and lf1
447             - datetime.timedelta(duration) <
448             primaryconstraintdate:
449                 lf1 = primaryconstraintdate + datetime.timedelta(
450                 duration)
451             ls = adddays(lf1,-duration,calendar)
452
453         conn.execute("UPDATE activities SET lf = ? WHERE
454         projectid = ? AND activityid = ?;",(lf1.isoformat(),
455         projectid,activityid))
456         conn.execute("UPDATE activities SET ls = ? WHERE
457         projectid = ? AND activityid = ?;",(ls.isoformat(),
458         projectid,activityid))
459         count += 1
460         log("      -> set %s activity "%count)
461         conn.execute("UPDATE activities SET tf = JULIANDAY(lf) - JULIANDAY(ef)
462         WHERE projectid = ?;",(projectid,)) # calculate the total float
463         # update projects
464         conn.execute("UPDATE projects SET totalactivities = (SELECT COUNT(*) FROM
465         activities WHERE projects.projectid = activities.projectid);")
466         conn.execute("UPDATE projects SET criticalactivities = (SELECT COUNT(*) FROM
467         activities WHERE projects.projectid = activities.projectid AND activities.tf
468         = 0);")
469         conn.commit()
470         conn.execute('VACUUM;')
471         conn.commit()
472         log(' + %s Done.'%(datetime.datetime.now() - starttime,))
473
474     elif cond == 'opt':# FRONT CALCULATION ONLY FOR THE OPTIMUM. This will not
475     randomize the lags, it will only calculate upon them
476     for projectid in projects: # loop for each project
477         log(' > %s Project %s/%s with %s activity'%(datetime.datetime.now() -
478         starttime,projects.index(projectid) + 1, len(projects),conn.execute(
479         "SELECT COUNT(*) FROM activities WHERE projectid = ?;",(projectid,)).
480         fetchall()[0][0]))
481         conn.execute("UPDATE activities SET os = DATE((SELECT JULIANDAY(start)
482         FROM projects WHERE projectid = ?) + lag) WHERE projectid = ? AND
483         activityid NOT IN (SELECT activity2id FROM relationships WHERE
484         relationships.projectid = ?);",(projectid,projectid,projectid))
485         conn.execute("UPDATE activities SET of = DATE(JULIANDAY(os) + duration)
486         WHERE projectid = ? AND os IS NOT NULL;",(projectid,))
487         while conn.execute("SELECT COUNT(*) FROM activities WHERE projectid = ?
488         AND os IS NULL;",(projectid,)).fetchall()[0][0] > 0: # loop while there
489         are unscheduled activities
490             log(' + %s New front - Remaining activities = %s activity'%(
491             datetime.datetime.now() - starttime,conn.execute("SELECT COUNT(*)
492             FROM activities WHERE projectid = ? AND os IS NULL;", (projectid,)).
493             fetchall()[0][0]))
494             acts = [a[0] for a in conn.execute("SELECT DISTINCT(activity2id)
495             FROM biq WHERE projectid = ? AND activity2os IS NULL AND activity2los
496             IS NOT NULL;",(projectid,)).fetchall()]
497             log("      -> Focusing on %s activity"%len(acts))

```

- 10 -

```

465 count = 0
466 for activityid in acts: # loop on each unscheduled activity
467     d = conn.execute('SELECT activitylos,activitylof,
activity2duration,type, rlag FROM big WHERE projectid = ? AND
activity2id = ?;',(projectid,activityid)).fetchall()
468     os1l = [a[0] for a in d]
469     of1l = [a[1] for a in d]
470     dur2l = [a[2] for a in d]
471     rtypel = [a[3] for a in d]
472     rlags = [a[4] for a in d]
473     laq, es2 = conn.execute('SELECT laq, es FROM activities WHERE
projectid = ? AND activityid = ?;',(projectid,activityid)).
fetchall()[0]
474     lag = int(laq)
475     es2 = parse date(es2)
476     project start = parse date(conn.execute("SELECT start FROM
projects WHERE projects.projectid = ?",(projectid,)).fetchall()[0]
)
477     if None not in os1l + of1l + dur2l + rtypel and d != []: # If
this is true, then the activity can be scheduled because all its
predecessors are set
478         os1l = [parse date(a) for a in os1l]
479         of1l = [parse date(a) for a in of1l]
480         possible os2 = [project start,es2 + datetime.timedelta(lag)]
481         for os1,of1,dur2,rtype,rlag in zip(os1l,of1l,dur2l,rtypel,
rlags):
482             if rlag == None:
483                 rlag = 0
484             if rtype in ['fs','FS','fS','Fs']:
485                 possible os2.append(of1 + datetime.timedelta(rlag))
486             if rtype in ['ss','SS','sS','Ss']:
487                 possible os2.append(os1 + datetime.timedelta(rlag))
488             if rtype in ['ff','FF','fF','Ff']:
489                 possible os2.append(of1 - datetime.timedelta(dur2) +
datetime.timedelta(rlag))
490             if rtype in ['sf','SF','sF','Fs']:
491                 possible os2.append(os1 - datetime.timedelta(dur2) +
datetime.timedelta(rlag))
492             os2 = max(possible os2)
493
494     # compare if there is a constraint of the activity
495     primaryconstraint, primaryconstraintdate,duration,calendar =
conn.execute("SELECT primaryconstraint,
primaryconstraintdate,duration,calendar FROM activities
WHERE projectid = ? AND activityid = ?",(projectid,
activityid)).fetchall()[0]
496     if primaryconstraint != None:
497         primaryconstraintdate = parse date(primaryconstraintdate)
498         if primaryconstraint == "Finish On or Before" and os2 +
datetime.timedelta(duration) > primaryconstraintdate:
499             os2 = primaryconstraintdate - datetime.timedelta(
duration)
500         elif primaryconstraint == "Start On or After" and os2 <
primaryconstraintdate:
501             os2 = primaryconstraintdate
502         of = adddays(os2,duration,calendar)
503
504     conn.execute("UPDATE activities SET os = ? WHERE projectid =
? AND activityid = ?;",(os2.isoformat(),projectid,activityid))
505     conn.execute("UPDATE activities SET lag = JULIANDAY(os) -
JULIANDAY(es) WHERE projectid = ? AND activityid = ?;",(
projectid,activityid))
506     conn.execute("UPDATE activities SET of = ? WHERE projectid =
? AND activityid = ?;",(of.isoformat(),projectid,activityid))
507     count += 1
508     log("      -> set %s activity "%count)
509 # -----
510 log('Calculating Cash flow ')

```

- 11 -



```

511     if cond == '':
512         # Calculate cost and price of projects
513         conn.execute("UPDATE projects SET cost = (SELECT SUM(cost) FROM activities
                    WHERE projects.projectid = activities.projectid);")
514         conn.execute("UPDATE projects SET price = cost * (1+markup);")
515     # Initiate the cash flow table
516     conn.execute("DELETE FROM cashflow%s;"%cond)
517     conn.execute("DELETE FROM cashflowall%s;"%cond)
518     # create the dates -----
519     first date = conn.execute("SELECT DATE(MIN(JULIANDAY(es))) FROM activities;").
                    fetchall()[0][0]
520     finish date = conn.execute("SELECT DATE(MAX(JULIANDAY(ls))) FROM activities;").
                    fetchall()[0][0]
521     max payment period = int(conn.execute("SELECT MAX(paymentperiod) FROM projects;")
                    .fetchall()[0][0])
522     max retention period = int(conn.execute("SELECT MAX(retentionperiod) FROM
                    projects;").fetchall()[0][0])
523     first date = datetime.date(int(first date.split("-")[0]),int(first date.split("-"
                    ) [1]),int(first date.split("-")[2]))
524     finish date = datetime.date(int(finish date.split("-")[0]),int(finish date.split(
                    "-" ) [1]),int(finish date.split("-") [2]))
525     last date = finish date + datetime.timedelta(max(max payment period,
                    max retention period)+10)
526     curr date = first date
527     projects = [a[0] for a in conn.execute("SELECT projectid FROM projects;")]
528     while curr date <= last date:
529         for project in projects:
530             conn.execute("INSERT INTO cashflow%s (date,projectid) VALUES (?,?);"%cond
                    ,(curr date.isoformat(),project))
531             curr date += datetime.timedelta(1)
532     log(" + %s Filled cash flow with dates"%(datetime.datetime.now() - starttime,))
533     for projectid in projects: # loop for each project NOTE: For some reason, it may
                    better to do it this way
534         log(" - Calculating cash project %s/%s"%(projects.index(projectid)+1,len(
                    projects)))
535         # Fill cash out
536         if cond == '':
537             conn.execute("UPDATE cashflow%s SET cashout = (SELECT SUM(cost/duration)
                    FROM activities WHERE projectid = ? AND cashflow%s.date >= activities.%s
                    and cashflow%s.date < activities.%s) WHERE projectid = ?;"%(cond,cond,
                    'es',cond,'ef'),(projectid,projectid))
538         elif cond == 'opt':
539             conn.execute("UPDATE cashflow%s SET cashout = (SELECT SUM(cost/duration)
                    FROM activities WHERE projectid = ? AND cashflow%s.date >= activities.%s
                    and cashflow%s.date < activities.%s) WHERE projectid = ?;"%(cond,cond,
                    'os',cond,'of'),(projectid,projectid))
540     conn.execute("UPDATE cashflow%s SET cashout = 0 WHERE cashout IS NULL AND
                    projectid = ?;"%cond,(projectid,))
541     log(" + %s Calculated cash out"%(datetime.datetime.now() - starttime,))
542     conn.execute("UPDATE cashflow%s SET cashin = 0 WHERE projectid = ?;"%cond,(
                    projectid,))
543     # Fill the invoices issued without considering the downpayment and retention
544     #~ conn.execute("UPDATE cashflow%s SET cashin = cashin + (SELECT
                    SUM(cashout) * (1+(SELECT markup from projects WHERE projectid = ?)) FROM
                    cashflow%s as c2 WHERE projectid = ? AND DATE(cashflow%s.date, 'start of
                    month') = DATE(c2.date, 'start of month')) WHERE date = DATE(date, 'start of
                    month', '+1 month', '-1 day') AND projectid =
                    ?;"%(cond,cond,cond),(projectid,projectid,projectid))
545     conn.execute("UPDATE cashflow%s SET cashin = cashin + (SELECT SUM(cashout)
                    FROM cashflow%s as c2 WHERE projectid = ? AND
                    DATE(JULIANDAY(cashflow%s.date) - (SELECT paymentperiod FROM projects
                    WHERE projectid = ?), 'start of month', '+1 month', '-1 day') = DATE(c2.date, 'start
                    of month', '+1 month', '-1 day')) WHERE DATE(JULIANDAY(date) - (SELECT
                    paymentperiod FROM projects WHERE projectid = ?)) = DATE(JULIANDAY(date) -
                    (SELECT paymentperiod FROM projects WHERE projectid = ?), 'start of
                    month', '+1 month', '-1 day') AND projectid = ?;"%(cond,cond,cond),(projectid,
                    projectid,projectid,projectid,projectid))

```



