An On-Event Based Model for Resource Constrained Scheduling of Aircraft Heavy Maintenance Tasks

by

Elizabeth Croteau

Submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
November 2015

Dedication

To my parents, Jane & Jim, and my boyfriend, Dave, for all their patience, support, humour, and above all – their love

Table of Contents

List of Figures

List of Tables

Abstract

This thesis has resulted from a joint industry project with Jazz Aviation outlines the development of a decision tool to improve the scheduling of aircraft heavy maintenance tasks. Given the high operating costs and strong competition, it is advantageous to minimize the time that an aircraft spends in the maintenance bay, while also maintaining high resource utilization; thus, it is desirable to reliably forecast the length of an aircraft "check" for a given resource allocation. The problem is further complicated by the addition of unexpected (non-routine) tasks during the check, as repairs and replacements discover and generate new tasks to be added to the schedule. Therefore, the main area for improvement lies in developing an efficient sequence for the known tasks while accommodating potential non-routine tasks. A new event-based formulation is developed for this resource constrained project scheduling problem (RCPSP). Several numerical experiments are conducted to compare the performance of the new model against existing ones. Analysis of the results shows the conditions under which the new formulation outperforms existing models. This new formulation is appropriate for large-scale problems with a relatively high number of resources available and relatively low resource demand.

A three-phase approach consisting of running the new event-based model in series to minimize the makespan of the heavy maintenance “check" of an individual aircraft while accommodating industry requirements and computational limitations is then proposed. The approach is used to schedule the heavy maintenance task cards on a Bombardier Q400 aircraft. Prior to developing the main schedule, data regarding prerequisites, task duration, and task resource requirements is run through a pre-processing phase to determine the network structure of the problem, as well as earliest and latest possible start times for each task. Pre-processing decreases overall computation time. In each phase, a schedule is developed using an event-based linear programming model for a given task set. Between phases, data from the preceding phase is used to accommodate precedence relations for the next phase. Phase 1 optimizes the scheduling of the first 90 tasks that open and inspect the aircraft, including area-based block tasks. Phase 2 optimizes a second set of tasks, which includes modification tasks as well as the accommodation of non-routine tasks resulting from phase 1. Phase 3 optimizes all remaining known tasks to close and test the aircraft, as well as non-routine tasks. A post-processing algorithm combines the results from the three phases into a single schedule.

List of Abbreviations Used

RCPSP	Resource Constrained Project Scheduling Problem
PMRP	Project Materials Requirements Planning Problem
TCTP	Time/Cost Tradeoff Problem
PSP	Payment Scheduling Problem
MRCPSP	Multi-mode Resource Constrained Project Scheduling Problem
RCMPSP	Resource Constrained Multi-Project Scheduling Problem
MILP	Mixed Integer Linear Programming
LP	Linear Programming
TNC	Total Network Complexity
NDNC	Non-Dummy Network Complexity
RF	Resource Factor
RC	Resource Constrainedness
DR	Disjunction Ratio
PR	Processing Range
FAC	Forward Activity Count
RAC	Reverse Activity Count
PSPLIB	Project Scheduling Problem Library
NR	Non-Routine Task
SNR	Shadow Non-Routine
M	Mechanical
S	Structural
E	Electrical

Acknowledgements

I am eternally grateful to my supervisors, Dr. Claver Diallo & Dr. Alireza Ghasemi, for their guidance, wisdom, and patience; I know I was not the easiest of graduate students, yet you steered me true and unwavering, and I do not have the words to express my appreciation.

I would like to extend a heartfelt thank you to Dr. Eldon Gunn for teaching me a few of the many tips and tricks of the optimization world, while showing me glimpses of the vast, interesting, beautiful world of applied mathematics. If and when I return to academia, it will be because of your inspiration, and the door you opened.

I am indebted to my friends and colleagues at Jazz Aviation for welcoming me into their organization and teaching me everything I never thought I would know about aircraft maintenance & aviation in general. Special thanks goes to Lisa Vad for her generosity and mentorship, and to Bryan, Scott, Justin, & Mark for all their time spent in Meeting Room Alpha – I'm sure the vitamin D deficiency will fade in time.

My deepest thanks to all my professors for their teachings throughout my time at Dalhousie – especially to Dr. Corinne MacDonald, whose passion for industrial engineering shone through my very first class and is a constant reminder of why I chose this particular path, and to Dr. Guy Kember, whose unbridled enthusiasm for mathematics never ceases to bring a smile to my face.

I would like to express my sincere appreciation to Dr. Josh Leon, for his encouragement, kindness, and support – which allowed me to develop all those people skills I never got to practice in the classroom.

Lastly, to all my friends, roommates, co-conspirators, study buddies, travel companions, and trivia team members who ever listened with feigned interest while I excitedly explained the nuances of mathematical modelling – I owe you all a round.

Chapter 1. Introduction

One of the few universal constants across all types of industry is the concept of logistics; specifically, the scheduling of a variety of items – be they physical tasks, milestone events, or human resources – within a variety of constraints. A well-constructed schedule can be the difference between a profit margin for a company, or a loss; the safe execution of an activity, or an incident; the meeting of an important deadline, or a costly delay. The scheduling of sequences of tasks according to prerequisite requirements and within resource constraints is known as the Resource Constrained Project Scheduling Problem (RCPSP).

RCPSP is the general classification for problems which model precedence-constrained scheduling scenarios that are subject to resource constraints. The goal of such scheduling problems is usually to minimize the total time taken to complete the project. The resource constraints can be limited quantities of material, time available on machines, or number of employees. RCPSPs are inherently valuable in real-world applications; there are a wide variety of relevant uses, from automotive production to construction management to computer memory allocation. In this thesis, an RCPSP is used to model the optimal scheduling of aircraft maintenance at Jazz Aviation in Halifax, Nova Scotia.

Within the aviation industry, the scheduling of when to perform aircraft maintenance is a remarkably trivial concern; maintenance safety margins are determined well in advance by numerous transportation safety authorities, and the window in which an aircraft must have maintenance performed along with the type of maintenance required is strictly defined. In comparison, the order in which required maintenance tasks must be completed is relatively flexible. While a preliminary schedule for these maintenance tasks is developed by the aircraft manufacturer, these preliminary schedules are based on the capacity of the aircraft manufacturing facility to perform maintenance; as a result, they are based on an ideal scenario where only one aircraft is undergoing maintenance by a consistent team of employees.

The reality of aircraft maintenance environments usually includes shift-work schedules with changing resource profiles and multiple aircraft in a hangar at any given time. Jazz Aviation's Heavy Maintenance facility in Halifax, Nova Scotia is such a real-world facility, requiring an automated scheduling tool to assist with optimal sequencing of aircraft maintenance tasks. This

thesis presents the development of an RCPSP model and the theoretical applications of this model, as well as the practical application and implementation at Jazz Aviation.

1.1 Jazz Aviation & Maintenance Scheduling

Jazz Aviation is one of the few companies in North America to perform their own heavy maintenance in-house, and the only company in North America to do so with a variety of aircraft. Heavy maintenance checks are scheduled up to 8 months in advance, in accordance with aircraft safety regulations; the list of tasks to be performed during the maintenance checks is also determined in advance. This list of tasks is forwarded electronically to Production Control to be scheduled. The scheduling of the sequence of these tasks is done manually, though they are tracked though the company database.

The scheduling of one aircraft check currently happens in two phases: the full schedule creation and the day-to-day scheduling. The full schedule creation occurs in the weeks before an aircraft is due to enter the hangar. The current advance scheduling process can be seen in Figure 1.

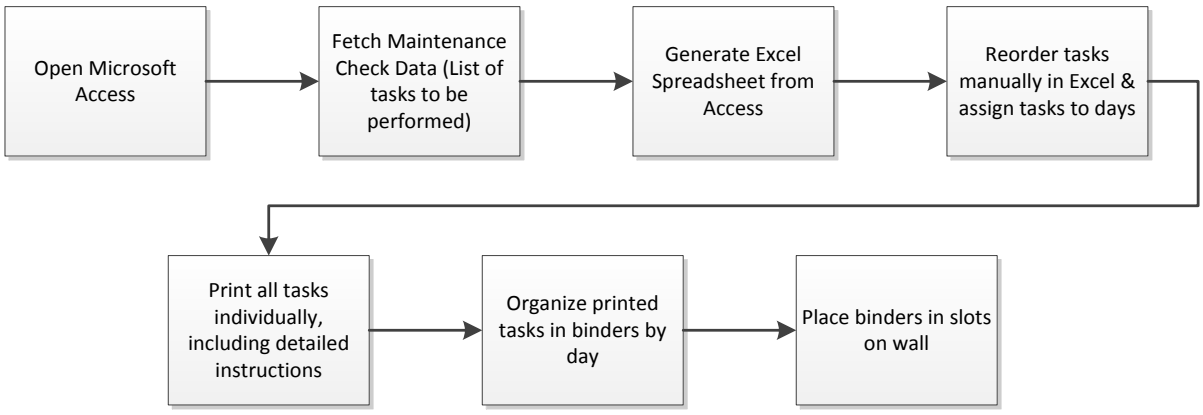

Figure 1 – Current Advanced Scheduling Process

The day to day scheduling happens every morning and during the day as unexpected tasks, referred to as “non-routines”, are generated and new information is obtained. The day to day scheduling process is:

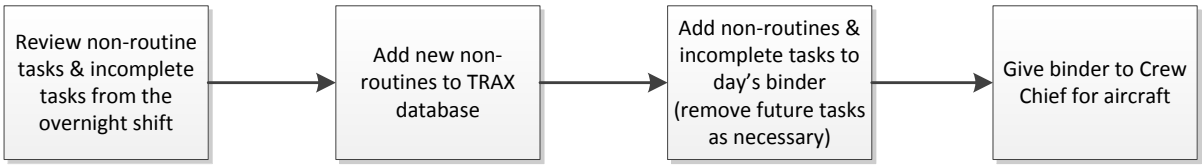

Figure 2 – Current Day-to-Day Scheduling Process

1.2 Problem Structure

The basic problem is a *resource-constrained project scheduling problem*, which will be expounded upon in the literature review. Jazz Aviation has a specific goal in mind: namely, developing a scheduling tool for their Heavy Maintenance division in Halifax, Nova Scotia. A scheduling model is needed to order tasks in a way that efficiently utilizes the resources available, including manpower, equipment, and aircraft space available at the Jazz Heavy Maintenance hangar. Such a model will have to account for routine tasks, non-routine tasks, and aircraft configuration changes. In addition, the scheduling tool resulting from this model must be implementable in Jazz's IT infrastructure at a realistic cost level.

The following sections detail the specific aspects of the Jazz problem that must be accommodated by the model to ensure an accurate and relevant schedule.

1.2.1 Resources

The resources required for each task include the employees, which encompasses both the type of employee (electrical, mechanical, or structural) and the number of employees; the location on an aircraft (as each location has a limited amount of space for an employee to perform a task); and the equipment required (which is restricted to large equipment that must be shared). Items like tools are assumed to be assigned with the employee – that is, each employee will bring all tools necessary to do the task. Items such as parts and their availability will not be integrated into the model at this time; however, they may be integrated into the model in the future, and the model should be flexible enough to accommodate this possibility.

1.2.2 Routine Tasks

In Heavy Maintenance, there are numerous "routine" tasks to schedule, which are defined by the aircraft manufacturer before the plane has even been purchased by an airline. These tasks are listed as individual "task cards", and stored both as physical hard

copy at the Heavy Maintenance location at Halifax airport, and also as digital soft copy on Jazz Aviation's database servers. Many of these tasks have precedence constraints and cannot be started until the precedence tasks have been completed; this data is usually included on the "task card", but is not listed elsewhere – in other words, precedence constraints cannot be known before the task card is accessed and the task already assigned. The resources required for each task, and the duration of each task, are also listed on these task cards; as a result, resources required and activity durations are considered deterministic rather than stochastic.

1.2.3 Non-Routine Tasks

In addition to "routine" tasks, there are also "non-routine" tasks that are created as the aircraft in question is undergoing maintenance. Routine tasks such as inspections will generate "findings" – essentially, an issue that must be fixed before the aircraft can be released from the hangar. Each finding generates a unique "non-routine" task to address this finding and correct the defect that has been found. This thesis must address these unexpected tasks, and incorporate them into the scheduling algorithm.

1.2.4 Multiple Aircraft

There are multiple aircraft in the hangar at any given time, each with their own unique set of routine tasks, which will generate a unique set of non-routine tasks. Ideally, a scheduling model will be required to not only schedule an individual aircraft, but up to four aircraft concurrently sharing many of the resources (notably employees and equipment). In this thesis, a model is presented for scheduling an individual aircraft only; however, recommendations are made for future enhancement using heuristics to decide optimal assignment of employees and equipment to aircraft.

1.2.5 Aircraft Configuration

The aircraft configuration refers to the state of the aircraft in three areas: power, ground, and hydraulics. Tasks exist to turn power on and off at the aircraft, lift the aircraft on and off the ground, and apply hydraulic power to the aircraft. These configuration changes must be accommodated by the model.

Chapter 2. Literature Review

This section provides an overview of the Resource Constrained Project Scheduling Problem, including relevant applications, common formulations, typical solving methods, and relevance to the Jazz Aviation problem.

2.1 Overview of the Resource Constrained Project Scheduling Problem

The resource constrained project scheduling problem (RCPSP) is one of the most practical mathematical modeling problems in academia today. RCPSP is a generalized form of the classic job shop problem, and is NP-hard. While there are many variations of the RCPSP, the problem is aptly described as "…the scheduling of project activities subject to resource and precedence constraints, under the objective of minimizing the project makespan" [1]. Under such a definition, where "makespan" is the total time for a project, it is easy to see why the RCPSP problem is highly applicable; a plethora of real-world problems from manufacturing to production to computer scheduling can be modeled using RCPSP. As a result, there are many different types of RCPSP models. The purpose of this literature review is to provide an overview of the existing model types and solving methods for RCPSP in the literature, evaluate these models and methods, and identify which models and methods may be applicable to a project scheduling problem at the Heavy Maintenance division of Jazz Aviation in Halifax, Nova Scotia.

2.1.1 RCPSP Classification

The Resource Constrained Project Scheduling Problem can be defined by both the model type of RCPSP, which refers to the specific constraints and formulation of the RCPSP model, and the solving method of RCPSP, which refers to the method used to solve the RCPSP. While the solving type is typically well defined – such as linear programming, constraint programming, or heuristic methods – the model types are inconsistently classified, and the nomenclature is not universally applied. The majority of the classifications used in this document have been taken from Brucker et al's 1999 paper on notation and classifications of the RCPSP [2]. This taxonomy is mathematically robust for the project management sphere of problems. It should be noted that this classification scheme has been criticized for being inadequate for machine and job-shop scheduling [3]; however, as this paper addresses a project management problem, Brucker et al's taxonomy is taken as adequate.

2.1.2 RCPSP Applications

The potential applications for RCPSP models are numerous and significant. Any scheduling problem with prerequisite requirements and limited resources – which is essentially any scheduling problem of note in modern industry – is an RCPSP and can benefit from RCPSP modelling. Specific applications include maintenance [4]; production lines [5]; computer memory allocation [6]; construction [7]; and equipment installation [8].

RCPSPs can also be integrated with other types of problems, such as project material requirements planning problems (PMRP) [9]; lot sizing problems [10]; and time/cost trade off problems (TCTP) and payment scheduling problems (PSP) [11]. Such integrations tend to focus on new metrics for RCPSP application other than merely makespan minimization [12].

2.2 RCPSP Model Types

Within the literature, there are multiple types of RCPSP models. As per Brucker et al [2], common across all models is a set of activities, V, with n real activities indexed 1..n. dummy activities 0 and n+1 index the starting event and terminating event, respectively. Each activity has a duration denoted by p_j. These activities are constrained by a set of precedence or temporal constraints, E, as indicated by a directed graph $G = (V, E)$. Precedence constraints are binary pairs where (i,j) indicates that activity i is a predecessor of activity j. Temporal constraints are given by minimum and maximum start time lags between activities, where S_j is the start time of activity j, and if activity i is a predecessor for activity j, then $S_j - S_i \geq p_j$. Alternatively, $dmin_{ij}$ and $dmax_{ij}$ may indicate the minimum and maximum time lag between the start times of activities i and j, where $dmin_{ij}$ is greater than or equal to p_j. The time horizon is given by T, usually divided into t periods.

Outside of the basics, models begin to diverge in terms of what is considered and how the formulation is developed. While the vast majority focus on the objective of minimizing the makespan of a particular schedule, some variations on the RCPSP focus on minimizing cost or even multi-objective models [7]. More commonly, the constraints and set up of the model are altered to accommodate variations instead. This section will present a brief overview comparing some of the typical types of RCPSPs.

2.2.1 Multi- vs. Single-Mode

The most common distinction is that of multi- vs. single-mode applications. Multi-mode scheduling, referred to as MPS [2] or MRCPSP [13], corresponds to an option for different lengths of time for a job depending on how many resources are allocated to it – for example, a wall can either be built in three days by one worker, or in two days by two workers [1]. Single mode implies that the number of hours is constant related to the number of resources – in other words, the wall can only be built in a given number of days by a given number of workers, and the model will not consider other possibilities. Multi-mode is typically used in machine job-shop scheduling, as different machines make take different amounts of time to do the same task. Single-mode is typical when the resources in question are employees. Many MRCPSPs focus on selecting a single mode for each task, and then solving the RCPSP afterward [14].

2.2.2 Time Windows

Another variation between RCPSP models is the accommodation of time windows, which limit the possible start and end times of each task to a predefined period. Such formulations are referred to as PS/temp [2] or RCPSP/max [15]. The PS/temp notations indicates the use of temporal constraints as mentioned above – that is, the time in which an activity must occur is formulated in relation to other activities. In comparison, the RCPSP/max model gives times when an activity must occur in relation to the time horizon, and is more useful for modelling a delivery window or a limited availability resource. RCPSP/max problems can be combined with other model types, such as the multi-model problem, to create MRCPSP/max [16].

2.2.3 Stochastic Activity Times

The activity duration vector p_j may also consist of stochastic activity times instead of deterministic activity times, notated as $p_j = sto$ [2]. Whether stochastic (or "fuzzy") activity times should be accommodated or converted to deterministic activity time vectors is the source of some debate, though guidelines exist for choosing when to work with stochastic measures [17]. Most literature deals with deterministic activity times or "defuzzifying" activity times so they may be treated as deterministic [18]. Such proactive policies are usually relevant when the network structure is known, and the activity durations can be modeled accurately; these policies usually result in an expected end

time and a minimized maximum makespan, rather than a minimum expected makespan [19]. However, it is possible to actively accommodate uncertainty for certain types of scheduling with limited data using possibilistic models, though this typically results in a scheduling decision tool for short-term schedules rather than long term plans [20].

2.2.4 Renewable vs. Non-Renewable Resources

In basic RCPSP models, resources are considered renewable. These resources are available at a pre-determined time level for each time period *t* [2]. In comparison, some formulations include non-renewable resources that are "used up" during the schedule. This can be due to these resources being consumed during set-up stages of a schedule [21], or due to their being a limited amount of resources available over the entire planning horizon [14]. The problem as a whole may be notated as RCPSP/CPR [22].

2.2.5 Cyclic vs. Linear

While the majority of the literature focuses on linear RCPSP, where there is a clear start and end activities denoted as activities "0" and "n+1", some research explores cyclic RCPSP, where the problem continues infinitely. Common applications of cyclic RCPSP include data-flow computations and software pipelining [23] and cyclic production shop scenarios which are mass producing an item [6]. Cyclic RCPSPs can also be used for continuous maintenance scheduling where there is prerequisite that some sort of central or core precedence – such as a safety inspection requirement – must be met by a certain point in the cycle [24]. It is typical to look at cyclic problems from the angle of maximizing throughput rather than minimizing makespan, though this is not always the case [25].

2.2.6 Pre-emptive vs. Non-pre-emptive

Non-pre-emptive models will not allow for an activity to be interrupted; once the activity is scheduled, the entire duration of the activity must pass consecutively [8]. In comparison, pre-emptive RCPSPs allow for activities to be interrupted so the resources may be used elsewhere, and then the activity may resume after a period of time. Models which allow pre-emption may either be modelled such that pre-emption is not disallowed [26], or modelled by actively splitting the activities into smaller sections [8].

2.2.7 Multi-Project Management & Support

Lastly, RCPSPs can be used to model management of multiple projects, usually referred to as the Resource Constrained Multi-Project Scheduling Problem (RCMPSP) [27]. An example of this is for resource allocation across projects, such as line balancing across production or maintenance lines; this can range from simple theoretical line balancing [5] to more complex real-world applications in automotive manufacturing [28]. RCPSPs can also be used to address the resource dedication problem (RDP) in multi-project environments [29]. Another use in project management is modelling the potential disruption of one project for the accommodation of another more urgent project, referred to as the extended RCPSP or x-RCPSP [30]. Such x-RCPSPs may utilize priority rules; there has been discussion on which priority rules to use for which project types [31].

2.3 RCPSP Solving Methods

The solving methods for RCPSP fall into the same categories as the majority of operations research problems – exact methods using linear programming, constraint programming and satisfiability testing, and heuristic/metaheuristic methods.

2.3.1 Exact Methods (Linear Programming)

Throughout the 1970s and 1980s, much research focused on the development of MILP models and the application of exact solving methods to scheduling problems [32]. As exact methods have been shown to solve only small sets of tasks to optimality [13], the focus has been not on Linear Programming methods as a whole, but on specific LP formulations and selecting the correct formulation based on the problem at hand. Recent work has demonstrated that"…when exact solving through a commercial solver is involved, no formulation class dominates the other ones, and that the appropriate formulation has to be selected depending on instance characteristics" [33]. In general, Linear Programming models are considered the starting point of model development; however, the limitations of exact solving methods has meant the research focus of the 1990's through into the new century has been the development of heuristics to improve solving time for large scale problems [27].

2.3.2 Heuristic and Metaheuristic Methods

Since the 1990's, the majority of the literature has been focused on heuristic and metaheuristic methods (see [13], [34], [35] [15], [1]). Priority rules based heuristics such as X-pass, single pass, and multi-pass methods [34] are the basis for many heuristic and metaheuristic methods in the literature; metaheuristic methods include simulated annealing, tabu search, and genetic algorithms [34], and ant colony optimization and backward-forward hybrid algorithms [36].

The popularity of heuristics is mainly due to the computational limitations on more exact solving methods; however, heuristics are also used to more accurately model real-world scenarios. For example, it is possible to accommodate stochastic RCPSPs by first developing a schedule based on deterministic data, and then add buffers via a heuristic [37]. Another combination-type application is utilizing a powerful meta-heuristic such as the Artificial Bee Colony to maximize the robustness and flexibility of a model [38].

2.3.3 Constraint Programming and Satisfiability Testing

Relatively new additions to the RCPSP solver field are constraint programming and satisfiability testing, which may be combined with each other and with heuristic or mathematical programming methods [39]. Constraint programming is a very flexible programming language, usually reliant on logic programming in commercially available solvers using constraints to define problems that may be solved very efficiently [40], [41]. Satisfiability testing focuses on using Boolean statements to solve problems [42]. Constraint programming approaches convert typical RCPSP MILP models into relevant sub-constraints that can be solved as sub-problems and slotted into the main problem appropriately [43]. Satisfiability testing approaches typically capitalize on artificial neural network techniques to solve RCPSPs that have been reduced to conjunctive normal form formulations of combinatorial constraints [42]. As these two topics are complex and diverse in their own right, the focus of this literature review is on heuristic and linear programming methods for solving RCPSP.

2.4 MILP Formulations

There are three primary LP formulations: discrete time, continuous flow, and event based, outlined comprehensively in the 2011 paper by Kone, Artigues, Lopez & Mongeau, hereafter referred to as Kone et al [33]. These LP Formulations are outlined in this section and compared.

