AN APPLICATION OF MATHEMATICAL OPTIMIZATION TO AUTOCLAVE PACKING AND SCHEDULING IN A COMPOSITES MANUFACTURING FACILITY

by

Andres Collart

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
September 2015

Table of Contents

List of Tables

List of Figures

Abstract

An industry problem regarding autoclave packing and scheduling in a composites manufacturing facility is examined. The hill-climbing method is modified and applied to solving the packing of autoclaves, which is a modified bin packing problem. A mixed-integer linear program is formulated and applied to solve the scheduling of autoclave loads throughout the planning horizon. The scheduling problem takes into account utilization of resources at steps prior to the curing process, autoclave blackout times for anticipated downtime, weekend overtime, tool desired recycle times, incentivizes the creation of usable gaps, and minimizes peak power consumption. Both the autoclave packing problem and the scheduling problem are combined in a software which has been tested and delivered to industry. Solution of the mixed-integer linear program makes use of cloud technology to allow for flexible licensing options of an expensive state-of-the-art mathematical solver.

List of Abbreviations and Symbols Used

Abbreviation/Symbol:	Definition:
$\forall$	For every
$\sum$	Sum
AOG	Aircraft on Ground
API	Application Programming Interface
APP	Autoclave Packing Problem
ASTA	Aerospace Technologies of Australia
AWS	Amazon Web Services
BPP	Bin Packing Problem
ConWIP	Constant Work In Process
GA	Genetic Algorithms
KW	Kilowatt
MILP	Mixed-Integer Linear Programming
MIP	Mixed-Integer Program
MRP	Materials Requirements Planning
NP-Complete	Non-Polynomial Complete
PM	Program Manager
POA	Plane on Assembly
PP	Production Planner
PPL	Program Production Leader
RTN	Resource-Task Network
SBP	Scheduling of the Batch Processor
SPT	Shortest Processing Time
STN	State-Task Network
TIP	The Industry Partner
TOC	Theory of Constraints

TSP	Travelling Salesman Problem
WIP	Work In Process

Acknowledgements

First I would like to thank my parents, Juan Ramon Collart and Vera Daetz de Collart, for their unending support and encouragement throughout my life and academic pursuits. Without their guidance and support, I would not have dreamed of completing a Bachelors and Masters of Industrial Engineering in Canada.

My friends and family have also played a very important role in helping me complete this work, especially by balancing the provision of distractions to maintain my sanity and understanding when I cannot be there for them. My girlfriend, Melanie Gillis, has been the leader in this role and I thank her endlessly.

Dr. Eldon Gunn, my supervisor, has been a great source of knowledge and help throughout the last year. His wise words, experiences, and knowledge have taught me so much and helped me navigate many obstacles during the completion of this thesis.

From the industry side, both Richard Ernst and Mario Stevaux provided great support for the project. Richard helped compile large databases, explained the basics of autoclave packing and scheduling to me, and sifted through many potential schedules to help determine if they would work or not. Mario helped provide the management support for this project and aided in providing the right resources for implementation.

Finally I would like to thank my examination committee comprised of Dr. Eldon Gunn, Dr. Corinne MacDonald, Dr. Uday Venkatadri, and Dr. M. Ali Ülkü for taking the time to read through and with your comments and suggestions help refine the final version of this thesis.

I would also like to acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through an Engage grant awarded to Dr. Eldon Gunn.

Chapter 1 Introduction

The research in this thesis was completed in the context of an industry partner who for confidentiality will remain anonymous and will be referred to as The Industry Partner (TIP). TIP manufactures composite parts for the aerospace industry. Composite parts are made up of several layers of different materials bound together by resin. For the resin to bind the layers both heat and pressure must be applied for several hours, this process is called curing and is done via the use of specialized machines called autoclaves. The size, initial cost, and energy usage of autoclaves make them some of the most expensive pieces of equipment to run in the facility.

This thesis will examine the processes surrounding the curing of composite parts in the aerospace industry and then analyze the objectives regarding autoclave packing and scheduling. A heuristic is developed to tackle the Autoclave Packing Problem (APP) and is followed by a mixed-integer program (MIP) to handle the scheduling side of the problem. From the literature examined in section 2.0, this problem has never been tackled to this extent before. This research brings together more knowledge about the APP than previous attempts at solving it, develops an MIP to solve the scheduling problem, and makes use of emerging cloud technologies to deliver an implementable solution. The solution was delivered to TIP in the form of a program that could solve their autoclave packing and scheduling problem. Computational results for a test case of both the packing and scheduling algorithms are shown and discussed. From the work shown here it will be evident that autoclave packing and scheduling is a very difficult problem to tackle, however with the right tools and approaches it is possible to yield better results than experienced human operators.

1.1 Background

TIP manufactures composite parts for the aerospace industry. Their customers include some of the largest aerospace companies in the world. They produce items such as wing tips, flaps, nose shells, and other structural and fuselage components. Their products are mostly made from carbon fiber and resin, which requires a curing process to be fully fused together and harden. This curing process takes place in autoclaves or ovens, although for the purposes of this study the ovens were ignored. Awaiting increases in demand, and realizing that their autoclaves are currently near maximum capacity while running under the current processes, TIP had requested help in improving the efficiency of their curing process. First a project between Dalhousie, two Industrial Engineering final year students, and TIP was carried out (Devine & Milne, 2014). This initial study determined that the autoclaves were the bottleneck of their operation. They also mapped out the process and recommended to change from a push system to a pull system. To aid in this process they provided a manual scheduling tool that helped a planner schedule the autoclaves and determine which parts that go inside them easily.

Figure 1: Large autoclave at TIP (TIP, 2015)

Autoclaves are large pressure and heat vessels, an example from TIP is shown in Figure 1. Every part requires a specific heat and pressure profile to cure, this is commonly referred to as a recipe. A typical heat profile is shown in Figure 2. Parts for similar applications usually follow the same recipe and are cured together as long as they fit within the physical and other constraints of the autoclave.

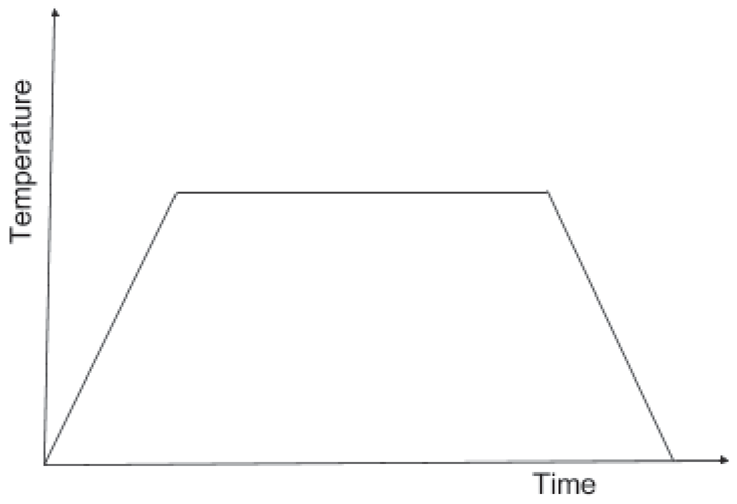


Figure 2: Typical Heat Profile for Autoclave Cure

The process of producing a composite part starts in the Kit Cutting department where rolls of several materials, mostly carbon fiber, are cut to form the different layers of a part. Layers that form a single part are packed together as a kit. These kits then are transferred to the layup department responsible for the customer who ordered that part. Experienced layup personnel trained in that specific part's manufacturing then lay up the different layers of the part onto a mould. These moulds will follow the part through the curing centre until they reach demoulding. Then when enough parts to fill an autoclave load have been laid up they are sent to the autoclave centre and get loaded together into an autoclave. They cure with the same recipe for several hours in the autoclave and then proceed to demoulding where the cured part and the mould are separated. The mould is cleaned and made available for another part to be laid up on it.

This process as described follows a push system, where parts move on to the next station as they are completed. This has been the cause, in large part, of inefficiencies in processes downstream. For example, if parts arrive at the autoclave department sporadically they may get cured in several batches. Each cure costs money due to heating up the autoclave, as well as often underutilizing capacity of the autoclave. The cost to heat up an autoclave is relatively fixed to a recipe and autoclave, mostly independent from the number of parts in the cure. This is due to the mass of the autoclave being much larger than the mass of the parts as well as the cost of pressurizing a certain volume being fixed. Therefore reducing the number of cures and making more effective use of the capacity for each cure has the potential to significantly reduce the overall cost to produce the parts and to increase the throughput capacity of the plant.

1.2 Production of Composites

Here we give a brief description of the manufacturing process for advanced composites. This description is mostly based on Gutowski (1997). Other good references include Harper (1996) and Mazumdar (2001). There are several methods of manufacturing composites. The first difference is in the laying of the fiber structure that makes up the part. The two major variants are linear or interlaced laying of the fiber. The linear structure is laid up sheets of aligned fibers that are usually impregnated with a resin. The interlaced structure is produced through a weaving process and can produce 2D or 3D structures. In the process considered in this project only the linear structure is found, thus the explanation of the interlaced structure manufacturing process will not be part of this report. However a good explanation the interlaced structure process can be found in Gutowski (1997)

The second difference in manufacturing composites is the wetting of the fiber structure. This can be either online or offline and constitutes when the resin is placed into the fiber. Off-line wetting is most common at TIP, although some on-line wetting does take place for some materials. Materials that go through the off-line wetting process are usually called "prepregs" for "pre impregnated". These materials must be refrigerated until they are used, given that they have a limited material life.

The third critical difference is thermoset and thermoplastic parts. Thermoset parts require heat to trigger a chemical reaction that solidifies the parts. Thermoplastic parts on the other hand solidify via a cooling process. All parts at TIP are thermoset parts and use autoclaves to provide the necessary heat and pressure for the chemical reaction to take place.

The several layers of fiber structures are laid up on tools by hand. Ply material properties are usually not isotropic, therefore layers are laid in different directions to account for the material properties. The most dimensionally critical side of the part is laid on the tool face, while the other side is covered by a vacuum bag that aids in reducing voids during the curing process by applying uniform pressure as is shown in Figure 3. Once laid up, the part and tool are placed inside an autoclave to cure. This triggers a chemical reaction in thermoset parts.

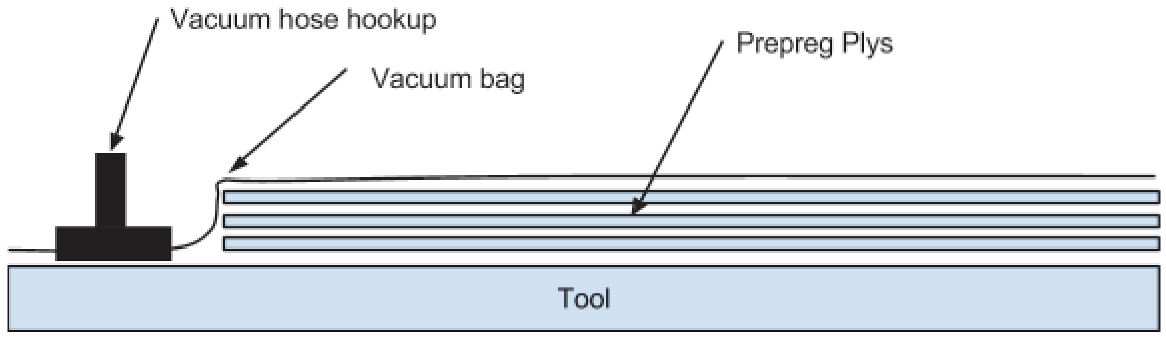


Figure 3: Simplified Tool Schematic

1.3 Production Process at TIP

Using a linear thermoset pre-preg material, the process found in Figure 4 illustrates that used at the TIP manufacturing facility. The sheets of material are cut into kits at the kit-cutting operation. Each kit contains the different layers required for a part. The roll of material is taken out of the freezer to thaw and that is when the material shelf life begins. When the shelf life runs out, the material is obsolete if it has not been cured. The material is then cut into the kits and the kits are sent to the layup room.

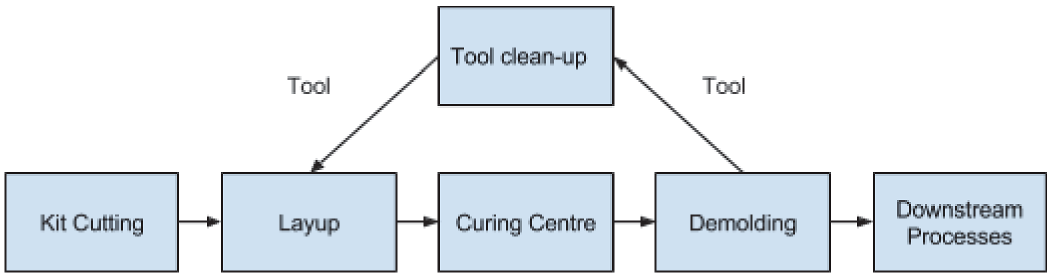


Figure 4: General Workflow of Plant Processes

Then the hand-layup and curing process takes place. The main steps here are: cleaning of layup tool surface, apply the different layers that make up the part, attach vacuum bag and seal, load tool into autoclave, turn on autoclave, wait for cycle, turn off autoclave, unload from autoclave. Resources for a more complete and detailed list of hand-layup and autoclave processes are available in literature (Harper, 1996). Finally the part is removed from the tool (demolded) and the tool is cleaned. The cured part then follows downstream processes, such as trimming, drilling, painting, assembly and testing before being shipped off to the customer (Devine & Milne, 2014).

At TIP there are several autoclaves of widely varying size, and given a diverse customer mix there is a also a wide variety of parts. There are also several layup rooms, with each layup room serving a different customer. Parts vary in several aspects including size, recipe, cure time, material, as well as the number of cures in a few occasions. Some parts are two-stage cures, where a part is laid up and is cured, followed by another layup and a final cure. These kinds of operations are called re-entrant from the autoclave perspective since the part enters the autoclave two times. They are not common at TIP and are almost exclusively restricted to one layup room which has very stable demand. Due to the complexity surrounding these, as well as the stable demand, there are cure slots scheduled permanently for these parts several times per week.

Currently the process of scheduling the autoclaves actually starts with layup Program Production Leaders (PPLs) and Program Managers (PMs) agreeing on an acceptance of the workload. Each program has a Production Planner (PP) who schedules the work for their own areas independently. Simultaneously, the Curing PPL begins assigning cure slots for the upcoming week based on meeting the requirements for the top priority programs as identified by the Production Director. While determining cure slots care is taken to stagger cure start times whenever possible in order to reduce peak power consumption. If possible usable gaps, unscheduled slots of 4 or more hours, are also left to provide a slot in case later changes must be made. A draft cure schedule is submitted to all PPs and PPLs (by Wednesday) and a meeting is scheduled to allocate any remaining cure slots (no later than Friday). Conflicts are dealt with by discussion and a schedule is finalized at this meeting. However, the schedule continues to change as layup PPLs approach the autoclave PPL throughout the week to make additional changes. Some reasons for the changes are as follows: tooling issues, kitting/material issues, attendance or manpower issues, work delays, work completed early, scrap replacement, Critical Parts, Aircraft on Ground (AOG), Plane on assembly (POA), material outlife issues or material testing. A similar but separate meeting happens with the kit-cutting department to ensure material flows into the layup rooms on time. An Aircraft On Ground (AOG) situation occurs when a part must be produced urgently to accommodate an aircraft that is waiting for repair; a POA is a similar situation related to the assembly of new aircraft. These are rare, but production of any AOG/POA part is critical and supplants any production schedule. Cures in an autoclave will not be stopped (preemption is not allowed), but any cure slot that is needed will be taken to service the AOG/POA part. Critical Parts are identified (tagged) during the production sequence. There are many reasons why a part can become critical; usually it involves an elevated Internal or External need for the part. In most cases Critical Parts will not supplant any cure schedule, but they are processed as first priority throughout other areas of production.