```

547 # increase profit deduction for downpayment and retention
548 conn.execute("UPDATE cashflow%s SET cashin = cashin * (1+(SELECT markup from
549 projects WHERE projectid = ?)) WHERE projectid = ? AND cashin != 0;"%cond,(
projectid,projectid))
550 conn.execute("UPDATE cashflow%s SET cashin = cashin - ((cashin / (SELECT
price FROM projects WHERE projectid = ?)) * ((SELECT downpayment*price FROM
projects WHERE projectid = ?) + (SELECT retention*price FROM projects WHERE
projectid = ?))) WHERE cashin != 0 AND projectid = ?;"%(cond,),(projectid,
projectid,projectid,projectid))
551 # Fill the downpayments
552 conn.execute("UPDATE cashflow%s SET cashin = cashin + (SELECT
downpayment*price FROM projects WHERE projectid = ?) WHERE date = (SELECT
start FROM projects WHERE projectid = ?) AND projectid = ?;"%(cond,),(
projectid,projectid,projectid))
553 # Fill the retention received
554 conn.execute("UPDATE cashflow%s SET cashin = cashin + (SELECT
retention*price FROM projects WHERE projectid = ?) WHERE date=(SELECT
DATE(JULIANDAY(finish)+retentionperiod) FROM projects WHERE projectid = ?)
AND projectid = ?;"%(cond,),(projectid,projectid,projectid))
555 log(" + %s Calculated cash in"%(datetime.datetime.now() - starttime,))
556 # Fill cash out cumulative
557 conn.execute("UPDATE cashflow%s SET cashoutcum = (SELECT SUM(cashout) FROM
cashflow%s as temp WHERE projectid = ? AND JULIANDAY(cashflow%s.date) >=
JULIANDAY(temp.date)) WHERE projectid = ?;"%(cond,cond,cond), (projectid,
projectid))
558 # Fill cash in cumulative
559 conn.execute("UPDATE cashflow%s SET cashincum = (SELECT SUM(cashin) FROM
cashflow%s as temp WHERE projectid = ? AND JULIANDAY(cashflow%s.date) >=
JULIANDAY(temp.date)) WHERE projectid = ?;"%(cond,cond,cond), (projectid,
projectid))
560 log(" + %s Calculated cummulative"%(datetime.datetime.now() - starttime,))
561 # Fill the overdraft
562 conn.execute("UPDATE cashflow%s SET overdraft = cashincum - cashoutcum WHERE
projectid = ?;"%cond,(projectid,))
563 #Fill the discounted values
564 conn.create function("pv",2,pv) # Creates a new function in SQLITE to
calculate the present value
565 conn.execute("UPDATE cashflow%s SET cashoutdisc = cashout / pv((SELECT
interest from projects WHERE projectid = ?),JULIANDAY(date) - (SELECT
MIN(JULIANDAY(start)) FROM projects)) WHERE projectid = ?;"%(cond,),(
projectid, projectid))
566 conn.execute("UPDATE cashflow%s SET cashindisc = cashin / pv((SELECT
interest from projects WHERE projectid = ?),JULIANDAY(date) - (SELECT
MIN(JULIANDAY(start)) FROM projects)) WHERE projectid = ?;"%(cond,),(
projectid,projectid))
567 conn.execute("UPDATE cashflow%s SET cashoutcumdisc = cashoutcum / pv((SELECT
interest from projects WHERE projectid = ?),JULIANDAY(date) - (SELECT
MIN(JULIANDAY(start)) FROM projects)) WHERE projectid = ?;"%(cond,),(
projectid, projectid))
568 conn.execute("UPDATE cashflow%s SET cashincumdisc = cashincum / pv((SELECT
interest from projects WHERE projectid = ?),JULIANDAY(date) - (SELECT
MIN(JULIANDAY(start)) FROM projects)) WHERE projectid = ?;"%(cond,),(
projectid, projectid))
569 conn.execute("UPDATE cashflow%s SET overdraftdisc = overdraft / pv((SELECT
interest from projects WHERE projectid = ?),JULIANDAY(date) - (SELECT
MIN(JULIANDAY(start)) FROM projects)) WHERE projectid = ?;"%(cond,),(
projectid, projectid))
570 log(" + %s Calculated discounted"%(datetime.datetime.now() - starttime,))
571 # fill into the cashflow all table
572 conn.execute("DELETE FROM cashflowall%s;"%cond)
573 # create the dates
574 first_date = conn.execute("SELECT DATE(MIN(JULIANDAY(start))) FROM projects;").
fetchall()[0][0]
575 last_date = conn.execute("SELECT
DATE(MAX(MAX(JULIANDAY(finish)+paymentperiod),MAX(JULIANDAY(finish)+retentionperio
d)) + 50) FROM projects;").fetchall()[0][0]
576 first_date = datetime.date(int(first_date.split("-")[0]),int(first_date.split("-"

```

- 13 -

```

577 ) [1], int(first date.split("-")[2]))
578 last date = datetime.date(int(last date.split("-")[0]), int(last date.split("-")[1
579 ]), int(last date.split("-")[2]))
580 curr date = first date
581 while curr date <= last date:
582     conn.execute("INSERT INTO cashflowall%s (date) VALUES ('%s')"%(cond, curr date
583     .isoformat()))
584     curr date += datetime.timedelta(1)
585 # fill in the values in the
586 conn.execute("UPDATE cashflowall%s SET projectid = 'all';"%cond)
587 for col in ['cashin', 'cashout', 'cashincum', 'cashoutcum', 'cashindisc',
588 'cashoutdisc', 'cashincumdisc', 'cashoutcumdisc', 'overdraft', 'overdraftdisc']:
589     conn.execute("UPDATE cashflowall%s SET %s = (SELECT SUM(%s) FROM cashflow%s
590     WHERE cashflow%s.date = cashflowall%s.date);"% (cond, col, col, cond, cond))
591 # Fill the present values and the npv into the projects table
592 conn.execute("UPDATE projects SET cashinpv%s = (SELECT SUM(cashindisc) FROM
593 cashflow%s WHERE cashflow%s.projectid = projects.projectid);"% (cond, cond, cond))
594 conn.execute("UPDATE projects SET cashoutpv%s = (SELECT SUM(cashoutdisc) FROM
595 cashflow%s WHERE cashflow%s.projectid = projects.projectid);"% (cond, cond, cond))
596 conn.execute("UPDATE projects SET npv%s = cashinpv%s - cashoutpv%s;"% (cond, cond,
597 cond))
598 conn.execute("UPDATE projects SET maxoverdraftdisc%s = (SELECT
599 MAX(overdraftdisc) FROM cashflow%s WHERE cashflow%s.projectid =
600 projects.projectid);"% (cond, cond, cond))
601 conn.execute("UPDATE projects SET minoverdraftdisc%s = (SELECT
602 MIN(overdraftdisc) FROM cashflow%s WHERE cashflow%s.projectid =
603 projects.projectid);"% (cond, cond, cond))
604 # FILL the cashflow values in the portfolio table
605 if cond == '':
606     conn.execute("UPDATE portfolio SET cost = (SELECT SUM(cost) FROM projects);")
607     conn.execute("UPDATE portfolio SET price = (SELECT SUM(price) FROM projects);")
608     conn.execute("UPDATE portfolio SET cashinpv%s = (SELECT SUM(cashindisc) FROM
609     cashflowall%s);"% (cond, cond))
610     conn.execute("UPDATE portfolio SET cashoutpv%s = (SELECT SUM(cashoutdisc) FROM
611     cashflowall%s);"% (cond, cond))
612     conn.execute("UPDATE portfolio SET npv%s = cashinpv%s - cashoutpv%s;"% (cond, cond,
613     cond))
614     conn.execute("UPDATE portfolio SET maxoverdraftdisc%s = (SELECT
615     MAX(overdraftdisc) FROM cashflowall%s);"% (cond, cond))
616     conn.execute("UPDATE portfolio SET minoverdraftdisc%s = (SELECT
617     MIN(overdraftdisc) FROM cashflowall%s);"% (cond, cond))
618     conn.commit()
619     conn.execute("VACUUM;")
620     conn.commit()
621     log(" + %s Done."%(datetime.datetime.now() - starttime,))
622
623 def export(): # export a lot of files for further analysis
624     log("Exporting")
625     if not os.path.exists(export folder):
626         os.mkdir(export folder)
627     # Remove old files
628
629     files = os.listdir(export folder)
630     for file in files:
631         try:
632             os.remove(export folder+file)
633         except:
634             log(' [!] Error removing file "%s" from export folder!%file)
635     log(' - Removed old files from export folder')
636
637 # export database summary
638 txtfile = export folder + 'summary.txt'
639 with open(txtfile, 'w') as f:
640     tablenames = [a[0] for a in conn.execute("Select name FROM sqlite master
641     WHERE type='table' or type='view';").fetchall()]
642     for name in tablenames:
643         f.write('-> '+name+'\n')
644         columnnames = [a[1] for a in conn.execute("PRAGMA table_info(%s);" %name)]

```

- 14 -

```

627         columntypes = [a[2] for a in conn.execute("PRAGMA table info(%s);" %name)]
628         for col,t in zip(columnnames,columntypes):
629             f.write('         -> '+col + ' -> '+ t +'\n')
630
631     # Export excel file
632     excel file = export folder + 'output.xlsx'
633     log(' - Exporting to Excel File "%s"' %excel file)
634     wb = xlswriter.Workbook(excel file)
635     bold = wb.add format({'bold': True})
636     for table in [a[0] for a in conn.execute("SELECT name FROM sqlite master WHERE
type='table';").fetchall()]:
637         ws = wb.add worksheet(table)
638         ws.repeat rows(0)
639         ws.freeze panes(1, 1)
640         ws.set portrait()
641         ws.set paper(4)
642         ws.center horizontally()
643         ws.center vertically()
644         ws.set footer('&CPage &P of &N')
645         ws.fit to pages(1, 0)
646         row = 0
647         col = 0
648         heads = [a[0] for a in conn.execute("PRAGMA table info(%s);" %table)]
649         for head in heads:
650             ws.write(row,col,head,bold)
651             col += 1
652             ws.set column(0,col,15)
653         row = 1
654         sql = 'SELECT * FROM %s;'%table
655         for each in conn.execute(sql).fetchall():
656             col = 0
657             for cell in each:
658                 ws.write(row,col,cell)
659                 col += 1
660             row += 1
661     conn.close
662     wb.close()
663
664     # export csv file for every table
665     log(" - Exporting csvs")
666     tables = [a[0] for a in conn.execute("SELECT name FROM sqlite master WHERE
type='table';").fetchall()]:
667     for table in tables:
668         with open(export folder+'%s.csv'%table,'w',newline='') as csvfile:
669             w = spamwriter = csv.writer(csvfile)
670             data = conn.execute("PRAGMA table info(%s);" %table).fetchall()
671             data = [a[1] for a in data]
672             w.writerow(data)
673             data = conn.execute("SELECT * FROM %s;"%table)
674             for r in data:
675                 w.writerow(r)
676
677     # export portfolio charts
678     log(" - Exporting portfolio charts")
679     export file name = export folder + 'portfoliosummary' + figure export format
680     data = conn.execute("SELECT projectid,totalactivities,criticalactivities FROM
projects;").fetchall()
681     projectids = [a[0] for a in data]
682     totalactivities = [int(a[1]) for a in data]
683     criticalactivities = [int(a[2]) for a in data]
684     noncriticalactivities = [a[1] - a[0] for a in zip(criticalactivities,
totalactivities)]
685     lw = 10
686     plt.vlines(range(len(projectids)),[0 for a in projectids], criticalactivities,
color = 'red', label = 'Critical Activities', lw = lw)
687     plt.vlines(range(len(projectids)),criticalactivities,totalactivities, color =
'blue', label = 'Non-critical Activities', lw = lw)
688     plt.xlabel('Projects')

```

- 15 -