2.4.1 Formulation Notation Definitions

Constants:

n	Number of activities to be scheduled, including dummies
H	Planning Horizon

Sets:

IP	Set of precedence index pairs; (i,j) in IP means activity i must precede activity j
R	Set of resource types; denoted by r
A	Set of activities; denoted by a; default 1..n where 1 is dummy start & n is dummy end
E	Set of events; denoted by e

Parameters:

p_a	Parameter for the duration of activity a
b_r	Parameter for the budget; amount of resource r available
$d_{a,r}$	Parameter for resource demand; amount of resource r required by activity a
ES_a	Parameter for the earliest possible start time of activity a
LS_a	Parameter for the latest possible start time of activity a

2.4.2 Discrete Time Formulation

The basic discrete time formulation consists of one binary decision variable x_{it}, which is equal to 1 if activity i starts at time t. Modifications include if the activity i is in process at time t. Discrete time formulations are more efficient for smaller scale problems, and are much simpler to model; however, as they require time indexes for each scheduled unit of time, they can grow very large very quickly, especially with large time horizons and great variation in activity durations. For example, if one task takes 13 minutes, then

a time interval of one minute is logical; however, if another task takes 7 hours, this will require 420 decision variables to accommodate. As a result, discrete time formulations are recommended for smaller-scale problems, or problems with shorter time horizons. The basic discrete time formulation is as follows:

Variable:

$x_{a,t}$ Binary; if activity *a* starts at time *t*

Objective:

$$\min \sum_{t=0}^{LS_n} t x_{n,t}$$

Subject to:

$$\sum_{t=ES_j}^{LS_j} t x_{jt} \geq \sum_{t=ES_i}^{LS_i} t x_{it} + p_i \qquad \forall (i,j) \in IP$$

$$\sum_{a=1}^{n} d_{a,r} \sum_{\tau=t-p_a+1}^{t} x_{a,\tau} \leq b_r \qquad \forall t \in H, \qquad \forall r \in R$$

$$\sum_{t=ES_a}^{LS_a} x_{a,t} = 1 \qquad \forall a \in A, a \neq n$$

$$x_{1,0} = 1$$

$$x_{a,t} = 0 \qquad \forall a \in A, a \neq n,$$

$$\forall t \in H, H < ES_a, H > LS_a$$

$$x_{a,t} \in \{0,1\} \qquad \forall a \in A, a \neq n, \qquad \forall t \in \{ES_a, LS_a\}$$

2.4.3 Flow-based Continuous Time Formulation

The flow-based continuous time formulation consists of three variables: continuous variable S_i indicating the start time of activity *i*, binary variable x_{ij} to indicate if activity *i* is

processed before activity *j*, and continuous variable f_{ijk} which is equal to the quantity of resource *k* that is transferred from activity *i* to activity *j*. Flow-based continuous time formulations are most logical in machine shop environments, where the resource is being handed off to the next activity and may process many activities at once. In a project scheduling environment, most resources can handle being assigned to at most one activity – employees are not seen to be concurrently working on multiple tasks equally. As a result, this formulation is less appropriate to project scheduling. The flow-based continuous time formulation is below.

Additional Parameters:

M_{ij} Large constant for (*i,j*) in *A*; equal to $ES_i - LS_j$

g_{ar} Amount of resource *r* being consumed per time period during the execution of activity *a*

$\check{g}_{ar}$ Indicates if resource is a source or sink; equals g_{ar} except where a=0 and a=*n*; in these cases, $\check{g}_{0r} = \check{g}_{nr} = b_r$

Variables:

s_a *C*ontinuous; start time of activity *a*

$y_{i,j}$ Sequential binary; if activity *i* is processed before activity *a*

$f_{i,j,r}$ *C*ontinuous; amount of *r* transferred from activity *i* to activity *j*

Objective:

$$\min s_n$$

Subject To:

$$y_{i,j} + y_{j,i} \leq 1 \qquad \forall (i,j) \in A^2, i < j$$

$$y_{i,k} \geq y_{i,j} + y_{j,k} - 1 \qquad \forall (i,j,k) \in A^3$$

$$s_j - s_i \geq -M_{i,j} + \left(p_i + M_{i,j}\right) * y_{i,j} \qquad \forall (i,j) \in A^2$$

$$f_{i,j,r} \leq \min(ğ_{ir}, ğ_{jr})\, y_{ij} \qquad \forall (i,j) \in A^2,\ \forall r \in R$$

$$\sum_{j \in A} f_{ijr} = ğ_{ir} \qquad \forall i \in A,\ \forall r \in R$$

$$\sum_{i \in A} f_{ijr} = ğ_{jr} \qquad \forall j \in A,\ \forall r \in R$$

$$f_{n,0,r} = b_r \qquad \forall r \in R$$

$$y_{ij} = 1 \qquad \forall (i,j) \in IP$$

$$y_{ji} = 0 \qquad \forall (i,j) \notin IP$$

$$f_{ijr} \geq 0 \qquad \forall (i,j) \in A^2, \forall r \in R$$

$$S_0 = 0$$

$$ES_a \leq S_a \leq LS_a \qquad \forall a \in E$$

$$y_{ij} \in \{0,1\}$$

2.4.4 Event Based Formulation

The event based formulations can be either start/end based or on/off based, and are focused on defining events in relation to the beginning of activities. The first type, the start/end formulation, was originally proposed by Zapata et al., [44], and defines an event a moment when an activity starts or ends. A binary x_{ie} variable determines if an activity starts at event e, and a second binary variable y_{ie} determines if an activity ends at event e. A continuous time variable t_e determines the time of event e. Start/End formulations are shown to be less intuitive, however, and perform poorly in relation to on/off event based formulations [33].

The on/off based formulation has been shown to be more robust than the start/end formulation, as it consists of one binary decision variable z_{ie}, which is equal to 1 if activity *i* starts or is being processed immediately after event *e*, along with the continuous variable *t* which represents the start time of event *e*. In this formulation, the number of events is exactly equal to the number of activities. It is indicated in the literature that the on/off event model involves data preprocessing; however, this is not detailed in the Koné et al [33].

2.5 MILP Formulation Metrics

Koné et al [33] summarizes the comparison of these models after running tests on PSPLIB instances. Some of the instance indicators used include Network Complexity (denoted TNC for "Total Network Complexity" in this thesis), Resource Factor (RF), Disjunction Ratio (DR) and Processing Range (PR) (ibid). Other metrics used in this paper include Resource Constrainedness (RC) [45] and Non-Dummy Network Complexity (NDNC), introduced in this thesis. These metrics may be divided into three groups: network indicators, resource indicators, and instance indicators.

2.5.1 Network Indicators

Network indicators are based on the given network architecture, and describe that architecture through a ratio of arcs to nodes. In this thesis network indicators are TNC and NDNC. TNC is defined as the average number of precedence arcs per activity; generally, higher network complexity decreases the difficulty of the problem [33]. NDNC is defined as the average number of non-dummy precedence arcs per non-dummy activity; this eliminates arcs between real activities and a dummy start or dummy end activity. Previous work has not clearly distinguished between these two metrics, when it is evident that the two may present very different values.

Take as an example an extreme network with n activities and n-1 total connections, resulting in a serial configuration of activities with no decisions to be made regarding scheduling. In this case, only two arcs are dummies – the first arc connecting the dummy start to the first real activity, and the last arc connecting the last real activity to the dummy end. Thus, the difference between the TNC of (n-1)/n, and the NDNC of (n-3)/(n-2), is negligible when n is large. However, the other extreme example is a network with no prerequisites between real activities – that is, zero non-dummy arcs. The NDNC of such a network is 0/(n-2), or 0; however, the TNC of such a network is (2*(n-2))/n, as there must be an arc from the dummy start to every activity, and every activity to the dummy end. The difference in this case is significant; as a result, NDNC is introduced in this thesis to clarify the difference between these two indicators.

2.5.2 Resource Indicators

Resource indicators are based on the resource demands and availabilities of a given instance. RF is the average number of resources required by all activities, and is

therefore defined by the activities themselves via parameter $d_{a,r}$. In general, the higher resource factor, the greater the difficulty of the problem [33]. RC is defined as the RF for a given resource, divided by the resource budget [45]. As a result, RC is defined by both parameters $d_{a,r}$ and the available resource budget b_a.

2.5.3 Instance Indicators

Instance indicators combine network and resource indicators to give a general impression of the entire instance. DR integrates precedence and resource features and defines cumulative instances as having a low disjunction ratio, while disjunctive instances have a high disjunction ratio [33]. Cumulative instances are generally network-budget combinations where it is possible to have many activities in process simultaneously, while the solutions to disjunctive instances have very few activities in process simultaneously. PR is defined as the ratio of the longest possible processing time to the shortest possible processing time, with a high PR indicating a wide range of possible durations, and a low PR indicating a small range of possible durations.

Koné et al's 2011 paper indicates there are general trends comparing different MILP formulations across the instance indicators. These are summarized below [33].

Table 1 – Synthesis of Experiments from Kone et al 2011

	High DR	**Low DR**
Low PR	Discrete > Flow > Event	Discrete > Event > Flow
High PR	Flow > Event > Discrete	Event > Flow > Discrete

2.5.4 Indicator Evaluation of Jazz Aviation

While pure data from Jazz Aviation will be evaluated at a later section in this thesis, the general particulars of the Heavy Maintenance Scheduling Problem are known. As a result, it is possible to evaluate qualitatively the likely instance values.

Network Complexity and Non-Dummy Network Complexity are likely to be high since most tasks rely on the completion of previous tasks – for example, a panel must be opened before a part is inspected; the part must be inspected before the repair is completed; the repair must be completed before the safety check performed; and the safety check performed before the panel is closed.

Resource Factor is likely to be low; most maintenance tasks are performed by one resource only due to limited space on the aircraft. Resource constrained-ness is also likely to be low, as there are between eight and fifteen employees assigned to each aircraft.

Disjunction ratio is likely to be low, as the number of activities that could be scheduled at any given moment is relatively high and the problem in general is highly cumulative. As task durations can range from 5 minutes to over 8 hours, the processing range will be very high.

2.6 Conclusion of Literature Review

Jazz Aviation's scheduling problem is a linear, single-mode project scheduling problem with deterministic activity times. As a result, existing MILP formulations are appropriate for modeling the problem, though they will likely require heuristics to improve solving time due to the size of the problem – Jazz Aviation will have an average of 250 routine tasks to be scheduled, as well as up to 500 non-routine tasks which must be accounted for as well. As a result, selecting a formulation and solving method that will accommodate such a large problem is of utmost importance.

Ideally, a scheduling tool that solves the Jazz problem would allow for some sort of balancing across the multiple aircraft which are pulling from a pooled set of resources – the employees and equipment. However, to the author's knowledge, such a problem type is not reflected in the literature. This lack of depth of research combined with processing time restrictions leads the author to recommend that the scheduling tool focus on scheduling one aircraft at a time, with the number of employees assigned to a given aircraft being determined by production control personnel. This will also allow production control to retain discretion in scheduling, which will be invaluable for change management during the implementation phase of the project. Further development of the model to accommodate multiple projects with pooled resources will be recommended for future study.

While Koné et al (2011) recognizes that discrete time formulations are beneficial for low processing time range problems, high processing time range problems are best served by both flow based continuous time formulations and the on/off event based model. Flow-based formulations are most appropriate when there are highly valuable resources that serve as

bottlenecks combined with straightforward precedence relations (such as machines in job-shop problems); in other terms, when instances are highly disjunctive with a high DR.

The final instance indicator set, with a high PR and low DR, is likely best served by an event-based model. As the Jazz Aviation problem definitely has a high PR and will likely have a low DR, it may be concluded that the initial results – though presented with the caveat of caution – indicate that an event type model should be preferred for the Jazz Aviation problem.

Chapter 3. Event-based Scheduling Models

An on/off event based model was selected as the base model that would be modified to improve performance for large-scale problems such as the one at Jazz Aviation. Event-based models have shown promise in effectively modelling large-scale problems with long time horizons and many tasks. This section will present the formulation of the On/Off Event formulation proposed by Koné et al, and identify shortcomings in their formulation. This thesis will propose a new On-Only Event formulation, developed jointly with Eldon Gunn, to address these shortcomings. Trade-off comparisons between both models are then conducted and the results discussed.

3.1 Event Formulation Notation

The Event Formulations use the same notation as outlined in section 2.4.1, with the addition of the following variables:

Variables

t_e Continuous variable, the time of event *e*

$z_{a,e}$ Binary variable, if event *e* is the start of activity *a*

C_{max} Continuous variable, equal to the makespan

3.2 On/Off Event Formulation

The on/off event formulation defines the z variable as equal to 1 if a given activity *a* is in process at or immediately after event *e*; as a result, for a given activity, there are multiple values of *e* for which $z_{a,e}$ can equal 1. It also means an event occurs at the start and end of each activity; however, if the start of an activity coincides with the end of another activity, there is only one event rather than two. The resultant number of non-zero *z* values is at least equal to *n*. A visual representation of this definition is presented below, as well as the mathematical formulation, and an overview of the limitations of this formulation.

3.2.1 On/Off Event Formulation Visualization

Figure 3 shows the Gantt diagram of the solution to an example problem with 5 activities, and the resulting event times using the On/Off Event formulation.

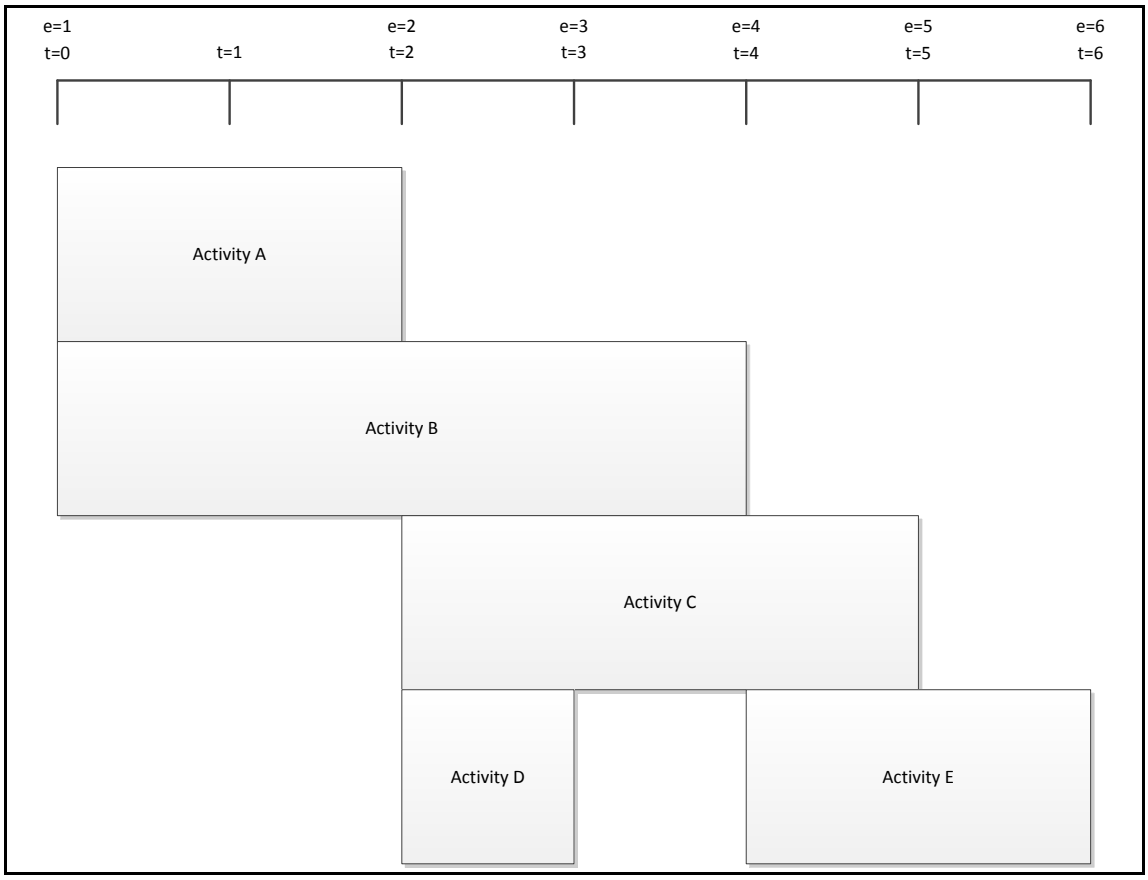

Figure 3 – On/Off Event Formulation Diagram

In this example, the total number of activities is 5; the number of events is 6; and the number of non-zero *z* values ($z_{a,e}$ = 1) would be 10. Relevant *z* values are shown in Table 2 – On/Off Event Formulation Example Z-Variable Values.

Table 2 – On/Off Event Formulation Example Z-Variable Values

Activity *a*	Event *e*	Value
A	1	1
A	2	0
B	1	1
B	2	1
B	3	1
B	4	0
C	2	1
C	3	1
C	4	1
C	5	0
D	2	1

Activity *a*	Event *e*	Value
D	3	0
E	4	1
E	5	1
E	6	0

3.2.2 On/Off Objective & Constraints

Objective

$$Minimize\ C_{max}$$

Subject To

$\sum_{e \in E} z_{a,e} \geq 1$ $\forall a \in A$
(1)

$C_{max} \geq t_e + \left(z_{a,e} - z_{a,e-1}\right) * p_a$ $\forall e \neq 1 \in E, \forall a \in A$
(2)

$t_1 = 0$
(3)

$t_{e+1} \geq t_e$ $\forall e \neq n \in E$
(4)

$t_{e_2} \geq t_{e_1} + \left(\left(z_{a,e_1} - z_{a,e_1-1}\right) - \left(z_{a,e_2} - z_{a,e_2-1}\right) - 1\right) * p_a$

(5) $\forall (e_1, e_2, a) \in \{E \times E \times A\}, e_2 > e_1$

$\sum_{f=1}^{e-1} z_{a,f} \leq e * \left(1 - \left(z_{a,e} - z_{a,e-1}\right)\right)$ $\forall e \neq 1 \in E$
(6)

$\sum_{f=e}^{n} z_{a,f} \leq (n - e) * \left(1 + \left(z_{a,e} - z_{a,e-1}\right)\right)$ $\forall e \neq 1 \in E$
(7)

$z_{i,e} + \sum_{f=1}^{e} z_{j,f} \leq 1 + \left(1 - z_{i,e}\right) * e$ $\forall e \in E, \forall (i,j) \in IP$
(8)

$\sum_{a=1}^{n} d_{a,r} * z_{a,e} \leq b_r$ $\forall e \in E, \forall r \in R$
(9)

$ES_a * z_{a,e} \leq t_e$ $\forall e \in E, \forall a \in A$
(10)

$t_e \leq LS_a * \left(z_{a,e} - z_{a,e-1}\right) + H * \left(1 - z_{a,e} + z_{a,e-1}\right)$ $\forall e \in E, \forall a \in A$
(11)

$$z_{a,e} \in \{0,1\} \qquad \forall a \in A, e \in E$$

(12)

$$t_e \geq 0 \qquad \forall e \in E$$

(13)

**Note: the original model includes an unnecessary constraint enforcing earliest & latest start times that is not required when constraints 10 & 11 are modelled separately.*

3.2.3 On/Off Constraint Details

The constraints are described as follows:

(1) Assigns each activity at least one event, ensuring it is scheduled

(2) Defines the makespan, $C_{max,}$ as the end of the last activity

(3) Initializes the first event to the start time of zero

(4) Ensures events happen in order

(5) Links the binary *z* variable to the corresponding *t* variable

(6) Ensures non-pre-emption from previous events

(7) Ensures non-pre-emption for future events

(8) Enforces precedence constraints

(9) Enforces resource capacity constraint

(10) Enforces earliest start times

(11) Enforces latest start times

(12) Binary z definition

(13) Continuous non-negative t definition

3.2.4 On/Off Formulation Limitations

The on/off formulation is not an intuitive one; the definition tying events to activities means that a given activity may have more than one event for which $z_{a,e}$ = 1. While this simplifies the resource capacity constraints, as it is allows for summing the activities that are on at a given event to determine resource demand, it also convolutes the prerequisite constraints and necessitates the non-pre-emption constraints. Most importantly, it makes it difficult to reduce the solution space by limiting which activities may be assigned to which events; such limitations are not obvious when each activity is assigned to multiple events.

3.3 On-Only Event Formulation

The on-only event formulation serves to rectify the limitations identified in the on/off formulation by making the formulation more intuitive and easier to manipulate. The on-only event formulation changes the definition of the *z* variable, so that each activity is assigned to one event, and one event only; at this event, the activity starts. As a result, the total number of non-zero *z* variables will always be equal to the number of activities, *n*. This definition means it is possible to capitalize on the underlying structure of the prerequisite network to limit which events may be assigned to the given activity, based on where the activity sits in the prerequisite network.

3.3.1 On-Only Event Formulation Visualization

The following diagram visualizes the variable values for the same final schedule as in example 3.2.1.

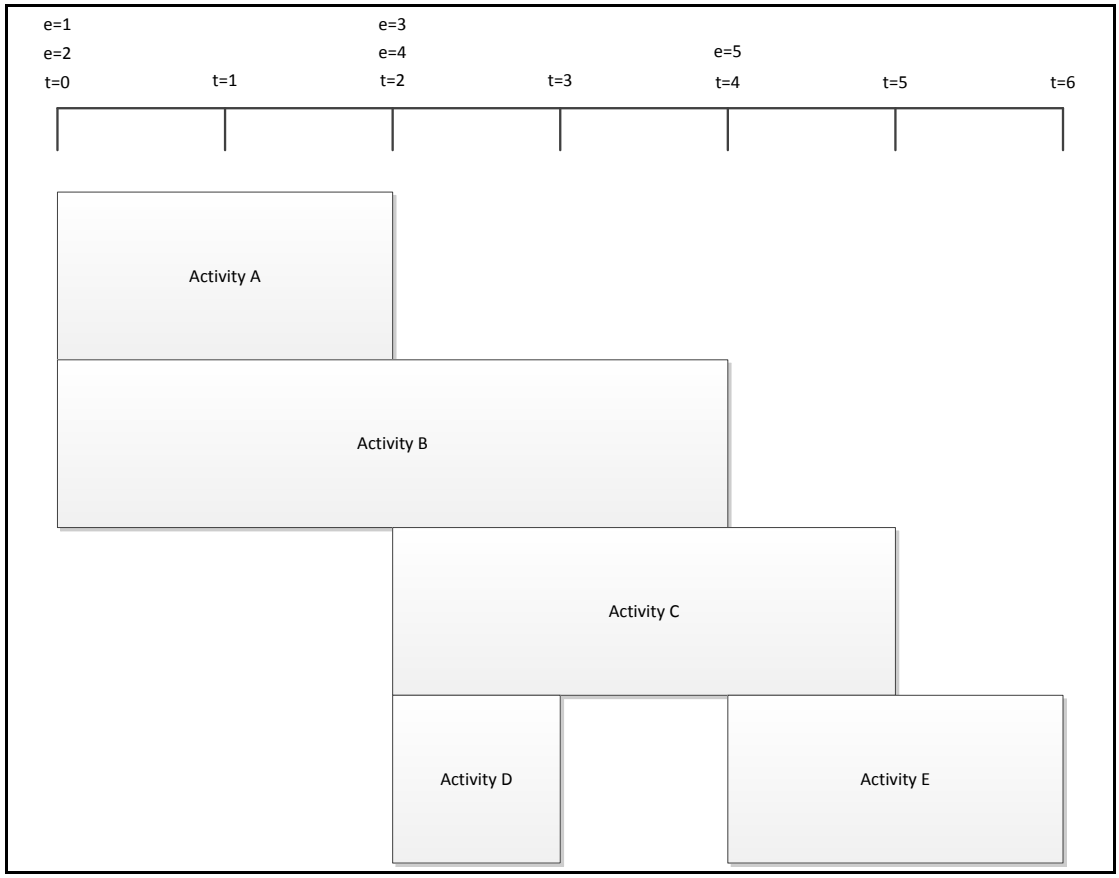

Figure 4 – On-Only Event Formulation Diagram

As can be seen, for the same solution there are only 5 events rather than 6, and the number of non-zero *z* variables is equal to 5 rather than 1. This is clearly shown in Table 3.