Given the current process, the layup rooms ensure first that they have correctly levelled their manpower requirements for every day. This sometimes results in less than full loads in the autoclaves, resulting in more cures throughout the week. Any change to scheduling the autoclave cures based first on full loads must also take into account the manpower levelling requirements at the layup rooms.

1.4 Previous Work at TIP

A 5th year Industrial Engineering group worked on an earlier version of this research (Devine & Milne, 2014). This study helped to formally identify the problem as well as many constraints and objectives. The team identified the autoclave centre as the bottleneck of the operation and provided a semi-automatic Excel tool that allows the user to manually add parts to a bin and determine when that bin will be cured. As parts are added from the demand list, the list of parts that can also be added to that same cure is reduced based on the packing constraints (recipe, volume, etc.). Several statistics can be tracked for the bins and cure schedule in general.

A key recommendation from the team's report is to move from a push system to a pull system that is driven by the cure centre. The recommendation is due to the cure centre being the most expensive operation as well as the bottleneck of the plant. They also state that a mathematical model could better optimize the process, which is in part the reason for the research included in this thesis. Moving to a pull system would require the creation of batches of parts to be cured together as well as scheduling these batches in a manner that is more favorable to the efficiency of the autoclave centre.

1.5 Thesis Outline and Results

Making the appropriate use of the autoclaves through load creation and load scheduling is the focus of this thesis, it is organized as follows. Chapter 2 shows a collection of relevant literature to the solution of the autoclave packing and scheduling problem. It presents brief explanations of relevant solution approaches including mixed-integer linear programming and metaheuristics. Previous work relating to scheduling of autoclaves is also presented.

Chapter 3 presents the reader with two solution approaches to the autoclave packing and scheduling problem, the ideal and the proposed. The ideal solution approach is unobtainable due to several factors explained, and therefore the proposed model is presented. Solutions for the packing and scheduling portions of the problem are presented separately.

Chapter 4 discusses the information, features, and software structure required to implement the proposed model at TIP. Information regarding all the inputs to the program, assumptions, and modifications to data are presented. The output from the software is then presented. An overview of how both cloud computing and flexible licensing of the mathematical solver are used in this thesis is also shown in this chapter.

Chapter 5 displays the computational results from the software when used with real data from TIP. The data itself is hidden due to confidentiality, however the performance of the model running under different parameters is examined and the ideal parameters to use are selected.

Finally, chapter 6 presents the conclusion. Section 6.1 includes a list of future projects that could be completed to improve the performance and expand the scope of the optimization models. Sections of code for the proposed model are found in the Appendices section.

Chapter 2 Literature Review

This chapter will provide an overview of some literature relevant to the autoclave packing and scheduling problem. Section 2.1 provides an introduction to Theory of Constraints, an idea which shows why improvement at the curing centre, the bottleneck at TIP, is necessary. Section 2.2 will discuss the reasoning behind production planning and examine push and pull systems. Section 2.3 will discuss literature relating to the general scheduling problem, showing the different methods of dealing with material balances and time representations. It will also explain some common scheduling objectives. Section 2.4 will discuss the Bin Packing Problem. Section 2.5 will discuss the Batching Problem. Section 2.6 and section 2.7 give a brief introduction to mixed-integer linear programming and meta-heuristics since these are tools used in solving the autoclave packing and scheduling problem. Section 2.8 shows some of the common mathematical programming languages available which are used to describe mixed-integer linear programs. Section 2.9 then presents a few of the most prominent commercial and noncommercial mathematical solvers used to solve mixed-integer linear programs and shows a performance comparison between them. Section 2.10 explains the concept behind cloud computing and how some mathematical solvers use it to provide flexible licensing options. Finally, section 2.11 presents previous work done in the field of the autoclave packing and scheduling.

2.1 Theory of Constraints

The theory of constraints, proposed by Goldratt, Cox, and Whitford, is centered around the notion that every organization has at least one constraint that is preventing it from making more of what it needs to make, usually profit (Goldratt, Cox, & Whitford, 2004). Constraints can be in anything from demand and sales department to manufacturing and supply chain. It provides a five step process to improve the performance of organizations based on this theory. A key piece of insight provided is that "an hour lost at a bottleneck is an hour lost for the entire system" which serves to elevate the importance of improving the constraint operation as much as possible.

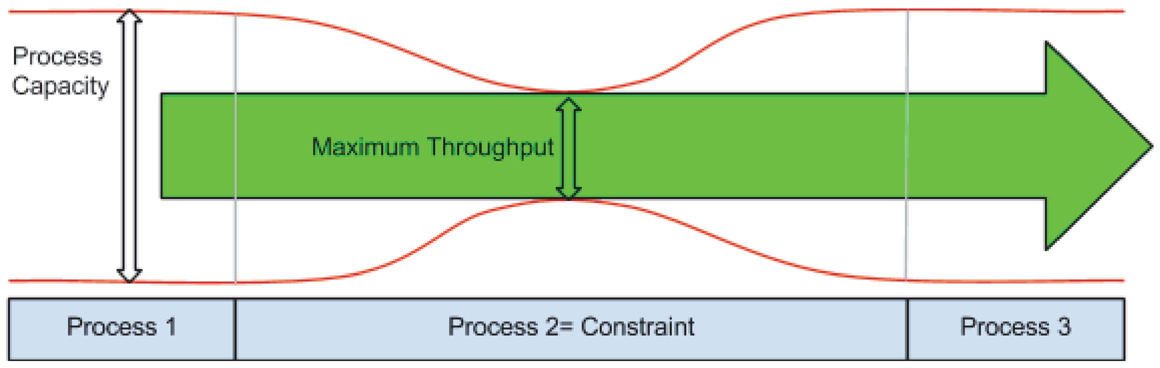


Figure 5: Illustration of Bottleneck Process Dictating Maximum Throughput

The five step process designed by Goldratt, Cox, and Whitford and explained in their book "The Goal" is as follows:

1. Identify the system's bottlenecks.
    a. In Figure 5 this would be Process 2.
2. Decide how to exploit the bottlenecks.
    a. This could be by reducing or eliminating downtime or making the process more efficient in some way.
3. Subordinate everything else to the above decision.
    a. All other non-constraint processes should behave to support the maximum efficiency of the constraint process.
4. Elevate the system's bottlenecks.
    a. If steps 2 and 3 have not relieved the bottleneck then adding more capacity or substituting the process may be required. These are more expensive and time-consuming improvements.
5. If, in a previous step, a bottleneck has been broken go back to step 1.
    a. Once a bottleneck has been broken a new one surfaces, to continuously improve the next bottleneck should be identified and the process repeated.

One method of subordinating non-constraints to a constraint process is to shift from pushing work to the constraint process to having the constraint process pull the work. However, this also involves pushing of work from the constraint downstream. This is explained further in the next section.

2.2 Production Planning

Production planning is needed to coordinate the different processes, inventories, and resources that work together to form a final product. The proper coordination of these resources leads to lowered costs and a production process that is better able to meet customer expectations. Anthony's hierarchy splits managerial activities into three areas: strategic planning, tactical planning, and operational control (Silver et al., 1998). When applied to a production system, the strategic planning area takes a very long term view (2+ years) and generally accomplishes decisions surrounding acquisition of resources. The tactical planning area plans for utilization of resources and typically makes decisions for a 6 to 24 month planning horizon. The operational control area makes decisions for a very short term planning horizon that are very detailed, such as individual resource scheduling. This final area of operational control is the focus of this thesis; therefore, this section will explore short-term production planning and control systems.

2.2.1 Push and Pull Systems

In a push system, production planning is done for all levels of production in advance, while in a pull system products are only produced at a level when the next level requests it. Both systems have advantages and disadvantages. Push systems are able to forecast demand and use this to inform production decisions as well as to determine lot sizes to their advantage and thus reduce setup and batch costs. However, push systems tend to have a much higher level of work-in-process inventory than pull systems since push systems do not place a cap on work-in-process. Pull systems reduce work-in-process inventory which leads to a reduction of associated costs like holding costs and outdated work-in-process. Pull systems are also able to quickly pinpoint quality issues before large batches of defective products build up. This is due to their more piece by piece flow instead of batches that are more associated with push systems. Two common pull systems are examined in the next couple paragraphs, ConWIP and Kanban, followed by a description of Materials Requirements Planning (MRP), a push system.

ConWIP is the simplest way to establish an WIP cap. A maximum amount of WIP is established for a production line or process and new materials are not released into the process until the WIP is below the established maximum amount. This creates an hybrid push-pull system where materials within the production line controlled by ConWIP are pushed from one step to the next, but the line as a whole pulls material into it when one leaves the process. This process results in almost constant WIP (Spearman, Woodruff, & Hopp, 1990).

The Kanban system was developed at Toyota, the word kanban is Japanese for "card". The system runs on a set of cards where production authorization cards are attached to finished goods inventory. When a part is removed from finished goods inventory the card is sent back to the workstation that feeds the inventory point. This card gives authorization to produce a new part to replace the one just taken out of inventory. This system feeds back and parts are pulled through the system as they are required. Two types of cards were used in the original system, production cards and move cards. Production cards authorize the production of a part, which triggers production of a part if the necessary components are available. Move cards trigger the move of material to a stockpoint from a previous step or inventory, stockpoints are where components necessary for production are kept. Although traditionally cards are moved back and forth throughout the plant to trigger these authorizations with newer technology this system can be computerized and card movements done instantaneously (Hopp & Spearman, 2011; Nahmias & Olsen, 2015).

A typical push system is Materials Requirements Planning (MRP) where a master production schedule based on forecasted demand is converted to a build schedule using a set of rules. The build schedule defines when and how many parts will be built at each level of production. This system works with both end items, or items that are sold to customers or exit the production facility, and lower-level items which are WIP or raw materials. The bill of materials defines the relationship of which and how many lower-level items are required to build an end item.

There is a large amount of literature surrounding production planning and scheduling. However, it is beyond the scope of this thesis to review it in detail. The textbooks by Hopp and Spearman as well as Nahmias and Olsen are excellent references for those wishing more detail.

2.3 Scheduling Problem

The scheduling of tasks is a very active area of research in academia as well as industry. This is mainly due to the wide variety of real-world problems that relate to scheduling, as well as the impact that a good schedule can have on system performance. There are many ways to formulate a scheduling model using linear integer programming, however the two main decisions address how material balances will be handled as well as how time will be represented. The following two sections explore these decisions in detail. The third section discusses the different objectives that can be present in a scheduling problem.

2.3.1 Material Balances

Two main ways of dealing with material balances exist. The monolithic approach simultaneously deals with optimization of the batches as well as determining what goes in each batch. The second approach creates the batches first, then uses this as input to determine when to schedule these batches.

Monolithic approaches are further broken down into two models: state-task network (STN) and resource-task network (RTN). RTN based representations consume and release resources at the start and end of tasks. STN instead uses a directed graph with "(i) state nodes representing feeds, intermediates and final products; (ii) task nodes representing the process operations which transform material from one or more input states into one or more output states and; (iii) arcs that link states and tasks indicating the flow of materials" (Méndez et al., 2006).

Given that both of these approaches simultaneously deal with the creation of batches as well as the batch scheduling, this yields a very large and complex model. Due to this, only small problems or those with a short time-horizon have been be solved using these methods (Méndez et al., 2006). In the TIP case, using a monolithic method would attempt to both determine the optimal autoclave packings and schedule them in the same model.

The second approach uses lots that are pre-specified. Since the lots are created independently from the schedule, a global optimal solution cannot be guaranteed. However, "this two stage approach can address much larger practical problems than monolithic methods" (Méndez et al., 2006). By breaking up the problem the scheduling portion only has to deal with how resources are allocated to the lots, making the problem much smaller and easier to solve.

2.3.2 Time Representations

There are two main options as to how to represent time: continuously or using discrete intervals. Continuous time representations can usually better represent the problem at hand, however they can be much harder to solve. Several ways of modelling continuous time representations exist, including: global time intervals, global time points, unit-specific time events, time slots, and precedence-based (Castro & Grossmann, 2005; Castro & Grossmann, 2006; Gupta & Karimi, 2003; Méndez et al., 2006). For discrete time the only way is using time intervals.

There are several continuous-time representations available in academic literature. However, most are only applied to small to medium-sized problems (Castro & Grossmann, 2005; Castro & Grossmann, 2006; Floudas & Lin, 2004; Gupta & Karimi, 2003). The discrete time representation using time intervals can handle large problems. However, if the time intervals do not exactly line up with task times the task times must be rounded up. This creates an approximation of the problem, resulting in suboptimal solutions to the real problem (Castro & Grossmann, 2006). A solution to this is increasing the number (decreasing the width) of time intervals, which can lead to a very large problem.

2.3.3 Scheduling Objectives

Many objective functions have been studied for scheduling problems, these range from makespan minimization which has been studied extensively to profit maximization (or its counterpart cost minimization). Makespan minimization attempts to schedule all tasks in the shortest amount of time possible. Profit maximization (and cost minimization) usually involve constraints that determine the latest time tasks can take place at or penalize tasks scheduled after a certain time, but their objective is mostly to maximize profit (minimize cost) from the completion of these tasks. Other objective functions can attempt to minimize average or maximum task lateness assuming that tasks have a due date (Floudas & Lin, 2004; Maravelias & Grossmann, 2003; Méndez et al., 2006; Simpson & Abakarov, 2009; Ye, Li, Chen, Ma, & Leng, 2014).

2.4 Bin Packing Problem

The classical Bin Packing Problem (BPP) consists of a set of items that need to be packed into a set of bins, each with capacity C, such that the number of bins used is minimized and none of the physical constraints are violated (Coffman Jr, Csirik, Galambos, Martello, & Vigo, 2013). The BPP is within a larger class of problems called order independent minimum grouping problems (Lewis, 2009) . Most literature regarding the BPP focuses on one or two dimensions, usually with identical bin sizes (J. V. de Carvalho, 2002; J. V. de Carvalho, 1999; S. Martello, Pisinger, & Vigo, 2000; S. Martello & Toth, 1990). However work with non-identical bin sizes also exists (J. V. de Carvalho, 2002; Friesen & Langston, 1986; Haouari & Serairi, 2009; Kang & Park, 2003; Murgolo, 1987).