```

689 plt.ylabel('Number of Activities')
690 plt.title(title + 'Summary of Activities')
691 plt.xticks(range(len(projectids)),projectids)
692 plt.legend(loc='best',fancybox=True,framealpha=0.5, fontsize = 8)
693 plt.margins(0.05)
694 plt.savefig(export file name,transparent=True)
695 plt.close('all')
696
697 # export gantt charts for portfolio and projects
698 log(' - Exporting Gantt Charts')
699 projects = conn.execute("SELECT projectid, start, finish FROM projects;").
700 fetchall()
701 projects.reverse()
702 projectids = [a[0] for a in projects]
703 projectstarts = [parse date(a[1]) for a in projects]
704 projectfinishes = [parse date(a[2]) for a in projects]
705 fig, ax = plt.subplots(1)
706 lw = 8
707 color = 'blue'
708 ax.hlines(range(len(projectids)),projectstarts,projectfinishes,lw=lw, color =
709 color)
710 fig.autofmt xdate()
711 plt.xlabel('Time')
712 plt.ylabel('Projects')
713 plt.yticks(range(len(projectids)),projectids)
714 xticks = [min(projectstarts) ,max(projectfinishes)]
715 plt.xticks(xticks,xticks)
716 plt.title(title + 'Portfolio Gantt Chart')
717 plt.margins(0.05)
718 plt.savefig(export folder+'portfoliogantchart'+figure export format,transparent=
719 True)
720 plt.close('all')
721 for p in projectids:
722     export file name = 'gantchart' + p + figure export format
723     data = conn.execute("SELECT activityid,es,ef,lf FROM activities WHERE
724     projectid = '%s';"%p).fetchall()
725     data.reverse()
726     activityid = [a[0] for a in data]
727     activityn = range(len(activityid))
728     adict = {}
729     for aid, an in zip(activityid, activityn):
730         adict[aid] = an
731         es = [parse date(a[1]) for a in data]
732         ef = [parse date(a[2]) for a in data]
733         lf = [parse date(a[3]) for a in data]
734         fig, ax = plt.subplots(1)
735         lw = 2
736         ax.hlines(activityn,es,ef,lw=lw, color = color, label= 'Non-critical
737         Activities')
738         if None not in lf:
739             ax.hlines(activityn,ef, lf,lw=lw/1.5, color = 'green', label = 'Total
740             Float')
741             critaid = []
742             ces = []
743             cef= []
744             clf = []
745             for a in zip(range(len(activityid)),es,ef,lf):
746                 if a[2] == a[3]:
747                     critaid.append(a[0])
748                     ces.append(a[1])
749                     cef.append(a[2])
750                     clf.append(a[3])
751             ax.hlines(critaid,ces,cef,lw=lw, color = 'red', label= 'Critical
752             Activities')
753         plt.yticks(activityn,['' for a in activityid], size = 2)
754         fig.autofmt xdate()
755         # add arrows for the relationships
756         data = conn.execute("SELECT activitylid, activityles, activitylef,

```

```

750         activity2id, activity2es, activity2ef, type FROM big WHERE projectid = '%s';"
%p).fetchall()
751     for activitylid, activityles, activitylef, activity2id, activity2es,
752     activity2ef, rtype in zip([a[0] for a in data], [parse date(a[1]) for a in
753     data], [parse date(a[2]) for a in data], [a[3] for a in data], [parse date(a[
754     4]) for a in data], [parse date(a[5]) for a in data], [a[6] for a in data]):
755         activity1n = adict[activitylid]
756         activity2n = adict[activity2id]
757         if rtype in ['fs', 'FS', 'fS', 'Fs']:
758             plt.annotate("", xy=(activitylef, activity1n), xycoords='data',
759             xytext=(activity2es, activity2n), textcoords='data', arrowprops=dict(
760             arrowstyle="<-", lw = 0.2))
761         if rtype in ['ss', 'SS', 'sS', 'Ss']:
762             plt.annotate("", xy=(activityles, activity1n), xycoords='data',
763             xytext=(activity2es, activity2n), textcoords='data', arrowprops=dict(
764             arrowstyle="<-", lw = 0.2))
765         if rtype in ['ff', 'FF', 'fF', 'Ff']:
766             plt.annotate("", xy=(activitylef, activity1n), xycoords='data',
767             xytext=(activity2ef, activity2n), textcoords='data', arrowprops=dict(
768             arrowstyle="<-", lw = 0.2))
769         if rtype in ['sf', 'SF', 'sF', 'Sf']:
770             plt.annotate("", xy=(activityles, activity1n), xycoords='data',
771             xytext=(activity2ef, activity2n), textcoords='data', arrowprops=dict(
772             arrowstyle="<-", lw = 0.2))
773     plt.xlabel('Time')
774     xticks = [min(es), max(ef)]
775     plt.xticks(xticks,xticks)
776     plt.ylabel('Activities')
777     plt.title(title + 'Gantt Chart - ' + p)
778     plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
779     plt.margins(0.05)
780     plt.savefig(export folder+export file name,transparent=True)
781     plt.close('all')
782
783 # Export cashflow charts
784 log(" - Exporting cashflow")
785 data = conn.execute("SELECT
786 date,cashincum,cashoutcum,overdraft,cashincumdisc,cashoutcumdisc,overdraftdisc
787 from cashflowall;").fetchall()
788 dates = [a[0] for a in data]
789 dates = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split('-')
790 [2])) for a in dates]
791 cashincum = [a[1] for a in data]
792 cashoutcum = [a[2] for a in data]
793 overdraft = [a[3] for a in data]
794 cashincumdisc = [a[4] for a in data]
795 cashoutcumdisc = [a[5] for a in data]
796 overdraftdisc = [a[6] for a in data]
797 plt.close('all')
798 fig, ax = plt.subplots(1)
799 lw = 0.5
800 for a, l in ((cashincum,'Cash In Cummulative'),(cashoutcum,'Cash Out Cummulative'
801 ),(overdraft,'Overdraft'),(cashincumdisc,'Cash Out Cummulative Discounted'),(
802 cashoutcumdisc,'Cash Out Cummulative Discounted'),(overdraftdisc,'Overdraft
803 Discounted')):
804     ax.plot(dates,a,label = l, lw=lw)
805     fig.autofmt xdate()
806     plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
807     plt.xlabel('Time')
808     plt.ylabel('EGP')
809     plt.title(title + 'Cash-flow')
810     plt.grid(True)
811     plt.savefig(export folder+'cashflow'+figure export format,transparent=True)
812     plt.close('all')
813
814 # export chart of trials
815 log(" - Exporting chart for the trials")
816 data = conn.execute("SELECT trialid, initialnpv, trialnpv, bestnpv FROM trials;"

```

```

799     ).fetchall()
800     trialid = [a[0] for a in data]
801     initialnpv = [a[1] for a in data]
802     trialnpv = [a[2] for a in data]
803     bestnpv = [a[3] for a in data]
804     fig, ax = plt.subplots(1)
805     lw = 0.5
806     for a, l in ((initialnpv, 'Initial NPV'), (bestnpv, 'Best NPV')):
807         ax.plot(trialid, a, label = l, lw= 2 * lw)
808     ax.plot(trialid, trialnpv, 'o', label = 'Trial NPV', lw=lw)
809     plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
810     plt.xlabel('Trial #')
811     plt.ylabel('NPV')
812     plt.title(title + 'Optimization trials')
813     plt.grid(True)
814     plt.margins(0.05)
815     plt.savefig(export folder+'optimization trials'+figure export format, transparent=
816     True)
817     plt.close('all')
818
819 # export optimization gantt chart
820 try:
821     log(' - Exporting Optimized Gantt Charts')
822     projects = conn.execute("SELECT projectid, start, finish FROM projects;").
823     fetchall()
824     for p in projectids:
825         export file name = 'optimizedganttchart' + p + figure export format
826         data = conn.execute("SELECT activityid,es,ef,lf,os,of FROM activities
827         WHERE projectid = '%s';"%p).fetchall()
828         data.reverse()
829         activityid = [a[0] for a in data]
830         activityn = range(len(activityid))
831         adict = {}
832         for aid, an in zip(activityid, activityn):
833             adict[aid] = an
834             es = [a[1] for a in data]
835             ef = [a[2] for a in data]
836             lf = [a[3] for a in data]
837             ost = [a[4] for a in data]
838             of = [a[5] for a in data]
839             es = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
840             ('-')[2])) for a in es]
841             ef = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
842             ('-')[2])) for a in ef]
843             lf = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
844             ('-')[2])) for a in lf]
845             ost = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.
846             split('-')[2])) for a in ost]
847             of = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
848             ('-')[2])) for a in of]
849             fig, ax = plt.subplots(1)
850             lw = 2
851             ax.hlines(range(len(activityid)),es,lf,lw=0.7*lw, color = 'grey', label=
852             'Activity Range (ES to LF)')
853             #~ ax.hlines(range(len(activityid)),of,lf,lw=0.7*lw, color = 'grey',
854             label= 'Total Float')
855             ax.hlines(range(len(activityid)),ost,of,lw=lw, color = 'blue', label=
856             'Non-critical Activities')
857             if None not in lf:
858                 critaid = []
859                 ces = []
860                 cef = []
861                 clf = []
862                 for a in zip(range(len(activityid)),es,ef,lf):
863                     if a[2] == a[3]:
864                         critaid.append(a[0])
865                         ces.append(a[1])
866                         cef.append(a[2])
867                         clf.append(a[2])

```

- 18 -

```

855         clf.append(a[3])
856         ax.hlines(critaid,ces,cef,lw=lw, color = 'red', label= 'Critical
Activities')
857     plt.yticks(range(len(activityid)),[' for a in activityid], size = 2)
858     fig.autofmt xdate()
859     # add arrows for the relationships
860     data = conn.execute("SELECT activitylid, activitylos, activitylof,
activity2id, activity2os, activity2of, type FROM big WHERE projectid =
'%s';"%p).fetchall()
861     for activitylid, activitylos, activitylof, activity2id, activity2os,
activity2of, rtype in zip([a[0] for a in data], [parse date(a[1]) for a
in data], [parse date(a[2]) for a in data], [a[3] for a in data], [
parse date(a[4]) for a in data], [parse date(a[5]) for a in data], [a[6]
for a in data]):
862         activityln = adict[activitylid]
863         activity2n = adict[activity2id]
864         if rtype in ['fs','FS','fS','Fs']:
865             plt.annotate("", xy=(activitylof, activityln), xycoords='data',
xytext=(activity2os, activity2n), textcoords='data', arrowprops=
dict(arrowstyle="<", lw = 0.2))
866         if rtype in ['ss','SS','sS','Ss']:
867             plt.annotate("", xy=(activitylos, activityln), xycoords='data',
xytext=(activity2os, activity2n), textcoords='data', arrowprops=
dict(arrowstyle="<", lw = 0.2))
868         if rtype in ['ff','FF','fS','Fs']:
869             plt.annotate("", xy=(activitylof, activityln), xycoords='data',
xytext=(activity2of, activity2n), textcoords='data', arrowprops=
dict(arrowstyle="<", lw = 0.2))
870         if rtype in ['sf','SF','sF','Fs']:
871             plt.annotate("", xy=(activitylos, activityln), xycoords='data',
xytext=(activity2of, activity2n), textcoords='data', arrowprops=
dict(arrowstyle="<", lw = 0.2))
872     plt.xlabel('Time')
873     plt.ylabel('Activities')
874     plt.title(title + 'Optimized Gantt Chart - ' + p)
875     xticks = [min(es),max(ef)]
876     plt.xticks(xticks,xticks)
877     plt.legend(loc='best',fancybox=True,framealpha=0.5, fontsize = 8)
878     plt.margins(0.05)
879     plt.savefig(export folder+export file name,transparent=True)
880     plt.close('all')
881 except Exception as e:
882     log(" - Failed to export optimized gantt charts, skipping")
883
884 # export optimization gantt chart without relationship arrows
885 try:
886     log(' - Exporting Optimized Gantt Charts without relationship arrows')
887     projects = conn.execute("SELECT projectid, start, finish FROM projects;").
fetchall()
888     for p in projectids:
889         export file name = 'optimizedganttchartnoarrows' + p + figure export format
890         data = conn.execute("SELECT activityid,es,ef,lf,os,of FROM activities
WHERE projectid = '%s';"%p).fetchall()
891         data.reverse()
892         activityid = [a[0] for a in data]
893         activityn = range(len(activityid))
894         adict = {}
895         for aid, an in zip(activityid, activityn):
896             adict[aid] = an
897         es = [a[1] for a in data]
898         ef = [a[2] for a in data]
899         lf = [a[3] for a in data]
900         ost = [a[4] for a in data]
901         of = [a[5] for a in data]
902         es = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
('-')[2])) for a in es]
903         ef = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
('-')[2])) for a in ef]

```

- 19 -