Table 3 – On-Only Event Formulation Example Z-Variable Values

Activity *a*	Event *e*	Value
A	1	1
B	2	1
C	3	1
D	4	1
E	5	1

3.3.2 On-Only Additional Parameters & Variables

Parameters:

$ActivityForward_{a\ in\ A}$ Number of activities which must be completed for activity *a* to start

$ActivityReverse_{a\ in\ A}$ Number of activities which require activity *a* to be completed to start

$ZBounds_{e\ in\ E,a\ in\ A}$ Binary; default if((e<=(n-ActivityReverse[a])) and (e>ActivityForward[a])) then 1, else 0

Variables:

$Cov_{e1\ in\ E,\ e2\ in\ E}$ binary variable which represents if the activity which starts at event e1 overlaps with the activity which starts at event e2

$Rescv_{r\ in\ R,\ e1\ in\ E,\ e2\ in\ E}$ continuous variable, >=0, which represents number of resource *r* that is required by the activity at event *e1* when the activity at event *e2* starts

3.3.3 On-Only Objective & Constraints

Objective

$$Minimize\ C_{max}$$

Subject To

Defining Constraints:

$\sum_{e \in E} z_{ae} = 1 \qquad \forall a \in A$ (1)

$$\sum_{a \in A} z_{ae} = 1 \qquad \forall e \in E$$
(2)

$$t_e \leq t_{e+1} \qquad \forall e \neq n \in E$$
(3)

$$t_e + \sum_{a=1}^{a=n} (z_{a,e} * p_a) \leq C_{max} \qquad \forall e \in E$$
(4)

Start Time Constraints:

$$t_e \geq \sum_{a=1}^{a=n} (ES_a * z_{a,e}) \qquad \forall e \in E$$
(5)

$$t_e \leq \sum_{a=1}^{a=n} (LS_a * z_{a,e}) \qquad \forall e \in E$$
(6)

$$t_{e_2} \geq t_{e_1} + p_i * (z_{i,e_1} + z_{j,e_2} - 1) \qquad \forall e_1, e_2 \in E, \forall (i,j) \in IP;\ e_1 < e_2$$
(7)

Resource Constraints:

$$t_{e_1} + \sum_{a=0}^{a=n} p_a * z_{a,e} \leq t_{e_2} + H * cov_{e_1,e_2} \qquad \forall e_1, e_2 \in E;\ e_1 < e_2$$
(8)

$$t_{e_1} + \sum_{a=0}^{a=n} p_a * z_{a,e} \geq t_{e_2} - H * (1 - cov_{e_1,e_2}) \qquad \forall e_1, e_2 \in E;\ e_1 < e_2$$
(9)

$$rescv_{r,e_1,e_2} \leq cov_{e_1,e_2} * b_r \qquad \forall r \in R, \forall e_1, e_2 \in E;\ e_1 < e_2$$
(10)

$$rescv_{r,e_1,e_2} \geq \sum_{a=0}^{a=n} d_{a,r} * (cov_{e_1,e_2} + z_{a,e_1} - 1) \qquad \forall r \in R, \forall e_1, e_2 \in E;\ e_1 < e_2$$
(11)

$$rescv_{r,e_1,e_2} \leq -\sum_{a=0}^{a=n} d_{a,r} * (cov_{e_1,e_2} - z_{a,e_1} - 1) \qquad \forall r \in R, \forall e_1, e_2 \in E;\ e_1 < e_2$$
(12)

$$b_r \geq \sum_{a=0}^{a=n} d_{a,r} * z_{a,e} + \sum_{f=1}^{f<e} rescv_{r,f,e} \qquad \forall r \in R, \forall e \in E$$
(13)

$$z_{a,e} \in \{0,1\} \qquad \forall a \in A, e \in E$$
(14)

$$t_e \geq 0 \qquad \forall e \in E$$
(15)

$$C_{max} \geq 0$$

(16)

$$cov_{e1,e2} \in \{0,1\} \qquad \forall e1 \in E, e2 \in E$$

(17)

$$rescov_{r,e1,e2} \geq 0 \qquad \forall r \in R, e1 \in E, e2 \in E$$

(18)

3.3.4 On-Only Constraint Details

The on-only constraints are described as follows:

(1) Assigns each activity to exactly one event

(2) Assigns each event to exactly one activity

(3) Ensures events happen in order

(4) Defines the makespan, C_{max}, as the end time of the last scheduled activity

(5) Enforces earliest start times

(6) Enforces latest start times

(7) Enforces precedence constraints

(8) Defines coverage variable as equal to 1 if two activities overlap

(9) Defines coverage variable as equal to 1 if two activities overlap

(10) Limits value of resource coverage variable

(11) Defines resource coverage variable as amount of resource in use by overlapping activity

(12) Defines resource coverage variable as amount of resource in use by overlapping activity

(13) Enforces budget constraint

(14) Binary z definition

(15) Continuous non-negative t definition

(16) Continuous non-negative Cmax definition

(17) Binary cov definition

(18) Continuous non-negative rescov definition

3.3.5 Earliest & Latest Starts

The earliest and latest possible start times were calculated based on the network prerequisite structure; an earliest possible start time for an activity is equal to the sum of the durations of the critical path of activities required before the given activity can start. Complementarily, the latest possible start time is equal to the maximum time horizon minus the sum of the durations of the critical path of activities which are dependent upon the given activity finishing, including the duration of the given activity.

3.3.6 Z-Bounds Details

Z-Bounds are used to limit the solution space by limiting the number of *z* variables which can be equal to 1, by defining some values of *z* as equal to zero. The model capitalizes on the network architecture to limit the possible events that can be assigned to activities. Originally, the intention was to use activity counts up to a given node; however, it became apparent that a more efficient way to reduce the solution space would be to count the number of activities required for a given activity to start (the forward activity count, FAC) and the number of activities dependent on a given activity (the reverse activity count, RAC). Examples of these may be seen in Figure 5. These activity counts could then be used to set the Z-Bounds as tight as possible. As per the example, Activity 9 has FAC = 3. This means Activity 9 cannot be assigned events 1, 2, or 3; that is, events ≤ FAC. Complimentarily, the RAC for Activity 9 is 4, and Activity 9 cannot be assigned events 12, 11, 10, or 9; or events > *n*-RAC, where *n* is the total number of activities.

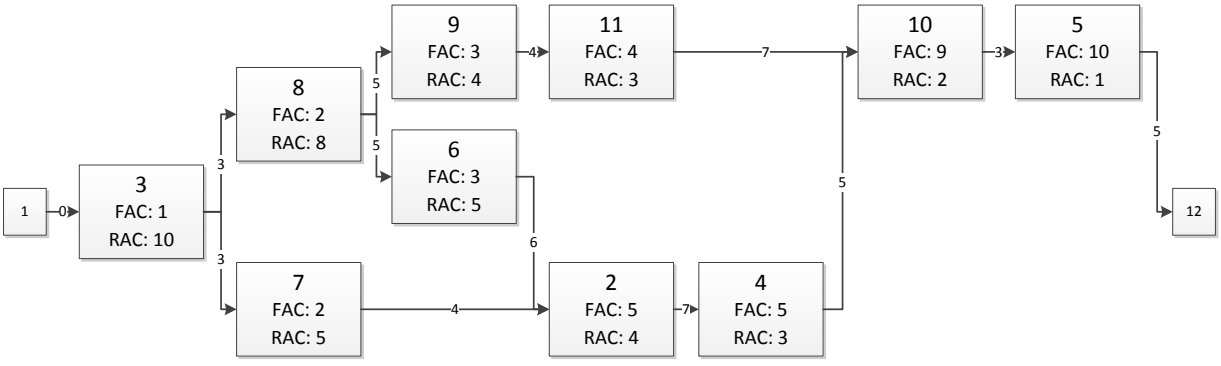

Figure 5 – Example of Activity Counts (Forward & Reverse

Chapter 4. Pre-processing Algorithm

The On-Only model requires data that is not included in the raw data set obtained from Jazz Aviation for implementation, but is rather the result of processing the raw data to draw out embedded information. This information includes the earliest and latest start times for a given activity, as well as information about the network architecture, such as where a given activity is in relation to other activities as defined in section 3.3.5. To determine this information, a pre-processing algorithm was written in C programming language for this paper, and converted to C# for implementation at Jazz Aviation.

While activity counts are easily discernable through visual inspection, it is surprisingly difficult to determine this information computationally. The pre-processing code makes use of a basic topological sort algorithm which is then modified with iterative sub-algorithms to determine the earliest start times, latest start times, and activity counts. While there are numerous topological sort algorithms available online, none took the inputs in the formats that were already available and being used for the On-Only model proposed in this thesis; as a result, the pre-processing code was entirely written by the author. An explanation of this code may be found in Appendix C.

4.1 Topological Sort Pseudocode

The data used for the pre-processing algorithm was the list of prerequisite pairs and the list of activities with their durations for a total of two input files. The pre-processing code outputs three files: earliest start times for each activity, latest start times for each activity, and the activity counts for each activity.

4.1.1 Node Levels & Earliest Start Time Pseudocode

The simplified pseudocode to find node levels and earliest start times is as follows:

A= set of Activities
E(i) = earliest start time of activity i
F(i) = earliest finish time of activity i
D(i) = duration of activity i
N(i) = nodelevel of activity i
P(j) = {I} where I is list of prerequisites to j, or 0 if no prerequisites
nLevel = current node level

1) For i ∈ A, set E(i) = 0 and F(i) = 0
2) Set nlevel = 1
3) If P(i) = 0, set N(i) = nlevel and F(i) = D(i)
4) For all i with N(i) = nlevel
    a. For all j ∈ A
        i. If P(j) includes i, set N(j) = nlevel+1
        ii. If E(j) < F(i), set E(j) = F(i) and set F(j) = F(i) + D(j)
    b. nlevel = nlevel + 1
5) Repeat step 4 until there are zero N(i) with current nlevel. Save (current nlevel – 1) as MaxNode.

4.1.2 Latest Start Time Pseudocode

The simplified pseudocode to find latest start times is as follows:

A= set of Activities
L(i) = latest start time of activity i
F(i) = latest finish time of activity i
D(i) = duration of activity i
N(i) = nodelevel of activity i
S(i) = {J} where J is list of successors to i, or 0 if no successors
nLevel = current node level

1) For i ∈ A, set L(i) = (Time Horizon – D(i))
2) Set nlevel = maxNode
3) If S(i) = 0, set N(i) = nlevel
4) For all j with N(j) = nlevel
    a. For all i ∈ A
        i. If S(i) includes j, set N(i) = nlevel-1
        ii. If L(i) > L(j) – D(i), set L(i) = L(j) – D(i)
    b. nlevel = nlevel - 1
5) Repeat step 4 until nlevel = 0.

4.1.3 Forward Activity Count Pseudocode

The simplified pseudocode to find forward activity counts is as follows:

A= set of Activities
C(i) = if Activity i has been counted
R(i) = if Activity i is pending prerequisite counting
FAC(i) = Forward Activity Count for activity i
P(j) = {I} where I is a list of prerequisites to j, or 0 if no prerequisites

1) For i ∈ A, set FAC(i) = 0, set R(i) = FALSE, and C(i) = FALSE
2) For each j ∈ A
    a. P(j) = I, for all i ∈ I
        i. If C(i) = FALSE, then set R(i) = TRUE, C(i) = TRUE and FAC(j) = FAC(j) + 1
    b. For all i where R(i) = TRUE
        i. P(i) = I, for all r ∈ I
            1. If C(r) = FALSE, then set R(r) = TRUE, C(r) = TRUE and FAC(j) = FAC(j) + 1
    c. Repeat 2.b until every C(r) = TRUE
    d. For i ∈ A, set R(i) = FALSE, and C(i) = FALSE

4.1.4 Reverse Activity Count Pseudocode

The simplified pseudocode to find reverse activity counts is as follows:

A= set of Activities
C(i) = if Activity i has been counted
R(i) = if Activity i is pending successor counting
RAC(i) = Reverse Activity Count for activity i
S(i) = {J} where J is a list of successors to i, or 0 if no successors

3) For j ∈ A, set RAC(j) = 0, set R(j) = FALSE, and C(j) = FALSE
4) For each i ∈ A
    a. S(i) = J, for all j ∈ J
        i. If C(j) = FALSE, then set R(j) = TRUE, C(j) = TRUE and RAC(i) = RAC(i) + 1
    b. For all j where R(j) = TRUE

i. S(j) = J, for all r ∈ J

1. If C(r) = FALSE, then set R(r) = TRUE, C(r) = TRUE and RAC(i) = RAC(i) + 1

c. Repeat 2.b until every C(r) = TRUE

d. For i ∈ A, set R(j) = FALSE, and C(j) = FALSE

Chapter 5. Test Design

As this model is being developed for a specific application – heavy maintenance scheduling at Jazz Aviation – the true relevant test is the implementation of the model to that end. However, from an academic standpoint, there was a desire to examine the relationship between network complexity and resource constraints. Project Scheduling Library (PSPLIB) instances are the typical benchmark [46]; however, the instances follow specific rules that are not always indicative of a real-world application. For example, PSPLIB problems typically have each activity demanding a known quantity of a single resource type, rather than multiple resource types, and their precedence relationships are randomly generated rather than following a known "typical project" network type. Essentially, PSPLIB networks are statistically generated based on known parameters, rather than known networks which can then be statistically measured by known parameters.

As a result, while the PSPLIB is essential for measuring benchmark problems across academic research, they tend to fall short of approximating real world applications. To accurately assess the performance of the on-only event model against the on/off event model, five networks and five budgets were developed to represent the type of problem these models were intended for. A total of 25 instances were generated and their details are outlined in this section.

5.1 Small Example Problem

The initial testing between the On/Off and On-Only models was performed against the example problem from Koné et al [33]. The diagram for this problem may be found in Figure 5; the problem consists of 10 real and 2 dummy activities. When this 12-activity example problem is modelled in GMPL and solved using the application GUSEK, both the On/Off and On-Only models solve to optimality quickly; the On/Off model solves in 0.2 seconds while the On-Only model solves in 0.0 seconds. As a result, a larger problem set was deemed necessary to truly evaluate differences in model performance. This resulted in the development of the 32-activity test networks.

5.2 Test Networks

All test networks were designed for 30 real activities and 2 dummies, for a total of 32 activities with varying degrees of interconnectivity. This degree of interconnectivity is measured by two metrics: total network complexity (TNC) and non-dummy network complexity (NDNC), as

outlined in section 2.5.1. The summary of networks may be seen in Table 4. Visual representations of the networks are shown in Appendix D.

Table 4 – Test Network Complexity Summary

Network	1	2	3	4	5
All arcs	36	42	48	54	60
Non-dummy arcs	32	37	43	49	54
TNC	**1.125**	**1.3125**	**1.5**	**1.6875**	**1.875**
NDNC	**1.067**	**1.233**	**1.433**	**1.633**	**1.800**

5.3 Test Activity Data

Across each network, the activity data remains constant – that is, the duration and resource demands of a given activity is the same for each network. Activity durations were randomly assigned between 1 and 30 units of time. To most appropriately approximate real-world scenarios, each activity was randomly assigned resource demands, with different upper limits on the amount of each resource that could be seized by the activity. This was especially relevant given that Jazz's activities have multiple resource demands, rather than simply one resource being seized by any given activity as in the PSPLIB problem sets. The activity durations and resource demands are in Table 5, with the Resource Factor (as outlined in section 2.5.2) at the end. The overall processing ratio (PR) for this set of activities is 30/1, or 30.

Table 5 – Activity Duration & Resource Demands

Activity	Duration	R1	R2	R3	R4
1	0	0	0	0	0
2	26	5	11	2	1
3	13	20	17	11	9
4	13	11	17	6	9
5	21	19	13	8	5
6	21	14	16	6	2
7	9	15	12	0	3
8	28	3	2	5	6
9	16	20	12	11	7
10	14	0	11	7	4

Activity	Duration	R1	R2	R3	R4
11	10	18	16	1	4
12	24	4	6	13	1
13	5	17	6	13	4
14	24	2	2	10	5
15	13	11	2	11	7
16	30	3	7	5	4
17	24	15	5	8	4
18	30	4	0	1	2
19	16	18	1	5	3
20	15	11	13	7	4
21	10	13	4	11	9
22	15	16	0	0	6
23	22	6	7	2	1
24	27	0	5	7	5
25	20	15	1	8	3
26	25	7	11	1	3
27	1	7	0	3	2
28	8	1	10	5	5
29	5	7	15	5	4
30	17	4	0	9	1
31	17	3	7	1	9
32	0	0	0	0	0
RF	N/A	9.03	7.16	5.69	4.13

5.4 Test Resource Budgets

Lastly, test budgets were created ranging from highly resource constrained (Budget 1) to relatively unconstrained (Budget 5). The highly constrained budget consisted of the same resource availabilities as the maximum demand, while the relatively unconstrained budget consisted of twice the maximum resource demand. The budgets are shown in Table 6.

Table 6 – Test Budgets

	Budget	1	2	3	4	5	Unconstrained
Resource	R1	20	25	30	35	40	289
	R2	17	21	25	29	34	229
	R3	13	17	20	23	26	182
	R4	10	12	15	18	20	132
	Total	**60**	**75**	**90**	**105**	**120**	**832**

The test budgets are measured by their resource constrainedness (RC); as outlined in section 2.5.2, this is equal to the resource factor seen in the final row of Table 5, divided by the available amount of that resource for each budget. The summary of resource constrainedness by resource is seen below in Table 7.

Table 7 – Budget Resource Constrainedness

Resource Constrainedness	Budget 1	Budget 2	Budget 3	Budget 4	Budget 5
Resource 1	0.452	0.361	0.301	0.258	0.226
Resource 2	0.421	0.341	0.286	0.247	0.21
Resource 3	0.438	0.335	0.284	0.247	0.219
Resource 4	0.413	0.344	0.275	0.229	0.206
Budget Average	**0.431**	**0.345**	**0.287**	**0.245**	**0.215**

This table shows numerically how Budget 1 is the most resource constrained, while Budget 5 is the least.

5.5 Test Instances

A total of 25 unique test instances were generated by applying the 5 budgets across each of the 5 networks. As outlined in section 2.5.3, it is possible to compare these instances in relation to each other using a disjunction ratio (DR) combining both network and budget metrics. The lower the disjunction ratio, the more cumulative the network – that is, the less resource constrained, and the less network complex. The higher the disjunction ratio, the more disjunctive the network – that is, the greater resource constrained, and the greater network complex. For these test instances, DR is calculated by multiplying the Total Network Complexity by the Resource Constrainedness. The results are shown in Table 8 below.

Table 8 – Disjunction Ratio for Test Instances

	Budget	1	2	3	4	5
Network	**TNC vs RC**	**0.431**	**0.345**	**0.287**	**0.245**	**0.215**
1	**1.125**	*0.485*	*0.388*	*0.323*	*0.276*	*0.242*
2	**1.3125**	*0.566*	*0.453*	*0.377*	*0.322*	*0.282*
3	**1.5**	*0.647*	*0.518*	*0.431*	*0.368*	*0.323*
4	**1.6875**	*0.727*	*0.582*	*0.484*	*0.413*	*0.363*
5	**1.875**	*0.808*	*0.647*	*0.538*	*0.459*	*0.403*

The instances are presented in Table 9 in order from the lowest DR to the highest DR.

Table 9 – Instances by Increasing Disjunction Ratio

Instance	DR
N1, B5	0.242
N1, B4	0.276
N2, B5	0.282
N2, B4	0.322
N1, B3	0.323
N3, B5	0.323
N4, B5	0.363
N3, B4	0.368
N2, B3	0.377
N1, B2	0.388
N5, B5	0.403
N4, B4	0.413
N3, B3	0.431
N2, B2	0.453
N5, B4	0.459
N4, B3	0.484
N1, B1	0.485
N3, B2	0.518
N5, B3	0.538
N2, B1	0.566
N4, B2	0.582
N5, B2	0.647
N3, B1	0.647
N4, B1	0.727
N5, B1	0.808

Chapter 6. Theoretical Results

Each of the 25 instances – each of the 5 networks across each of the 5 budgets – were tested using both the On/Off and On-Only models. They were tested on a 2013 Lenovo W530 running 64 bit Windows 8.1 with 16 GB RAM. The optimizer used was Gurobi version 5.6.3 under academic license.

6.1 Gap Comparison

Initial testing used a 20% heuristics parameter, with the MIPFocus parameter set to balance between finding new solutions and proving optimality. The time limit was 3600 seconds, with the percent gap – defined as the percentage difference between the best bound and found optimal at a given time – was recorded after 600 seconds, and again after the optimizer confirmed optimality or concluded after 3600 seconds. Full results may be found in Appendix B, and are displayed graphically in sections 6.1.1 and 6.1.2.