The BPP is traditionally an NP-Complete problem, thus making it very difficult to solve in large instances (Garey & Johnson, 1979). NP-Complete is a class of problems theorized impossible to solve in polynomial time, therefore named Non-Polynomial (NP) Complete. Although the lack of evidence of an algorithm that can solve these problems in polynomial time is not proof that they cannot be solved in polynomial time, it is generally agreed upon by the scientific community that certain problems fall in this realm (Blum & Rivest, 1992). Solving small NP-Complete problems to optimality is possible, however as the problem size grows solution times will grow dramatically, often making it impossible to reach an optimal solution. To solve these large problems usually heuristics or meta-heuristics are used.

The complexity intrinsic to the Autoclave Packing Problem (APP) is greater than that of the traditional BPP given the additional constraints as well as the need to level out the demand for autoclave resources. Given that the BPP is NP-Complete, optimally solving the APP is likely to be at least equally difficult.

2.5 Batching Problem

There are two batching problems in literature, the serial-batching problem and the parallel-batching problem. The serial-batching problem is where a set of jobs only requires one setup time if they are part of the same job family. Jobs are completed and can move on independently. In the parallel-batching problem, jobs of a particular batch both start and end the process together, often the machine process cannot be interrupted (preemption is not allowed). In literature this is known as the scheduling of the batch processor (SBP). The SBP problem is the one that will be discussed in this section as it is most relevant to the problem at TIP.

The SBP, or parallel-batching problem, is found in many industries. Predominantly research in this area has focused on the semiconductor industry, however it is also found in metalworking, show-manufacturing, helicopter-part production, and ceramics (Fanti et al., 1996; Koh et al, 2004; Luh et al, 1997; Ram & Patel, 1998). This problem must deal with how the batches are created as well as the scheduling of these batches in the time-horizon to best suit the objectives (usually makespan minimization). The creation of batches, while minimizing the time-horizon, carries an implied Bin Packing Problem. Additional difficulty is added to the problem as the batches are also scheduled simultaneously. A very thorough literature review on the subject relating to the semiconductor industry analyses the research in this area (Mathirajan & Sivakumar, 2006). Due to the difficulty of the problem, research in this area has dealt mostly with small single-machine (76% of articles in literature review), while a few others have dealt with two-machine problems and almost none tackle more than two machines. If multiple machines are considered, usually these have identical capacity. Little research exists for multiple machine problems with non-identical capacity. Capacity in all papers considered in the literature review used only a single capacity constraint (usually volume). Most papers studied in the literature review use heuristics (66%) instead of mathematical programming (30%) or simulation (4%).

While research in this field is related to the problem at TIP, no article was found that dealt with the level of complexity found in the problem at TIP. Most research considered at most two machines, often identical, and with only one capacity dimension. The problem at TIP has more than two machines which are not identical and have several capacity dimensions (Volume, thermocouples, tools, etc.). In addition, none of the problems integrate complex constraints such as manpower levelling, absolute minimum tool recycle times, desired tool recycle times, energy minimization, and other constraints found in the problem at TIP.

2.6 Mixed-Integer Linear Programming

Linear programming is used on problems that are linear in nature, therefore no "if" statements, multiplication or division of decision variable is allowed. All constraints and the objective function must be expressed linearly. A key word here is "expressed" since many nonlinear problems can be linearized by creative modelling techniques or approximations (Williams, 1999).

Mixed-Integer Linear Programs (MILP) deal with special subset of optimization problem where some of the decision variables must take on integer values instead of being allowed continuous values. Being able to take on only integer values greatly increases the types of problems that can be modelled. Especially useful are binary variables which can be used to turn constraints on or off depending on situations. Although there is an improvement in terms of modelling possibilities this also introduces an additional level of complexity over continuous optimization, both making it more difficult to find feasible solutions but also to prove optimality (Eiselt & Sandblom, 2000). MILPs are frequently used to model problems such as network problems, facility location problems, Bin Packing Problems and resource-task scheduling problems among others (Pochet & Wolsey, 2006; Salkin & Mathur, 1989).

Continuous linear programming models are easily solved by modern solvers using the simplex method, however for MILP models more sophisticated techniques are used. Most techniques rely on an initial "lower bound" computed by the relaxation of the integer program into a continuous program approximation and solved with the simplex method, this is followed by cutting-plane methods as well as branch-and-bound enumeration. Significant literature is available on these methods (Balas, 1971; Belov & Scheithauer, 2006; J. V. de Carvalho, 1999; Eiselt & Sandblom, 2000; S. Martello & Toth, 1990; Salkin & Mathur, 1989).

Branch and bound is at the core of all integer programming solvers. This method works by solving the linear relaxation of the integer program where all integer variables are made continuous. Once the linear relaxation is solved one of several things can happen. The first is that the optimal solution to the linear relaxation is integer, in which case the optimal solution to the integer problem has been found. The second is that there is no solution to the linear relaxation in which case the model is infeasible. The third option is that some or all of the variables for the linear relaxation are non-integer. In this case branching occurs, where at least one non-integer variable is set to be integer. For example if a decision variable is found to have a value of 3.8 in the linear relaxation two branches occur at 3 and at 4. This method is repeated until all variables are integer. If at any point a linear relaxation yields a worse result than an already found integer solution that branch is fathomed (the bounding section of the name). This is because any integer solution will always be worse or equal to the linear relaxation, never better. Therefore no better integer solution than the linear relaxation can result from this branch (Lawler & Wood, 1966).

While solving the branch and bound tree a "gap" parameter can be calculated. This gap indicates how far the currently best-found integer solution is from the best known bound on the optimal solution. It is calculated by dividing the value of the best linear relaxation minus the value of the best integer solution over the value of the best integer solution. When this gap reaches 0, an optimal solution has been found. Frequently solvers will have a default tolerance where a solution will be deemed optimal once the gap is below a certain point, typically 1e-04. A further improvement upon branch and bound is the implementation of cutting planes into the procedure which is common in mathematical solvers. This leads to the method called branch and cut. Cutting planes add linear inequalities as constraints which help bring the linear relaxation closer to an integer solution and thus speed up the procedure (Gilmore & Gomory, 1961; Jünger et al., 2009; Wolsey & Nemhauser, 2014).

Sometimes MILPs have a property called symmetry. Symmetry is found in a problem when "its variables can be permuted without changing the structure of the problem" (Margot, 2010). In non-mathematical terms, consider three loads being scheduled on a machine. Loads 2 and 3 have the exact same contents, but one is called Load 2 and one is called Load 3. Scheduling them in the sequence 1-2-3 and 1-3-2 would have no impact on the actual operations at the machine, however the MILP would view those as two different solutions. This example has one symmetry. In large problems thousands of symmetries can occur; these can result in a much larger solution space for the mathematical solver to explore. Identifying and reducing these symmetries in the problem formulation stage can greatly help the solver reach a solution faster.

2.6.1 Goal Programming

Goal programming can be used within MILPs when there are multiple objectives in a problem. In these cases there exist constraints, but also separately there are goals. The objective function measures if goals are met, and penalizes if they are not. There exist several approaches to meeting these goals. The lexicographic minimum method attempts to satisfy objectives in a set priority structure where the first objective to be satisfied is infinitely more important than the second, and the second infinitely more important than the third and so forth. A second method is using the weighting of objectives, where each objective function term is given a weight to show the relative importance versus other objective function terms. The weighting of objectives is based on utility theory, where each term holds a certain utility or value. There are several other methods of dealing with multi-objective optimization, however in this thesis the weighting of objectives will be used. Several good articles and books exist on this topic (Flavell, 1976; Ignizio, 2004; Ignizio & Cavalier, 1994; Tamiz et al. 1998).

2.7 Heuristics and Meta-Heuristics

When MILP is not able to solve the given problem either at all or requires too much time to solve for the application at hand, heuristics and meta-heuristics are a good alternative. Heuristics are policies, rules of thumb, that provide a simpler method to solving a problem and are usually much faster. However, they do not guarantee optimality. Sometimes heuristics can provide a 'worst case' scenario, where they guarantee that the solution is no worse than X% away from optimal. A good list for heuristics and meta-heuristics for the BPP can be found in Coffman et al. (2013). A brief description of first-fit, best-fit, shortest-processing-time, simulated annealing, genetic algorithms, tabu search, theory of constraints heuristic, ant colony optimization, and the hill-climbing method is shown in the following sections. Many other heuristics and meta-heuristics exist, however these are the most common and most relevant to the problems discussed in this report.

2.7.1 Heuristics

Heuristics can be categorized into two broad categories, general and problem-specific. General heuristics can be applied to almost any problem (greedy algorithm), while problem-specific heuristics are only applicable to a class of problems. First-fit and best-fit are usually problem-specific to Bin Packing Problems, but with modifications they can also be used to schedule. Shortest processing time (SPT) is specific to scheduling problems. The Theory of Constraints (TOC) heuristic is even more problem-specific as it is meant to schedule autoclaves with re-entrant operations.

2.7.1.1 First-Fit and Best-Fit

The First Fit heuristic packs each item into the lowest indexed bin in which it fits. If there isn't a bin where it fits a new bin is created . Best-Fit attempts to pack the item in the bin which will end up most full (but not exceeding capacity), if no bin can accommodate the item a new bin is created (Coffman Jr. et al., 2013; Johnson, 1974). Both of these are formulations of a greedy algorithm, where the local optimum solution is always selected. Although they are not guaranteed to produce globally optimal results, the heuristics run very fast. These two heuristics can be also used to schedule jobs on machines; in this case jobs are fed into the heuristic and scheduled into either the first machine they can be done on (First-Fit) or the best machine to process that job (Best-Fit).

2.7.1.2 Shortest Processing Time (SPT)

Shortest Processing Time is a heuristic usually utilized for single-machine scheduling. It schedules the jobs that must be processed in a machine by having the one with the shortest processing time first and all after have non-decreasing processing times. This heuristic is very fast for single-machine scheduling of jobs given that all it needs to do is sort the jobs in order of processing time and is good for just-in-time systems (Lee, 1991).

2.7.1.3 Theory of Constraints (TOC) Heuristic

Ye et al. proposed a heuristic in 2014 that could deal with reentrant operations in autoclave scheduling. Reentrance is sometimes found in composites manufacturing when a part is laid up, then cured, then laid up and cured at least once more. The TOC heuristic is based on priority rules, and its objective is to maximize "the mean comprehensive utilization of all autoclaves, and at the same time shortening the makespan as possible"(Ye et al., 2014).

2.7.2 Meta-Heuristics

Meta-heuristics insert randomness and typically run a basic heuristic multiple times, either having the latest solution as the best one or keeping a log of solutions and then picking the best one thus far at the end of the run. Meta-heuristics always give better solutions than just running the underlying heuristic once, and given the speed at which heuristics run, a meta-heuristic can have thousands of iterations in just a few seconds. Some common meta-heuristics are discussed below as well as some not so common ones that proved useful in solving the TIP problem. While most of the meta-heuristics discussed here are general and can be applied to a broad range of problems, the hill-climbing method is problem-specific to the Bin Packing Problem.

2.7.2.1 Simulated Annealing

Originally this method was developed independently by both Kirkpatrick et. al and Černý (Kirkpatrick, Scott and Gelatt, C Daniel and Vecchi, Mario P and others, 1983; Černý, 1985). Since then several an articles have applied simulated annealing heuristic to job shop scheduling (Osman and Potts, 1989; Van Laarhoven, Aarts, & Lenstra, 1992). Although good, their results show very long computing times are required for good solutions, even in the case of moderately sized problems.

Simulated Annealing works by exploring the neighbourhood around a given solution, then picking a solution in that neighbourhood. If the new solution is better than the given solution it is accepted and the process starts again. If not, it is accepted based on probability. This probability of acceptance is positively correlated to the temperature at that time and negatively correlated with how much worse the proposed solution is. The temperature is regularly decreased according to a cooling schedule, thereby reducing the probability of acceptance of non-improving solutions. The initial solution can be generated by a heuristic, or a linear relaxation of an integer program that although infeasible is penalized for infeasibilities. This heuristic is sometimes very fast, and in very large problem instances can provide a good solution where other methods struggle, it is also very easy to program. It is however very parameter sensitive, in particular the cooling schedule must be carefully designed (Bertsimas & Tsitsiklis, 1993; Eiselt & Sandblom, 2000; Hajek, 1988).

2.7.2.2 Genetic Algorithms

Genetic algorithms (GA) work very similarly to simulated annealing, however they are inspired in biology rather than metallurgy. For each iteration a set of possible solutions are present, a few of the best are selected to be the parents of the next iteration. Children are produced from these parents by selecting certain pieces of the two parents at random. The children and parents both continue on to the next iteration. Sometimes randomness is also inserted into the children via a mutation, where a part of the genome that makes up the solution is altered randomly. This mutation procedure allows genetic algorithms to make jumps across the solution space, while the parent-child relationship makes it search locally for an optimum (Eiselt & Sandblom, 2000). One drawback to genetic algorithms is the required tuning of parameters such as mutation rate, selection of next parents in an iteration, and others. This parameter tuning can greatly influence the performance of the algorithm and is very problem-specific.

2.7.2.3 Tabu Search

Tabu search, also called Steepest Descent Mildest Ascent method, was independently developed by Glover and Hansen (Glover, 1989; Glover, 1990; Hansen & Jaumard, 1990). The algorithm starts with an initial solution and much like simulated annealing searches the neighbourhood around that solution for the next one. Tabu search will search the whole neighbourhood and then select the most improving move; if no improving moves are available, then the least costly non-improving move is selected. This is in contrast to just selecting a random move like simulated annealing would do. Having this non-random procedure introduces the possibility of cycling between several solutions, to prevent this a tabu list of moves that not allowed is kept. Several parameters and policies to keep this list exist (Eiselt & Sandblom, 2000).

2.7.2.4 Ant Colony Optimization

First developed by Dorigo (1996), the latest version by Levine and Ducatelle includes an augmented local-search algorithm which yields very good results (Dorigo, Maniezzo, & Colorni, 1996; Levine & Ducatelle, 2004). The basic approach mimics how ants search for food. It uses a combination of heuristic information and a pheromone trail that is reinforced by the quality of the solution to build new and better solutions. Whereas initially this method was used to solve the well-known Travelling Salesman Problem (TSP), it can also be successfully applied to the Bin Packing Problem.

For the TSP the pheromone trail is encoded as a matrix of values which describe the favorability of visiting city j after city i. A similar concept is used when applying ant colony optimization to the BPP, where the pheromone matrix describes the favorability of having items of size i and size j in the same bin. This is especially good for solving problems that have identical bin sizes, but the structure breaks down when using variable bin sizes (Levine & Ducatelle, 2004).

2.7.2.5 Hill-Climbing Method

The hill-climbing method works based on a greedy algorithm (Lewis, 2009). First, unsorted items are inserted into bins on a first-fit rule, if it does not fit in any bins a new bin is created. Then the algorithm applies a problem-specific local optimization whose objective is to "improve the quality" of the bin packings. All this is done while preserving feasibility. Finally the items from the bins go through the algorithm again, being fed into it in either the order of the groupings or a random order of the groupings, while keeping the items within each bin in sequential order. Doing this process of re-inserting the items to the heuristic in this manner always yields a lesser or equal number of bins in each permutation according to the following theorem: "If each set of items Ui ∈ U (for i=1,..., G) is considered in turn, and all items u ∈ Ui are fed one-by-one into the greedy algorithm, the resultant solution U will be feasible, with |U||U|" (Lewis, 2009). The process stops when a user-specified stopping rule is met, usually number of iterations.