```

904         lf = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
905         ('-')[2])) for a in lf]
906         ost = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.
907         split('-')[2])) for a in ost]
908         of = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split
909         ('-')[2])) for a in of]
910         fig, ax = plt.subplots(1)
911         lw = 2
912         ax.hlines(range(len(activityid)),es,lf,lw=0.7*lw, color = 'grey', label=
913         'Activity Range (ES to LF)')
914         #~ ax.hlines(range(len(activityid)),of,lf,lw=0.7*lw, color = 'grey',
915         label= 'Total Float')
916         ax.hlines(range(len(activityid)),ost,of,lw=lw, color = 'blue', label=
917         'Non-critical Activities')
918         if None not in lf:
919             critaid = []
920             ces = []
921             cef = []
922             clf = []
923             for a in zip(range(len(activityid)),es,ef,lf):
924                 if a[2] == a[3]:
925                     critaid.append(a[0])
926                     ces.append(a[1])
927                     cef.append(a[2])
928                     clf.append(a[3])
929                 ax.hlines(critaid,ces,cef,lw=lw, color = 'red', label= 'Critical
930                 Activities')
931         plt.yticks(range(len(activityid)),[' for a in activityid], size = 2)
932         fig.autofmt_xdate()
933         plt.xlabel('Time')
934         plt.ylabel('Activities')
935         plt.title(title + 'Optimized Gantt Chart - ' + p)
936         xticks = [min(es) ,max(ef)]
937         plt.xticks(xticks,xticks)
938         plt.legend(loc='best',fancybox=True,framealpha=0.5, fontsize = 8)
939         plt.margins(0.05)
940         plt.savefig(export folder+export file name,transparent=True)
941         plt.close('all')
942     except Exception as e:
943         log(" - Failed to export optimized qantt charts, skipping")
944
945     # Export optimized cashflow charts
946     try:
947         log(" - Exporting optimized cashflow")
948         data = conn.execute("SELECT
949         date,cashincum,cashoutcum,overdraft,cashincumdisc,cashoutcumdisc,overdraftdisc
950         from cashflowall;").fetchall()
951         dates = [a[0] for a in data]
952         dates = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.split(
953         '-')[2])) for a in dates]
954         cashincum = [a[1] for a in data]
955         cashoutcum = [a[2] for a in data]
956         overdraft = [a[3] for a in data]
957         cashincumdisc = [a[4] for a in data]
958         cashoutcumdisc = [a[5] for a in data]
959         overdraftdisc = [a[6] for a in data]
960         data = conn.execute("SELECT
961         date,cashincum,cashoutcum,overdraft,cashincumdisc,cashoutcumdisc,overdraftdisc
962         from cashflowallopt;").fetchall()
963         datesopt = [a[0] for a in data]
964         datesopt = [datetime.date(int(a.split('-')[0]),int(a.split('-')[1]),int(a.
965         split('-')[2])) for a in datesopt]
966         cashincumopt = [a[1] for a in data]
967         cashoutcumopt = [a[2] for a in data]
968         overdraftopt = [a[3] for a in data]
969         cashincumdiscopt = [a[4] for a in data]
970         cashoutcumdiscopt = [a[5] for a in data]
971         overdraftdiscopt = [a[6] for a in data]

```

- 20 -



```

959 plt.close('all')
960
961 fig, ax = plt.subplots(1)
962 lw = 0.5
963 for a, l in ((cashincum, 'Cash In Cumulative'), (cashoutcum, 'Cash Out
Cumulative'), (overdraft, 'Overdraft'), (cashincumdisc, 'Cash Out Cumulative
Discounted'), (cashoutcumdisc, 'Cash Out Cumulative Discounted'), (
overdrafterdisc, 'Overdraft Discounted')):
964     ax.plot(dates, a, label = l, lw=lw)
965 for a, l in ((cashincumopt, 'Optimized Cash In Cumulative'), (cashoutcumopt,
'Optimized Cash Out Cumulative'), (overdrafteropt, 'Optimized Overdraft'), (
cashincumdiscopt, 'Optimized Cash Out Cumulative Discounted'), (
cashoutcumdiscopt, 'Optimized Cash Out Cumulative Discounted'), (
overdrafterdiscopt, 'Optimized Overdraft Discounted')):
966     ax.plot(dates, a, label = l, lw=lw)
967 fig.autofmt_xdate()
968 plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
969 plt.xlabel('Time')
970 plt.ylabel('EGP')
971 plt.title(title + 'Cash-flow - Combined')
972 plt.grid(True)
973 plt.savefig(export_folder+'optimized cashflow combined'+figure_export_format,
transparent=True)
974 plt.close('all')
975
976 fig, ax = plt.subplots(1)
977 lw = 0.5
978 for a, l in ((cashincum, 'Cash In Cumulative'), (cashoutcum, 'Cash Out
Cumulative'), (overdraft, 'Overdraft')):
979     ax.plot(dates, a, label = l, lw=lw)
980 for a, l in ((cashincumopt, 'Optimized Cash In Cumulative'), (cashoutcumopt,
'Optimized Cash Out Cumulative'), (overdrafteropt, 'Optimized Overdraft')):
981     ax.plot(dates, a, label = l, lw=lw)
982 fig.autofmt_xdate()
983 plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
984 plt.xlabel('Time')
985 plt.ylabel('EGP')
986 plt.title(title + 'Optimized Cash-flow (Comparision)')
987 plt.grid(True)
988 plt.savefig(export_folder+'optimized cashflow fv'+figure_export_format,
transparent=True)
989 plt.close('all')
990
991 fig, ax = plt.subplots(1)
992 lw = 0.5
993 for a, l in ((overdraft, 'Overdraft'), (overdrafteropt, 'Optimized Overdraft')):
994     ax.plot(dates, a, label = l, lw=lw)
995 fig.autofmt_xdate()
996 plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
997 plt.xlabel('Time')
998 plt.ylabel('EGP')
999 plt.title(title + 'Optimized Overdraft (Comparision)')
1000 plt.grid(True)
1001 plt.savefig(export_folder+'optimized cashflow overdraft'+figure_export_format
,transparent=True)
1002 plt.close('all')
1003
1004 fig, ax = plt.subplots(1)
1005 lw = 0.5
1006 for a, l in ((overdraft, 'Overdraft'), (overdrafteropt, 'Optimized Overdraft'), (
overdrafterdisc, 'Overdraft Discounted'), (overdrafterdiscopt, 'Optimized
Overdraft Discounted')):
1007     ax.plot(dates, a, label = l, lw=lw)
1008 fig.autofmt_xdate()
1009 plt.legend(loc='best', fancybox=True, framealpha=0.5, fontsize = 8)
1010 plt.xlabel('Time')
1011 plt.ylabel('EGP')
1012 plt.title(title + 'Optimized Overdraft Discounted (Comparision)')

```

```

1013         plt.grid(True)
1014         plt.savefig(export folder+'optimized cashflow overdraft discounted'+
figure export format,transparent=True)
1015         plt.close('all')
1016     except Exception as e:
1017         log(" - Could not export optimized cashflow, skipping")
1018
1019     log(" - Done")
1020
1021 def optimize(): # optimize
1022     log("OPTIMIZING")
1023     global conn
1024     starttime = datetime.datetime.now()
1025     # Create list of table names
1026     tables = [a[0] for a in conn.execute("Select name FROM sqlite master WHERE
type='table';").fetchall()]
1027     tables.remove('trials')
1028     tablesbackup = [a+'bck' for a in tables]
1029     # initiate the npv with the current npv using es and ef
1030     initialnpv = conn.execute("SELECT npv FROM portfolio;").fetchall()[0][0]
1031     bestnpv = initialnpv
1032     trialnpv = bestnpv
1033     conn.execute("DELETE FROM trials;")
1034     conn.commit()
1035     conn.execute("INSERT INTO trials (trialid, initialnpv, trialnpv, bestnpv) VALUES
('%s','%s','%s','%s');"%(0,initialnpv,trialnpv,bestnpv))
1036     # start the trials
1037     trialid = 0
1038     condition = True
1039     while condition:
1040         trialid += 1
1041         log(' <> %s Trial %s'%(datetime.datetime.now() - starttime,trialid))
1042         conn.execute("UPDATE activities SET lag = NULL;")
1043         conn.execute("UPDATE activities SET os = NULL;")
1044         conn.execute("UPDATE activities SET of = NULL;")
1045         conn.execute("UPDATE activities SET lag = 0 WHERE cost = 0 OR tf = 0;") #
activities that are critical or have no cost don't need to be optimized'
1046         conn.execute("UPDATE activities SET os = es WHERE lag = 0;")
1047         conn.execute("UPDATE activities SET of = ef WHERE lag = 0;")
1048         # randomize the lags, must be done outside of the database because the
random function in sqlite3 is biased
1049         for projectid in [a [0] for a in conn.execute("SELECT projectid FROM
projects;").fetchall()]:
1050             for activityid in [a[0] for a in conn.execute("SELECT activityid from
activities WHERE projectid = '%s' AND tf > 0 AND cost > 0;"%projectid).
fetchall()]:
1051                 tf = conn.execute("SELECT tf FROM activities WHERE projectid = '%s'
AND activityid = '%s';"%(projectid, activityid)).fetchall()[0][0]
1052                 lag = random.randint(0, tf)
1053                 conn.execute("UPDATE activities SET lag = '%s' WHERE projectid =
'%s' AND activityid = '%s'"%(lag, projectid, activityid)) # update
the lag
1054         # calculate the new schedule for the trial using the new lags
1055         calculate('opt')
1056         # get current opt npv and compare
1057         trialnpv = conn.execute("SELECT npvopt FROM portfolio;").fetchall()[0][0]
1058         if trialnpv > bestnpv: # check if the current trial yields a better result
and store it
1059             bestnpv = trialnpv
1060             for table, bck in zip(tables,tablesbackup):
1061                 conn.execute("DROP TABLE IF EXISTS %s;"%bck)
1062                 conn.execute("CREATE TABLE %s AS SELECT * FROM %s;"%(bck,table))
1063             d = [a[0] for a in conn.execute("select DISTINCT(bestnpv) from trials
Order BY bestnpv DESC LIMIT 2;").fetchall()]
1064             if len(d) >= 2:
1065                 if (bestnpv / (sum(d)/len(d))) < optimization stoppingpercentage:
1066                     condition = False
1067             if conn.execute("SELECT COUNT(bestnpv) FROM trials WHERE bestnpv = (SELECT

```

- 22 -