6.1.1 Gap Comparison after 600 seconds vs. 3600 seconds by Network

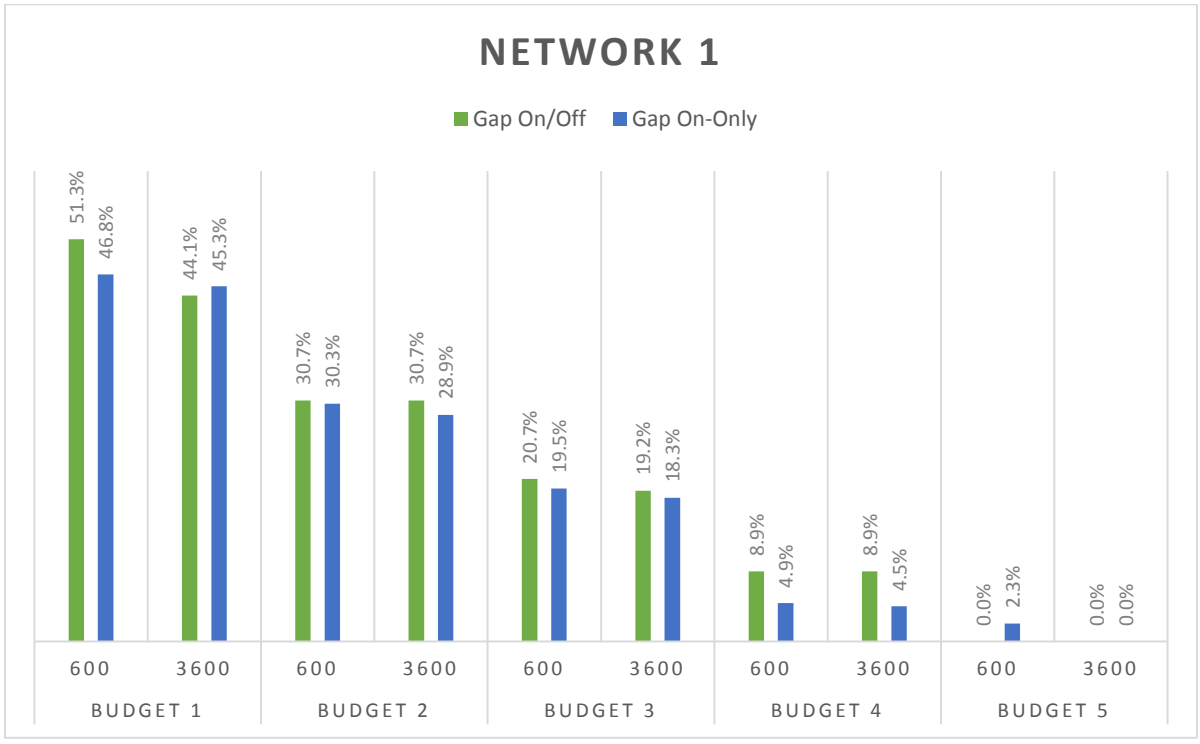

Figure 6 – Network 1 Initial Results by Budget

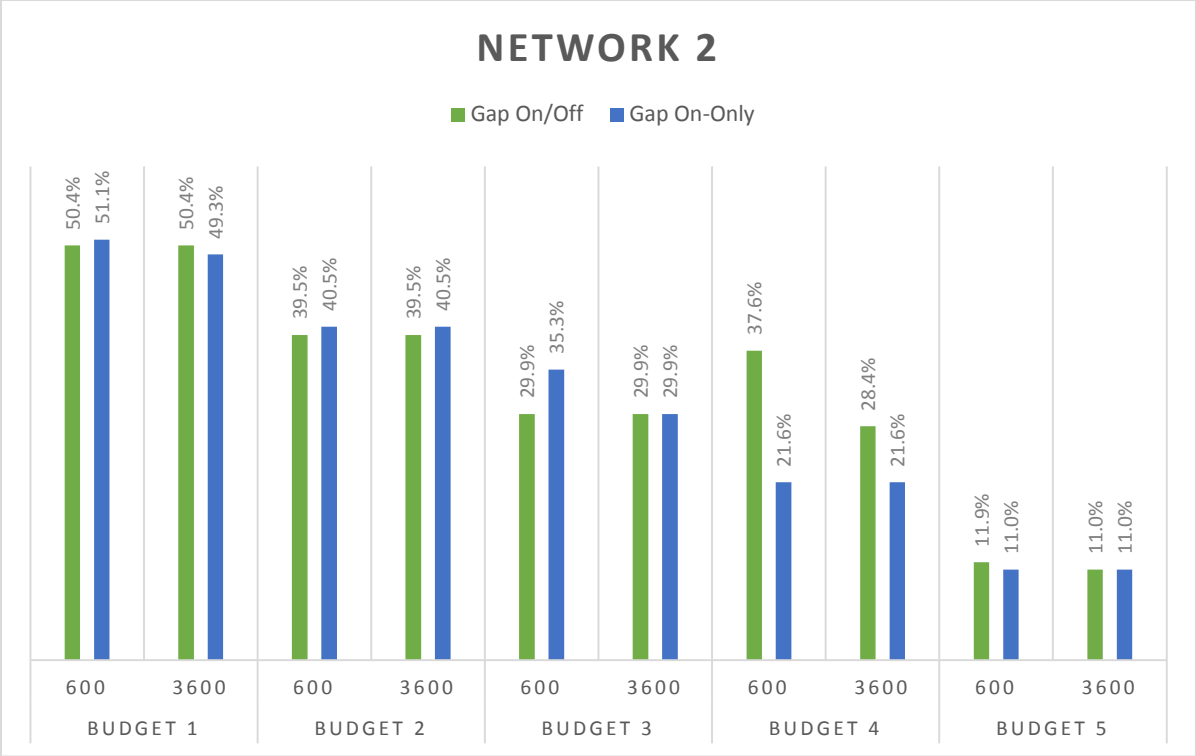

Figure 7 – Network 2 Initial Results by Budget

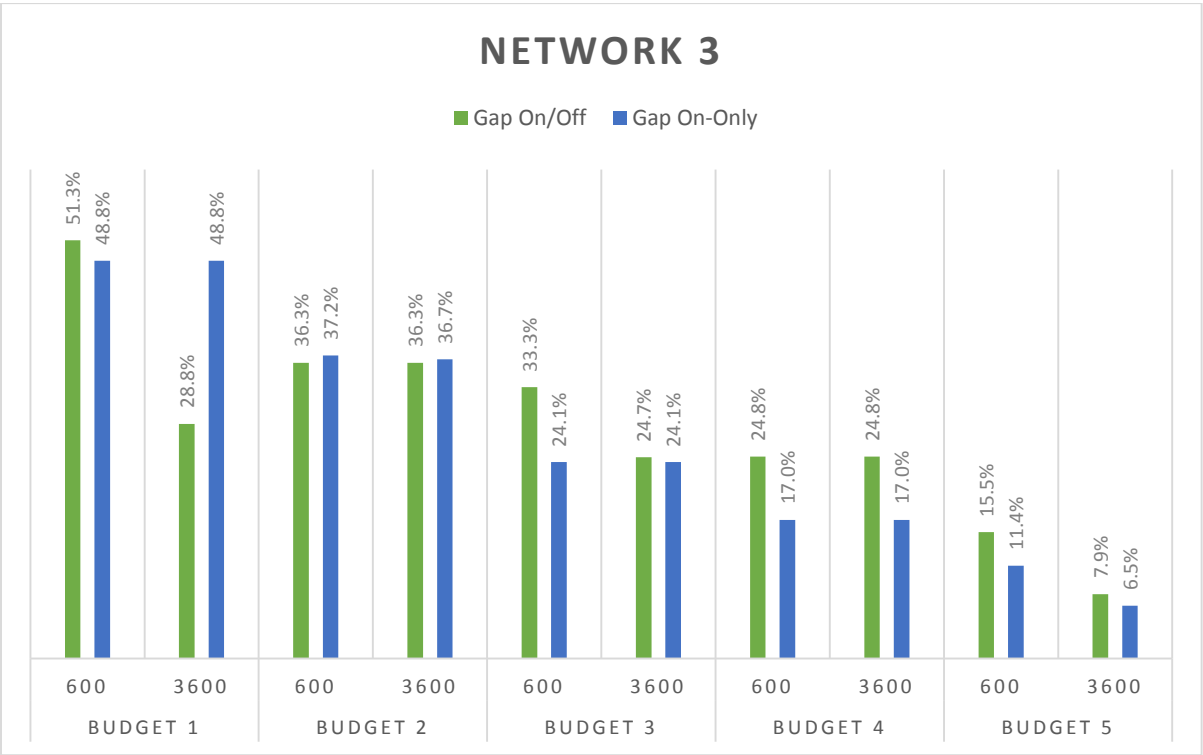

Figure 8 – Network 3 Initial Results by Budget

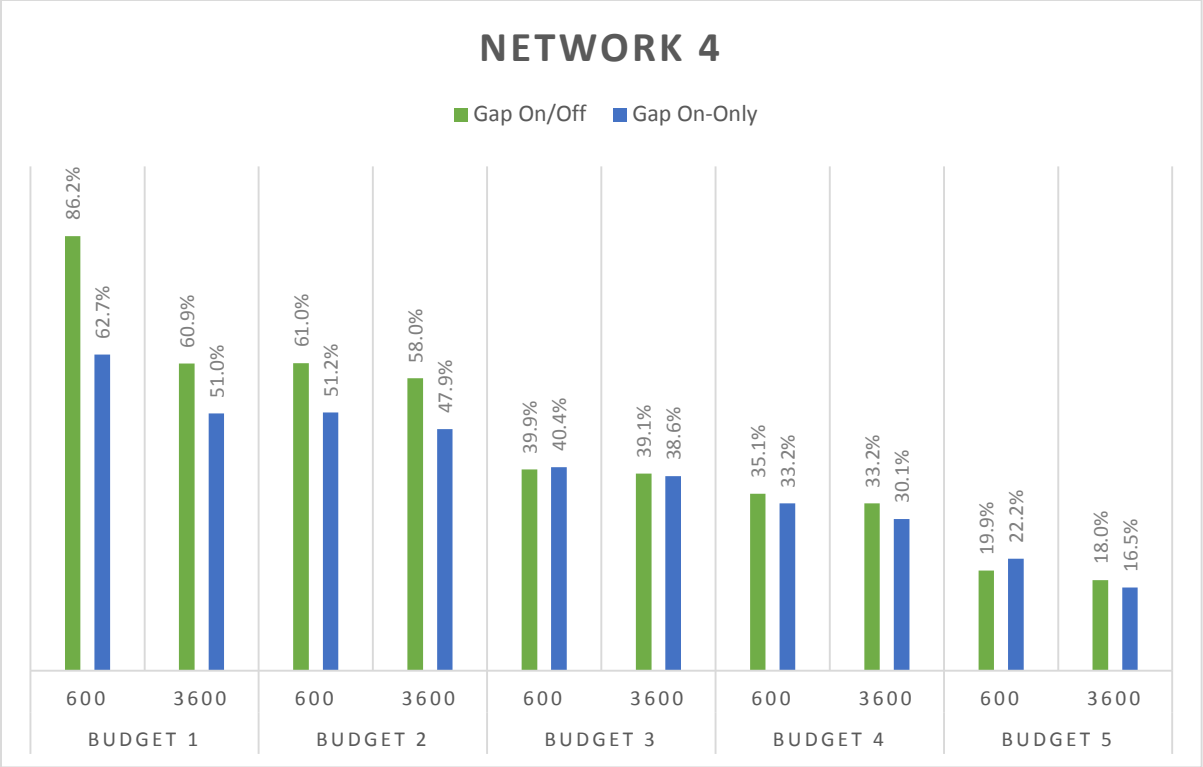

Figure 9 – Network 4 Initial Results by Budget

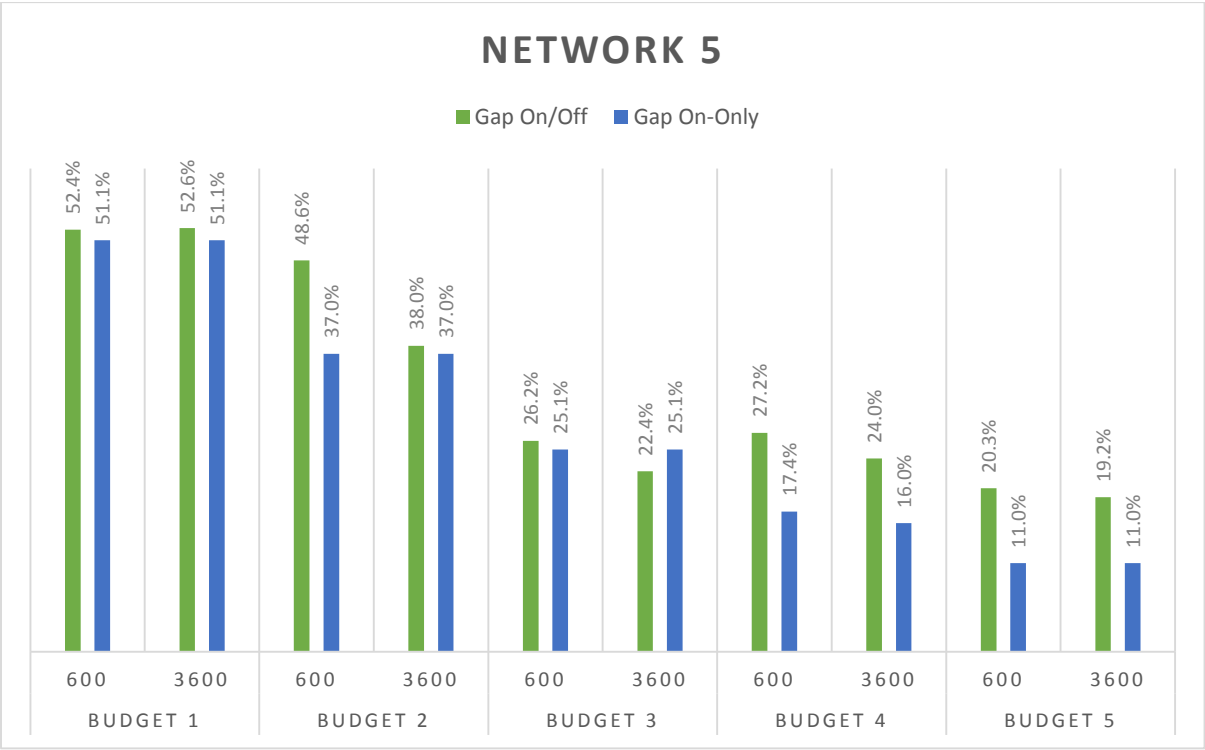

Figure 10 – Network 5 Initial Results by Budget

6.1.2 Gap Comparison after 600 seconds vs. 3600 seconds by Budget

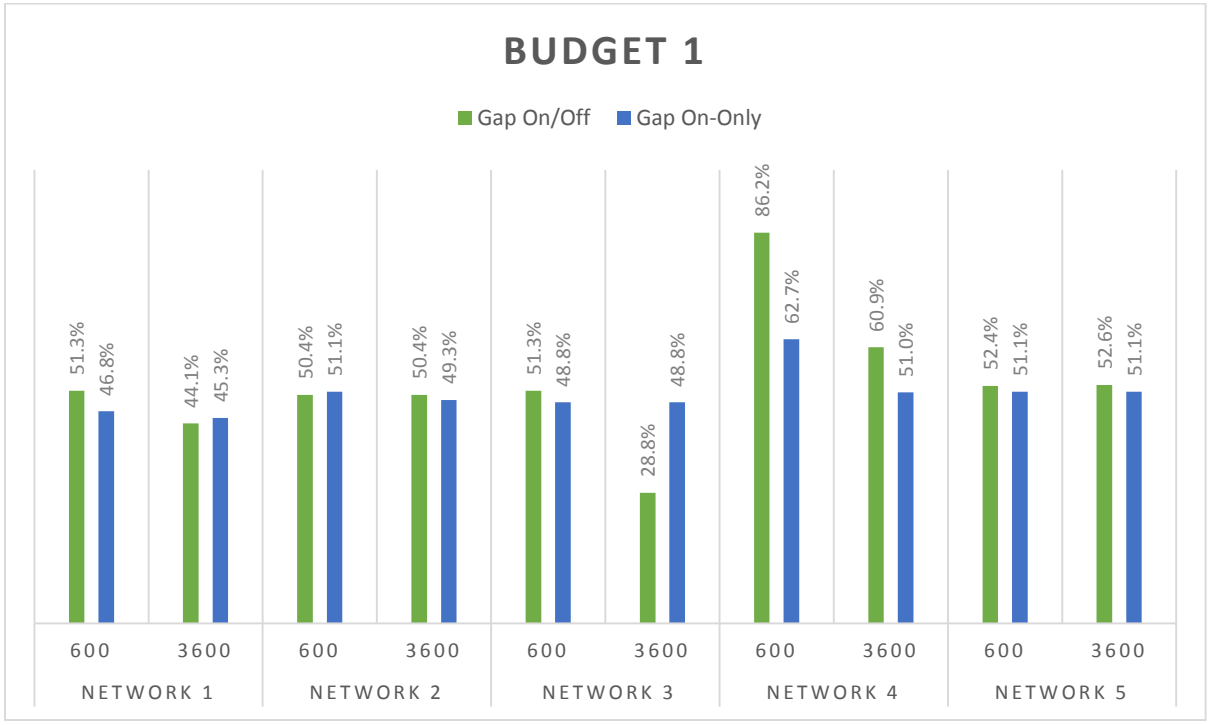

Figure 11 – Budget 1 Initial Results by Network

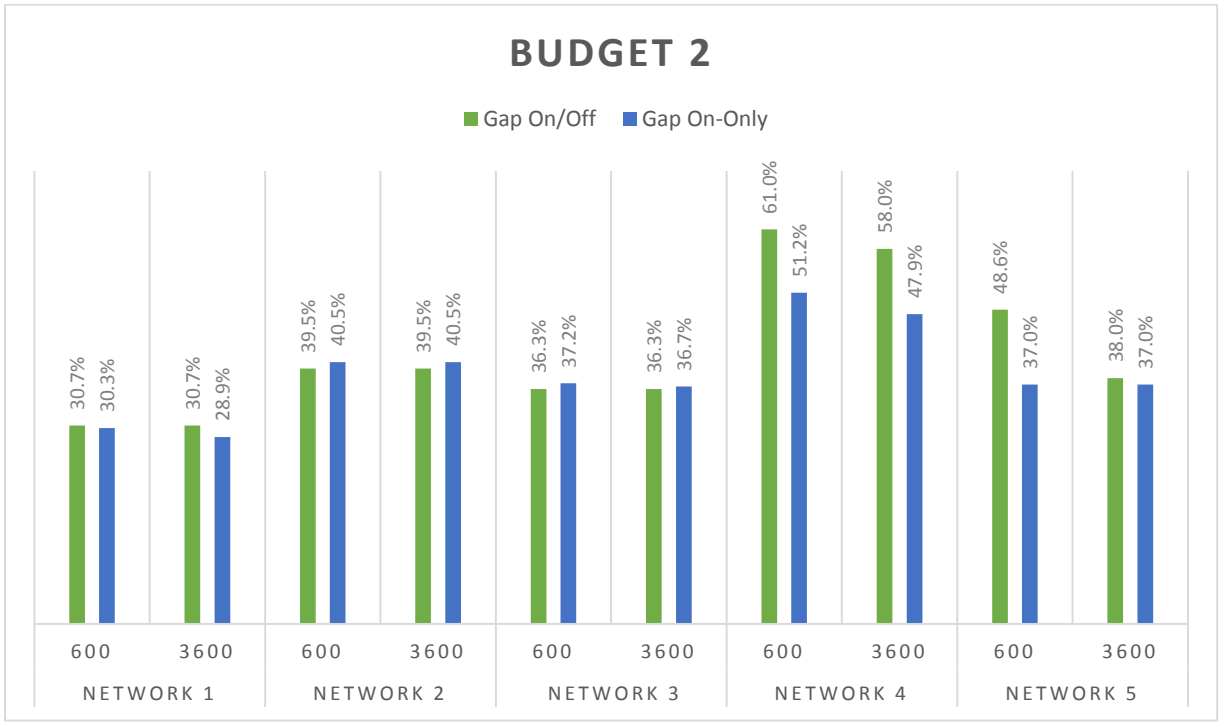

Figure 12 – Budget 2 Initial Results by Network

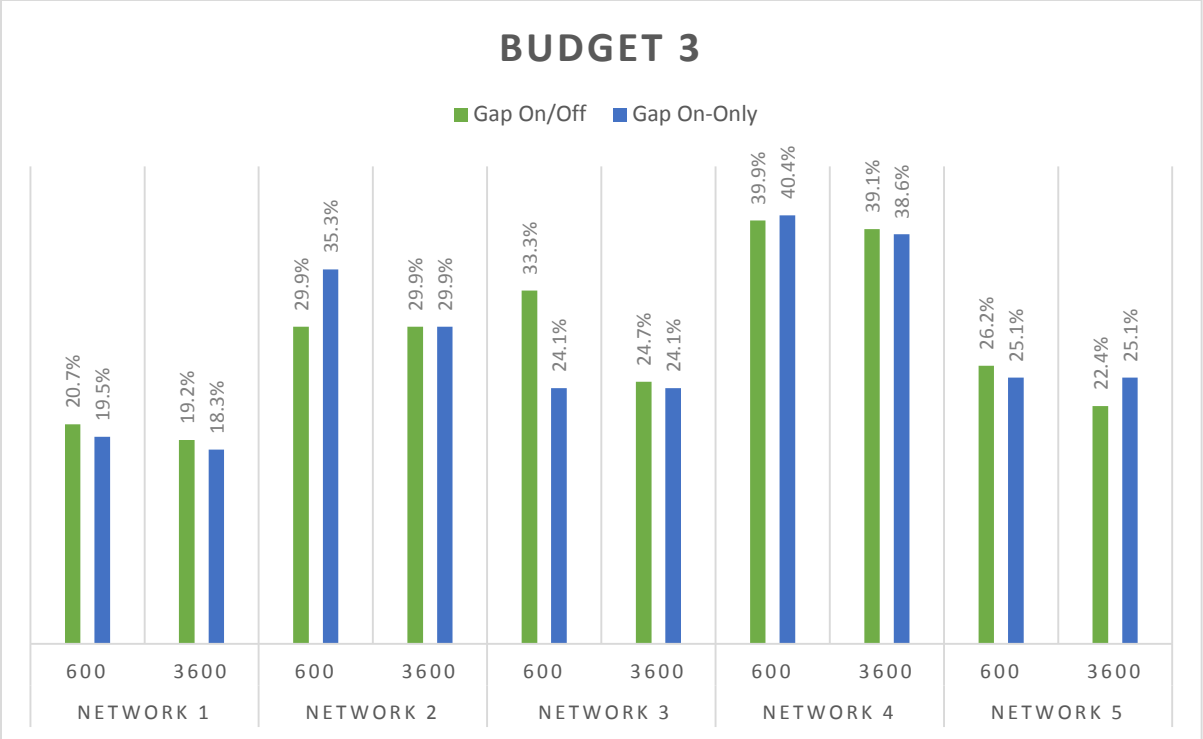

Figure 13 – Budget 3 Initial Results by Network

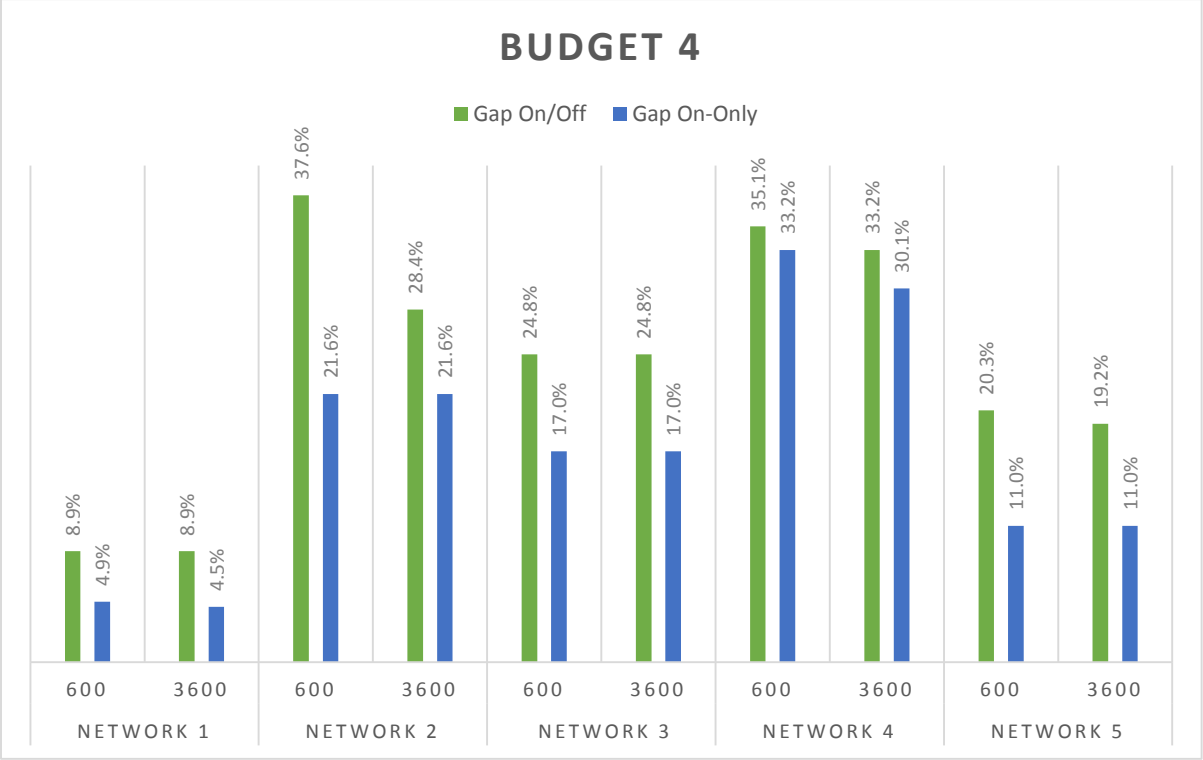

Figure 14 – Budget 4 Initial Results by Network

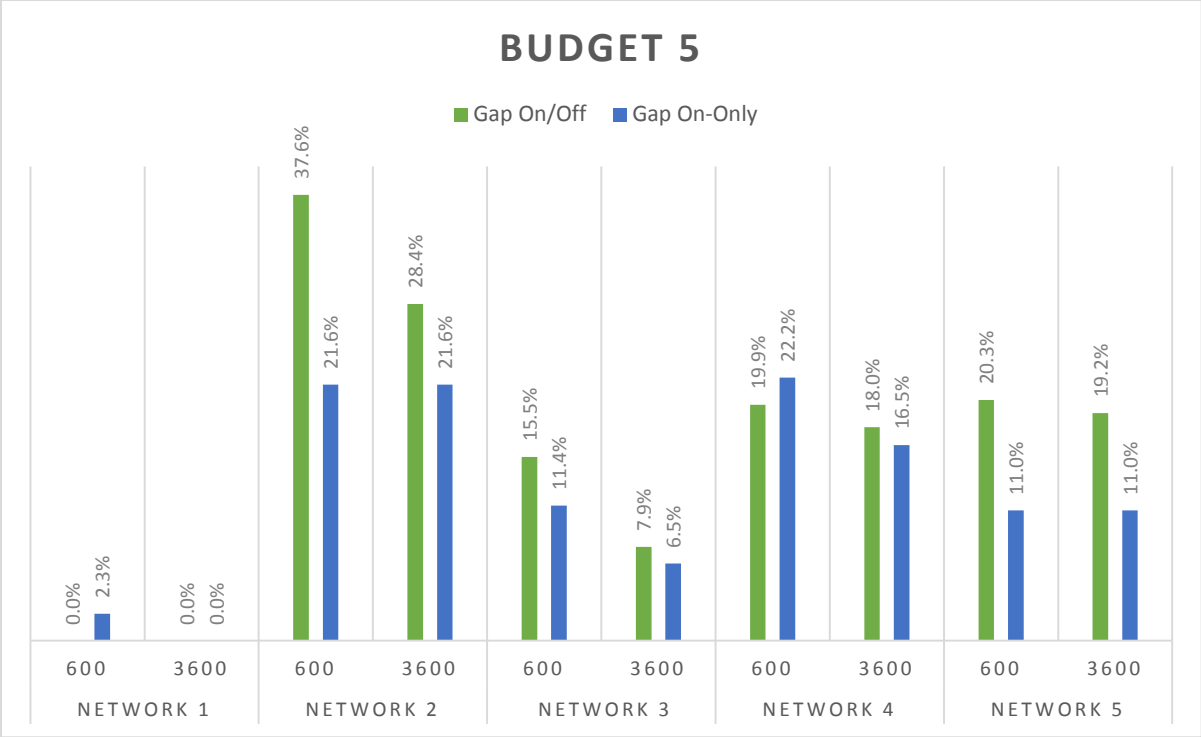

Figure 15 – Budget 5 Initial Results by Network

6.1.3 Disjunction Ratio vs. Gap

The results from the above instances were reordered to follow the increasing disjunction ratio shown in Table 9. The percent gap after 600 and 3600 seconds were then graphed against the increasing DR for each model, and trend lines determined. The results are shown in figures 16 and 17 below, and clearly show that the gap after both 600 and 3600 seconds can be clearly predicted by the disjunction ratio; as the DR increases, so does the gap, indicating that the instances become more intractable as the DR increases.