2.8 Mathematical Programming Languages

When a mathematical program is formulated it must then be translated into a mathematical programming language to be communicated to the solver. Some languages are not compatible with some solvers, therefore the decision between language and solver must be made taking both into account. This section reviews some of the prominent mathematical programming languages, the next section will discuss optimization packages that can be used with these solvers.

a) AMPL

AMPL is an acronym for "A Mathematical Programming Language", this language is developed by Robert Fourer, David M. Gay, and Brian W. Kernighan at AT&T Bell Laboratories. It continues to be developed, marketed, sold, and maintained by AMPL Optimization (Fourer, Gay, & Kernighan, 1990).

b) GMPL

GMPL is also called GNU Mathprog, this language is open source and based on the AMPL standard in Fourer et al. (1990). It is designed to be used with the GLPK open source solver but can also be used to generate models for other solvers (Pokutta, 2010).

c) GAMS

GAMS is an acronym for "General Algebraic Modeling System", it is developed and sold by the GAMS Development Corporation (GAMS Development Corporation, 2015). Its structure is similar to AMPL but provides access to a wider class of nonlinear problems (Brooke et al., 1998).

d) JuMP

JuMP is a mathematical programming language based on the Julia programming language. Julia is a language for technical computing and is very fast. It is designed for parallelization and distributed computation (Bezanson, Karpinski, Shah, & Edelman, 2015). It is a free language to use under the MPL License (Lubin, Dunning, & Huchette, 2014).

e) PuLP

This language is based on Python programming language and is used as a module that is installed onto Python. It was created by Stuart Mitchell at Stuart Mitchell Consulting and Michael O'Sullivan and Iain Dunning at the University of Auckland. The main advantage of this language is that pre-processing and post-processing work for the model can be done in the same interface. It is highly customizable and open source (Mitchell, O'Sullivan, & Dunning, 2011).

f) Pyomo

This language is also, like PuLP, based on Python. It was developed by William Hart, Jean-Paul Watson at Sandia National Laboratories along with David Woodruff at University of California Davis. The language has extensive documentation available and is very intuitive for Python users. It also has extensions for stochastic programming and distributed computation (Hart, Watson, & Woodruff, 2011; Hart et al., 2012).

2.8.1 Pyomo

The mathematical modelling work included in this thesis is described in the appendices using Pyomo. This language was selected among the several discuss in the previous section due to several advantages. One of the main advantages is that the language is based on Python, allowing several pre-processing steps and integration with other code sections to happen seamlessly. Additionally errors in code are easier to identify than with other languages. Both Pyomo and PuLP are based on Python, however the Pyomo community was found to be more active and provide more support. Early tests also found Pyomo to work faster than PuLP on a given mathematical model. Finally, Pyomo also provides an easy method of changing solvers (a single line of code is changed) which was important in early stages where the solver to be used was not yet identified. A brief tutorial on how Pyomo constraints are created and interpreted is found in section 4.2.1 Pyomo Basics.

2.9 Optimization Packages

Several optimization packages are available to solve linear integer programs. Three main ones are described below with the first two being commercial solvers and the last one being a free open-source solution.

a) Gurobi

Gurobi was developed by Robert Bixby, Zonghao Gu, and Edward Rothberg and first released in 2009 (Gurobi Optimization). It is written in C but like CPLEX can be connected to from multiple programming languages. This is a commercial solver with multiple licensing options. Due to the impressive performance that Gurobi has, its ability to make use of cloud computing to provide flexible licensing options, and its easy availability for academic research (during development), this solver was selected to tackle the scheduling problem at TIP.

b) CPLEX

CPLEX was developed by Robert Bixby in 1987, and through a series of acquisitions it is now owned, updated, and sold by IBM. This is a commercial solver with multiple licensing options. There are several ways to connect to CPLEX through multiple programming languages (IBM).

c) GLPK

GLPK stands for GNU Linear Programming Kit (Pokutta, 2010). It is a free open source solver easily available online. It can be connected to from multiple programming languages as well. Its performance is below that of the commercial solvers Gurobi and CPLEX.

Figure 6 and Figure 7 show the relative performance of each of these in a test scaled to Gurobi's performance. For this test the following versions were used: Gurobi 4.6.1, CPLEX 12.4.0.0, and GLPK 4.47 (Meindl & Templ, 2012). Clearly due to the running time and the percentage of instances solved, Gurobi's overall performance is the best among the considered optimization packages.

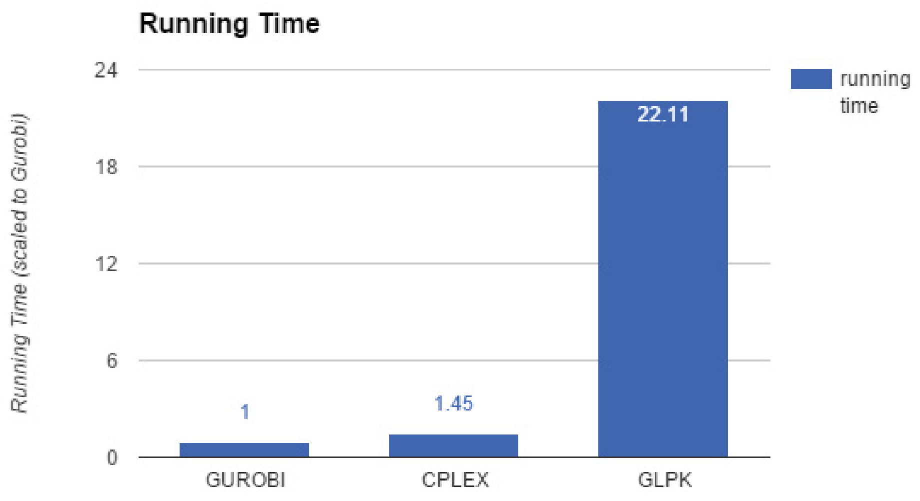


Figure 6: Running Time of Optimization Packages Scaled to Gurobi (Meindl & Templ, 2012)

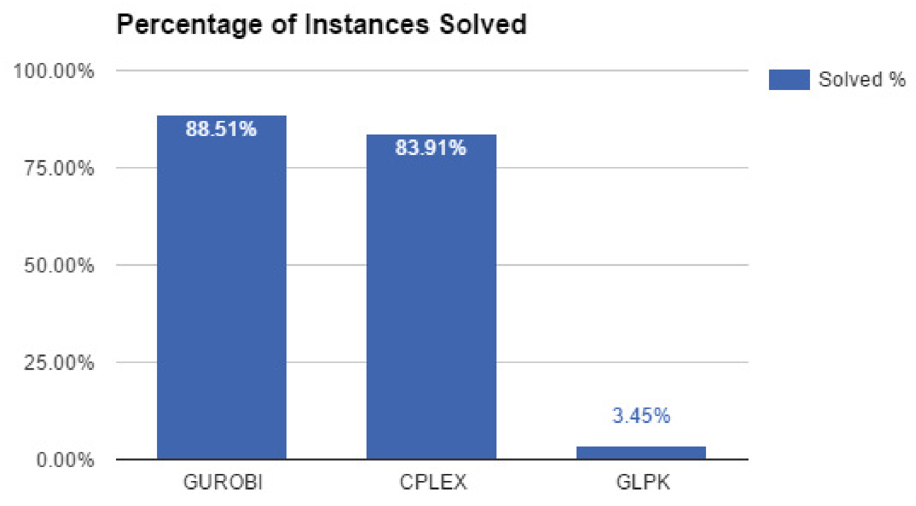


Figure 7: Percentage of Test Instances Solved (Meindl & Templ, 2012)

2.10 Cloud Computing

Cloud computing allows the purchase of computational power in a scalable way with a pay-as-you-go manner. This allows developers, companies, and the IT industry to not worry about the capital expenses required in launching a new product or service. The risk of having undersized in-house computing for a product that exceeds usage expectations or oversized in-house computing for a product that fails is no longer an issue. Instead large companies like Amazon, IBM, and Google have launched cloud computing services Amazon Web Services (AWS), BlueMix, and Google AppEngine respectively (Amazon Web Services ; Google ; IBM). They all provide different levels of integration, technical knowledge necessary for usage, and price. The main advantage they have over standard data centers is that due to their size they obtain massive utilization increases due to statistical multiplexing, spot market instances, and time shifting of their own computing needs (Armbrust et al., 2010).

In addition to the standard licenses discussed for the CPLEX and Gurobi optimization packages both have cloud computing options available. CPLEX costs $10 USD per hour online and Gurobi costs $25 per hour online (Gurobi Optimization, ; IBM, 2015). Their platforms are slightly different. CPLEX is hosted on IBM servers and the optimization software can be called through a web interface or an Application Programming Interface (API) call. Gurobi must run on AWS's servers, thus an instance must be started and a compute server loaded on. There is more technical work required to run Gurobi on the cloud than CPLEX on the cloud, but it also allows the user to specify the hardware that the software runs on.

Cloud computing is used in conjunction with Gurobi to solve the mathematical model discussed in this thesis. This is deployed on AWS, the details of this are discussed in the Implementation section. Cloud computing is used due to several important benefits it provides that are worth the added system complexity. From a financial perspective it allows the use of flexible licensing for mathematical solvers which, for the expected use (1 hour about 3 times per week) at TIP, provide cost savings. From a development standpoint it allows to easily update files remotely, it also lays the groundwork to allow different users at TIP to access optimization services from multiple locations. Finally, from a computational standpoint, it allows access to very well maintained, backed-up, high-performance computers at a low cost.

2.11 Previous Work on Autoclave Packing and Scheduling

Few work was found in literature regarding autoclave packing and scheduling for composite manufacturing plants. Section 2.11.1 presents work done at Aerospace Technologies of Australia which uses an MIP model to schedule autoclave loads. Section 2.11.2 examines a paper which uses a heuristic to schedule autoclave loads. Section 2.11.3 examines the Simul-8 commercial software application for autoclave packing which uses heuristics as well as simulation for autoclave packing and scheduling. Work done by the Lockheed Artificial Intelligence Centre on this subject is presented in Section 2.11.4. Finally, a paper dealing with on-line autoclave packing and scheduling is presented in section 2.11.5.

2.11.1 AeroSpace Technologies of Australia (ASTA)

Panton and Beaumont from the University of South Australia and Monash University respectively, developed an integer program for the scheduling of autoclave loads at ASTA in 1997. The problem they faced is similar to the one at TIP. Orders for parts in the problem were “usually regular and known months in advance”, the company “feels that the autoclave cell is the bottleneck”, and the number of tools is an important constraint. All of these features are also present at the TIP problem.

The team tackled the problem by enumerating 30 autoclave loads that would fill or nearly fill the assigned autoclave. Then the team designed an integer program that would attempt to schedule all loads as early as possible within the time-horizon established as shown in below (Panton & Beaumont, 1997).

Indices:

1...l for autoclave loads

1...v for autoclaves

1….s for shifts

Sets:

i autoclave loads

j autoclaves

k shifts

Parameters:

$a_{i,j}$ binary, 1 if logical set i can go into autoclave j

p_k integer, penalty parameter for late scheduling

b_i integer, demand for logical set i

c_i integer, longest cycle time for tools associated with logical set i

$n_{i,j}$ integer, number of shifts to process logical set i in autoclave j

Variables:

$X_{i,j,k}$, binary. 1 if logical set i is started in autoclave j in shift k, 0 otherwise.

Objective:

Minimize Z

Constraints:

1 $$\sum_{j=1}^{v} \sum_{k=1}^{s} a_{i,j}\, x_{i,j,k} = b_i \quad \forall i$$

2 $$\sum_{j=1}^{v} \sum_{t=1}^{c_i} a_{i,j}\, x_{i,j,(k+t-1)} \leq 1 \quad \forall i,\ \forall k \leq s - c_i + 1$$

3 $$\sum_{i=1}^{l} \sum_{r=1}^{n_{i,j}} a_{i,j}\, x_{i,j,(k-r+1)} \leq 1 \quad \forall j, k$$

4 $$\sum_{i=1}^{l} \sum_{j=1}^{v} \sum_{k=1}^{s} p_k\, x_{i,j,k} = Z$$

Constraint set (1) ensures demand for logical set i is satisfied. Constraint set (2) ensures tool recycle times are met. Constraint set (3) ensures autoclaves are processing at maximum one logical set at a time. Constraint set (4) is the objective function Z to be minimized. If p_i=k then there is a penalty associated with scheduling jobs late. This formulation uses a discrete time interval of 8 hours as shown by the "k" shifts set (Panton & Beaumont, 1997).

The problem being solved at ASTA had approximately 114 loads per month that were being scheduled around blackout periods. Blackout periods were used to schedule a large portion of very regular and single-product cures that would not be scheduled by the formulation provided by the researchers. These 114 loads equated to approximately 693 hours of cure-time on the five autoclaves at ASTA (Panton & Beaumont, 1997). For comparison, TIP cures approximately three times this amount every month on a similar number of autoclaves. Additionally the TIP problem has a more varied product mix as it requires the creation of new loads every month, instead of being able to enumerate 30 good loads and use those. There are also more complex constraints that stem from the objectives discussed in section 3.1.2.2 Scheduling Objectives that are present in the TIP problem but not at ASTA.

2.11.2 Scheduling Method for Reentrant Autoclave Moulding Operations

Ye et al designed a heuristic for scheduling of composite parts to be cured in autoclaves. Their heuristic uses priority rules to determine which pieces are scheduled when and simultaneously solves the autoclave packing problem and scheduling problem. It can also handle parts which have "reentrant" operations where multiple iterations of layup followed by curing must take place. The scheduling objective is "maximizing the mean comprehensive utilization of all autoclaves, and at the same time shortening the makespan as possible" (Ye et al., 2014). Given that the method used is a heuristic no mathematical model is provided in this paper.

The problem at TIP also includes reentrant operations; however, these are minimal and in most cases restricted to parts that have a very stable demand. Reentrant operations are not included in the proposed model of this thesis given the difficulty of incorporating them into the model as well as their minimal use. Therefore these are cured in designated slots that are permanently scheduled and can be blacked out by the user to restrict the model from scheduling other loads during those times. More information on this is found in subsection 4.4.7 Blackout Calendar of the Implementation section.

2.11.3 Simul-8 Planner

The Simul-8 Planner is a planning and scheduling tool based on simulation (Hindle & Duffin, 2006). It was developed by the Visual8 Corporation. It can simulate the entire composites manufacturing facility and identify how well a schedule will affect different steps of the process. The Simul-8 Planner uses sets of rules (a heuristic) to create schedules, this set of rules can be changed by the user. It also supports manual overrides to these rules. In the planner autoclaves run on a bus schedule to ensure resources are available on either end of the cure cycle for load and unload processes. When a given cycle on the autoclaves is ready to start, the system searches which parts are compatible to enter the autoclave as a batch. If several batches are available the one with the least slack time (hours left on material outlife) is selected. The paper referenced here does not provide indications as to the specific heuristics used in scheduling nor does it show computational results of any kind.