```

MAX(bestnpv from trials);").fetchall()[0][0] >=
optimization stoppingmaxtrials:
1068     condition = False
1069     conn.execute("INSERT INTO trials (trialid, initialnpv, trialnpv, bestnpv)
VALUES ('%s','%s','%s','%s');"%(trialid,initialnpv,trialnpv,bestnpv))
1070     log(' <> %s Trial %s ended. Trial NPV = %s, Best NPV = %s'%(datetime.datetime
.now() - starttime,trialid, trialnpv, bestnpv))
1071     if bestnpv > initialnpv:
1072         for table,bck in zip(tables,tablesbackup):
1073             conn.execute("DROP TABLE IF EXISTS %s;"%table)
1074             conn.execute("CREATE TABLE %s AS SELECT * FROM %s;"%(table,bck))
1075             conn.execute("DROP TABLE IF EXISTS %s;"%bck)
1076     conn.commit()
1077     conn.execute("VACUUM;")
1078     conn.commit()
1079     log(' <> %s Optimization ended after %s trial. Initial NPV = %s, Optimized NPV =
%s'%(datetime.datetime.now() - starttime,trialid, initialnpv, bestnpv))

1080
1081 def verificatate():
1082     new database()
1083     create a portfolio()
1084     #- import uptown projects()
1085     database info()
1086     calculate("normal")
1087     optimize()
1088     export()
1089     database info()
1090
1091 def validate():
1092     new database()
1093     #- create a portfolio()
1094     import uptown projects()
1095     database info()
1096     calculate("normal")
1097     optimize()
1098     export()
1099     database info()
1100
1101 # ----- GUI PART -----
1102 class Drop menu: # generic drop list menu for the GUI, because the one in tkinter sucks
1103     def show(self):
1104         try:
1105             self.menu.destroy()
1106         except:
1107             pass
1108         self.menu = tk.OptionMenu(self.master, self.var, *self.options)
1109         self.menu.pack()
1110
1111     def options(self, options):
1112         self.options = options
1113         if len(options) > 0:
1114             self.var.set(options[0])
1115         else:
1116             self.var.set('')
1117
1118     def init (self, master):
1119         self.master = master
1120         self.var = tk.StringVar()
1121         self.options = []
1122
1123 class Gantt chart: # Gantt chart for the whole portfolio normal or optimized
1124     margin = 40
1125     lw = 2
1126     project = 'all'
1127     deltat = 10
1128     deltaa = 15
1129     barwidth = 5
1130     def show(self):

```

```

1131     global conn
1132     self.canvas.delete('all')
1133     data = conn.execute("SELECT projectid, activityid, es, ef, lf FROM
activities;").fetchall()
1134     activityindex = {}
1135     for n, projectid, activityid in zip([a for a in range(len(data))], [a[0] for
a in data], [a[1] for a in data]):
1136         activityindex[projectid + activityid] = n
1137     esl = [parse date(a[2]) for a in data]
1138     efl = [parse date(a[3]) for a in data]
1139     lfl = [parse date(a[4]) for a in data]
1140     na = len(data)
1141     mint = min(esl)
1142     maxt = max(lfl)
1143     totalt = (maxt - mint).days
1144     self.canvas['scrollregion'] = (0, 0, (totalt * self.deltat) + 2*self.marqin,
(na * self.deltat) + 2 * self.margin)
1145     # marqins
1146     self.canvas.create_rectangle((self.margin, self.margin), (self.margin +
totalt * self.deltat, self.margin + na * self.deltat))
1147     for a in range(1, totalt):
1148         self.canvas.create_line((self.margin + a * self.deltat, self.margin), (
self.marqin + a * self.deltat, self.marqin + na * self.deltat)), fill =
'grey80')
1149         if a % 10 == 0:
1150             self.canvas.create_text((self.margin + a * self.deltat, self.margin -
10), anchor = 'center', text = str(mint + datetime.timedelta(a)))
1151             self.canvas.create_line((self.marqin + a * self.deltat, self.marqin
), (self.margin + a * self.deltat, self.margin + na * self.deltat)),
fill = 'black')
1152     # add activities
1153     for n, es, ef, lf in zip([a for a in range(len(data))], esl, efl, lfl):
1154         if ef == lf:
1155             self.canvas.create_rectangle((self.margin + (es - mint).days * self.
deltat, self.marqin + n * self.deltat), (self.margin + (ef - mint).
days * self.deltat, self.margin + n * self.deltat + self.barwidth)),
fill = 'red')
1156         else:
1157             self.canvas.create_rectangle((self.margin + (es - mint).days * self.
deltat, self.marqin + n * self.deltat), (self.margin + (ef - mint).
days * self.deltat, self.margin + n * self.deltat + self.barwidth)),
fill = 'green')
1158             self.canvas.create_rectangle((self.margin + (ef - mint).days * self.
deltat, self.margin + n * self.deltat + 0.3 * self.barwidth), (self.
margin + (lf - mint).days * self.deltat, self.margin + n * self.
deltat + 0.7 * self.barwidth)), fill = 'blue')
1159     # add relations
1160     data = conn.execute("SELECT activitylid, activityles, activitylef,
activity2id, activity2es, activity2ef, type, projectid FROM big").fetchall()
1161     for activitylid, activityles, activitylef, activity2id, activity2es,
activity2ef, rtype, projectid in zip([a[0] for a in data], [parse date(a[1])
for a in data], [parse date(a[2]) for a in data], [a[3] for a in data], [
parse date(a[4]) for a in data], [parse date(a[5]) for a in data], [a[6] for
a in data], [a[7] for a in data]):
1162         activity1n = activityindex[projectid + activitylid]
1163         activity2n = activityindex[projectid + activity2id]
1164         if rtype in ['fs', 'FS', 'fS', 'Fs']:
1165             self.canvas.create_line((self.margin + (activitylef - mint).days *
self.deltat, self.marqin + activity1n * self.deltat), (self.margin + (
activity2es - mint).days * self.deltat, self.margin + activity2n *
self.deltat)), arrow = 'last')
1166         if rtype in ['ss', 'SS', 'sS', 'Ss']:
1167             self.canvas.create_line((self.margin + (activityles - mint).days *
self.deltat, self.marqin + activity1n * self.deltat), (self.margin + (
activity2es - mint).days * self.deltat, self.margin + activity2n *
self.deltat)), arrow = 'last')
1168         if rtype in ['ff', 'FF', 'fS', 'Fs']:
1169             self.canvas.create_line((self.margin + (activitylef - mint).days *

```

- 24 -

```

1170         self.deltat, self.margin + activityln * self.deltaa),(self.margin + (
1171         activity2ef - mint).days * self.deltat, self.margin + activity2n *
1172         self.deltaa), arrow = 'last')
1173     if rtype in ['sf', 'SF', 'sF', 'Fs']:
1174         self.canvas.create_line(((self.margin + (activityles - mint).days *
1175         self.deltat, self.margin + activityln * self.deltaa),(self.margin + (
1176         activity2ef - mint).days * self.deltat, self.margin + activity2n *
1177         self.deltaa)), arrow = 'last')
1178
1179     def show_opt(self):
1180         global conn
1181         self.canvas.delete('all')
1182         data = conn.execute("SELECT projectid, activityid, es, ef, os, of, lf FROM
1183         activities;").fetchall()
1184         activityindex = {}
1185         for n, projectid, activityid in zip([a for a in range(len(data))], [a[0] for
1186         a in data], [a[1] for a in data]):
1187             activityindex[projectid + activityid] = n
1188         esl = [parse date(a[2]) for a in data]
1189         efl = [parse date(a[3]) for a in data]
1190         osl = [parse date(a[4]) for a in data]
1191         ofl = [parse date(a[5]) for a in data]
1192         lfl = [parse date(a[6]) for a in data]
1193         na = len(data)
1194         mint = min(esl)
1195         maxt = max(lfl)
1196         totalt = (maxt - mint).days
1197         self.canvas['scrollregion'] = (0, 0, (totalt * self.deltat) + 2*self.marqin,
1198         (na * self.deltaa) + 2 * self.margin)
1199         # margins
1200         self.canvas.create_rectangle((self.margin, self.margin), (self.margin +
1201         totalt * self.deltat, self.margin + na * self.deltaa))
1202         for a in range(1, totalt):
1203             self.canvas.create_line(((self.margin + a * self.deltat, self.margin), (
1204             self.margin + a * self.deltat, self.marqin + na * self.deltaa)), fill =
1205             'grey80')
1206             if a % 21 == 0:
1207                 self.canvas.create_text((self.margin + a * self.deltat, self.margin -
1208                 10), anchor = 'center', text = str(mint + datetime.timedelta(a)))
1209                 self.canvas.create_line(((self.marqin + a * self.deltat, self.marqin
1210                 ), (self.margin + a * self.deltat, self.margin + na * self.deltaa)),
1211                 fill = 'black')
1212         # add activities
1213         for n, es, ef, os, of, lf in zip([a for a in range(len(data))], esl, efl, osl
1214         , ofl, lfl):
1215             if ef == lf:
1216                 self.canvas.create_rectangle(((self.marqin + (es - mint).days * self.
1217                 deltat, self.margin + n * self.deltaa),(self.margin + (ef - mint).
1218                 days * self.deltat, self.marqin + n * self.deltaa + self.barwidth)),
1219                 fill = 'red')
1220             else:
1221                 self.canvas.create_rectangle(((self.marqin + (es - mint).days * self.
1222                 deltat, self.margin + n * self.deltaa + 0.3 * self.barwidth),(self.
1223                 marqin + (os - mint).days * self.deltat, self.margin + n * self.
1224                 deltaa + 0.7 * self.barwidth)), fill = 'grey')
1225                 self.canvas.create_rectangle(((self.margin + (os - mint).days * self.
1226                 deltat, self.margin + n * self.deltaa),(self.margin + (of - mint).
1227                 days * self.deltat, self.margin + n * self.deltaa + self.barwidth)),
1228                 fill = 'green')
1229                 self.canvas.create_rectangle(((self.margin + (of - mint).days * self.
1230                 deltat, self.margin + n * self.deltaa + 0.3 * self.barwidth),(self.
1231                 margin + (lf - mint).days * self.deltat, self.margin + n * self.
1232                 deltaa + 0.7 * self.barwidth)), fill = 'grey')
1233         # add relationships
1234         data = conn.execute("SELECT activitylid, activitylos, activitylof,
1235         activity2id, activity2os, activity2of, type, projectid FROM biq").fetchall()
1236         for activitylid, activitylos, activitylof, activity2id, activity2os,
1237         activity2of, rtype, projectid in zip([a[0] for a in data], [parse_date(a[1])

```