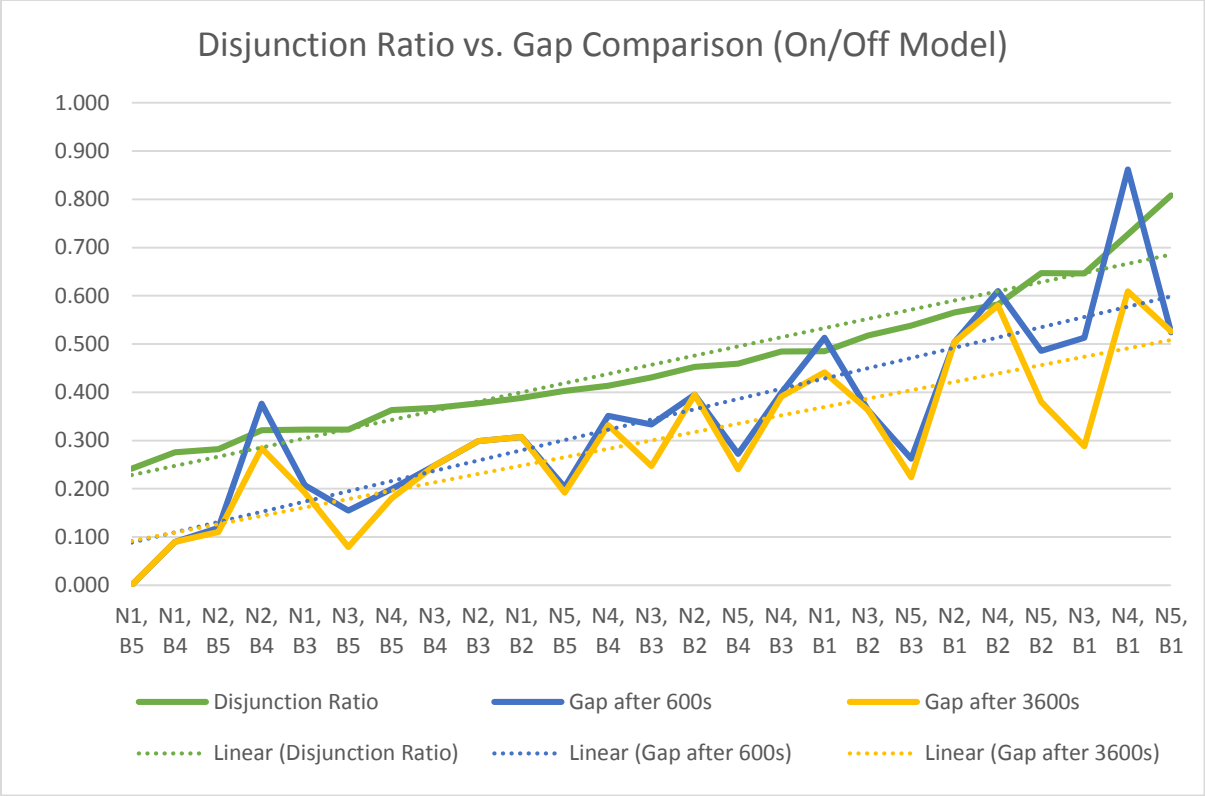

Figure 16 – Disjunction Ratio vs. Gap Comparison (On/Off Model)

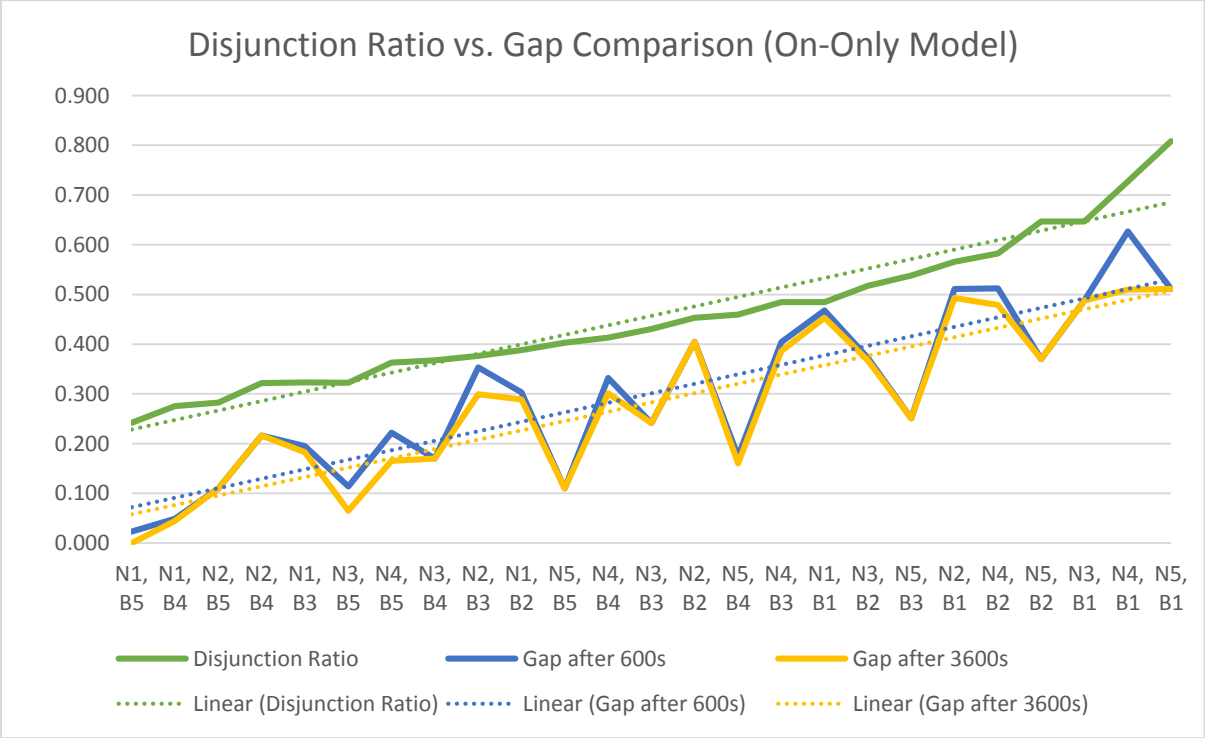

Figure 17 – Disjunction Ratio vs. Gap Comparison (On-Only Model)

6.2 Gap Improvement Analysis

As an overall trend, it can be remarked that the On-Only model reduces the gap more quickly than the On/Off model; the On-Only model had a smaller gap than the On/Off model in 72% of instances. In absolute numbers, the average gap improvement can be used to compare the results of the On/Off model vs the On-Only model in one metric.

The gap improvement is the difference between the gap for the on/off model, and the gap for the on-only model, as represented by $Gap_{on} - Gap_{on/off}$. A positive improvement means the on-only model performed better than the on/off model. The full results of these improvements can be seen in Table 10 – Gap Improvement Results below.

Table 10 – Gap Improvement Results

Network	Budget	Gap Improvement (600 s)	Gap Improvement (3600 s)
1	**1**	4.5%	-1.2%
1	**2**	0.4%	1.8%
1	**3**	1.2%	0.9%
1	**4**	4.1%	4.5%
1	**5**	-2.3%	0.0%
2	**1**	-0.7%	1.1%
2	**2**	-1.0%	-1.0%
2	**3**	-5.4%	0.0%
2	**4**	16.0%	6.8%
2	**5**	0.9%	0.0%
3	**1**	2.5%	-20.0%
3	**2**	-0.9%	-0.4%
3	**3**	9.2%	0.6%
3	**4**	7.8%	7.8%
3	**5**	4.1%	1.4%
4	**1**	23.5%	9.9%
4	**2**	9.8%	10.1%
4	**3**	-0.5%	0.5%
4	**4**	1.9%	3.1%
4	**5**	-2.3%	1.5%
5	**1**	1.3%	1.5%
5	**2**	11.6%	1.0%
5	**3**	1.1%	-2.7%
5	**4**	9.8%	8.0%
5	**5**	9.3%	8.2%

6.2.1 Gap Improvement by Network

When the results from Table 10 are averaged by network, a trend emerges as can be seen in shown in Table 11 – Gap Improvement Results by Network.

Table 11 – Gap Improvement Results by Network

Summary	**Gap Improvement**	
Network	600 s	3600 s
1	1.57%	1.20%
2	1.96%	1.38%
3	4.54%	-2.12%
4	4.54%	3.33%
5	6.62%	3.20%

There is a general trend of positive improvement of the on-only model over the on/off model, which increases as network complexity increases. This improvement is also more pronounced over shorter computation times; that is, the on-only model finds a better solution sooner than the on/off model. However, the on/off model closes the gap as the model is allowed to continue running. The only exception to this trend is network 3; this can be attributed to outliers in the data from network 3, budget 1. This also highlights the importance of resource constrainedness in solve time.

6.2.2 Gap Improvement by Budget

Similar to the network comparison, when comparing by budget it is possible to see clear improvement by the on-only model over the on/off model; however, the trend is not as pronounced as when viewed by network. The data in Table 10 can be averaged by budget to give Table 12 – Absolute Gap Improvement Results by Budget below. While in general there is an improvement by budget, there is no definite trend line connecting resource constrainedness to degree of improvement by the on-only model over the on/off model.

Table 12 – Absolute Gap Improvement Results by Budget

Summary	Gap Improvement	
Budget	600 s	3600 s
1	6.2%	-1.7%
2	4.0%	2.3%
3	1.1%	-0.1%
4	7.9%	6.0%
5	1.9%	2.2%

It is thus possible to conclude that it is more difficult to determine trends based on network complexity across a given budget than to predict instance tractability based on budget across a given network. Indeed, while previous results indicate that budget can predict the performance of a given network, it does not appear that the corollary is demonstrable.

6.3 Objective Value Comparison

The final objective value, representing the makespan of the schedule, was only proven in the case of one network-budget combination – network 1, budget 5. This is unsurprising, as this instance consisted of the simplest network with the lowest degree of resource constrainedness, and the resultant lowest disjunction ratio. However, the objective after 3600 seconds may be compared between the On/Off model and the On-Only model.

6.3.1 Objective Comparison by Network

As with the gap comparison by Network, there is general improvement by the On-Only model over the On/Off model; however, this improvement is marginal, no more than 2% over the On/Off result. These results may be seen in Figure 18 – Average Objective Value by Network (3600 seconds).

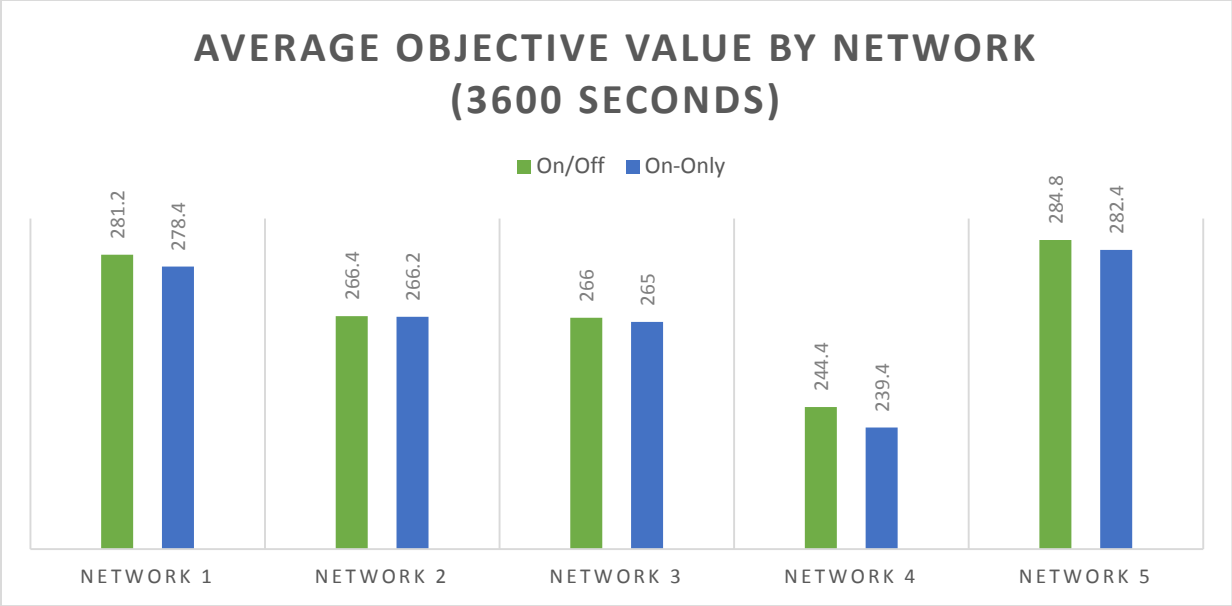

Figure 18 – Average Objective Value by Network (3600 seconds)

6.3.2 Objective Comparison by Budget

Similarly to the gap comparison by budget, the On-Only model makes small improvements over the On/Off model. Again, these improvements average no more than 2% over the On/Off result. This can be seen in Figure 19 – Average Objective Value by Budget (3600 seconds).

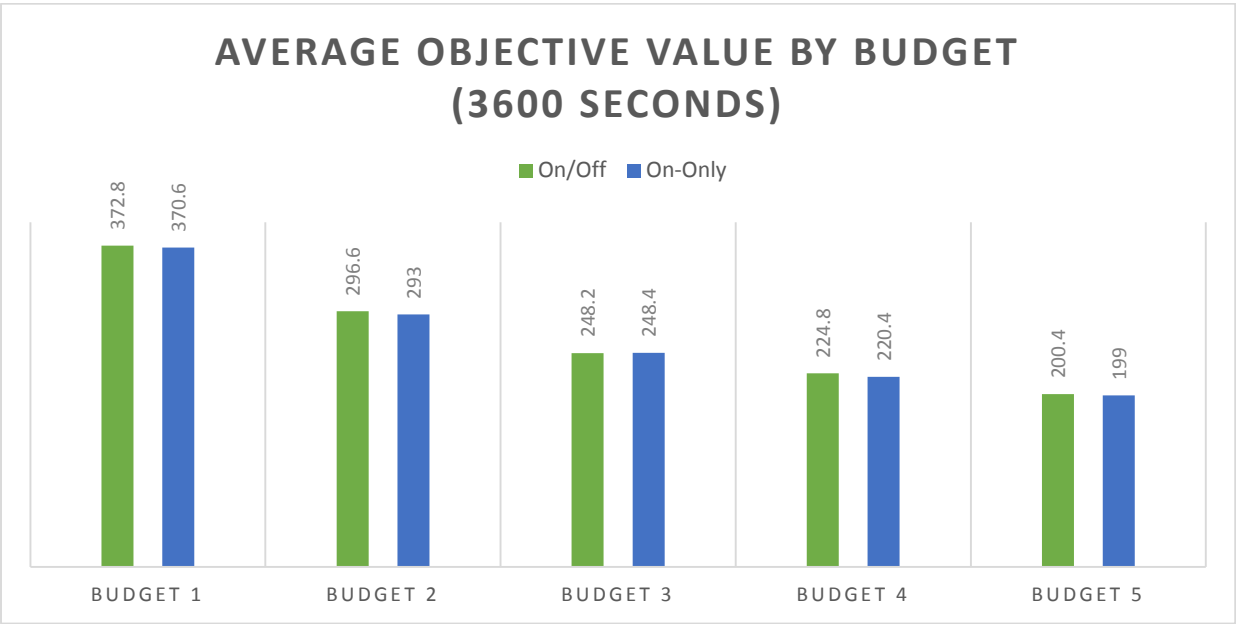

Figure 19 – Average Objective Value by Budget (3600 seconds)

6.4 Best Bound Improvement

The main area of improvement of the on-only model over the on/off model lies in the time to best bound. The average time to best bound comparison by network may be found in Table 13 – Average Time to Best Bound by Network.

Table 13 – Average Time to best Bound by Network

Summary	Average Time to Best Bound (s)	
Network	**On/Off**	**On-Only**
1	298	7
2	911	2
3	796	1
4	683	9
5	416	5

Similarly dramatic results are seen when comparing the average times to best bound by budget as seen in Table 14 – Average Time to Best Bound by Budget.

Table 14 – Average Time to Best Bound by Budget

Summary	Average Time to Best Bound (s)	
Budget	**On/Off**	**On-Only**
1	847	7
2	389	6
3	674	5
4	977	3
5	217	3

6.5 Computational Complexity Trade-off

Initial testing has, as expected, made apparent the computational limits of such large-scale problems. There is a trade-off for the on-only formulation between the size of the linear program generated, and the solve time in the optimizer.

An increase in the number of constraints – most notably, the increase in the number of iterations of the prerequisite constraint, which increases geometrically with the addition of more prerequisite pairs to the index pair set – results in a large linear program file very quickly. However, the perquisite constraints also serve to increase the accuracy and efficiency of the Z-bounds; the more arcs between non-dummy activities, the fewer event-to-activity

combinations, and thus fewer potential feasible solutions for the optimizer to analyze. As a result, as the number of prerequisite pairs increases, the time and memory required to compile the linear program increases, but the time for the optimizer to solve the linear program decreases.

The corollary to this is also true; fewer prerequisite pairs result in much smaller linear program files, which are easier to compile. However, this also decreases the efficacy of the Z-bounds by allowing for a larger solution space and thus a greater amount of options for the optimizer to consider. Accordingly, as the number of prerequisite pairs decreases, the time and memory required to compile the linear program decreases, but the time for the optimizer to solve the linear program increases.

Striking a balance between these two aspects is critical to the applicability and usability of the on-only event formulation; if the number of prerequisite pairs is too large, the memory available may be insufficient to generate the linear program file. In this case, the advantage of decreasing the optimizer solve time is irrelevant as the linear program cannot be built. Therefore, for each compiler there is a threshold number of prerequisite pairs after which the on-event model is inappropriate.

Chapter 7. Implementation Process at Jazz Aviation

The implementation of the on-event model at Jazz as a heavy maintenance scheduling tool was part of a major information technology infrastructure project designed to increase utilization of existing IT programs and decrease reliance on unreliable workarounds. As this model was chosen to meet Jazz Aviation's needs, the implementation and use of this model also stemmed from Jazz Aviation's requirements. This section outlines the proposed implementation process and the data required by Jazz Aviation for implementation. For confidentiality purposes, further details are not included in this thesis.

7.1 Proposed Process

After the implantation of a heavy scheduling tool, the scheduling process will be done automatically. The new process will be:

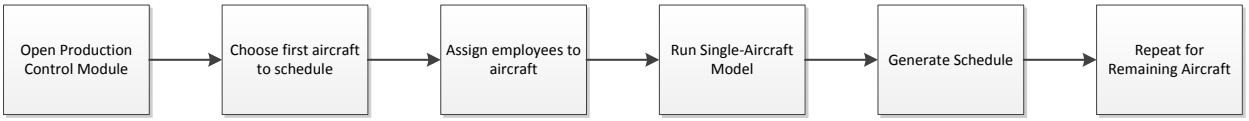

Figure 20 – Proposed Scheduling Process

Note that this will be done every day between 5:30 am and 6:30 am. As a result, the data from the night shift must be complete by 5:30 am every day for the model to run correctly. Production control will assign employees to aircraft based on shifts, and will schedule both the day and night shifts in the morning. In the future, the night shift may be scheduled later in the day for increased accuracy.

7.2 Data Requirements

The data from Jazz for the model is outlined below.

Table 15 – Data Requirements from Jazz Aviation

Data	Details	Status	TRAX
Prerequisites	Pairs of prerequisites for tasks	Employee knowledge	Data confirmation & entry required
Task duration	Length of task	From task card; accuracy uncertain	Available

Data	Details	Status	TRAX
Employees	Employees (number and type) for task	From task card	Available
Equipment	Large equipment for task	From task card	Available; uncertain if queryable
Location	Location on aircraft for task	From task card	Fields available to be populated
Configuration	Power on/off Blocks on/off Hydraulics on/off	Will be accommodated through prerequisites	Fields available to be populated

7.3 Data Details

The detailed explanation for the data requirements is summarized below.

7.3.1 Prerequisites

The general prerequisites for a Q400 were initially determined during a workout in November 2013. This data was a useful starting point for the full prerequisite list. To confirm the accuracy and fullness of this data, a new workout was scheduled for late 2014, which allowed for the determination of a more detailed prerequisite list. During the spring of 2015, these prerequisites were divided based on the Task Sets to which the tasks belonged, which will be discussed in section 8.3.

7.3.2 Task Duration

This data is available from the task cards to be queried. As the task is completed and items on the task card are signed off, the remaining task duration must be determined by summing the durations of the remaining items. This is to accommodate items on task cards taking longer than expected.

7.3.3 Employees

The type of employee and number required is easily accessible from the task card. The production control specialists will assign to each aircraft the number of employees of

each type available to work on that aircraft before generating the schedule each day (for example, Bay 3 has 14 “M” type employees, 5 “S”, and 2 “E” available for the day shift). Currently, task cards do not hold information about specializations for employee types; that is, whether or not a certain certification is required. As a result, this information will not be considered by the scheduling tool. In the future, the model can accommodate this information as it becomes available.

7.3.4 Equipment

As there is a minimal amount of large equipment, such as hydraulic carts, these will be assigned to the aircraft based on the production control specialist’s knowledge of where the aircraft is in a given point of a check. A drop-down list of which equipment is available will be provided in the web interface.

7.3.5 Area

On each task card, the aircraft is divided into zones. To increase the efficiency and decrease the run time of the model, the zones were grouped into areas, such as cockpit, tail, interior fuselage, left wing above, right wing below, etc. These areas have capacities assigned to them. On the task cards, a field called “Area” has been populated.

7.3.6 Configuration

To accommodate the configuration of the aircraft, task cards will be created that reflect physical changes to the aircraft, such as “Aircraft On Blocks” and “Aircraft Power Off”. These task cards will include manpower demands, location requirements, and durations, like other task cards. They will be the prerequisites for the necessary tasks – for example, “Aircraft Power Off” will be a prerequisite for any task that requires power off, which will in turn be prerequisites for the task “Aircraft Power On”.

Chapter 8. Application & Modification to Jazz Aviation

The on-event model that has been developed in this paper is appropriate for large scale, real-world applications such as the Jazz Aviation; however, computational limitations necessitated reducing the size of the problem from 250 routine tasks, plus unknown non-routine tasks, down to smaller datasets averaging 100 tasks. The unknown non-routine tasks also needed to be accommodated in the original schedule before they were known. This was accomplished in three ways:

1) Grouping inspections by area type;
2) Using historical data to generate "shadow" non-routines as placeholders; and
3) Dividing the given dataset into three sections, each small enough to be optimized

Once this was completed, a heuristic implementation process was developed, with iterative coding between the optimization of smaller datasets. In addition, production requirements for safety and change management resulted in data manipulation to fully capture the situation at Jazz Aviation. As mentioned in section 7, production control requires the model be run not only to generate an initial schedule before the aircraft arrives at the hangar, but also daily, with updated information about completed tasks. These completed tasks must also be accommodated in the implementation.