2.11.4 Lockheed Artificial Intelligence Centre

Two systems relating to autoclave packing and scheduling have been published by researchers from the Lockheed Artificial Intelligence Centre. Clavier is a system developed by Hennessy and Hinkle (1992) for autoclave packing. The second system, a case-based scheduler, uses the results from Clavier and applies heuristics to it to create a schedule for the production process, including the autoclaves.

Clavier uses case-based reasoning which compares the current set of parts to be cured, ordered by priority, to previous sets of parts that have been cured. If an exact match is found, then that is the load recommended. However, if an exact match is not found, it uses similar sets to propose a new load to the autoclave operator. The operator can then either accept or reject it, if accepted it gets added to the database of previously cured parts. The software takes into account the parts' layout within the autoclave as well as several heating characteristics experienced during the cure. These heating characteristics do not need to be taken into account in the TIP case as recipes are all assumed to work properly for the autoclaves they are rated for. The Clavier system was implemented at a composites manufacturing facility, however its use was changed as operators simply ran the software without restricting it to the set of parts currently at the curing centre. Operators then produced and cured full loads on the autoclaves and stored any excess parts for future use.

The second system, the case-based scheduler, uses the results from Clavier to create a schedule (Cook et al, 1990). It takes the loads created by Clavier and then applies a heuristic that arranges the loads on a schedule. One example given in the article is to arrange loads such that parts that take the longest to process are scheduled first. In this process resources are allocated to loads as required to monitor their usage. Following that process, a local improvement heuristic identifies decision points and improves the schedule by making changes based on a set of rules. Once no further changes can be made, or the scheduled is deemed "good enough" by operators, the system stops. The article does not explicitly show the heuristics used, present computational results, or show the size of problems that can be solved by these methods.

2.11.5 Heuristics for Multi-Server Batch Operations

Two papers were found in the field of parallel-batching problems dealing with aerospace applications very similar to those at TIP (Van Der Zee et al., 1997; Van Der Zee et al. 2001). Both papers conduct on-line scheduling, similar to that done by the Clavier system. The look-ahead heuristics provided have a cost minimization objective which takes into account waiting costs as well as setup costs. The 1997 paper is able to deal with multiple machines as long as they are identical. The 2001 paper presents a new heuristic which is better suited to handle non-identical machines. The methods presented here lack a way of dealing with several important aspects of the problem at TIP such as material outlife, energy costs, manpower levelling, complex autoclave packing rules, among others. Given the simple characteristics of the problem considered in these papers, it was not possible nor was it attempted to adapt these methods to solve the problem at TIP.

Chapter 3 Solution Approach

There are two main industrial engineering problems to solve here. The first is a modified Bin Packing Problem to determine what parts go into an autoclave load, the APP (Dyckhoff, 1990). The second problem deals with scheduling these loads around several constraints. These two problems can be solved either simultaneously or in series. If solved simultaneously the computation time may be prohibitively high, but yield a better solution. If solved in series, with the APP solution being the input of the scheduling problem, the computational time is less but the solution is just an upper bound on the true optimal solution.

The objective of the APP in this case is not entirely to minimize the total number of loads. Since there are several autoclaves with different compatibilities for parts, some autoclaves might have several times more loads associated with them than others if the only objective were to minimize the total number of loads. Given that the scheduling problem attempts to minimize makespan, another objective of the APP is to balance the utilization of all autoclaves. Therefore the objective is to balance the utilization of all autoclaves while keeping the number of loads to a minimum. Several constraints exist with respect to parts and autoclaves. These are listed below:

- Parts have a recipe with which they must be cured. Recipes which are not compatible with others must be cured separately.
- Autoclaves have thermocouple ports, and parts have differing numbers of thermocouples. The sum of all part thermocouples must not exceed the thermocouple ports available in an autoclave.
- The volume of all parts in an autoclave must not exceed 80% of the volume in an autoclave. This helps make all packings feasible given the complex geometry of the parts (Devine and Milne, 2014).

- Parts must individually be able to physically fit into the autoclave. This has been taken care of by pre selecting which autoclaves a part can fit into.
- Parts must be on tools when cured. The total number of a particular tool used in a single cure must not exceed the number of tools available.
- Certain parts cannot be cured with certain other parts, including replicas of themselves in some cases.

The objective of the scheduling problem is to minimize the overall cost and create a schedule that will be flexible to changes subject to the constraints of the problem. Frequently the objective of other scheduling problems, especially in job shops, is to minimize the makespan. However, minimizing makespan is not a priority for TIP, since parts have specific due dates and producing beforehand will just consume space at the facility. The aspect of due dates is solved prior to scheduling since demand is given to the model on a weekly basis and is therefore not included in the model.

In our approach to solving the TIP problem, the scheduling problem was solved after the APP. This scheduling problem has the following constraints:

- All autoclave loads must be scheduled within the week specified.
- Autoclave packings are only able to be scheduled on autoclaves for which they are compatible.
- Autoclaves cannot be scheduled to run more than one load at a time.
- All the tools used in a load have a certain recycle time post-curing for which they cannot be used to allow for cleaning. Therefore more tools than available at a given time cannot be used.
- There are blackout periods for which no work can be scheduled on the autoclaves. This is for maintenance or other programs not included in this optimization to be done.
- Re-scheduling must be kept to a minimum as changes are done to the schedule.

A more detailed explanation of the solution approaches and results for both problems will take place in the following pages.

3.1 Mathematical Models

The problem at TIP can be described using mixed integer linear programming (MILP) to be solved. Several different formulations can be carried out; in the following section the ideal formulation will be described followed by the problems which prevent this ideal formulation from being implemented. Then the proposed model, including separate formulations for autoclave packing and scheduling, will be shown.

3.1.1 Ideal Model

The ideal formulation for this problem would take into account both the APP and scheduling problem as a single model. This is necessary so trade-offs between individual autoclave utilization, manpower utilization, overtime, peak power, and schedule flexibility can all take place simultaneously. The ideal formulation would also use a continuous time model or discrete time model with very short time intervals. This would prevent rounding up of cure cycle times that can lead to suboptimal solutions. For example if two 1.5 hour cures are necessary these can take place in a total of three hours, but if rounding to the nearest hour were to take place they would be scheduled in a total of four hours. The Autoclave Packing Problem (APP) adds additional complexity to the BPP, a known NP-Complete problem, and given the large problem size at TIP is very difficult to solve (Garey & Johnson, 1979). Adding a scheduling problem to an NP-Complete problem and solving them simultaneously using integer programming is impractical and would prove very difficult to solve in large instances. This difficulty of the large size of the resulting solution space is compounded by symmetry that would be present in the problem. Symmetry can present itself in multiple situations in this problem. While solving the APP autoclave loads could be created which have different indices but have identical contents. Then again in the scheduling side when several identical loads are scheduled they can produce lots of symmetry in the problem.

3.1.2 Proposed Model

Given the reasons stated in the previous section for not solving the APP and the scheduling problem together, the proposed modelling approach deals with them separately. The proposed model uses a discrete time formulation instead of the ideal model's continuous time. Continuous time approaches require complex formulations of resource constraints which often use several big-M terms and can lead to difficulty in solving the problem. Discrete time models do not have this problem and solve much faster, due to this they are recommended in literature for large problems (Méndez et al., 2006).

First an autoclave packing heuristic to group sets of parts into loads will be discussed. Then a scheduling model will be discussed where the loads created by the autoclave packing heuristic are scheduled in the time horizon.

3.1.2.1 Autoclave Packing Problem Formulation

The problem at TIP is three dimensional, has non-identical bin sizes, and has other idiosyncratic constraints mentioned in the Introduction section. However, a simplification of the problem is achievable by only considering the volume of the autoclaves and parts as the physical constraint to the parts fitting in the autoclaves. The volume of the parts is approximated by using the length, width, and height of the part on the tool. Given the wasted space due to complex geometries the volume of the autoclaves is reduced by 20% in these calculations. Tests were carried out and the 20% value was found to produce feasible autoclave packings. This simplification was confirmed by the Cure Centre Program Production Leader as reasonable and necessary. Additionally, the objective of the problem at TIP is not purely to minimize the total number of loads but to also balance the utilization of all autoclaves. Therefore there are two sections to the APP objective function, to minimize the total number of bins and to balance the utilization among autoclaves.

The balancing of the utilization is accomplished by picking an autoclave with the lowest "weighted time", as calculated with the formula below, according to the autoclave packings already created. This formula was developed by the researcher and validated with the stakeholders at TIP.

Weighted Time=(1+0.1*au_priority)*au_time

For example, if autoclave 1 has 100 hours in it and is priority 1 while autoclave 2 has 50 hours allocated to it but is priority 2 for the part then the "weighted time" for autoclave 1 is 110 while autoclave 2 is 60. This helps balance between having an even utilization across all autoclaves and using the autoclaves with the best priority for a given set of parts. The weighted time function also takes into account blackout times in the au_time component as assigned work, blackout times will be discussed further in the implementation section. The weight of 0.1 for the au_priority section of the weighted time formula was obtained by trial and error and consultation with the Cure Centre PPL. The autoclave priorities are determined by the Cure Centre PPL, these take into account how well the current product mix fits in the autoclave as well as the relative cost of completing a run with each autoclave.

A modified version of the hill-climbing method discussed in the Meta-Heuristics section was developed and applied to the APP. Some modifications were necessary to carry this out. First it should be noted that the basic hill-climbing method yields an equal or lesser number of bins in each permutation. However, given that more than one limiting factor (volume, thermocouples, recipe max loads, etc.) is being considered, this is not always the case in the modified version. Instead it only improves some of the time, with an occasional increase in the number of bins. The pseudo code including the constraints of the problem is shown below in Figure 8. The full code is available in Appendix A1 and makes use of the Bin Class object in Appendix A2.

The objective is to minimize the total number of bins while keeping a balance between autoclave priority and autoclave utilization. The constraints are as follows:

1. Parts must be cured in a load with the right recipe.
2. Total number of thermocouples in an autoclave must be less than or equal to total number required by the parts in the load.
3. Volume of autoclave must be less than or equal to total volume of parts and tools. Volume used here is decreased by 20% to account for packing of complex geometries in a cylindrical tube.
4. Tool length must be less than autoclave length.
5. Number of parts in a load must be less than or equal to load's "Max Per Load" parameter.
6. Number of identical tools in a load must not exceed number of tools available.
7. There must be enough tools to hold all parts.

```
Start new iteration with items in some initial order:

Place items in first bin it fits in.
Item fits if:
    -Enough thermocouples are available
    -Item geometry is able to go into autoclave that bin uses
    -The right recipe is used
    -The number of parts in the load is less than the "Max per Load" for the recipe
    -Current tools in bin have excess capacity OR:
            -More tools are available AND
            -There is enough space for the new tool

If item does not fit:
    -A new bin is created for that item.
    -The new bin uses the autoclave with the lowest weighted time:
            -Weighted time=(1+0.1*au_priority)*au_time
            -au_time=Total time of autoclave for currently created bins for each
autoclave

After all items are placed:
Conduct local improvement:
    -Split all bins into two groups:
            -A: Are less than or equal to 30% full by volume
            -B: Are more than 30% full by volume
    -Attempt to move items from group A into B to create fuller bins.
            -Less full bins may be no longer required in future iterations.
Save the iteration as complete. Move on to the next.

For the next iteration randomize the order of bins, then take out all items from the
bins in that order and attempt to place them in the new bins with that order.

Stop iterations when desired number of iterations is reached. Best solution thus far
in terms of number of bins is selected.
```

Figure 8: Pseudo-Code for Hill-Climbing Algorithm

3.1.2.2 Scheduling Objectives

As the proposed model is separated into the scheduling problem and autoclave packing problem, the different objectives for both optimization problems will be discussed. This section will elaborate on the different objectives within the scheduling problem. Given that there are multiple objectives they must be balanced within the objective function by weighting their values accordingly. This aspect will be further discussed in the next section where the objective function is presented.

1. Minimize overtime

Overtime (OT) presents a cost for TIP in terms of people requiring to work additional hours at a premium rate. Given that the autoclave centre runs on three shifts, it only registers overtime when loads are started on the weekend. There are two components to this. One is minimizing the number of loads that start on overtime, the second is to reduce the number of overtime shifts required in total. This reduces both the amount of people required in the plant for a given shift, since there is less work that must be carried, out as well as the frequency at which people come in for overtime.

2. Minimize rescheduling

As changes are made to the schedule due to unforeseen circumstances like added demand, delays, or AOG situations, there may be a need to change the schedule. However, given that a problem may have several very different optimal solutions an effort must be made to make the solution to the rescheduled problem as similar as possible to the original schedule by minimizing changes. There are two ways to account changes, by the total number of changes made (regardless of magnitude) or the sum of the magnitude of all changes. The former is preferred by all stakeholders and leads to a few loads changing times by several hours or even days. The latter would move the cure times of many loads to accommodate changes, but each load’s cure time may be only a few hours different than before. These are best illustrated in Figure 9 below as load E is added to a pre-existing schedule. In this example Load E could have been inserted manually (Manual creation of loads is further discussed in section 4.7.4) to be scheduled during that week due to changes in customer demand. Minimizing the number of changes moves load B by a moderate amount forward, for example 5 time units. However, by moving both B and C in the magnitude of changes they can collectively move a total of 3 time units which minimizes the magnitude of changes. The minimization of the number of changes and not the magnitude, is preferred by stakeholders due to the coordination effort required for each move.

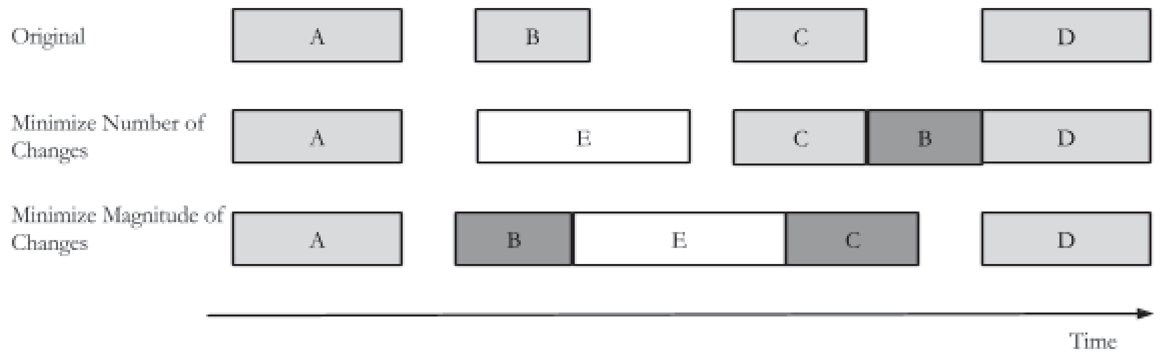


Figure 9: Different Rescheduling Methods When Inserting Load E

3. Usable gaps

As the schedule is created there are unused gaps between loads on a given autoclave. These unused gaps, if large enough, can be used to schedule or reschedule work without needing to modify the cure times for other loads. The minimum time to schedule a load is four hours, therefore any gap that is less than four hours is wasted time. The creation of usable gaps within the schedule provides it a certain level of flexibility that is desired. Figure 10 shows how a schedule without usable gaps would require rescheduling of the loads as load E is added, if usable gaps are already built into the schedule the scheduling of this load would leave all other loads unaffected. These usable gaps are achieved by the creation of dummy loads. Dummy loads contain no parts, use no recipe, and are only part of the scheduling problem. Several potential dummy loads are assigned to each autoclave, and if possible to be scheduled they are. When a load must be rescheduled, the dummy load can be taken out and the real load scheduled in its place.