```

1208     for a in data, [parse date(a[2]) for a in data], [a[3] for a in data], [
1209     parse date(a[4]) for a in data], [parse date(a[5]) for a in data], [a[6] for
1210     a in data], [a[7] for a in data]):
1211         activity1n = activityindex[projectid + activitylid]
1212         activity2n = activityindex[projectid + activity2id]
1213         if rtype in ['fs', 'FS', 'fS', 'Fs']:
1214             self.canvas.create_line(((self.margin + (activitylof - mint).days *
1215             self.deltat, self.margin + activity1n * self.deltaa), (self.margin + (
1216             activity2os - mint).days * self.deltat, self.margin + activity2n *
1217             self.deltaa)), arrow = 'last')
1218         if rtype in ['ss', 'SS', 'sS', 'Ss']:
1219             self.canvas.create_line(((self.margin + (activitylos - mint).days *
1220             self.deltat, self.margin + activity1n * self.deltaa), (self.margin + (
1221             activity2os - mint).days * self.deltat, self.margin + activity2n *
1222             self.deltaa)), arrow = 'last')
1223         if rtype in ['ff', 'FF', 'fF', 'Ff']:
1224             self.canvas.create_line(((self.margin + (activitylof - mint).days *
1225             self.deltat, self.margin + activity1n * self.deltaa), (self.margin + (
1226             activity2of - mint).days * self.deltat, self.margin + activity2n *
1227             self.deltaa)), arrow = 'last')
1228         if rtype in ['sf', 'SF', 'sF', 'Ff']:
1229             self.canvas.create_line(((self.margin + (activitylos - mint).days *
1230             self.deltat, self.margin + activity1n * self.deltaa), (self.margin + (
1231             activity2of - mint).days * self.deltat, self.margin + activity2n *
1232             self.deltaa)), arrow = 'last')
1233
1234     def init (self, master, normal or opt):
1235         self.frame = tk.Frame(master, bg = 'white')
1236         self.frame.pack(fill = 'both', expand = True)
1237         self.canvas = tk.Canvas(self.frame, bg = 'white')
1238         self.yscr = tk.Scrollbar(self.frame, orient = 'vertical', command = self.
1239         canvas.yview)
1240         self.xscr = tk.Scrollbar(self.frame, orient = 'horizontal', command = self.
1241         canvas.xview)
1242         self.canvas.configure(xscrollcommand = self.xscr.set, yscrollcommand = self.
1243         yscr.set)
1244         self.yscr.pack(fill = 'y', side = 'right')
1245         self.xscr.pack(fill = 'x', side = 'bottom')
1246         self.canvas.pack(fill = 'both', expand = True, side = 'left')
1247         if normal or opt == 'normal':
1248             self.show()
1249         elif normal or opt == 'opt':
1250             self.show opt()
1251
1252     class Table: # this is generic a table widget made using using ttk.treewiew
1253         width = 100
1254         global conn
1255         def delete(self, *arg):
1256             if self.table.focus() != '':
1257                 data = {}
1258                 for name, value in zip(self.table['columns'], self.table.item(self.table.
1259                 focus())['values']):
1260                     data[name] = value
1261                 if self.table scope == 'projects':
1262                     conn.execute("Delete FROM projects WHERE projectid = ?;", (data[
1263                     'projectid'], ))
1264                 elif self.table scope == 'activities':
1265                     conn.execute("Delete FROM activities WHERE projectid = ? AND
1266                     activityid = ?;", (data['projectid'], data['activityid']))
1267                 elif self.table scope == 'relationships':
1268                     conn.execute("Delete FROM relationships WHERE projectid = ? AND
1269                     activitylid = ? AND activity2id = ? AND type = ?;", (data['projectid'],
1270                     data['activitylid'], data['activity2id'], data['type']))
1271                 self.refresh()
1272                 conn.commit()
1273
1274     def create(self):
1275         Form_new(self.master, self.table_scope)

```

```

1253         self.refresh()
1254
1255     def refresh(self):
1256         for child in self.bottomframe.winfo_children():
1257             child.destroy()
1258         self.table = ttk.Treeview(self.bottomframe)
1259         self.table['show'] = 'headings'
1260         self.table['selectmode'] = 'browse'
1261         self.yscr = tk.Scrollbar(self.bottomframe, orient = "vertical", command =
1262             self.table.yview)
1263         self.xscr = tk.Scrollbar(self.bottomframe, orient = "horizontal", command =
1264             self.table.xview)
1265         self.yscr.pack(fill = 'y', side = "right")
1266         self.xscr.pack(fill = 'x', side = "bottom")
1267         self.table["yscrollcommand"] = self.yscr.set
1268         self.table["xscrollcommand"] = self.xscr.set
1269         self.table.pack(fill = "both", side = 'left')
1270         # get the data
1271         global conn
1272         cur = conn.cursor()
1273         cur.execute("SELECT * FROM %s"%self.table.scope)
1274         headings = [a[0] for a in cur.description]
1275         data = cur.fetchall()
1276         # set the columns
1277         self.table['columns'] = headings
1278         self.table['displaycolumns'] = headings
1279         for head in headings:
1280             self.table.column(head, width = self.width, minwidth = self.width,
1281                 stretch = False, anchor = 'center')
1282             self.table.heading(head, text = head)
1283         # set the data
1284         for r in data:
1285             self.table.insert("", 'end', values = r)
1286
1287     def init (self, master, table scope):
1288         self.master = master
1289         self.table scope = table scope
1290         self.frame = tk.Frame(master, bg = 'white')
1291         self.frame.pack(fill='both', expand = True)
1292         self.topframe = tk.Frame(self.frame)
1293         self.topframe.pack(fill = 'x')
1294         self.bottomframe = tk.Frame(self.frame)
1295         self.bottomframe.pack(fill = 'both', expand = True)
1296         # Create the buttons at the top
1297         if self.table scope in ('projects', 'activities', 'relationships'):
1298             tk.Button(self.topframe, text = 'Create New', command = self.create).pack
1299                 (side = 'left')
1300             tk.Button(self.topframe, text = 'Delete Selected', command = self.delete
1301                 ).pack(side = 'left')
1302             tk.Button(self.topframe, text = 'Refresh', command = self.refresh).pack(
1303                 side = 'left')
1304         self.refresh()
1305
1306 class Plot: # this is for generic financial plotting with dates on the x-axis
1307     title = ''
1308     data = []
1309     lw = 2
1310     colors = ['red', 'blue', 'green', 'brown', 'orange']
1311
1312     def clear(self):
1313         self.title = ''
1314         self.data = []
1315         self.canvas.delete('all')
1316         self.show()
1317
1318     def scalex(self,x):
1319         newx = (self.width + self.margin) + x * (self.width - self.margin - self.
1320             margin)/(self.widthself.maxx - self.minx)

```



```

1314
1315
1316 def scaley(self, y):
    newy = (self.height + self.topmargin) + y * (self.width - self.margin - self.
margin)/(self.widthself.maxx - self.minx)
1317
1318 def show(self, *ev):
1319     self.height = self.master.winfo height()
1320     self.width = self.master.winfo width()
1321     self.margin = max(0.05 * self.width, 0.05 * self.height)
1322     self.topmargin = 2 * self.margin
1323     self.canvas.delete('all')
1324     if self.data == []: # break if empty
1325         return 0
1326     # set the boundaries
1327     allx = []
1328     ally = []
1329     for plot in self.data:
1330         allx += plot['x']
1331         ally += plot['y']
1332     self.minx = min(allx)
1333     self.miny = min(ally)
1334     self.maxx = max(allx)
1335     self.maxy = max(ally)
1336     # create the borders
1337     self.canvas.create line((self.margin, self.height - self.margin), (self.
width-self.margin, self.height - self.margin))
1338     self.canvas.create line((self.margin, self.topmargin), (self.width-self.
margin, self.topmargin))
1339     self.canvas.create line((self.margin, self.height - self.margin), (self.
margin, self.topmargin))
1340     self.canvas.create line((self.width - self.margin, self.height - self.margin
), (self.width-self.margin, self.topmargin))
1341     # plot the lines
1342     for plot in self.data:
1343         x = [(a - self.minx)/(self.maxx - self.minx) * (self.width - self.marqin
- self.margin) for a in plot['x']]
1344         y = [(1 - (a - self.miny)/(self.maxy - self.miny)) * (self.height - self.
margin - self.topmargin) for a in plot['y']]
1345         x = [a + self.margin for a in x]
1346         y = [a + self.topmargin for a in y]
1347         x = [int(a) for a in x]
1348         y = [int(a) for a in y]
1349         self.canvas.create line([a for a in zip(x,y)], width = self.lw, fill =
plot['color'])
1350     # plot line at zero
1351     y = (1 - (0 - self.miny)/(self.maxy - self.miny)) * (self.height - self.
margin - self.topmargin)
1352     y = y + self.topmargin
1353     y = int(y)
1354     self.canvas.create line((self.margin, y),(self.width - self.margin, y))
1355     # create legend
1356     self.legendx = self.margin + 20
1357     self.legendy = self.topmargin + 20
1358     loc = self.legendy
1359     for plot in self.data:
1360         self.canvas.create line((self.legendx + 10, loc),(self.legendx + 30, loc
), fill = plot['color'], width = self.lw)
1361         self.canvas.create text((self.legendx + 40, loc), text = plot['title'],
anchor = 'w')
1362         loc += 15
1363     # add title
1364     self.canvas.create text((self.width/2, self.topmargin/2), text = self.title,
font = ("arial",20), anchor = 'center')
1365     # add the ticks
1366     for a in [self.minx, self.maxx] + [self.minx + datetime.timedelta( a * (self.
maxx - self.minx).days / 10) for a in range(10)]:
1367         location = (a - self.minx)/(self.maxx - self.minx) * (self.width - self.
margin - self.margin)

```

- 28 -



```

1368         location = location + self.margin
1369         location = int(location)
1370         self.canvas.create line((location, self.height - self.margin + 5), (
1371         location, self.height - self.margin + 10))
1372         self.canvas.create text((location, self.height - self.margin + 20), text
1373         = str(a), anchor = 'center')
1374         for a in [a for a in range(int(self.miny), int(self.maxy), int((self.maxy -
1375         self.miny) / 20))] + [int(self.maxy)] + [0]:
1376         location = (1 - (a - self.miny)/(self.maxy - self.miny)) * (self.height -
1377         self.margin - self.topmargin)
1378         location = location + self.topmargin
1379         location = int(location)
1380         self.canvas.create line((self.margin - 5, location), (self.margin - 10,
1381         location))
1382         self.canvas.create text((self.margin - 20, location), text = str(a),
1383         anchor = 'e')
1384
1385     def add_plot(self, title, x, y):
1386         plot = {}
1387         plot['title'] = title
1388         for a in y:
1389             if a == None:
1390                 y[y.index(a)] = 0
1391         plot['x'] = x
1392         plot['y'] = y
1393         plot['color'] = self.colors[len(self.data)]
1394         self.data.append(plot)
1395         self.show()
1396
1397     def set_title(self, title):
1398         self.title = title
1399         self.show()
1400
1401     def init (self, master):
1402         self.frame = tk.Frame(master)
1403         self.frame.pack(fill = 'both', expand = True)
1404         self.master = master
1405         self.canvas = tk.Canvas(self.frame, bg = 'white')
1406         self.canvas.pack(fill = 'both', expand = True)
1407         self.canvas.bind("<Configure>",self.show)
1408         self.show()
1409
1410     class Form new: # a new window to create new stuff
1411     def ok(self, *arg):
1412         if self.focus == 'activities':
1413             projectid = self.projectid_selector.get()
1414             activityid = self.activityid.get()
1415             activityname = self.activityname.get()
1416             duration = self.duration.get()
1417             cost = self.cost.get()
1418             conn.execute("INSERT INTO activities
1419             (projectid,activityid,activityname,duration,cost) VALUES (?,?,?,?,?);", (
1420             projectid,activityid,activityname,duration,cost))
1421         elif self.focus == 'projects':
1422             projectid = self.projectid.get()
1423             projectname = self.projectname.get()
1424             start = self.start.get()
1425             interest = self.interest.get()
1426             markup = self.markup.get()
1427             downpayment = self.downpayment.get()
1428             invoiceinterval = self.invoiceinterval.get()
1429             paymentperiod = self.paymentperiod.get()
1430             retention = self.retention.get()
1431             retentionperiod = self.retentionperiod.get()
1432             conn.execute("INSERT INTO projects
1433             (projectid,projectname,start,interest,markup,downpayment,invoiceinterval,p
1434             aymentperiod,retention,retentionperiod) VALUES (?,?,?,?,?,?);", (
1435             projectid, projectname,start,interest,markup,downpayment,invoiceinterval,

```