8.1 Area Blocks

To reduce the number of tasks, it was concluded that grouping inspection-type tasks by area on the aircraft would not only reduce the number of tasks at hand, but also serve to minimize movement between areas by employees. The aggregation of such tasks results in a "block" task, which the model treats as one individual task, but in reality contains anywhere from 4 to 15 individual tasks. The information about this aggregate block is stored on the Jazz Aviation database server and re-integrated into the schedule after optimization is complete.

The determination of the area blocks was done by Jazz Aviation employees through a series of Kaizen events. This was to properly capture atypical situations, as well as determine how many resources to assign to each area block. For all area opens and inspections, only "M" type employees are required; this results in the area blocks only seizing M-type employees. The following example clearly demonstrates the benefit of such an aggregation.

8.1.1 Area Block Example

For zone 100 on the sample aircraft, there are 6 tasks as shown in Table 16. The prerequisite constraints, as with most intra-zone inspections, are minimal; activity 5 precedes activity 1, and activity 1 precedes activity 6.

Table 16 – Area Block Example Data

Activity	Employees	Duration (min)
1	1	75
2	2	60
3	4	150
4	1	90
5	1	30
6	2	60

Visual manipulation of these activities within the two precedence constraints results in the following area block schedule shown in Figure 21.

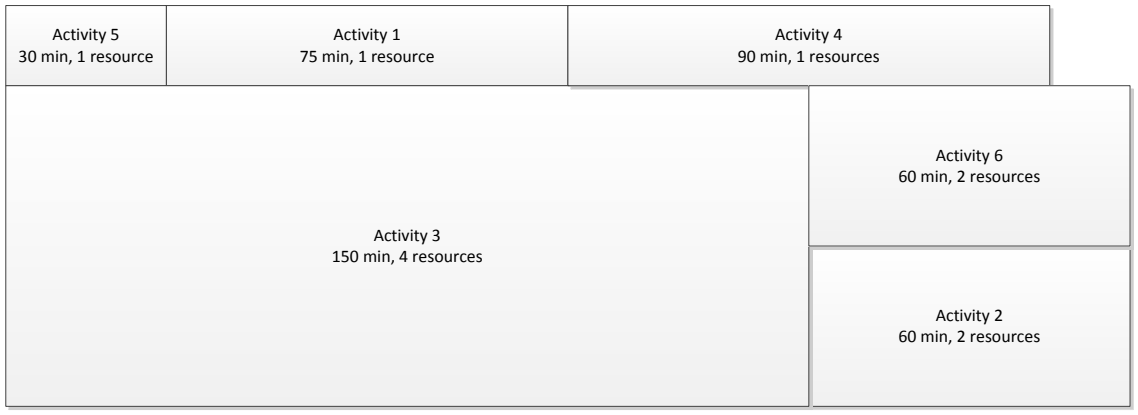

Figure 21 – Area Block Example Schedule

This area block for zone 100 can now be used as a single activity requiring 5 resources for a 210 minutes.

8.2 Shadow Non-Routines

To accommodate the uncertainty of non-routines, shadow non-routines, or SNRs, were calculated from historical data by averaging the amount of non-routine work generated by each routine task, referred to as a "parent routine task". The average duration of a non-routine task generated by a parent non-routine is then assigned to an SNR task which serves as a placeholder for the potential "real" non-routine tasks which has yet to be generated. The SNR has the parent

routine task as a prerequisite. Once the parent routine task is completed, the SNR will be deleted from the model dataset, as either there will be a real non-routine entered into the dataset which will take the place of the SNR, or there will be no further work required due to the parent task, and therefore none to schedule.

8.3 Dataset Splitting

To further reduce the size of the datasets to be optimized, the full list of tasks was divided into three Task Sets.

- Task Set 1 contains routine tasks that fall under the “pre-open”, “open” and “inspect” categories. Examples would include tasks such as removing fuel from the aircraft; washing the aircraft; or removing panels from a given section. It also contains the area block tasks and the few non-“M” type inspection tasks.
- Task Set 2 contains the shadow non-routines for Task Set 1, and the tasks that are neither “open/inspection” nor “close/test” category tasks. These tasks are typically modifications, such as “modification to existing Longeron”, which are done once in an aircraft’s lifecycle.
- Task Set 3 contains the shadow non-routines for Task Set 2, as well as the “close”, “functional test” and “operations test” category tasks. Examples would include testing the hydraulic system; refueling the aircraft; or the flight test. As the model is compiled daily, this dataset includes the real non-routines that are generated throughout the heavy maintenance check.

The benefit of this task set splitting also extends to reducing the number of prerequisites and therefore the size of the compiled linear program model, as mentioned in section 6.5. As the prerequisites between datasets must be accommodated through the heuristic implementation, there are fewer prerequisites within each dataset which decreases the size of the optimization model at each phase.

8.4 Heuristic Implementation

The implementation at Jazz Aviation must accommodate the three datasets, but also connect these datasets together accurately. As a result, in between these three optimization phases, iterative coding in C# is used to enforce precedence constraints between data sets.

Figure 22 shows the process, including the “midprocessing” steps which enforce prerequisite constraints between datasets. The midprocessing pulls from a separate prerequisite list that documents prerequisites between datasets, and takes the scheduled end time of a task from the previous Task Set, and assigns that as the “earliest possible start” time of the successor in the next Task Set. As a result, precedence relations are accommodated outside of the model.

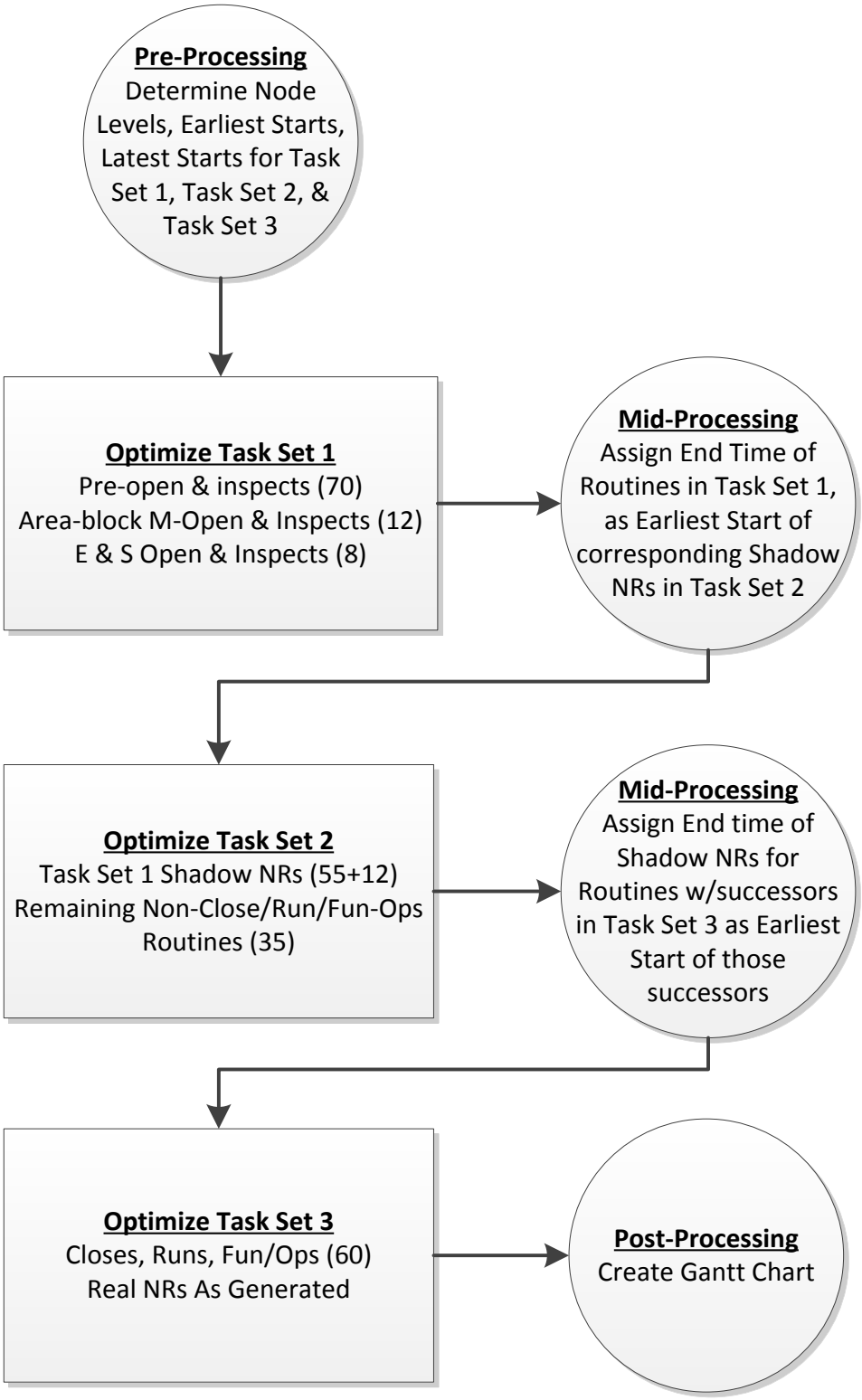

Figure 22 – Heuristic Implementation at Jazz Aviation

8.5 Dataset Updating

Lastly, as the model will be run daily, the datasets must be continually updated each morning. The new datasets delete completed tasks and their shadow non-routines, and add in real non-routines. This breaks the precedence constraints between the completed task and successors so successors may be scheduled as soon as possible. Once a Task Set is 80% completed, the tasks remaining in that task set are moved to the next Task Set and the number of optimization phases reduces by 1. Thus, as the maintenance check continues, the schedule becomes more accurate and quicker to generate.

8.6 Implementation Results

The results of the implementation at Jazz Aviation are not completed, but highly promising. Due to the sheer size of the time horizon of the heavy maintenance problem, solving to sufficient optimality is not only acceptable, but preferred over solving to true optimization. When this "sufficiently optimal" level is considered, results are truly optimistic. Task Set 1 consisted of 72 tasks with 21 resource types, 80 resources, and 137 prerequisite pairs. This task set has a Total Network Complexity of 1.9 – slightly higher than network 5 – and a Resource Constrainedness of 0.0286 – much lower than any of the test budgets, as expected. The resulting disjunction ratio is 0.05434, which is very low. Task Set 1 was solved using Gurobi version 5.6.3. The best bound of 1741 minutes was found in 17 seconds. Results are shown in Table 17.

Table 17 – Jazz Task Set 1 Implementation Results

Incumbent (minutes)	**Gap (percentage)**	**Time (seconds)**	**Time (minutes)**
2140	18.60%	229	3.8
1960	11.20%	231	3.9
1849	5.84%	253	4.2
1839	5.33%	257	4.3
1821	4.39%	323	5.4
1809	3.76%	512	8.5
1763	1.25%	791	13.2
1762	1.19%	953	15.9
1759	1.02%	1111	18.5
1757	0.91%	1188	19.8

Remarkably, the given Task Set is solved to within 95% optimal in under 6 minutes, and to within 99% optimal in under 20 minutes. As the differences between these solutions is small – approximately 1 hour difference in the makespan – Jazz Aviation can find sufficiently good solutions for their datasets within a highly reasonable time frame.

Since Task Sets 2 and 3 are of similar size to Task Set 1, it is reasonable to assume that Task Sets 2 and 3 will, when the data collection is completed, solve to sufficiently close to optimal in similar time periods. Mid-processing is expected to take a similar computer time as pre-processing, which is near-instantaneous. As a result, the entire process is expected to take less than one hour to solve to 99% optimality (not including sub-optimal conditions introduced by the heuristic implementation).

8.7 Post-Processing

The final step for Jazz Aviation will be converting the results from each optimize phase to a visualization in the form of a Gantt chart. This chart will effectively stitch together the results of each optimization to create strings of work for each resource. The Gantt chart is being integrated with existing IT infrastructure to take advantage of existing functionality within the Jazz Aviation database.

Chapter 9. Future Work & Recommendations

The on-only event based RCPSP formulation shows promise for large-scale applications in general, and has been proven to be effective and useful for the Jazz Aviation situation in particular. There is still opportunity, as always, for improvement both academically and in the practical implementation moving forward.

9.1 Academic Future Work

The obvious extension of the on-only formulation would be to accommodate existing variations of the RCPSP, such as the multi-mode formulation and RCPSP/Max time windows formulation. As the on-only event based model is most useful for large, real-world problems, accommodating varying resource profiles would be a rich area for future development; this could include defined resource schedules and renewable vs. non-renewable resources, as well as consumption and production of resources – highly applicable in manufacturing environments.

In addition to these more general areas of improvement, further development of maintenance specific applications would also be beneficial. This includes developing a "resource change penalty matrix", which assigns a time value for moving between certain areas – between the cockpit and the wings of an aircraft, for example. This change in resource usage – seizing space on the wing compared to space in the cockpit – requires time as the employee must obtain and don fall arrest and personal protective equipment. Modelling this reality more fully could impact large-scale projects significantly.

The complexity trade-off couple also be examined in more detail, with the goal of mathematically determining the optimal point where the number of prerequisites is large enough to reduce the solve time of the model, but still small enough to generate the model in a realistic amount of time under existing technological limitations. Additions to the PSPLIB with instances that more closely approximate real-world examples in maintenance, production, and construction would also be beneficial for those in academia whose work spans the theoretical and the practically applicable

Lastly, the current model accommodates uncertainty with static averages based on historical data – the predictive "shadow non-routines". A model that accommodated fuzzy or unpredictable activities and resource demands stochastically would be an interesting area of research.

9.2 Recommendations for Jazz Aviation

As with most industry implementation strategies, data management and validation as well as user acceptance testing is still required at Jazz Aviation; this is in progress and will likely be achieved in the coming months.

In the short term, numerous small improvements would also benefit Jazz Aviation, and are easily implementable in a company with such a robust culture of continuous improvement. One such improvement would be allowing production control supervisors to assign parts information to tasks by overriding the preprocessed earliest start time of a task with the arrival date of a relevant part or specialty tool, so that the model does not schedule the task until the part is available. Jazz Aviation could also continuously update the calculations for shadow NRs with new data as it becomes available so as to increase the predictive capacity of the model.

In the longer term, a tool that balances hangar resources between multiple aircraft would be ideal, whether this is achieved by a heuristic assigning resources between each aircraft and the on-only event model being run multiple times, or by modifying the on-only model to accommodate multiple aircraft implicitly.

Chapter 10. Conclusion

RCPSPs are a highly researched, highly applicable set of problems with a large variety of model types, solving methods, and practical applications. This thesis address the area of event-based models, demonstrating that the on-only model proposed in this thesis generally improves upon the solving ability of the existing on/off model. The on-only model is more intuitive and easier to manipulate than the on/off model; as a result, it is heavily improved by preprocessing to find bounds on the binary z decision variable. This model is appropriate for large scale, highly interconnected problems with relatively unconstrained resources, i.e. problems with high disjunction ratios.

A significant contribution of the On-Only model to the body of research, though, is clearly the best bound solve time; the development of the preprocessing code to provide a mathematical representation of activity location within a network, and the resultant capitalization on this data has resulted in a robust and accurate model for finding the best bound very quickly. While the model may not quickly find the solution that results in this best bound, knowing the best possible objective value is of great importance to large-scale scheduling problems in general and Jazz Aviation in specific.

The implementation results also prove that real-world scenarios are far more tractable than theoretical experiments – even when those experiments have been designed to approximate real-world scenarios. Neither event models found optimal solutions within an hour for the majority of 32 activity test instances, yet the On-Only event model was able to solve the real Jazz Aviation problem, with 72 tasks, to 99% optimality in just 20 minutes. While an exciting result, it does highlight the insufficiency of current test cases for predicting the true usefulness of existing models in industrial applications.

This specific implementation accommodates the complexities of the situation at Jazz Aviation, and was developed with the ultimate goal of practical usability for the company. This resulted in modifications to the model to accommodate data uncertainties, as well as the use of a heuristic implementation process. Combined, these modifications drastically reduce the solve time for the overall scheduling process at Jazz Aviation by taking the initial schedule generation from 4 or 5 days to under an hour. The tool also allows for up-to-date daily scheduling which utilizes the existing IT infrastructure at Jazz Aviation to its full capacity.

As implementation is currently in progress and Jazz Aviation stakeholders are highly satisfied with the tool, this thesis has met its goal for not only contributing to the body of knowledge in the area of RCPSPs, but also meeting the requirements of those who will be affected by this model every day.

References

[1] F. E. D. &. W. H. Deblaere, "Reactive Scheduling in the Multi-Mode RCPSP," *Computers & Operations Research,* pp. 63-74, 2011.

[2] P. Brucker, A. Drexl, R. Mohring, K. Neumann and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research,* pp. 3-41, 1999.

[3] W. Herroelen, E. Demeulemeester and B. De Reyck, "A note on the paper “Resource-constrained project scheduling: Notation, classification, models and methods” by Brucker et al.," *European Journal of Operational Research,* pp. 679-688, 2001.

[4] F. Marmier, C. Varnier and N. Zerhouni, "Proactive, dynamic and multi-criteria scheduling of maintenance activities," *International Journal of Production Research,* pp. 2185-2201, 2009.

[5] A. Sprecher, "A competitive branch-and-bound algorithm for the simple assembly line balancing problem," *International Journal of Production Research,* pp. 1787-1816, 1999.

[6] A. Bonfietti, M. Lombardi, L. Benini and M. Milano, "A Constraint Based Approach to Cyclic RCPSP," in *Lecture Notes in Computer Science*, Berlin, Springer, 2011, pp. 130-144.

[7] G. Fernando and D. Ramirez Rios, "Multi-objective Optimization of the Resource Constrained Project Scheduling Problem (RCPSP)," *International Journal of Computer Science & Applications,* 2013.

[8] J. Cheng, J. Fowler, K. Kempf and S. Mason, "Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting," *Computers & Operations Research,* pp. 275-287, 2015.

[9] H. Neng Chiu and D. Maw Tsai, "An integer linear programming model and a modified branch and bound algorithm for the project material requirements planning problem," *Journal of Information and Optimization Sciences,* pp. 151-196, 2003.

[10] H. Stadtler, "Multilevel capacitated lot-sizing and resource-constrained project scheduling: an integrating perspective," *International Journal of Production Research,* pp. 5253-5270, 2005.

[11] O. Icmeli, S. Selcuk Erenguc and C. J. Zappe, "Project Scheduling Problems: A Survey," *International Journal of Operations & Production Management,* pp. 80-91, 1993.

[12] I. Cohen and M. Iluz, "When cost–effective design strategies are not enough: Evidence from an experimental study on the role of redundant goals," *Omega,* pp. 99-111, 2015.

[13] R. Kolisch and A. Drexl, "Local search for nonpreemptive mult-mode resource constrained project scheduling," *IIE Transactions,* pp. 987-999, 1997.

[14] J. Coelho and M. Vanhoucke, "Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers," *European Journal of Operations Research,* pp. 73-82, 2011.

[15] A. Oddi and R. Rasconi, "Iterative Flattening Search on RCPSP/max Problems: Recent Developments," pp. 99-115, 2009.

[16] F. Ballestine, A. Barrios and . V. Valls, "Looking for the best modes helps solving the MRCPSP/max," *International Journal of Production Research,* pp. 813-827, 2013.

[17] F. Ballestin, "When it is worthwhile to work with the stochastic RCPSP?," *Journal of Scheduling,* pp. 153-166, 2007.

[18] T. Bhaskar, M. N. Pal and A. K. Pal, "A heuristic method for RCPSP with fuzzy activity times," *European Journal of Operational Research,* pp. 57-66, 2011.

[19] F. Deblaere, E. Demeulemeester and W. Herroelen, "Proactive policies for the stochastic resource-constrained project scheduling problem," *European Journal of Operational Research,* pp. 308-316, 2011.

[20] M. Masmoudi and A. Hait, "Fuzzy uncertainty modelling for project planning: application to helicopter maintenance," *International Journal of Production Research,* pp. 3594-3611, 2012.

[21] H. Okubo, T. Miyamoto, S. Yoshida and K. Mori, "Project scheduling under partially renewable resources and resource consumption during setup operations," *Computers & Industrial Engineering,* pp. 91-99, 2015.

[22] O. Kone, C. Artigues, P. Lopez and M. Mongeau, "Comparison of mixed-integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources," *Flexible Services and Manufacturing Journal,* pp. 25-47, 2013.

[23] F. Gasperoni and U. Schwiegelshohn, "Generating close to optimum loop schedules on parallel processors," *Parallel Process Lett,* pp. 291-403, 1994.

[24] Z. Hanzalek and C. Hanen, "The impact of core precedences in a cyclic RCPSP with precedence delays," *Journal of Scheduling,* pp. 275-284, 2015.

[25] B. Dupont de Dinechin and A. Munier Kordon, "Converging to periodic schedules for cyclic scheduling problems with resources and deadlines," *Computers and Operations Research,* pp. 227-236, 2014.

[26] V. Van Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operations Research,* pp. 409-418, 2010.

[27] F. Villafañez, A. Lopez-Paredes and J. Pajares, "From the RCPSP to the DRCMPSP: Methodological Foundations," in *Proceedings of the International Conference on Artificial Intelligence*, Las Vegas, 2014.

[28] N. Grangeon, P. Leclaire and S. Norre, "Heuristics for the re-balancing of a vehicle assembly line," *International Journal of Production Research,* pp. 6609-6628, 2011.

[29] U. Beşikci, U. Bilge and G. Ulusoy, "Resource dedication problem in a multi-project environment," *Flexible Services and Manufacturing Journal,* pp. 206-229, 2013.

[30] J. Kuster, D. Jannach and G. Friedrich, "Extending the RCPSP for modeling and solving disruption management problems," *Applied Intelligence,* pp. 234-253, 2009.

[31] E. Vazquez, M. Calvo and P. Ordonez, "Learning process on priority rules to solve the RCMPSP," *Journal of Intelligent Manufacturing,* pp. 123-138, 2015.

[32] J. Blazewicz, J. K. Lenstra and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics,* pp. 11-24, 1983.

[33] O. Kone, C. Artigues, P. Lopez and M. Mongeau, "Event-based MILP models for resource-constrained project scheduling problems," *Computers & Operations Research,* pp. 3-13, 2011.

[34] R. Kolisch and S. Hartmann, "Experimental evaluation of state-of-the-art heuristics for the resource constrained project scheduling problem," *European Journal of Operational Research,* pp. 394-407, 2000.

[35] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European Journal of Operational Research,* pp. 23-37, 2006.

[36] B. P. Rao and K. M. Chaitanya, "Resource Constrained Project Scheduling Problems - A Review Article," *International Journal of Science and Research,* pp. 1509-1512, 2013.

[37] M. B. Aryanezhad and B. Ashtiani, "Preemptive resource constrained project scheduling problem with uncertain resource availabilities: Investigate worth of proactive strategies," in *Industrial Engineering and Engineering Management (IEEM)*, Macao, 2010.

[38] A. H.-L. Chen, Y.-C. Liang and J. D. Padilla, "An Entropy-Based Upper Bound Methodology for Robust Predictive Multi-Mode RCPSP Schedules," *Entropy,* pp. 5032-5067, 2014.

[39] A. Schutt, T. Feydy, P. J. Stuckey and M. G. Wallace, "Solving the Resource Constrained Project Scheduling Problem with Generalized Precedences by Lazy Clause Generation," 2010.

[40] T. Fruhwirth and S. Abdennadher, Essentials of Constraint Programming, Springer-Verlag, 2003.

[41] W.-J. van Hoeve, "Introduction to Constraint Programming," 24 September 2012. [Online]. Available: http://www.andrew.cmu.edu/user/vanhoeve/summerschool/slides/introduction_1up.pdf.