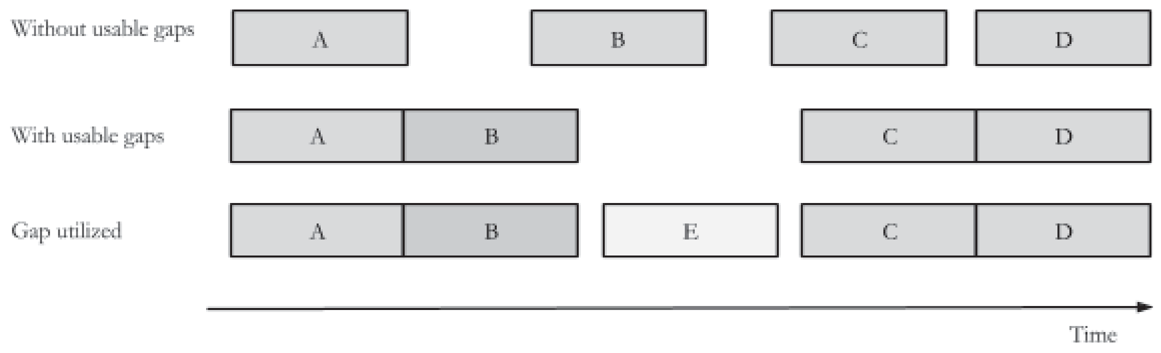


Figure 10: Demonstration of Usable Gaps When Inserting Load E

4. Peak Power

There are two main portions to the electricity bill that companies pay. The first is total usage which is a sum of the power consumed. The second is the peak power charge which is the maximum power drawn at any given time in a 24-hour period and is billed daily. Figure 11 shows two autoclaves running one cure every day. In the first day peak power is not accounted for in the schedule and both loads on different autoclaves start at the same time. In this example this results in a peak power of 20 kilowatts (KW). On day two, the same two loads are run but their start times are separated so they do not overlap. This results in a much reduced peak power of 10KW. Although both days use the same amount of energy in total, the peak power they use is very different. For peak power, day two is much more preferable than running a schedule like day 1.

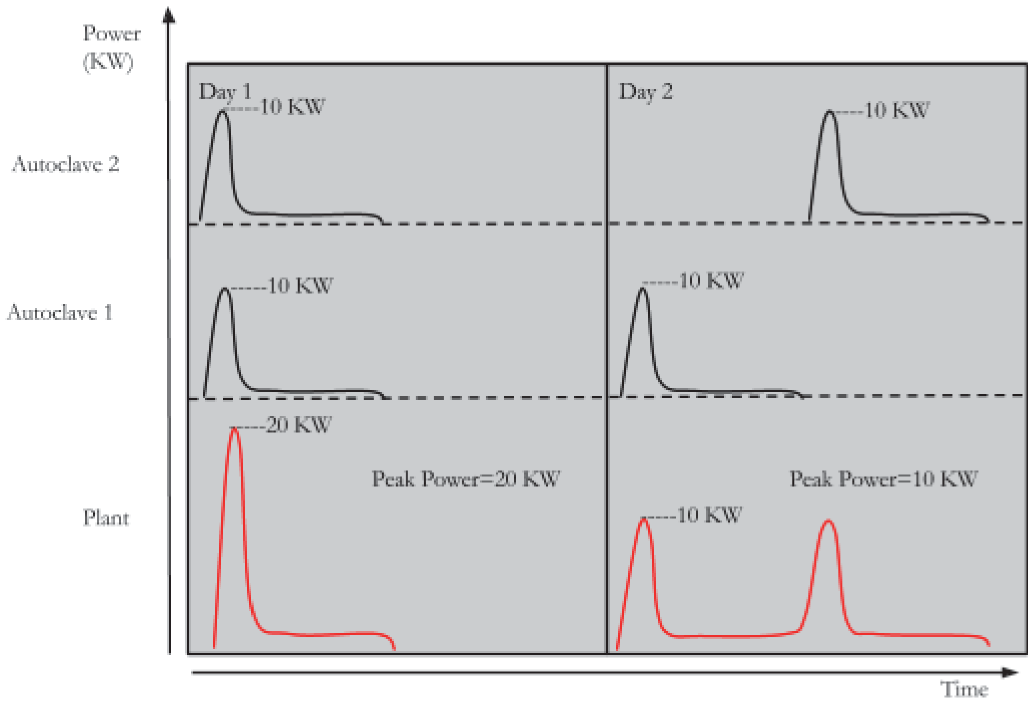


Figure 11: Demonstration of Two Peak Power Scenarios

5. Manpower Levelling

Scheduling production of the parts that go into each of the loads at the layup stage is not directly included in the optimization. In order to ensure that the schedules provided by the optimization do not negatively impact the layup areas too much, it is necessary to even out every area's workload throughout the week. This is done by setting as an objective to minimize the maximum workload on any given day for every area. Figure 12 below shows an example of the maximum workload identified for a given week. By bringing this upper bound down, the work should be evened out throughout the week and not concentrated on only a few days. It is important to measure the manpower levelling in terms of percentage of manpower available so areas with differing numbers of manpower do not contribute different amounts towards the objective function. Otherwise, this would lead to smaller areas ending up with more uneven schedules.

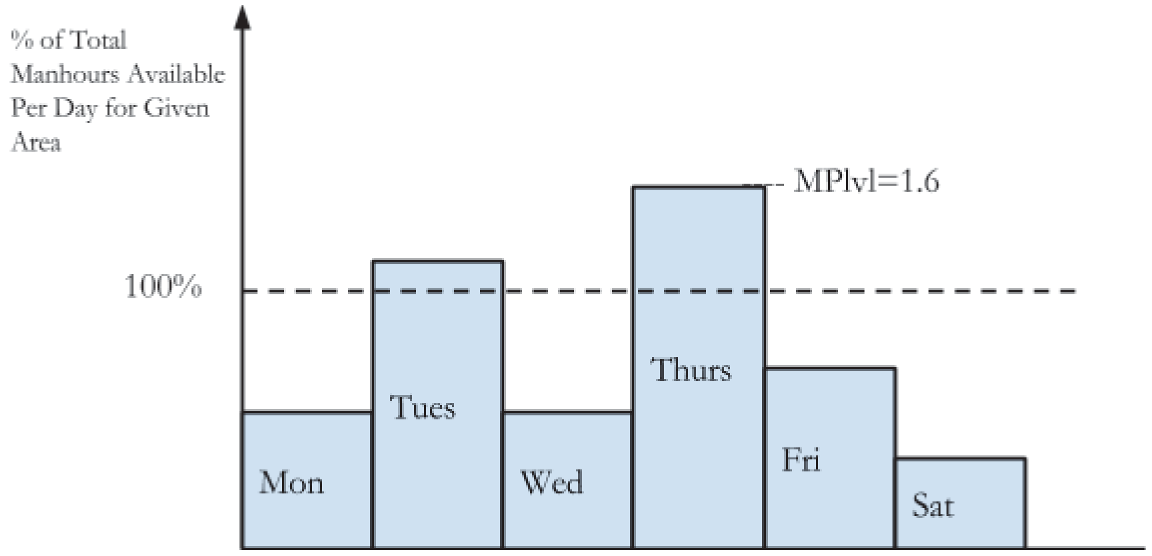


Figure 12: Example of Manpower Levelling at Layup Areas

6. Desired Tool Recycle Times

The absolute minimum tool recycle times are hard constraints that must be met. A tool cannot be taken out of an autoclave, demoulded, laid up, and be ready for curing before a set minimum amount. However, this minimum amount may not be the best for all areas within the production process to work together. If possible, a few extra hours allows for jobs not to be rushed through and this is possible through proper planning. In addition to the absolute minimum tool recycle time, information is provided on the desired tool recycle time for each tool, a time which all stages of production can comfortably meet. A good schedule would meet all, if not most, of the desired tool recycle times.

7. Early Starts

Setting a load to be cured on Monday morning at 2am is feasible in many situations, but usually undesirable. This is because meeting this cure time would require the parts to be laid up on Friday and have them wait throughout the weekend, with the potential of having material outlife issues if the material runs out of shelf-life post thawing. Due to this, it is preferable that loads be scheduled such that they can be laid up in the same week as they are cured. Scheduling a part so it can be laid up in the same week as it is cured depends on two factors: the maximum number of shift-hours required for all parts in the load, the shift schedule for the area laying up the part. Properly identifying these desired earliest start times and adhering to them in a good schedule minimizes the WIP that is kept over the weekend and the potential for material outlife issues. Suppose that a layup room works on a one shift schedule where only day shift is worked. In Figure 13, "A" and "B" are loads to be cured at those times and the black lines preceding them are the shift-time required to build those loads. Load "A" here would be violating the earliest start preference, given that the layup room would be unable to make the parts that week to be cured at that time slot. Load "B" would not be, since the layup room has sufficient time that week to lay up the required parts on the Monday day shift to be cured that afternoon.

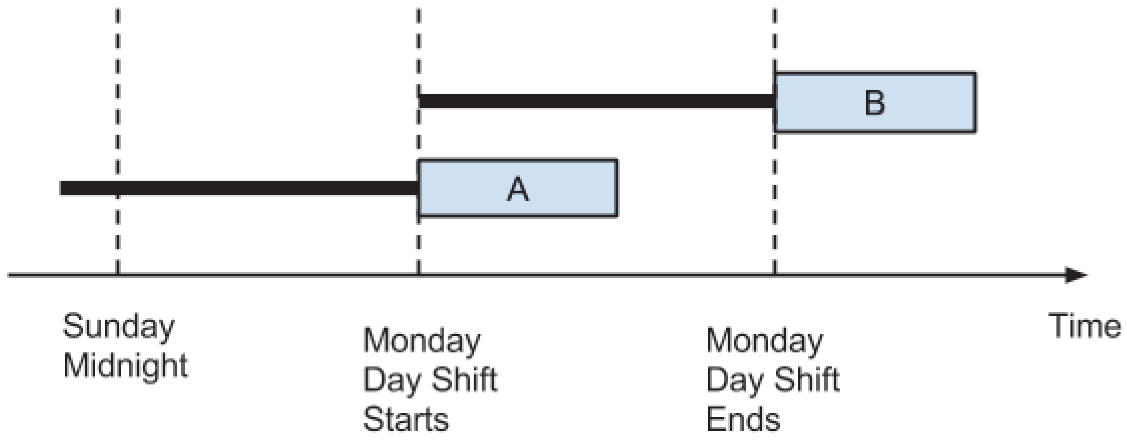


Figure 13: Example on Early Starts in a Week

3.1.2.3 Scheduling Formulation

The Mixed-Integer Linear Program formulation is done using discrete time slots of hours to schedule the autoclave packings. The resources used and considered in the model are the autoclaves, energy, tools, and to a certain extent manpower in the layup rooms. The full mathematical model is shown below.

Sets:

P=Parts “z” to be packed

Au=Autoclaves “j” available for packing

W= Set of all weeks “w”

L=Set of all loads “i”

DL=Dummy logical sets “i”. Subset of L.

LL_w=Set of loads “i” to be scheduled in week “w”. Subset of L.

Ar=Set of “r” available areas (layup)

S=Set of all “k” available times.

SW_w=Set of timeslots “k” in week “w”

IL_z=Set of loads “i” that part “z” is in.

$blkt_j$=Set of blocked off times for each autoclave “j”

weekends=Set of times “k” which fall on a weekend

Wsame=Set of pairs of loads “i” that are identical in terms of autoclaves and tools

Wsomesame=Set of pairs of loads “i” that are not in Wsame but both use at least one tool to its maximum capacity

AUX2=Set of all loads “i” that can be scheduled at time “k”. Used for tool recycle time constraints..

AUX3_1=Set of possible pairs of loads “i” starting or running at times “k”. This aids in dealing with loads that start late on Sunday and end on Monday. Used for autoclave cycle time constraints.

AUX6=Set of all load “i” and time “k” pairs in violation of blocked off times for autoclave “j”.

Parameters:

$MaxInWeek_w$= Last time slot in week “w”

$MinInWeek_w$=First time slot in week “w”

$WinS_k$=Week that timeslot “k” belongs to

MPA_r=Manpower for area “r”

SAR_r=Number of shifts for area “r”

S_Time=Start time of day shift

MPL_i=Manpower reqd load “i”

ALS_i=Layup area for load “i”

PL_{iz}=Binary, 1 if part “z” is in load “i”

A_j=Indicates which LS’s are in autoclave “j”.

C_z=Int, absolute minimum tool recycle time for each P_z.

CD_z=Int, desired tool recycle time for each P_z.

sh_i=Full shifts required to complete job “i”

sh_h_i=Hours in excess of full shifts required to complete job “i”

$tools_z$=Int, number of tools available for P_z.

n_i=Int, Cycle time for L_i

Wsametime=Minimum time units apart that a pair of Wsame autoclave packings can be scheduled at given tool recycle times

Wsomesametime=Minimum time units apart that a pair of Wsomesame autoclave packings can be scheduled at given tool recycle times

$Earliest_i$=Earliest time “k” that load “i” can be scheduled if laid up the same week as curing

$Repenalize_i$=1 if rescheduling load “i” should be penalized, 0 otherwise.

Variables:

$Load_{wik}$=Binary, 1 if L_i is scheduled in week “w” at time “k”, 0 otherwise.

OT= Integer, number of starts on weekend

$SatWeekend_w$=Binary, 1 if saturday shift is used for OT on week “w”, 0 otherwise.

$SunWeekend_w$=Binary, 1 if sunday shift is used for OT on week "w", 0 otherwise.

$Energy_w$=Integer, counts peak power usage within each week "w"

$Tool\_Inf_z$=Integer, slack variable to represent tool infeasibility for tool "z"

Au_Inf_j=Integer, slack variable to represent autoclave infeasibility for autoclave "j"

Resched=Integer, number of bins which were rescheduled in the model

$Early_i$=Binary, 1 if parts in load "i" are not able to be laid up in week they are cured, 0 otherwise.

MP_lvl_{wr}=Non-negative real, manpower leveler per week "w" per area "r"

$Desired\_Tool\_Rec_z$=Non-negative real, counts number of violations of desired tool recycle time for tool "z"

$Before_{i,ii}$=Binary, 1 if bin "i" is scheduled before bin "ii", 0 otherwise.