```

1425         paymentperiod,retention,retentionperiod))
1426     elif self.focus == 'relationships':
1427         projectid = self.projectid selector.get()
1428         activity1id = self.activity1 selector.get()
1429         activity2id = self.activity2 selector.get()
1430         relationship type = self.type selector.get()
1431         conn.execute("INSERT INTO relationships
1432         (projectid,activity1id,activity2id,type) VALUES (?,?,,?)", (projectid,
1433         activity1id,activity2id,relationship type))
1434     conn.commit()
1435     self.root.destroy()
1436
1437 def selected a project(self, *ev):
1438     global conn
1439     try:
1440         activities = [a[0] for a in conn.execute("SELECT activityid FROM
1441         activities WHERE projectid = ?;", (str(self.project selector.get()), )).
1442         fetchall()]
1443         self.activity1 selector['values'] = activities
1444         self.activity1 selector.set(activities[0])
1445         self.activity2 selector['values'] = activities
1446         self.activity2 selector.set(activities[0])
1447     except:
1448         pass
1449
1450 def create entry(self, description, widget name):
1451     tk.Label(self.root, text = description).grid(column = 0, row = self.row,
1452     sticky = 'w')
1453     exec("self.%s = ttk.Entry(self.root)"%widget name)
1454     exec("self.%s.grid(column = 1, row = self.row, sticky = 'w')"%widget name)
1455     self.row += 1
1456
1457 def init (self, master, projects or activities or relationships):
1458     self.focus = projects or activities or relationships
1459     self.master = master
1460     self.root = tk.Toplevel(self.master)
1461     if self.focus == 'projects': title = 'Create New Project'
1462     if self.focus == 'activities': title = 'Create New Activity'
1463     if self.focus == 'relationships': title = 'Create New Relationship'
1464     self.root.title(title)
1465     self.root.geometry('+300+100')
1466     self.root.resizable(height = False, width = False)
1467     self.row = 0
1468     if self.focus in ['activities', 'relationships']:
1469         tk.Label(self.root, text = 'Select Project id:').grid(column = 0, row =
1470         self.row, sticky = 'w')
1471         self.project selector = ttk.Combobox(self.root)
1472         try:
1473             projects = [a[0] for a in conn.execute("SELECT projectid FROM
1474             projects;").fetchall()]
1475             self.project selector['values'] = projects
1476             self.project selector.set(projects[0])
1477         except:
1478             self.project selector['values'] = []
1479         self.project selector.bind("<<ComboboxSelected>>", self.selected a project)
1480         self.project selector.grid(column = 1, row = self.row, sticky = 'w')
1481         self.row += 1
1482     if self.focus in ['activities']:
1483         for name, widget in [{"New Activity ID: ", "activityid"}, ['Activity
1484         Name: ', 'activityname'], ['Activity Duration: ', 'duration'], ['Activity
1485         Cost: ', 'cost']]:
1486             self.create entry(name, widget)
1487     if self.focus in ['projects']:
1488         for name, widget in [['Project ID: ', 'projectid'], ['Project Name: ',
1489         'projectname'], ['Start (yyyy-mm-dd): ', 'start'], ['Interest: ',
1490         'interest'], ['Markup: ', 'markup'], ['Downpayment: ', 'downpayment'], [
1491         'Invoice Interval (days): ', 'invoiceinterval'], ['Payment Period
1492         (days): ', 'paymentperiod'], ['Retention: ', 'retention'], ['Retention

```

```

1479         Period (days): ', 'retentionperiod']]:
1480             self.create entry(name, widget)
1481     if self.focus in ['relationships']:
1482         tk.Label(self.root, text = 'Select Activity1 id:').grid(column = 0, row =
1483             self.row, sticky = 'w')
1484         self.activity1 selector = ttk.Combobox(self.root)
1485         self.activity1 selector.grid(column = 1, row = self.row, sticky = 'w')
1486         self.row += 1
1487         tk.Label(self.root, text = 'Select Activity2 id:').grid(column = 0, row =
1488             self.row, sticky = 'w')
1489         self.activity2 selector = ttk.Combobox(self.root)
1490         self.activity2 selector.grid(column = 1, row = self.row, sticky = 'w')
1491         self.row += 1
1492         tk.Label(self.root, text = 'Select Relationship type:').grid(column = 0,
1493             row = self.row, sticky = 'w')
1494         self.type selector = ttk.Combobox(self.root)
1495         self.type selector['values'] = ('FS', 'SS', 'FF', 'SF')
1496         self.type selector.set('FS')
1497         self.type selector.grid(column = 1, row = self.row, sticky = 'w')
1498         self.row += 1
1499         tk.Button(self.root, text = "Ok", command = self.ok, width = 15).grid(column =
1500             0, row = self.row, colspan = 2)
1501         self.root.bind("<KeyPress-Return>", self.ok)
1502         self.selected a project()
1503
1504     class Main window:
1505         def clear(self):
1506             for child in self.frame.winfo children():
1507                 child.destroy()
1508
1509         def create table from sql(self, table scope):
1510             self.clear()
1511             self.table = Table(self.frame, table scope)
1512
1513         def show gantt chart(self, normal or opt):
1514             self.clear()
1515             Gantt chart(self.frame, normal or opt)
1516
1517         def show plot(self, arg):
1518             self.clear()
1519             global conn
1520             data = conn.execute("SELECT
1521                 date, cashincum, cashoutcum, overdraft, cashincumdisc, cashoutcumdisc, overdraftdisc
1522                 from cashflowall;").fetchall()
1523             dates = [a[0] for a in data]
1524             dates = [datetime.date(int(a.split('-')[0]), int(a.split('-')[1]), int(a.split(
1525                 '-')[2])) for a in dates]
1526             cashincum = [a[1] for a in data]
1527             cashoutcum = [a[2] for a in data]
1528             overdraft = [a[3] for a in data]
1529             cashincumdisc = [a[4] for a in data]
1530             cashoutcumdisc = [a[5] for a in data]
1531             overdraftdisc = [a[6] for a in data]
1532             data = conn.execute("SELECT
1533                 date, cashincum, cashoutcum, overdraft, cashincumdisc, cashoutcumdisc, overdraftdisc
1534                 from cashflowallopt;").fetchall()
1535             datesopt = [a[0] for a in data]
1536             datesopt = [datetime.date(int(a.split('-')[0]), int(a.split('-')[1]), int(a.
1537                 split('-')[2])) for a in datesopt]
1538             cashincumopt = [a[1] for a in data]
1539             cashoutcumopt = [a[2] for a in data]
1540             overdraftopt = [a[3] for a in data]
1541             cashincumdiscopt = [a[4] for a in data]
1542             cashoutcumdiscopt = [a[5] for a in data]
1543             overdraftdiscopt = [a[6] for a in data]
1544             plot = Plot(self.frame)
1545             if arg == 'overdraft':
1546                 plot.clear()

```

```

1536         plot.add plot('Overdraft', dates, overdraft)
1537         plot.set title("OVERDRAFT")
1538     elif arg == 'overdraftopt':
1539         plot.clear()
1540         plot.add plot('Overdraft', dates, overdraft)
1541         plot.add plot('Overdraft Optimized', dates, overdraftopt)
1542         plot.set title("OVERDRAFT (Normal vs. Optimized)")
1543     elif arg == 'cashflow':
1544         plot.clear()
1545         plot.add plot('CashIn cumulative', dates, cashincum)
1546         plot.add plot('CashOut cumulative', dates, cashoutcum)
1547         plot.set title("Cash-Flow")
1548     elif arg == 'cashflowopt':
1549         plot.clear()
1550         plot.add plot('CashIn cumulative', dates, cashincum)
1551         plot.add plot('CashOut cumulative', dates, cashoutcum)
1552         plot.add plot('CashIn cumulative Optimized', dates, cashincumopt)
1553         plot.add plot('CashOut cumulative Optimized', dates, cashoutcumopt)
1554         plot.set title("Cash-Flow (Normal vs. Optimized)")
1555     elif arg == 'overdraftdisc':
1556         plot.clear()
1557         plot.add plot('Overdraft Discounted', dates, overdraftdisc)
1558         plot.set title("OVERDRAFT Discounted")
1559     elif arg == 'overdraftdiscopt':
1560         plot.clear()
1561         plot.add plot('Overdraft Discounted', dates, overdraftdisc)
1562         plot.add plot('Overdraft Discounted Optimized', dates, overdraftdiscopt)
1563         plot.set title("OVERDRAFT Discounted (Normal vs. Optimized)")
1564     elif arg == 'cashflowdisc':
1565         plot.clear()
1566         plot.add plot('CashIn cumulative Discounted', dates, cashincumdisc)
1567         plot.add plot('CashOut cumulative Discounted', dates, cashoutcumdisc)
1568         plot.set title("Cash-Flow Discounted")
1569     elif arg == 'cashflowdiscopt':
1570         plot.clear()
1571         plot.add plot('CashIn cumulative Discounted', dates, cashincumdisc)
1572         plot.add plot('CashOut cumulative Discounted', dates, cashoutcumdisc)
1573         plot.add plot('CashIn cumulative Discounted Optimized', dates,
1574             cashincumdiscopt)
1575         plot.add plot('CashOut cumulative Discounted Optimized', dates,
1576             cashoutcumdiscopt)
1577         plot.set title("Cash-Flow Discounted (Normal vs. Optimized)")
1578
1579     def initiate_toolbar(self):
1580         self.menubar = tk.Menu(self.root)
1581         self.root['menu'] = self.menubar
1582         for menu in ('file', 'create', 'portfolio', 'projects', 'activities',
1583             'calculations', 'plot'):
1584             menu = menu.lower()
1585             label = menu.capitalize()
1586             exec("self.menubar.%s = tk.Menu(self.menubar, tearoff = 0)"%menu)
1587             exec("self.menubar.add cascade(label = '%s', menu = self.menubar.%s)"%(
1588                 label, menu))
1589         self.menubar.file.add command(label = 'Clear All', command = new database)
1590         self.menubar.file.add separator()
1591         self.menubar.file.add command(label = 'Create a Random Portfolio', command =
1592             create a portfolio)
1593         self.menubar.file.add command(label = 'Import Validation Projects', command =
1594             import uptown projects)
1595         self.menubar.file.add separator()
1596         self.menubar.file.add command(label = 'Database Info', command = database info)
1597         self.menubar.file.add command(label = 'Clean Database', command =
1598             clean database)
1599         self.menubar.file.add separator()
1600         self.menubar.file.add command(label = 'Export', command = export)
1601         self.menubar.file.add separator()
1602         self.menubar.file.add command(label = 'Verificate (random)', command =
1603             verificate)

```