[42] A. Horbach, "A Boolean satisfiability approach to the resource-constrained project scheduling problem," *Annals of Operations Research,* pp. 89-107, 2010.

[43] O. Liess and P. Michelon, "A constraint programming approach for the resource-constrained project scheduling problem," *Annals of Operations Research,* pp. 25-36, 2008.

[44] J. Zapata, B. Hodge and G. Reklaitis , "The multimode resource constrained multiproject scheduling problem: alternative formulations," *AIChE Journal,* pp. 2101-2119, 2008.

[45] G. M. Kopanos, T. S. Kyriakidis and M. C. Georgiadis, "New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems," *Computers and Chemical Engineering,* pp. 96-106, 2014.

[46] R. Kolisch, "PSPLIB - A project scheduling problem library," *European Journal of Operations Research,* vol. 96, no. 1, pp. 205-216, 1997.

[47] T. Berthold, S. Heinz, M. E. Lubbecke, R. H. Mohring and J. Schulz, "A Constraint Integer Programming Approach for Resource-Constrained Project Scheduling," in *Lecture Notes in Computer Science*, Berlin, Springer, 2010, pp. 313-317.

[48] G. Brandinu and N. Trautmann, "Sequential selection and heuristic scheduling of multiple resource-constrained projects," *13th International Conference on Project Management and Scheduling,* pp. 102-105, 2012.

[49] F. Deblaere, E. Demeulemeester and W. Herroelen, "Reactive scheduling in the multi-mode RCPSP," *Computers & Operations Research,* pp. 63-74, 2011.

[50] R. Kolisch, A. Sprecher and A. Drexl, "Characterization and Generation of a General Class of Resource Constrained Project Scheduling Problems," *Management Science,* vol. 41, no. 10, pp. 1693-1703, 1995.

Appendix A – GMPL Code

The following is the GMPL code, which was developed in GUSEK. The model represents the 12-activity example problem from Koné et al [33].

```
#On-Only Event Model

param A := 12; #Max number of Activities is set as A, including dummies
param R := 2; #Number of resource types
param T := 49; #Max Time Horizon

set Activities, default{1..A}; #list of activities
set Events, default{1..A}; #list of events
set IP within Activities cross Activities; #list of precedence index pairs
set Resources, default{1..R}; #list of resource types


param Duration{a in Activities}; #duration of activity a
param Budget{r in Resources}; #budget of resource type r
param Demand{a in Activities, r in Resources}; #demand of resource r needed by
activity a

param ES{a in Activities}; #earlist start of activity a
param LS{a in Activities}; #latest start of activity a

param ActivityFront{a in Activities}; #number of activities required for activity a to start
param ActivityReverse{a in Activities}; #number of activities dependent on activity a
param zBounds{e in Events, a in Activities}, default if((e<=(A-
ActivityReverse[a])) and (e>ActivityFront[a])) then 1 else 0;

var t{e in Events}, >= 0; #continous time of event e
var z{e in Events, a in Activities}, binary <= zBounds[e,a]; #binary; if activity a
starts at event e
var Cmax, >= 0; #continuous makespan variable

var cov{e1 in Events, e2 in Events: e1<e2}, binary; #binary, if activity at event 1
overlaps activity at event 2
var rescv{r in Resources, e1 in Events, e2 in Events: e1<e2}, >=0; #continous,
number of resource r in use by event 1's activity at event 2

minimize makespan: Cmax;

s.t. One_Do_Activities{a in Activities}: sum{e in Events} z[e,a] = 1;
s.t. Two_Do_Events{e in Events}: sum{a in Activities} z[e,a] = 1;

s.t. Three_EventsIncreasing{e in Events: e<A}: t[e] <=t[e+1];

s.t. Four_EarlyStart{e in Events}:  t[e] >= sum{a in Activities}(ES[a]*z[e,a]);
s.t. Five_LateStart{e in Events}:  t[e] <= sum{a in Activities}(LS[a]*z[e,a]);

s.t. Six_Precedence{e1 in Events, e2 in Events, (i,j) in IP: e1<e2 }: t[e2]>=
t[e1] + Duration[i]*(z[e1,i] +z[e2,j]-1);

s.t. Seven_SetCov1{e1 in Events, e2 in Events: e1<e2}: t[e1] + sum{a in
Activities}(Duration[a]*z[e1,a]) <= t[e2] + T*cov[e1,e2];
```

```
s.t. Eight_SetCov2{e1 in Events, e2 in Events: e1<e2}: t[e1] + sum{a in
Activities}(Duration[a]*z[e1,a]) >= t[e2] - T*(1 - cov[e1,e2]);

s.t. Nine_BoundResCov{r in Resources, e1 in Events, e2 in Events: e1<e2}:
rescv[r,e1,e2] <= cov[e1,e2]*Budget[r];

s.t. Ten_SetResCov1{r in Resources, e1 in Events, e2 in Events: e1<e2}:
rescv[r,e1,e2] >= sum{a in Activities}(Demand[a,r]*(cov[e1,e2]+z[e1,a]-1));
s.t. Eleven_SetResCov2{r in Resources, e1 in Events, e2 in Events: e1<e2}:
rescv[r,e1,e2] <= -sum{a in Activities}(Demand[a,r]*(cov[e1,e2]-z[e1,a]-1));

s.t. Twelve_ResourceConstraint{r in Resources, e in Events}: Budget[r] >= sum{a
in Activities}(Demand[a,r]*z[e,a]) + sum{e1 in Events: e1<e} rescv[r,e1,e];

s.t. Thirteen_CMaxCalc{e in Events}: Cmax>= t[e] + sum{a in Activities}
z[e,a]*Duration[a];


solve;

display Cmax;
printf "Event           Activity      Resources 1    Resource 2 \n";
printf "-------------   -----------   ------------   -----------\n";
printf{e in Events, a in Activities: z[e,a] != 0}: "%13s   %11s       %11g
%11g \n", e,a,Demand[a,1], Demand[a,2];

printf "Times  t[e]  \n ";
printf {e in Events}: "%13s   %11g  \n", e,t[e];

printf "e               a             zBounds[e,a]      \n";
printf "-------------   -----------   ------------  \n";
printf{e in Events, a in Activities: zBounds[e,a] != 0}: "%13s   %11s     %11g
\n", e,a,zBounds[e,a];

data;

set IP :=
(1,3),(2,4),(3,7),(3,8),(4,5),(4,10),(5,12),(6,2),(7,2),(8,6),(8,9),(9,11),(10,
5),(11,10); #index pair of precedents

param Duration :=
1  0
2  7
3  3
4  5
5  5
6  6
7  4
8  5
9  4
10 3
11 7
12 0;

param Budget :=
1       3
2       3;

param Demand: 1 2 :=
1 0    0
2 0    2
3 2    1
4 3    3
```

```
5 3    2
6 2    1
7 1    0
8 1    3
9 1    1
10 1   1
11 3   1
12 0   0;

param : ActivityFront ActivityReverse :=
1 0 11
2 5 4
3 1 10
4 6 3
5 10 1
6 3 5
7 2 5
8 2 8
9 3 4
10 9 2
11 4 3
12 11 0;

param ES :=
1 0
2 14
3 0
4 21
5 29
6 8
7 3
8 3
9 8
10 26
11 12
12 34;

param LS :=
1 15
2 29
3 15
4 36
5 44
6 23
7 25
8 18
9 30
10 41
11 34
12 49;

end;
```

Appendix B – Test Data

Table 18 – Full Initial Test Results

Version	Network	Budget	Heuristic	Gap at 600 s	Obj	Best Bound	Time BB Found	Gap at End	Time
On/Off	1	1	0.2	51.3%	383	214	778	44.1%	3600
On/Off	1	2	0.2	30.7%	309	214	215	30.7%	3600
On/Off	1	3	0.2	20.7%	265	214	220	19.2%	3600
On/Off	1	4	0.2	8.9%	235	214	220	8.9%	3600
On/Off	1	5	0.2	0.0%	214	214	55	0.0%	58
On/Off	2	1	0.2	50.4%	359	178	259	50.4%	3600
On/Off	2	2	0.2	39.5%	294	178	185	39.5%	3600
On/Off	2	3	0.2	29.9%	254	178	715	29.9%	3600
On/Off	2	4	0.2	37.6%	225	161	3195	28.4%	3600
On/Off	2	5	0.2	11.9%	200	178	200	11.0%	3600
On/Off	3	1	0.2	51.3%	363	186	243	28.8%	3600
On/Off	3	2	0.2	36.3%	292	186	266	36.3%	3600
On/Off	3	3	0.2	33.3%	247	186	1906	24.7%	3600
On/Off	3	4	0.2	24.8%	226	170	1225	24.8%	3600
On/Off	3	5	0.2	15.5%	202	186	340	7.9%	3600
On/Off	4	1	0.2	86.2%	350	137	2862	60.9%	3600
On/Off	4	2	0.2	61.0%	275	115.5	430	58.0%	3600
On/Off	4	3	0.2	39.9%	225	137	27	39.1%	3600
On/Off	4	4	0.2	35.1%	205	137	43	33.2%	3600
On/Off	4	5	0.2	19.9%	167	137	53	18.0%	3600
On/Off	5	1	0.2	52.4%	409	194	95	52.6%	3600
On/Off	5	2	0.2	48.6%	313	194	850	38.0%	3600
On/Off	5	3	0.2	26.2%	250	194	500	22.4%	3600
On/Off	5	4	0.2	27.2%	233	177	200	24.0%	3600
On/Off	5	5	0.2	20.3%	219	177	435	19.2%	3600
On-Only	1	1	0.2	46.8%	391	214	11	45.3%	3600

Version	Network	Budget	Heuristic	Gap at 600 s	Obj	Best Bound	Time BB Found	Gap at End	Time
On-Only	1	2	0.2	30.3%	301	214	12	28.9%	3600
On-Only	1	3	0.2	19.5%	262	214	11	18.3%	3600
On-Only	1	4	0.2	4.9%	224	214	0	4.5%	3600
On-Only	1	5	0.2	2.3%	214	214	0	0.0%	1042
On-Only	2	1	0.2	51.1%	351	178	4	49.3%	3600
On-Only	2	2	0.2	40.5%	299	178	4	40.5%	3600
On-Only	2	3	0.2	35.3%	254	178	1	29.9%	3600
On-Only	2	4	0.2	21.6%	227	178	3	21.6%	3600
On-Only	2	5	0.2	11.0%	200	178	0	11.0%	3600
On-Only	3	1	0.2	48.8%	363	186	1	48.8%	3600
On-Only	3	2	0.2	37.2%	294	186	1	36.7%	3600
On-Only	3	3	0.2	24.1%	245	186	1	24.1%	3600
On-Only	3	4	0.2	17.0%	224	186	1	17.0%	3600
On-Only	3	5	0.2	11.4%	199	186	1	6.5%	3600

Version	Network	Budget	Heuristic	Gap at 600 s	Obj	Best Bound	Time BB Found	Gap at End	Time
On-Only	4	1	0.2	62.7%	351	137	14	51.0%	3600
On-Only	4	2	0.2	51.2%	263	137	7	47.9%	3600
On-Only	4	3	0.2	40.4%	223	137	8	38.6%	3600
On-Only	4	4	0.2	33.2%	196	137	6	30.1%	3600
On-Only	4	5	0.2	22.2%	164	137	9	16.5%	3600
On-Only	5	1	0.2	51.1%	397	194	5	51.1%	3600
On-Only	5	2	0.2	37.0%	308	194	5	37.0%	3600
On-Only	5	3	0.2	25.1%	258	194	6	25.1%	3600
On-Only	5	4	0.2	17.4%	231	194	4	16.0%	3600
On-Only	5	5	0.2	11.0%	218	194	4	11.0%	3600

Appendix C – Preprocessing Code

This appendix explains, in detail, the preprocessing code developed for this paper.

The code begins by initializing an array of durations (dur_array), where dur_array[i] corresponds to the duration of activity i+1, and an array of prerequisites (prereq_array), where prereq_array[i][0] stores a predecessor and prereq_array[i][1] stores a successor. The maximum time horizon is found as well and stored as maxLate. Initialization commands are seen below in Code Capture 1.

```
41      /*initialization commands */
42      for(i=0; i<ActivityNum; i++){
43          FNL_array[i] = 0;           //initializes array of node levels, to all activities have nodel level 0
44      }
45
46      for(i=0; i<ActivityNum; i++){
47          fscanf(fdurs, "%d %d", &temp, &dur_array[i]);   //initializes array of activity durations
48          maxLate = maxLate + dur_array[i];               //sum all durations; gives max makespan if everything is done one at a time
49      }
50
51      printf("%d\n", maxLate);
52
53      for(i=0; i<PairNum; i++){
54          fscanf(fprereqs, "%d %d", &prereq_array[i][0], &prereq_array[i][1]);    //initializes prereq array
55      }
56
57      for(i=0; i<ActivityNum; i++){
58          ES_array[i] = 0;            //initializes earliest start time of activity as 0
59          LS_array[i] = maxLate;      //initialize latest start time of activity as maxlate
60          FNA_array[i] = 0;           //initialize the number of activities at each node from the front as zero
61          RNA_array[i] = 0;           //initialize the number of activities at each node from the reverse as zero
62          FAC_array[i] = 0;           //initialize the number of activities required for an activity to start as zero
63          RAC_array[i] = 0;           //initialize the number of activities dependent on an activity as zero
64
65      }
66
```

Code Capture 1 – Pre-processing Code Initialization Commands, lines 41-66

Topological Sort

The first step to determine the general structure of the network was to build a basic topological sort code that could be built up to determine the earliest start times, latest start times, and activity counts. A topological sort is generally used in computer science to determine a feasible order of activities in a network – not an optimal order, but rather a list of activities that does not violate any precedence constraints.

At the time of developing the topological sort, node levels were being used to create the Z-Bounds parameter values in the On-Only model. It was later determined that activity counts would be able to limit the solution space further than node levels. However, node levels were still used for the basic topological sort and to determine the earliest and latest start times.

The node levels are defined as the number of levels deep into the network where an activity is located based on a string of activities that comes before it. This can be visualized most effectively in Figure 23. In the figure, nodes are represented by the circles at the bottom of the sample network diagram, and color-coded to the corresponding activity. For example, activity 10 is at node level 7 because the longest string of activities leading up to activity 10 is six activities long {1, 3, 8, 6, 2, 4}. By using the node levels, it is possible to create a topological sort that will give a single-dimension array that does not violate any precedence constraints; from the following sample network, the topological sort would return {1, 3, 8, 7, 9, 6, 11, 2, 4, 10, 5, 12}.

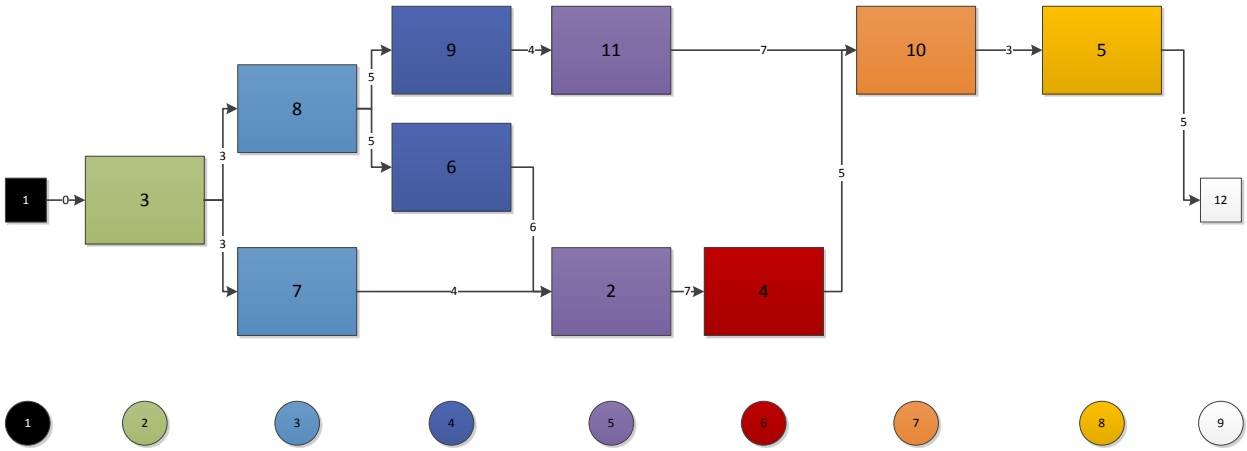

Figure 23 – Example of Node Levels (Forward)

Similarly, it is possible to build node levels in reverse – that is, what is the longest string of activities that is dependent on a given activity. As can be seen in Figure 24, the reverse node levels changes the position of some activities. In more complex networks, these shifts are even larger. For the reverse topological sort, an order that would not violate precedence constraints would be {1, 3, 8, 6, 7, 9, 2, 11, 4, 10, 5, 12}. While very similar to the forward topological sort, activities that could be done in parallel are switched.

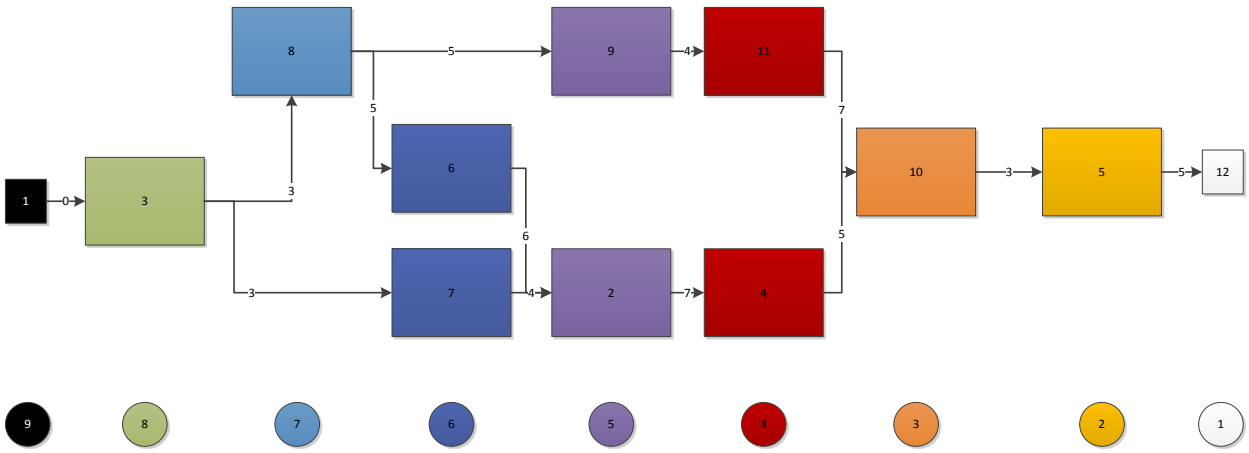

Figure 24 – Example of Node Levels (Reverse)

Earliest Start Times

Earliest start times were found in conjunction with the node levels and the topological sort. The algorithm begins by initializing all activities as having no predecessors (Code Capture 2, lines 68 to 70), and then sorting through the list of prerequisite pairs to assign a pred_exists value of 1 to any task that is in the "successor" column (Code Capture 2, lines 72 to 75). This isolates the starting task, which has no prerequisites as the only task with a pred_exists value of '0'. This starting task is assigned a node level of 1, and an earliest start time in the ES_array of '0' (Code Capture 2, lines 77 to 82).

```
67      /* beginning of Topological Sort for node levels & earliest start times */
68      for(i=0; i<ActivityNum; i++){
69          pred_exists[i] = 0;          //initializes array saying no predecessors exist
70      }
71
72      for(i=0; i<PairNum; i++){
73              successor = prereq_array[i][1];
74              pred_exists[successor-1] = 1;          //identifies tasks with prerequisites
75      }
76
77      for(i=0; i<ActivityNum; i++){
78          if(pred_exists[i] == 0){
79              FNL_array[i] = FnodeLevel;          //assigns tasks with no prerequisites a node level of "1"
80              ES_array[i] = 0;                    //assigns tasks with no prerequisites an Earliest Start time of "zero"
81          }
82      }
```

Code Capture 2 – Pre-processing Code, lines 67-82

Once the first node level has been initialized, a while-loop is started that will keep running as long as predecessors exist to be explored. This can be seen in Code Capture 3, lines 84-120. The algorithm resets all predecessors existing for a given activity to '0' (Code Capture 3, lines 86-88). Then it goes through all activities, and if it finds an activity at the given forward node level, it

goes through the prerequisite pairs to find if that activity has a successor. If it does, then the successor is assigned a pred_exists value of '1' (Code Capture 3, line 95). Lastly, if the duration of the predecessor activity being checked, plus the earliest start time of the predecessor activity, is greater than the earliest start time of the successor that has been found for that activity (Code Capture 3, line 97), then the earliest start time of the successor is set to the duration of the predecessor activity added to the earliest start time of the predecessor activity (line 98).

After the for loop in Code Capture 3 lines 90-103 has been executed, all successors that were found for activities at the current node level have a pred_exists value of 1. The current node level (stored at FnodeLevel) is increased by 1, and these successors are assigned the new node level (Code Capture 3, lines 106-110). Then, the indicator predsExist is initialized to '0' (line 112), and the pred_exists array is checked. If there are any successors left to check, predsExist is set to '1', and the while-loop repeats with the next node level. Once all activities have been checked and earliest start times assigned, the predsExist value will remain at '0', and the while-loop will exit. This ends the topological sort for node levels and earliest start times.

```
84  while(predsExist == 1){
85
86      for(i=0; i<ActivityNum; i++){
87          pred_exists[i] = 0;          //resets predecessor exists to negatory
88      }
89
90      for(i=0; i<ActivityNum; i++){
91          if(FNL_array[i] == FnodeLevel){          //finds tasks with node level equal to what we are currently checking (first loop, nodeLevel = 1)
92              for(j=0; j<PairNum; j++){
93                  if(prereq_array[j][0] == i+1){      //goes through prereq array for tasks with node level equal to curent nodeLevel;
94                      successor = prereq_array[j][1]; //finds successors to tasks at current nodeLevel
95                      pred_exists[successor-1] = 1;       //assigns pred-exists to be 1
96
97                      if((dur_array[i] + ES_array[i]) > ES_array[successor-1]){       //if ("current task" + "duration of current task") is greater than current time to successor
98                          ES_array[successor-1] = dur_array[i] + ES_array[i];         //set the "time to the current successor" to ("time to the current task" + "duration of current task
99                      }
100                 }
101             }
102         }
103     }
104     FnodeLevel++;           //increase nodeLevel
105
106     for(i=0; i<ActivityNum; i++){
107         if(pred_exists[i] == 1){
108             FNL_array[i] = FnodeLevel;              //assigns tasks with prerequisites at node level 1, a node level of 2
109         }
110     }
111
112     predsExist = 0;         //initialize indicator
113
114     for(i=0; i<ActivityNum; i++){
115         if(pred_exists[i] == 1){                //if there is a precedence left to accomodate,
116             predsExist = 1;                     //predecessors still exist, keep noding
117         }
118     }
119
120 }
121 /*End Topological Sort for node levels & earliest start times */
```

Code Capture 3 – Pre-processing Code, lines 84-121

Latest Start Times

The latest start times topological sort works similarly to the earliest start times, except starting with the tasks that have no successors, and therefore checking for predecessors instead of successors. Code Capture 4 is comparable to Code Capture 2, as it initializes the suc_exists array to no successors existing, identifies tasks with successors, then assigns tasks with no successors

the maximum node level and a latest start time of the time horizon (stored at maxLate), minus the duration of that activity.

```
167     /*Beginning of sort for latest start times */
168     for(i=0; i<ActivityNum; i++){
169         suc_exists[i] = 0;          //initializes array saying no successors exist
170     }
171
172     for(i=0; i<PairNum; i++){
173             predecessor = prereq_array[i][0];
174             suc_exists[predecessor-1] = 1;          //identifies tasks with successors
175     }
176
177     for(i=0; i<ActivityNum; i++){
178         if(suc_exists[i] == 0){
179             RNL_array[i] = RnodeLevel;              //assigns tasks with no successors a node level of maxNode (carried over from the earliest start time sort
180             LS_array[i] = maxLate - dur_array[i];   //assigns tasks with no successors a latest start time of (maxLate - duration of that activity)
181         }
182     }
```

Code Capture 4 – Pre-processing Code, lines 167-182

Similarly, Code Capture 5 is comparable to Code Capture 3; a while-loop finds activities at each node starting at the maxNode, and finds predecessors for these activities, assigns them a node level one less than the current node level, and checks if the latest start time should be updated. Once there are no more predecessors, the network has been mapped, and the while-loop exits.

```
184     while(sucsExist == 1){
185
186         for(i=0; i<ActivityNum; i++){
187             suc_exists[i] = 0;          //resets successor exists to negatory
188         }
189
190         for(i=0; i<ActivityNum; i++){
191             if(FNL_array[i] == FnodeLevel){         //finds tasks with node level equal to what we are currently checking (first loop, maxNode)
192                 for(j=0; j<PairNum; j++){
193                     if(prereq_array[j][1] == i+1){          //goes through prereq array for tasks with node level equal to curent nodeLevel; has to be i+1 because count starts at 0
194                         predecessor = prereq_array[j][0];   //finds predecessor to tasks at current nodeLevel
195                         suc_exists[predecessor-1] = 1;      //assigns suc_exists to be 1
196
197                         if((LS_array[i] - dur_array[predecessor-1]) < LS_array[predecessor-1]){
198                             LS_array[predecessor-1] = LS_array[i] - dur_array[predecessor-1];
199                         }
200                     }
201                 }
202             }
203         }
204
205         FnodeLevel--;           //decrease nodeLevel
206         RnodeLevel++;           //increase reverse nodeLevel Count
207
208         for(i=0; i<ActivityNum; i++){
209             if(suc_exists[i] == 1){
210                 RNL_array[i] = RnodeLevel;
211             }
212         }
213
214         sucsExist = 0;          //initialize indicator
215
216         for(i=0; i<ActivityNum; i++){
217             if(suc_exists[i] == 1){             //if there is a precedence left to accomodate,
218                 sucsExist = 1;                      //predecessors still exist, keep noding
219             }
220         }
221     }
```

Code Capture 5 – Pre-processing Code, lines 184-222

Activity Counts

Originally, the code counted the number of activities in the network at a given node. This was meant to reduce the solution space through the use of Z-Bounds; however, as mentioned in section 3.2.5, the Z-Bounds were changed to be dependent on activity counts.