The objective of the MILP is in simplest terms to schedule all loads feasibly and at the lowest cost. Since actual costs were not available work was completed with the TIP team to identify relative weights of the different variables such as to produce a good schedule for them that took into account the preferences of different stakeholders. The weights (a_1 to a_{11}) are placeholders for values that can be changed based on the implementation and depend on the process manager's priorities. This objective function is based on the weighting of objectives, as discussed in section 2.6.1 Goal Programming.

$$\begin{aligned} Minimize:\ & a_1 * OT - a_2 * \sum_{w\in W} \sum_{i\in DL\cap LL_w} \sum_{k\in S} Load_{w,i,k} + a_3 * \sum_{w\in W} Energy_w \\ & + a_4 * \sum_{z\in P} Tool\_Inf_z + a_5 * \sum_{j\in Au} Au\_Inf_j + a_6 * \sum_{w\in W} \sum_{r\in Ar} MP\_lvl_{w,r} \\ & + a_7 * \sum_{z\in P} Desired\_Tool\_Rec_z + a_8 * Resched + a_9 * \sum_{i\in L} Early_i \\ & + \sum_{w\in W} a_{10} * SatWeekend_w + a_{11} * SunWeekend_w \end{aligned}$$

The first term penalizes each start that takes place during overtime hours, namely on the weekend. The second term provides an incentive to scheduling dummy loads, this is required since the constraints leave it up to the optimization to determine which and how many dummy loads are scheduled. The third term penalizes the Energy$_w$ variable for every week, disincentivizes the optimization from scheduling two autoclave starts on the same hour. The fourth and fifth terms represent slack variables for the tools and autoclave resources, the weights a_4 and a_5 should accordingly be large enough to make them the first priority of the model. If either of these two terms is present the model is infeasible and the infeasibilities are reported to the user. The sixth term, MP_lvl$_{w,r}$ penalizes the peak manpower for each area used in a week. This penalization makes the layup manpower smoother throughout the week. The seventh term, Desired_Tool_Rec$_z$ penalizes schedules that do not meet the desired tool recycle times but only the absolute minimum tool recycle times. The eighth term, Resched, penalizes rescheduling of bins. This term is appropriately weighted above the reward given for scheduling dummy nodes so that dummy nodes are eliminated instead of rescheduling more autoclave packings. The ninth term, Early$_i$, penalizes scheduling an autoclave packing to be cured at a time when the layup room for that autoclave packing could not have laid it up in that week. The final two terms, SatWeekend$_w$ and SunWeekend$_w$, penalize the number of shifts required on the weekend. Thus if a single load is starts on a Saturday shift it gets the penalty of 8 plus the penalty of 4 due to the OT term. This system makes sure that a greater penalty is accumulated for starting on different days of the weekend since employees would be required to come in on different days to start them.

The constraints of the MILP proposed are as follows. These are named according to the system used in the Pyomo code as shown in Appendix A3 in order to maintain consistency.

$$C1: \sum_{k \in SW_w} Load_{w,i,k} = 1 \quad \forall\ w \in W,\ i \in LL_w\ if\ DL_i <> 1$$

$$C101: \sum_{k \in SW_w} Load_{w,i,k} \leq 1 \quad \forall\ w \in W,\ i \in LL_w\ if\ DL_i = 1$$

The constraint C1 ensures that all bin packings that are not dummy loads are scheduled in the week that they are meant for. Constraint C101 allows the opportunity to schedule dummy bins in each week, but does not require all dummy bins to be scheduled. This is due to dummy loads being only present to increase schedule flexibility in case rescheduling is necessary; if there is no space to fit a dummy load the model will not become infeasible.

$$C2_0 : \Big(\sum_{t=0}^{t=min(C_z,ss-k)} \sum_{i\in AUX2_{min(k+t,ss-1)}} Load_{WinS_{k+t},i,min(k+t,ss-1)} * PL_{i,z}\Big)/tools_z \le 1 + Tool\_Inf_z$$

$$\forall k\in SW_0,\ z\in P_z$$

$$C2_{w,w+1} : \Big(\sum_{t=0}^{t=min(C_z,ss-k)} \sum_{i\in AUX2_{min(k+t,ss-1)}} Load_{WinS_{k+t},i,min(k+t,ss-1)} * PL_{i,z}\Big)/tools_z \le 1 + Tool\_Inf_z$$

$$\forall k\in SW_w \cup SW_{w+1},\ z\in P_z$$

$$C22_0 : \Big(\sum_{t=0}^{t=min(CD_z,ss-k)} \sum_{i\in AUX2_{min(k+t,ss-1)}} Load_{WinS_{k+t},i,min(k+t,ss-1)} * PL_{i,z}\Big)/tools_z \le 1 + Desired\_Tool\_Rec_z$$

$$\forall k\in SW_0,\ z\in P_z$$

$$C22_{w,w+1} : \Big(\sum_{t=0}^{t=min(CD_z,ss-k)} \sum_{i\in AUX2_{min(k+t,ss-1)}} Load_{WinS_{k+t},i,min(k+t,ss-1)} * PL_{i,z}\Big)/tools_z \le 1 + Desired\_Tool\_Rec_z$$

$$\forall k\in SW_w \cup SW_{w+1},\ z\in P_z$$

The constraints shown above make use of auxiliary set AUX2. Auxiliary sets like this one can be created by preprocessing to narrow down which variables or parameters a constraint must use when the mathematical expressions to define this set of variables or parameters are complex. A more in depth explanation of auxiliary sets can be found in section 4.2.1 Pyomo Basics. Constraint $C2_0$ and $C22_0$ are only present when a single week (week 0) is being scheduled. At all other times constraints $C2_{w,w+1}$ and $C22_{w,w+1}$ are activated for every week from week 0 to the penultimate week being scheduled. The C2 constraints ensure that the absolute minimum tool recycle times are met. These count for every time interval present and every tool that could be used at that time how many tools are being used, by dividing that by the total number of tools available that number should be less than or equal to one. In contrast, the C22 constraints give the Desired_Tool_Rec variable a positive value for each violation of the desired tool recycle time; it is calculated in a similar fashion as the absolute minimum recycle time constraint C2. The C22 constraint is not perfect since a single violation for a tool (not infeasible, but the slack variable Desired_Tool_Rec$_z$being 1) will enable other violations to go unpenalized, however due to computational time and the unlikeliness that this happens it was decided to leave it as is. The same happens for the C2 constraint, however here a slack variable being 1 does make the model infeasible. The model will still compute since the slack variable is explicit and the infeasibility will be reported to the user to help identify potential issues. This is further explained in the Implementation section.

$$C3_0 : \sum_{i,kk \in AUX3\_1_{j,k}} Load_{WKL_i ,i ,kk} \leq 1 + Au\_Inf_j \quad \forall j \in Au, k \in SW_0$$

$$C3_{w,w+1} : \sum_{i,kk \in AUX3\_1_{j,k}} Load_{WKL_i ,i ,kk} \leq 1 + Au\_Inf_j \quad \forall j \in Au, k \in SW_w \cup SW_{w+1}$$

Constraint $C3_0$ is only present when a single week (week 0) is being scheduled. Like C2 and C22, at all other times $C3_{w,w+1}$ is activated for every week from week 0 to the penultimate week being scheduled. This set of constraints ensures that two load's run times do not overlap in the same autoclave. It does this by summing the Load variable for all pairs (i,kk) of loads that if started would either start at or be running at time k. If more than one occupies the autoclave at the same time the Au_Inf$_j$ variable is turned on for that autoclave, indicating an infeasibility and being reported to the user, like in the C2 constraint.

$$C4 : \sum_{i \in L \cap LL_w if\ DL_i = 0} Load_{w,i,k} \leq Energy_w \quad \forall w \in W,\ k \in SW_w$$

The constraint C4 counts the highest number of simultaneous starts in each week and this number is penalized in the objective function for each week.

$$C5 : \sum_{w \in W} \sum_{i \in L \cap LL_w} \sum_{k\ \in\ weekends \cap SW_w} Load_{w,i,k} \leq OT$$

Constraint C5 counts the number of overtime (OT) starts for the schedule. The OT variable is penalized in the objective function.

$$C6 : \sum_{(i,k)\in AUX6_j} Load_{WKL_i,i,k} = 0 \qquad \forall j \in Au$$

Constraint C6 does not allow loads to to start or run during blackout times. It uses the AUX6 auxiliary set to identify the times when starting a load would interfere with a blackout period.

$$C7 : \sum_{i\in L\ if\ Rpenalize_i=1\ and\ DL_i<>1} (1 - Load_{WKL_i,\ i,\ B\_iter_i.scheduled}) <= \ Resched$$

Constraint C7 penalizes the rescheduling of loads where the Rpenalize parameter for that load is 1. This parameter comes from the AWS_d.p pickle file and can be changed by the user as required.

$$C8 : \sum_{k=MinInWeek_{WKL_i}}^{Earliest_i} Load_{WKL_i,\ i,\ k} \ \leq Early_i \qquad \forall i \in L$$

Constraint C8 uses the Early$_i$ variable which is penalized in the objective function to disincentivize loads from being scheduled at a time when they would require to be laid up the previous week by the layup room. This is done to decrease the likelihood of material outlife issues. The parameter Earliest defines the earliest time load "i" can be scheduled to be cured if laid up that week. When calculating this parameter in preprocessing the different layup room's shift schedules are taken into account.

$$C9 : \ (\sum_{i\in L\ if\ ALS_i=r} \ \sum_{k=WKL_i*7*24+d*24}^{k=WKL_i*7*24+(d+1)*24} Load_{WKL_i,\ i,\ k} * MPL_i)/(MPA_r * SAR_r * 8) \ \leq MP\_lvl_{w,r}$$

$$\forall w \in W,\ r \in Ar,\ d \in xrange(0, 7)$$

Constraint C9 calculates the highest manpower required from a each layup room on a single day for every week. By penalizing this variable, this reduces the peak manpower required, thus making it easier for layup rooms to have an even workload throughout the week.

$$C10: \sum_{i \in LL_w} \sum_{k=w*7*24+5*24}^{k=w*7*24+5*24+17} Load_{w,\,i,\,k} \leq SatWeekend_w * 100 \quad \forall w \in W$$

$$C11: \sum_{i \in LL_w} \sum_{k=w*7*24+5*24+17}^{k=w*7*24+5*24+24} Load_{w,\,i,\,k} \leq SunWeekend_w * 100 \quad \forall w \in W$$

Constraints C10 and C11 calculate loads that cause employees to come in on the weekend. In contrast to C5 where each overtime load is calculated and penalized in the objective function, these constraints count overtime shifts. By penalizing these Saturday and Sunday overtime shifts in the objective function the program incentivizes that if several autoclave loads must be started on a weekend they be done on a single day.

$$C12: \sum_{k \in SW_w} Load_{w,i,k} * k + wsame\_time_{i,ii} \leq \sum_{k \in SW_w} Load_{w,ii,k} * k + 1 \quad \forall w \in W, (i, ii) \in Wsame_w$$

Constraint 12 is present to reduce symmetry in the mathematical model and help it achieve better results faster, it does not limit the real solution in any way. Using the list Wsame of pairs of bins that are exactly the same it simply restricts one to be scheduled before the other, with the minimum tool recycle time separating them. Therefore the option of scheduling them with “ii” before “i” is removed, reducing the solution space. The computational improvement this constraint yields will be further explained in the Results section.

$$C13: \sum_{k \in SW_w} Load_{w,i,k} * k + wsomesame\_time_{i,ii} - \sum_{k \in SW_w} Load_{w,ii,k} * k - 1 \leq$$
$$0 + (MaxInWeek_0 + 1) * (1 - Before_{i,ii}) \qquad \forall w \in W, (i, ii) \in Wsame_w$$

$$C14:\ Before_{i,ii} + Before_{ii,i} == 1 \qquad \forall w \in W,\ (i, ii) \in Wsomesame_w$$

Constraints 13 and 14 generalize what constraint 12 does for identical bins to those bin pairs where both use the maximum number of tools available for one or more tools. Given that they are both using the maximum number of tools available it is impossible for their tool recycle times to overlap in any way, therefore one must be strictly scheduled before or after the other while separated by the tool recycle time. A binary variable called Before turns constraint 13 on or off depending on which one is scheduled before the other.

3.1.3 Proposed Model Cost vs Ideal Cost

Given that the model that uses hourly approximations this requires the rounding up of cure times to the nearest hour. This creates a perceived increase in cure-time demand by the model. This perceived increase was calculated with the equation shown below.

$$Perceived\ Increase = \frac{\sum_{all\ recipes} (Cure\ Time_{recipe}\ rounded\ up\ to\ nearest\ time\ interval - Cure\ Time_{recipe})}{\sum_{all\ recipes} Cure\ Time_{recipe}}$$

Using the recipe cure-times provided by TIP, and making the assumption that all recipes are used equally, the following table describes the loss of capacity under different discrete-time approximation methods. Although this is not an exact calculation given the previously stated assumption it does provide an idea of the magnitude of the time cost of scheduling using certain discrete time intervals. As can be seen below in Table 1, there is a large jump from using a time interval of 1 hour to 30 minutes. Therefore it may be of interest in the future to explore the possibility of modifying the model for a 30 minute time interval.

Table 1: Perceived Increase in Cure-Time Demand Due to Discrete Time Intervals

Time Interval	Perceived Increase in Cure-Time Demand
1 minute	0%
15 minutes	0.42%
30 minutes	1.56%
1 hour	7.22%
2 hours	16.15%

3.2 Other Methods Attempted

Other methods were attempted in the early stages of this research to better understand the problem and see if these methods provided a better or faster solution. The methods described in this section were found to have a lower performance than the proposed model methods.

3.2.1 Integer Program for Bin Packing

A binary integer program to solve the APP was considered. This program was coded using Pyomo and its objective was to minimize the number of bins used, therefore being a Bin Packing Problem as opposed to the one in the APP proposed model which also contained a secondary objective to also level out the load between autoclaves.

Table 2: Sets, Parameters, and Variables for Integer Bin Packing

Sets: B=list of all possible bins per autoclave, b Au= list of all possible autoclaves, j Parts= list of all parts, z Types= list of all part types, t R= list of all recipes, c right_type$_j$=list of parts z where Au_Feas$_{z,j}$=1 **Parameters:** Recipes$_{z,c}$=Identifies which recipe is attached to which part, binary PVolume$_z$=Volume of part z, integer PThermos$_z$=Thermocouples needed by part z, integer Priority$_{z,j}$=Priority level of autoclave j for part z Au_Feas$_{z,j}$=If part z can go in autoclave j	AuTime$_{z,j}$=Time in autoclave j for part z PType$_z$=Part type for part z Tools$_t$=Number of tools for part type t MaxPTool$_t$=Max parts per tool for type t AuVolume$_j$=Volume of autoclave j AuThermos$_j$=Thermocouples available in autoclave j **Variables:** vBinRec$_{b,j,c}$=Binary, if bin b on autoclave j uses recipe c. vPart$_{z,b,j}$=Binary, if part z goes into bin b of autoclave j. vBinOpen$_{b,j}$=Binary, if bin b of autoclave j is being used.