```

1596 self.menubar.file.add command(label = 'Validate (UPTOWN)', command = validate)
1597 self.menubar.file.add separator()
1598 self.menubar.file.add command(label = 'Exit', command = self.root.destroy)
1599 self.menubar.create.add command(label = 'New Project', command = functools. ↵
partial(Form new, self.root, "projects"))
1600 self.menubar.create.add command(label = 'New Activity', command = functools. ↵
partial(Form new, self.root, "activities"))
1601 self.menubar.create.add command(label = 'New Relationship', command = ↵
functools.partial(Form new, self.root, "relationships"))
1602 self.menubar.portfolio.add command(label = 'Show Portfolio', command = ↵
functools.partial(self.create table from sql,"portfolio"))
1603 self.menubar.projects.add command(label = 'Show Projects', command = ↵
functools.partial(self.create table from sql,"projects"))
1604 self.menubar.activities.add command(label = 'Show Activities', command = ↵
functools.partial(self.create table from sql,"activities"))
1605 self.menubar.activities.add separator()
1606 self.menubar.activities.add command(label = 'Show Relationships', command = ↵
functools.partial(self.create table from sql,"relationships"))
1607 self.menubar.calculations.add command(label = 'Calculate', command = ↵
functools.partial(calculate,"normal"))
1608 self.menubar.calculations.add separator()
1609 #- self.menubar.calculations.add command(label = 'Optimize (10 trials)', ↵
command = functools.partial(optimize,10))
1610 #- self.menubar.calculations.add command(label = 'Optimize (20 trials)', ↵
command = functools.partial(optimize,20))
1611 #- self.menubar.calculations.add command(label = 'Optimize (50 trials)', ↵
command = functools.partial(optimize,50))
1612 #- self.menubar.calculations.add command(label = 'Optimize (100 trials)', ↵
command = functools.partial(optimize,100))
1613 self.menubar.calculations.add command(label = 'Optimize', command = optimize)
1614 self.menubar.plot.add command(label = 'Gantt Chart', command = functools. ↵
partial(self.show gantt chart, "normal"))
1615 self.menubar.plot.add command(label = 'Gantt Chart Optimized', command = ↵
functools.partial(self.show gantt chart, 'opt'))
1616 self.menubar.plot.add separator()
1617 self.menubar.plot.add command(label = 'Overdraft', command = functools. ↵
partial(self.show plot, 'overdraft'))
1618 self.menubar.plot.add command(label = 'Overdraft Optimized', command = ↵
functools.partial(self.show plot, 'overdraftopt'))
1619 self.menubar.plot.add command(label = 'Cashflow', command = functools.partial ↵
(self.show plot, 'cashflow'))
1620 self.menubar.plot.add command(label = 'Cashflow Optimized', command = ↵
functools.partial(self.show plot, 'cashflowopt'))
1621 self.menubar.plot.add separator()
1622 self.menubar.plot.add command(label = 'Overdraft Discounted', command = ↵
functools.partial(self.show plot, 'overdraftdisc'))
1623 self.menubar.plot.add command(label = 'Overdraft Discounted Optimized', ↵
command = functools.partial(self.show plot, 'overdraftdiscopt'))
1624 self.menubar.plot.add command(label = 'Cashflow Discounted', command = ↵
functools.partial(self.show plot, 'cashflowdisc'))
1625 self.menubar.plot.add command(label = 'Cashflow Discounted Optimized', ↵
command = functools.partial(self.show plot, 'cashflowdiscopt'))
1626
1627 def init (self):
1628     self.root = tk.Tk()
1629     self.root.minsize(500,500)
1630     self.root.geometry(' 1400x900+200+0')
1631     self.root.title("Portfolio Cash Flow Optimization")
1632     self.initiate toolbar()
1633     self.frame = tk.Frame(self.root, bg = 'lightgrey')
1634     self.frame.pack(fill = 'both', expand = True)
1635     self.root.mainloop()
1636
1637 def time test():
1638     number of cases = 100 # this is the number of cases to try
1639     test numbers = []
1640     n activities = []
1641     n_relationships = []

```

```

1642 n activitiesxrelationships = []
1643 n activitiesprelationships = []
1644 times = []
1645 for test in range(1,number of cases + 1):
1646     new database()
1647     create a portfolio2(3,50,2000) # change this to change the min and max
1648     number of activities
1649     startt = datetime.datetime.now()
1650     calculate('normal')
1651     optimize()
1652     endt = datetime.datetime.now()
1653     time = (endt - startt).total seconds()
1654     activitiesn = conn.execute('SELECT COUNT(*) FROM activities').fetchall()[0][0]
1655     relationshipsn = conn.execute('SELECT COUNT(*) FROM relationships').fetchall
1656     test numbers.append(test)
1657     n activities.append(activitiesn)
1658     n relationships.append(relationshipsn)
1659     n activitiesxrelationships.append(activitiesn * relationshipsn)
1660     n activitiesprelationships.append(activitiesn + relationshipsn)
1661     times.append(time)
1662     filename = 'time test.csv'
1663     with open(filename,'w') as csv file:
1664         csvw = csv.writer(csv file)
1665         csvw.writerow(['Test #', 'Number of activities', 'Number of relationships',
1666             'Number of activities x Number of relationships', 'Number of activities +
1667             Number of relationships', 'Time (secs)'])
1668         for row in zip(test numbers,n activities,n relationships,
1669             n activitiesxrelationships,n activitiesprelationships,times):
1670             csvw.writerow(row)
1671
1672 def sensitivity analysis():
1673     new database()
1674     create a portfolio()
1675     conn.execute('Alter Table activities add column originalduration int(10);')
1676     conn.execute('Update activities set originalduration = duration;')
1677     npvs = []
1678     interests = []
1679     conn.execute('Update activities set duration = originalduration * 10')
1680     interest = 0
1681     while interest <= 0.5:
1682         conn.execute('Update projects set interest = %s;%interest)
1683         calculate('normal')
1684         npvs.append(float(conn.execute('select npv from portfolio;').fetchall()[0][0]))
1685         interests.append(interest*100)
1686         interest += 0.02
1687     plt.plot(interests,npvs, 'o-')
1688     plt.xlabel("Interest %")
1689     plt.ylabel('Net Present Value (NPV) EGP')
1690     plt.title("Interest Rate Sensitivity Analysis")
1691     plt.savefig("interest.pdf")
1692     plt.close()
1693     #~ # ----- Cost
1694     new database()
1695     create a portfolio()
1696     conn.execute('Alter Table activities add column originalcost float(10);')
1697     conn.execute('Update activities set orignalcost = cost;')
1698     costs = []
1699     npvs = []
1700     m = 1
1701     while m <= 2:
1702         conn.execute('Update activities set cost = originalcost * %s'%m)
1703         calculate('normal')
1704         npvs.append(float(conn.execute('select npv from portfolio;').fetchall()[0][0]))
1705         costs.append(conn.execute('select sum(cost) from activities;').fetchall()[0][
1706             0])
1707         m += 0.1
1708     plt.plot(costs,npvs, 'o-')

```

```

1704 plt.xlabel("Cost EGP")
1705 plt.ylabel('Net Present Value (NPV) EGP')
1706 plt.title("Cost Sensitivity Analysis")
1707 plt.savefig("cost.pdf")
1708 plt.close()
1709 # ----- Cost + interest
1710 new database()
1711 create a portfolio()
1712 conn.execute('Alter Table activities add column originalduration int(10);')
1713 conn.execute('Update activities set originalduration = duration;')
1714 conn.execute('Alter Table activities add column originalcost float(10);')
1715 conn.execute('Update activities set originalcost = cost;')
1716 conn.execute('Update activities set duration = originalduration * 2')
1717 m = 1
1718 while m <= 10:
1719     npvs = []
1720     interests = []
1721     costs = []
1722     interest = 0
1723     conn.execute('Update activities set cost = originalcost * %s'%m)
1724     while interest <= 0.5:
1725         conn.execute('Update projects set interest = %s;%interest)
1726         calculate('normal')
1727         npvs.append(float(conn.execute('select npv from portfolio;').fetchall()[0][0]
1728         interests.append(interest*100)
1729         interest += 0.02
1730     label = "Cost Multiplier = " + str(m)
1731     plt.plot(interests,npvs, 'o-', label=label)
1732     m += 1
1733 plt.xlabel("Interest %")
1734 plt.ylabel('Net Present Value (NPV) EGP')
1735 plt.legend()
1736 plt.title("Interest Rate Sensitivity Analysis")
1737 plt.savefig("interestpluscost.pdf")
1738 webbrowser.open("interestpluscost.pdf")
1739 #~ # ----- Cost + interest - percentage
1740 new database()
1741 create a portfolio()
1742 conn.execute('Alter Table activities add column originalduration int(10);')
1743 conn.execute('Update activities set originalduration = duration;')
1744 conn.execute('Alter Table activities add column originalcost float(10);')
1745 conn.execute('Update activities set originalcost = cost;')
1746 conn.execute('Update activities set duration = originalduration * 2')
1747 conn.execute('Update projects set interest = 0;')
1748 calculate('normal')
1749 initial npv = float(conn.execute('select npv from portfolio;').fetchall()[0][0])
1750 m = 1
1751 while m <= 10:
1752     npvs = []
1753     interests = []
1754     costs = []
1755     interest = 0
1756     conn.execute('Update activities set cost = originalcost * %s'%m)
1757     while interest <= 0.5:
1758         conn.execute('Update projects set interest = %s;%interest)
1759         calculate('normal')
1760         npvs.append(float(conn.execute('select npv from portfolio;').fetchall()[0][0]
1761         interests.append(interest*100)
1762         interest += 0.02
1763     label = "Cost Multiplier = " + str(m)
1764     plt.plot(interests,npvs, 'o-', label=label)
1765     m += 1
1766 plt.xlabel("Interest %")
1767 plt.ylabel('Net Present Value (NPV) %')
1768 plt.legend()
1769 plt.title("Interest Rate Sensitivity Analysis")

```



```

1770     plt.savefig("interestpluscostpercent.pdf")
1771     webbrowser.open("interestpluscostpercent.pdf")
1772     plt.close()
1773
1774     # ----- final level -----
1775
1776     start time = datetime.datetime.now()
1777
1778     database file name = 'database.db' # filename used for the database
1779     export folder = './export/'
1780     figure export format = '.pdf'
1781     log file name = 'log.txt'
1782     if os.path.exists(log file name):
1783         os.remove(log file name)
1784     title = 'thesis'
1785     optimization stoppingpercentage = 1.00002
1786     optimization stoppingmaxtrials = 20
1787
1788
1789     conn = sqlite3.connect(database file name)
1790
1791     #~ time test()
1792
1793     #~ for a in range(1,5+1):
1794         #~ title = 'Verification Trial %s - '%a
1795         #~ export folder = './exportverification%/'%a
1796         #~ verificate()
1797
1798     #~ verificate()
1799
1800     #~ export folder = './exportvalidation/'
1801     #~ validate()
1802
1803     Main window()
1804
1805     #~ sensitivity analysis()
1806
1807     #~ new database()
1808     #~ import uptown projects()
1809     #~ calculate("normal")
1810     #~ optimize()
1811     #~ export()
1812
1813
1814     conn.close()
1815
1816     end time = datetime.datetime.now()
1817
1818     log("Start Time was " + str(start time))
1819     log("End Time was " + str(end time))
1820     log("Difference is " + str(end time - start time))
1821

```