To find the activity counts from the preprocessing code proved trickier than expected; no algorithms could be easily found to mimic the process. An iterative, rather than a recursive, algorithm was desired so as to limit issues with data handling and passing pointers through the code. The resulting algorithm may be seen in Code Capture 6.

The for-loop (lines 239-269) initializes a “prime” activity, indicated by “m”. For a given prime, the found and pending arrays are initialized to zero, with the exception of the prime activity which is pending exploration. A while-loop (lines 246-268) is then used to run through all the activities leading up to the prime activity. Any “pending” activities are searched for predecessors through the prereq_array (line 251). If a predecessor is found, then the algorithm checks if this predecessor has been found before (line 252). If the predecessor has not been found before, it is deemed to be pending (line 253) and found (line 254), and the forward activity count for the prime activity is increased (line 255). The successor activity that was originally found is now no longer pending (line 260). If there are no more activities pending, all activities that are required for the prime activity have been found, and the while-loop exits, which causes the for-loop to move to the next prime activity.

```
236     int complete, prereqFound, currTest, k, l, m;
237     int found[ActivityNum], pending[ActivityNum];
238
239     for(m=0; m<ActivityNum; m++){                    //run through this for each activity, called the "prime" activity for this loop
240         for(l=0; l<ActivityNum;l++){                 //array of "found" activities for a given prime is initialized to zero
241             found[l] = 0;
242             pending[l] = 0;
243         }
244         pending[m] = 1;
245         complete = 0;
246         while(complete==0){                  //Loop the loop
247
248             for(j=0;j<ActivityNum;j++){
249                 if(pending[j] == 1){
250                     for(k=0; k<PairNum; k++){                //run through all the prereq pairs again
251                         if(prereq_array[k][1] == j+1){       //if the activity you are currently testing is in the second column, you want its prereq
252                             if(found[(prereq_array[k][0]-1)] != 1){ //you have found prereq and counted this one before
253                                 pending[prereq_array[k][0]-1] = 1;                          //this is in line to be looked at
254                                 found[prereq_array[k][0]-1] = 1;                                    //track you found this prereq
255                                 FAC_array[m] = FAC_array[m]+1;  //increase forward activity count
256                             }
257                         }
258                     }
259                 }
260                 pending[j] = 0;
261             }
262             complete = 1;
263             for(l=0;l<ActivityNum;l++){
264                 if(pending[l] == 1){
265                     complete = 0;
266                 }
267             }
268         }
269     }
```

Code Capture 6 – Pre-processing Code, lines 236-269

The reverse activity counts are found in a similar fashion as seen in Code Capture 7. This time, the while-loop checks for successors rather than predecessors, and counts the number of successors required for the given prime activity.

```
273     for(m=0; m<ActivityNum; m++){                    //run through this for each activity, called the "prime" activity for this loop
274         for(l=0; l<ActivityNum;l++){                 //array of "found" activities for a given prime is initialized to zero
275             found[l] = 0;
276             pending[l] = 0;
277         }
278         pending[m] = 1;
279         complete = 0;
280         while(complete==0){                  //loop the loop
281
282             for(j=0;j<ActivityNum;j++){
283                 if(pending[j] == 1){
284                     for(k=0; k<PairNum; k++){                //run through all the prereq pairs again
285                         if(prereq_array[k][0] == j+1){       //if the activity you are currently testing is in the second column, you want its prereq
286                             if(found[(prereq_array[k][1]-1)] != 1){ //you have not found prereq and counted this one before
287                                 pending[prereq_array[k][1]-1] = 1;                      //this is in line to be looked at
288                                 found[prereq_array[k][1]-1] = 1;                                //track you found this prereq
289                                 RAC_array[m] = RAC_array[m]+1;  //increase forward activity count
290                             }
291                         }
292                     }
293                 }
294                 pending[j] = 0;
295             }
296             complete = 1;
297             for(l=0;l<ActivityNum;l++){
298                 if(pending[l] == 1){
299                     complete = 0;
300                 }
301             }
302         }
303     }
```

Code Capture 7 – Pre-processing Code, lines 273-303

Appendix D – Test Network Diagrams

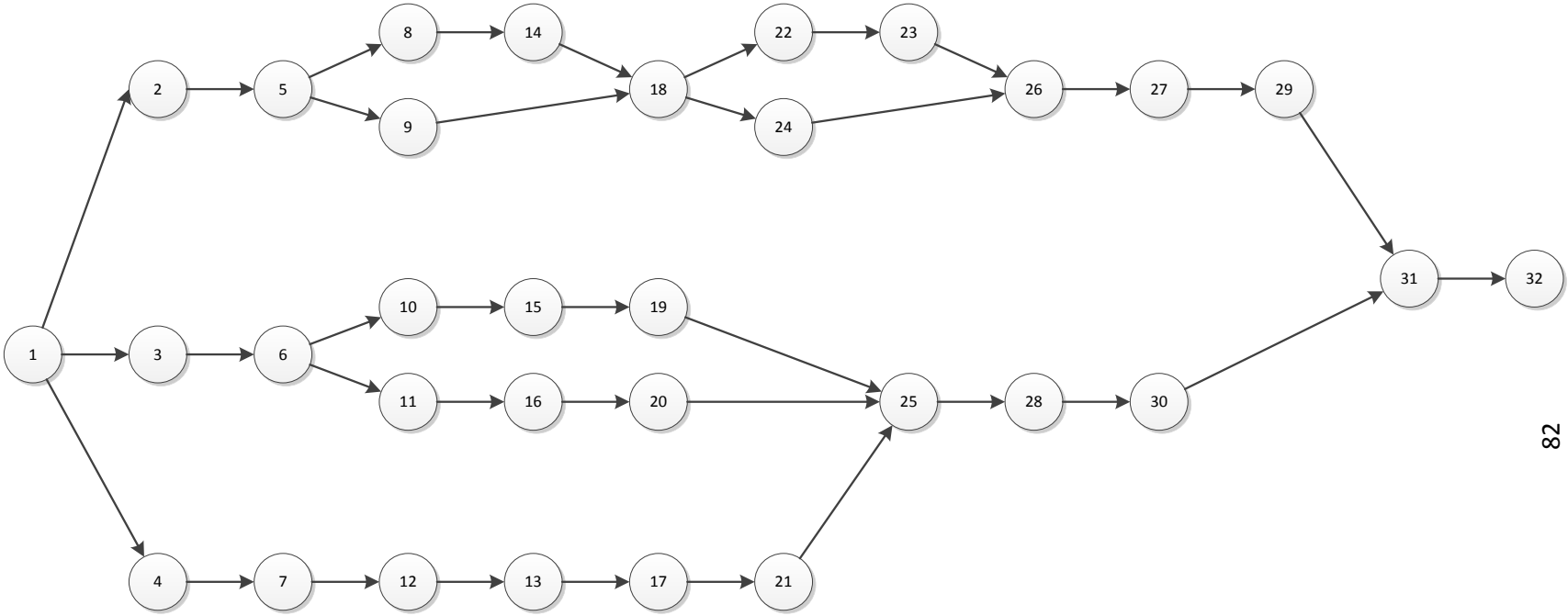

Figure 25 – Diagram of Test Network 1

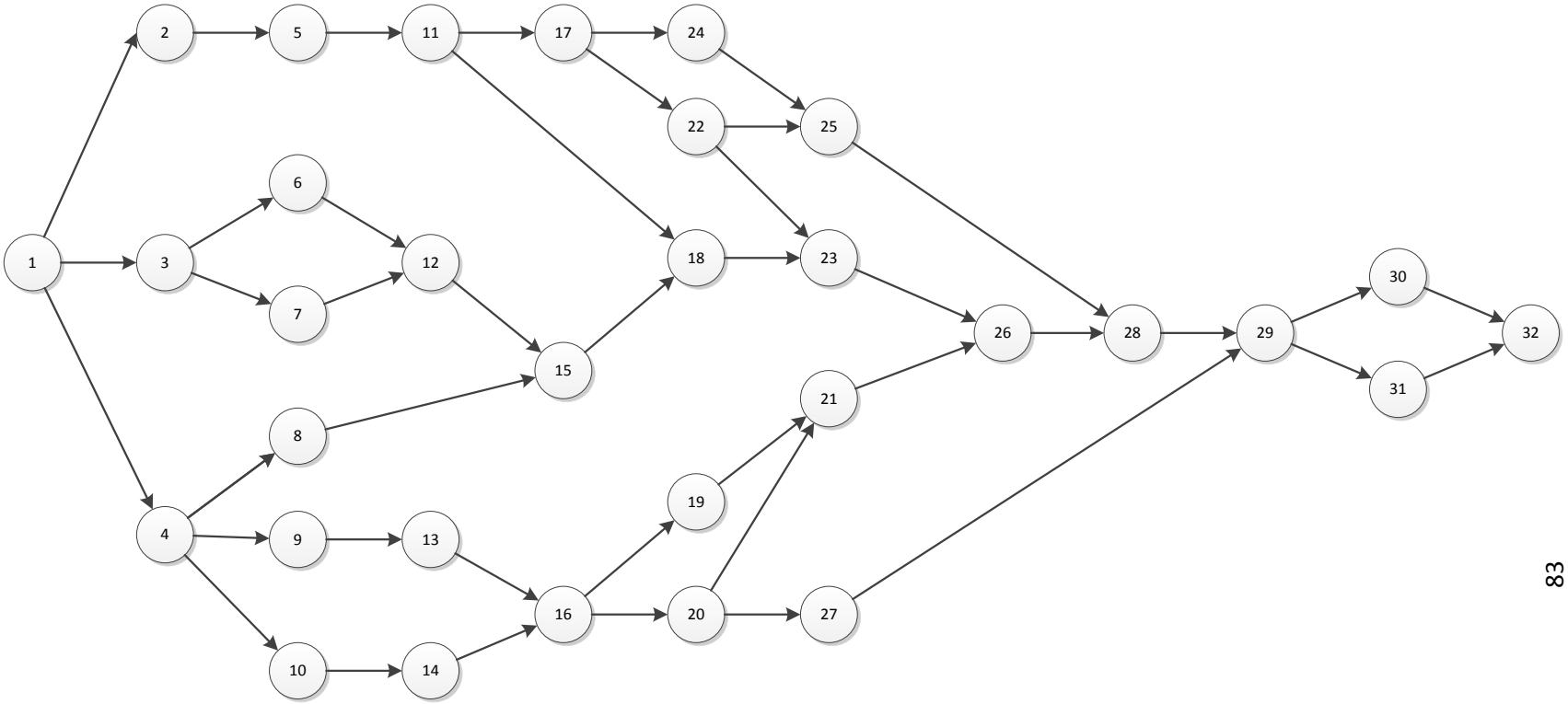

Figure 26 – Diagram of Test Network 2

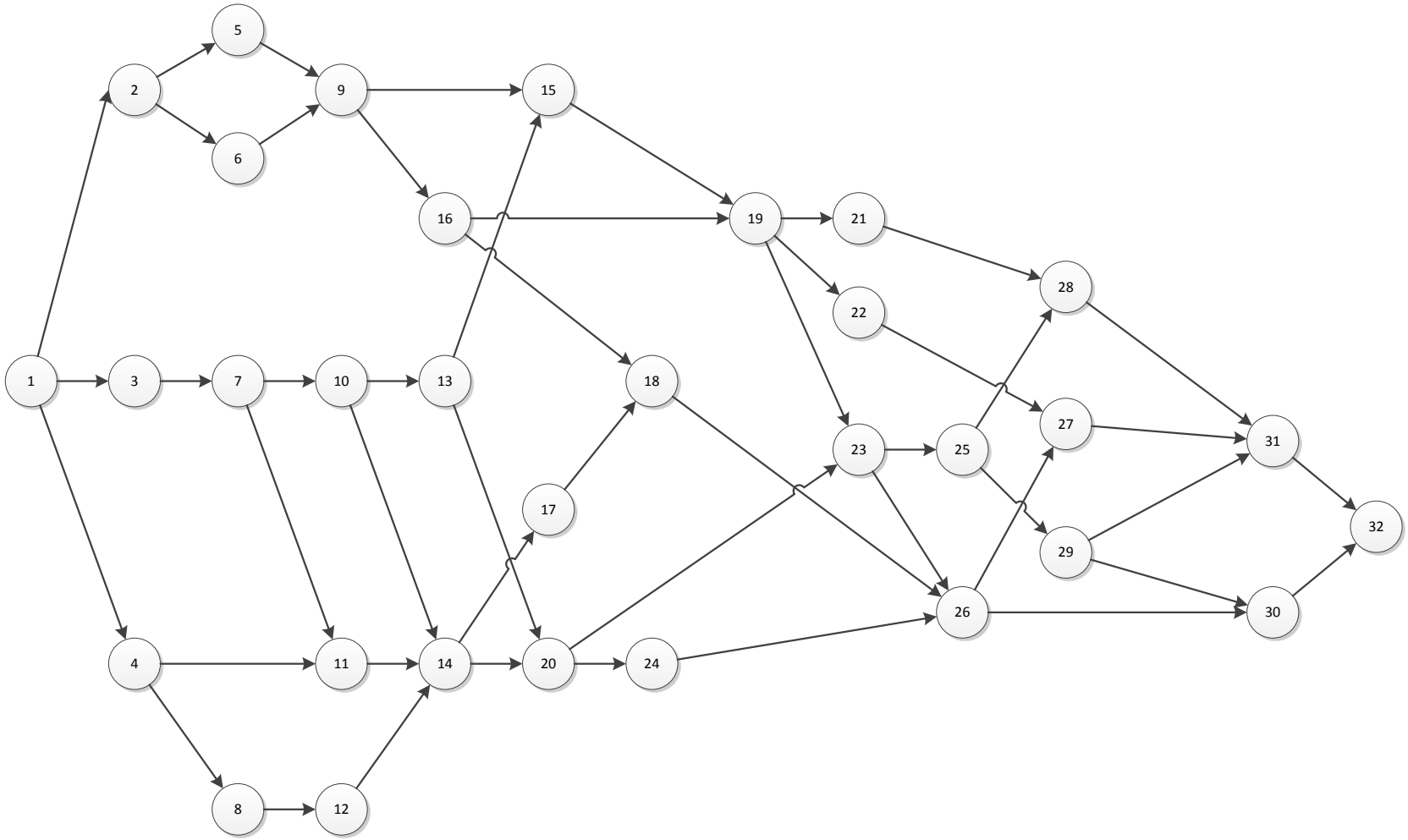

Figure 27 – Diagram of Test Network 3

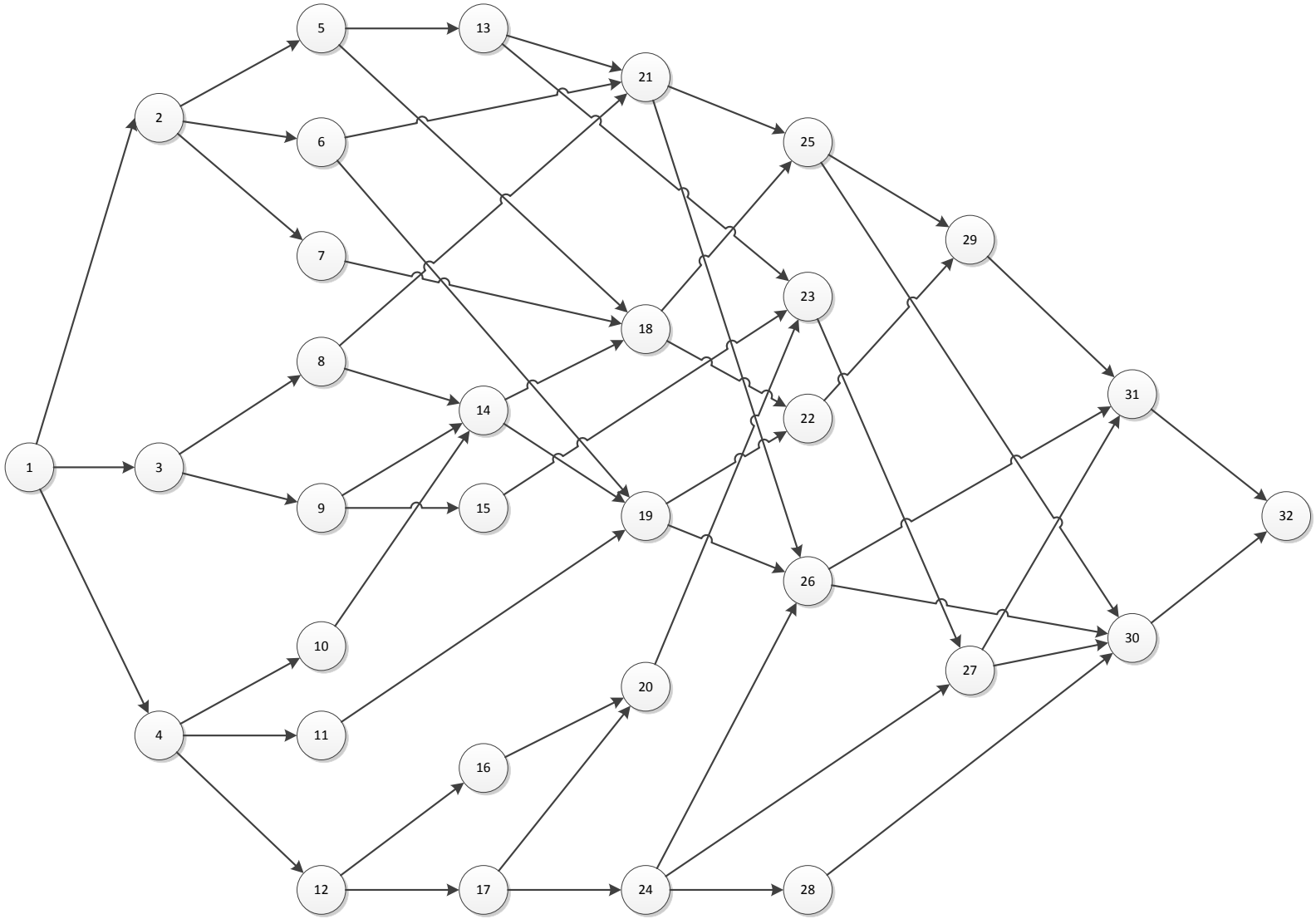

Figure 28 – Diagram of Test Network 4

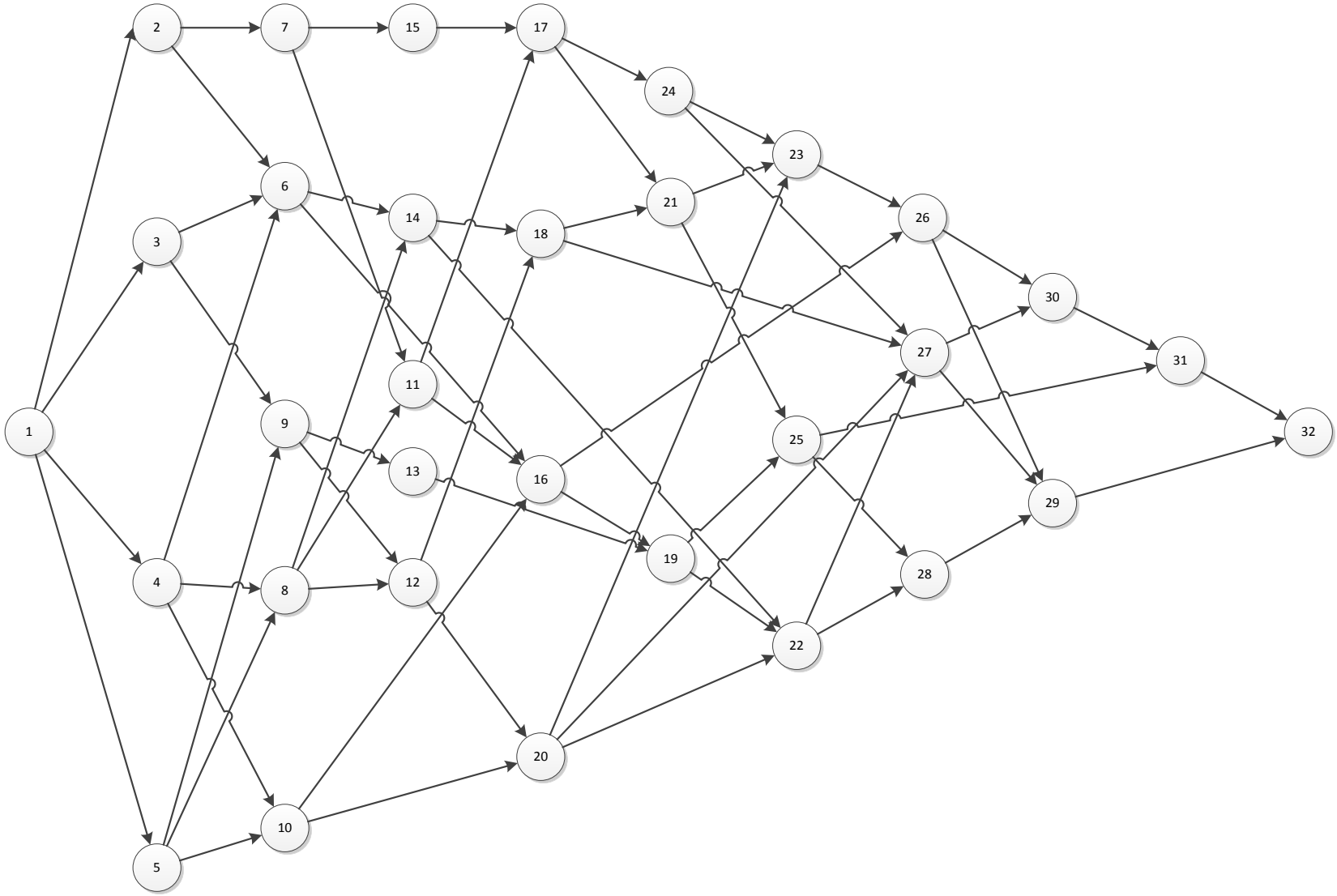

Figure 29 – Diagram of Test Network 5