Objective:

$$Minimize: \sum_{b \in B} \sum_{j \in Au} vBinOpen_{b,j}$$

Constraints:

$$C1: \sum_{b \in B} \sum_{j \in Au} vPart_{i,b,j} = 1 \quad \forall i \in Parts$$

$$C2: \sum_{c \in R} vBinRec_{b,j,c} = 1 \quad \forall b \in B,\ j \in Au$$

$$C3: (2 - vPart_{z,b,j} - vBinRec_{b,j,c}) \geq 1 \quad \forall\ b \in B,\ j \in Au\ , (z \in Parts, c \in R) \cap (Recipes_{z,c} = 0)$$

$$C4: \sum_{z\ \in\ right\_type_j} PVolume_z * vPart_{z,b,j} \leq AuVolume_j \quad \forall b \in B,\ j \in Au$$

$$C5: \sum_{z\ \in\ right\_type_j} PThermos_z * vPart_{z,b,j} \leq AuThermos_j \quad \forall b \in B,\ j \in Au$$

$$C6: \sum_{z\ \in\ right\_type_j} vPart_{z,b,j} \leq MaxPTool_t * Tools_t \quad \forall b \in B,\ j \in Au, t \in Types$$

$$C7: \sum_{z\ \in\ Parts} vPart_{z,b,j} \leq 200 * vBinOpen_{b,j} \quad \forall b \in B,\ j \in Au$$

$$C8: vBinOpen_{b-1,j} \leq vBinOpen_{b,j} \quad \forall j \in Au,\ b \in B\ if\ b <> 0$$

C1 ensures that every part has a bin such that demand is met. C2 restricts each bin to have exactly one recipe. C3 ensures the right recipe for a part is attached to the bin it goes into. C4 and C5 ensure the parts fit into the autoclaves in terms of volume and thermocouples required. C6 makes sure that enough tools are available for the parts that go into a single bin. C7 uses 200 as a Big-M term to allow up to 200 parts to go into a cure if that bin is opened. C8 removes some symmetry in the problem by allowing only consecutive indices from 0 to be used for open bins.

This method of solving the APP proved to be feasible. However it proved difficult to prove optimality of a solution. For example, in a small test case (approximately 2 days worth of production) the less than the lowest number of bins obtained after 1200 seconds of computation was 22 bins. That number was initially computed after 45 seconds and the rest of the time is spent bringing the lower bound up, reaching 59.1% gap after 1200 seconds. The hill-climbing method in the proposed model section reached the solution of 22 bins in less than 1 second. Therefore, given the simplicity of the hill-climbing method, its good performance, and the lack of need for an advanced mathematical solver to provide a solution, the IP method for solving the APP was not considered further.

3.2.2 Simulated Annealing for Scheduling

For scheduling of autoclave packings a simulated annealing approach was attempted. This approach was tried early on and both the data used to test it as well as the constraints included do not accurately represent the full problem. The method as developed over the course of the project relied primarily on two variable vectors. The first vector represents the sequence that loads are scheduled in. The second vector represents a "slack" vector, where loads are scheduled later than the earliest time they can be scheduled at. The method starts by computing these two vectors then feeding items from the sequence vector to a scheduling function. This function determines the earliest time that particular load can be scheduled given all current loads already scheduled, then it adds that load's slack time to the earliest possible to arrive at the scheduled time as shown in the equation below.

$$Time_{load} = Earliest\ possible + \ slack_{load}$$

By going through this procedure the objective is essentially makespan minimization. The slack times were included in the method to allow flexibility around scheduling during weekends. Given that the algorithm could not foresee when weekends happened, or if it would be necessary to schedule something on a weekend, it proved to have very poor performance in terms of weekend overtime scheduling.

This method proved to be very parameter sensitive, especially in respect to the cooling schedule used. Several rudimentary cooling schedules were tried as well as several iterations of the method were also run in an attempt to coax better solutions from it. However it proved to be of worse performance than the preliminary MILP method that was refined into the proposed model and thus work on simulated annealing for scheduling was discontinued.

Chapter 4 Implementation

As of the time of this writing the project has been tested with realistic scale problems but has not been fully implemented at TIP. However, the necessary steps to implement it have been carried out and management is planning to fully implement the project in the following months. Several adjustments, features, and data modifications were necessary to take the proposed mathematical models and design a complete solution that could be used in an operational environment.

To carry out the implementation at TIP it was necessary to conduct multiple plant visits. Almost weekly meetings were held for a six month period. During these meetings work was done with TIP to define the problem objectives and constraints. The limitations to implementing any program that would be given to them were also identified. Once early models were created, it was possible to use the output from these to help identify further constraints. The output from early models was shown to several stakeholders and they were able to point out system constraints which were being broken in the output since they were not included in the model. These were then added to the model and the review process started again. After several iterations of this process with different stakeholder groups it was possible to arrive at a schedule and set of objective weights everyone agreed on would be feasible and provide a good schedule. A further explanation on the objective weights is found in 4.1 Objective Function.

A second important aspect of the implementation work done here was to verify the accuracy of data being fed into the program. As shown later in section 4.4 Data Inputs, there are several Excel sheets which contain various pieces of data. Given the large amount of data as the learning curve in creating these documents, sometimes pieces of data that were entered into the Excel sheets would be inaccurate. When the demand for a week that has already passed and had been produced was fed into the model for testing purposes, sometimes it would return infeasible. This clearly was impossible and often resulted with the identification of inaccurate data. This process required extensive troubleshooting, data modification, and analysis of model inputs and outputs.

The following sections will provide details on how the proposed model was transformed into a solution that would solve the TIP problem. Section 4.1 will present the objective function weights used at TIP. Section 4.2 will discuss the programming languages used to code the software that was delivered to TIP, including the proposed model. Section 4.3 presents the software structure used. Sections 4.4 and 4.5 and will present the data inputs and outputs to the software. Any data shown in these sections is for demonstration purposes only and is not real data due to confidentiality. Following the data inputs and outputs the data modifications necessary to accurately represent the problem are explained. Section 4.6 presents data modifications necessary to make the provided data conform to model inputs. Section 4.7 presents a list of software features that allow for modification on how the problem is solved. Finally, section 4.8 presents a solution method to improve computational performance of the problem.

4.1 Objective Function

A major step in implementation of the software at TIP was determining the right weights for the objective function. The weights shown in the objective function below were obtained by first setting priorities, for example the weight of 3 was given to Energy while the 0.3 was given to the dummy load variables to ensure that the scheduling of dummy loads did not lead to higher peak power. Similarly the Desired Tool Recycle Time variable was given a weight of 2 while OT was given a weight of 4 to show that the reduction of overtime is more important than meeting desired tool recycle times. While these initially were slightly different, by going through several scheduling runs with the PPL's and Cure Centre personnel they provided feedback as to what would make a better schedule. The weights were changed as necessary until the optimization provided a schedule that best suited their needs. Future work could refine the objective function so that it takes into account actual costs, making the objective to minimize the cost of running the schedule.

$$Minimize: \; 4 * OT - 0.3 * \sum_{w \in W} \sum_{i \in DL \cap LL_w} \sum_{k \in S} Load_{w,i,k} + 3 * \sum_{w \in W} Energy_w$$

$$+ 1000 * \sum_{z \in P} Tool\_Inf_z + 1000 * \sum_{j \in Au} Au\_Inf_j + 4 * \sum_{w \in W} \sum_{r \in Ar} MP\_lvl_{w,r}$$

$$+ 2 * \sum_{z \in P} Desired\_Tool\_Rec_z + 1.5 * Resched + 1.4 * \sum_{i \in L} Early_i$$

$$+ \sum_{w \in W} 8 * SatWeekend_w + 20 * SunWeekend_w$$

4.2 Programming Languages

Due to the researcher's familiarity with Python, its ease of programming, and its ability to easily interface with thousands of pre-built modules including several mathematical programming languages this language was chosen to build the software on. Two main mathematical programming languages interface well with Python, these being PuLP and Pyomo. Both were tried in early stages and Pyomo proved to be a better option given its ease of programming, speed, and active development. This last point is exemplified by the fact that during the implementation phase of the project at least one large update to Pyomo happened which changed its structure, resulting in simpler and more efficient coding.

4.2.1 Pyomo Basics

A detailed description of one of the model constraints as coded in Pyomo will be provided in this section. The full model as written in Pyomo is found in Appendix A3. The first constraint of the proposed model, C1, is as follows:

$$C1: \; \sum_{k \in SW_w} Load_{w,i,k} = 1 \quad \forall \; w \in W, \; i \in LL_w \; if \; DL_i <> 1$$

This constraint's function is to ensure that all non-dummy loads are scheduled. The sum of Load variables over all time intervals "k" during the week for which load "i" should be scheduled must sum to 1. This is carried out for every week "w" and for all loads "i" within that week that are not dummy loads. Dummy loads are used to increase schedule flexibility and are thus dealt with separately in constraint 101. The C1 constraint is modelled in Pyomo as shown below.

```
1 def set_init1 ( model ):
2      return [( w , i ) for w in model . W for i in set ( model . LL [ w ])\
3             if model . DL [ i ]<> 1]
4 model . AUX1 = cpr . Set ( dimen = 2 , initialize = set_init1)
5
6 def con1_rule ( model , w , i ):
7      return sum ( model . Load [ w , i , k ]\
8             for k in model . SW [ w ])== 1
9 model . con1 = cpr . Constraint ( model . AUX1 , rule = con1_rule , name = 'Con1')
```

Lines 1 through 4 define an auxiliary set, while the constraint is constructed in lines 6 through 9. The auxiliary set is constructed via the use of a function, "set_init1". The function is given the model object, "model", which is what contains all constraints, parameters, variables and sets. Lines 2 and 3 define the contents of the auxiliary set. In this case, the auxiliary set is a list of tuples, a pair of values, which stand for the week "w" and load number "i". This pair is created for every week in model.W and for every load within that week (denoted in line 2 by "for i in set (model.LL[w])), these loads "i" are further filtered to not include any dummy loads as shown in line 3.

Once the auxiliary set is created, the constraint itself must be defined. Line 6 through 8 define the function by which the constraint is created. This function is called for every pair of "w" and "i" given to it, this is denoted by giving the inputs of model, "w", and "i" to the function in line 6. The constraint starts with the sum of the model.Load variable for all "k" in the week during which model "i" is supposed to be scheduled as shown in lines 7 and 8. This sum must equal 1 as stated in line 8. Finally, once the function has been declared it must be called over a given set of “w” and “i” pairs. This set is the previously declared auxiliary set AUX1. Line 8 defines the object model.con1 as a constraint (from the cpr/Pyomo module) and gives it AUX1 to be called over. It also provides the function as a rule over which AUX1 is called, finally it gives it a name “Con1” by which it will appear in any .LP file that is created.

4.3 Software Structure

The scheduling problem was modelled using a Mixed-Integer Linear Program (MILP) which requires a mathematical solver to obtain a solution. To solve the MILP problem Gurobi Cloud solver is used. This solver was selected due to its speed, ability to solve complex problems, and flexible licensing option through their cloud service. For the TIP case the on-demand cloud option was selected since using this option under the expected usage yielded minimum cost.

Gurobi Cloud is deployed on Amazon Web Services (AWS), where a computer instance is rented for approximately \$1 per hour to run the software and the licensing of Gurobi Cloud costs \$25 per hour or part thereof. The AWS instance can be started and stopped automatically using an Application Programming Interface (API) call. This allows TIP to launch the program and all the solving. Instance starting, licensing of software, and instance stopping is done in the background, without their involvement every time.

Due to the multiple computers interacting together a more complicated system is created. A high-level view of how the software is structured is found in Figure 14. The data is contained in Microsoft Excel sheets and those are read in by the local program on-site at TIP. The data from these is stored in the pickle file which will be further discussed in Section 4.4.8. The local program then calls the AWS side program to run any optimization required, and this carries out the solving of the APP as well as creates the scheduling MIP. A third computer is then called which contains Gurobi Cloud and this one solves the MIP. Finally the solution is sent back to the local computer and the output is shown to the user.

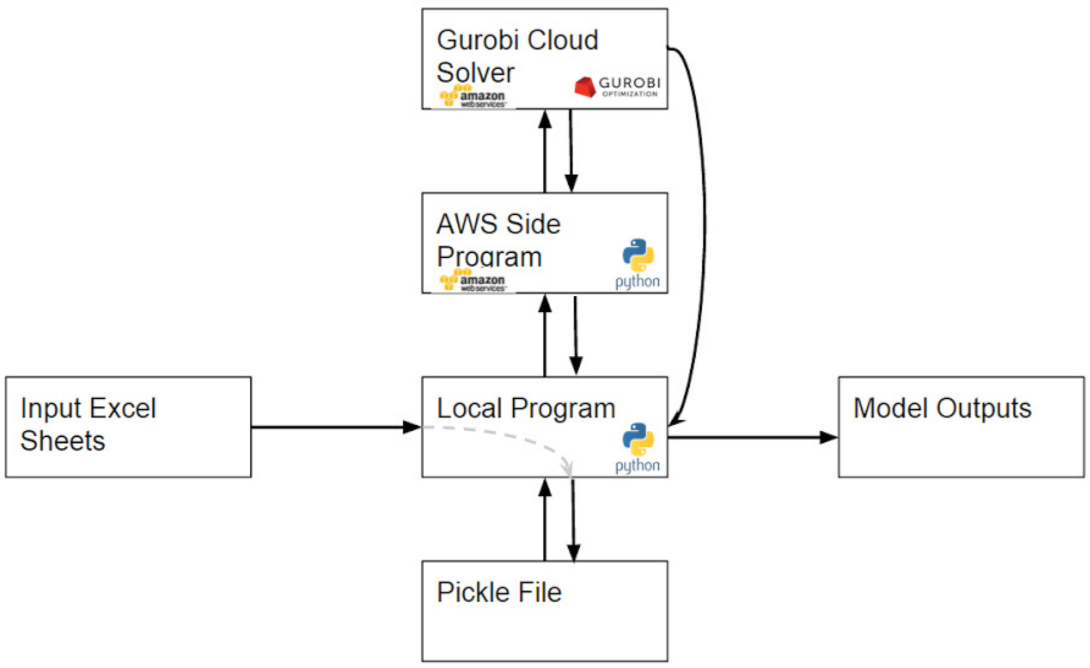


Figure 14: High-Level Software Structure

A workflow view of the whole process can be seen in Figure 15. This process is done via the use of Secure Shell (SHH) calls, application programming interface (API) calls, and using Gurobi's own communication channels. AWS is the host used for the Gurobi Cloud solver and uses one instance. Another instance was utilized to solve the APP, create the MILP problem, and receive the MILP solution prior to it being downloaded to TIP's computer. As shown in Figure 14 the process starts at the local machine where inputs such as demand, blackout periods, and other scheduling preferences are determined. At the local computer all inputs that are sensitive are encoded and then the linux server on AWS is started via an API call. The problem data is sent and the APP is solved on this computer. Following the APP solution the MILP problem is coded using Pyomo and via another API call the Gurobi Cloud Server is set up. Once the GCS is set up the MILP problem starts solving on it. Once the MILP problem is solved the solution is downloaded to Pyomo and written out into a file which can be downloaded to the local machine. A message is sent to the local machine at TIP that the problem has finished solving, this triggers the downloading of the solution file and shuts down all AWS instances. Finally the solution is decoded and the results are displayed to the user as shown in the Output section.

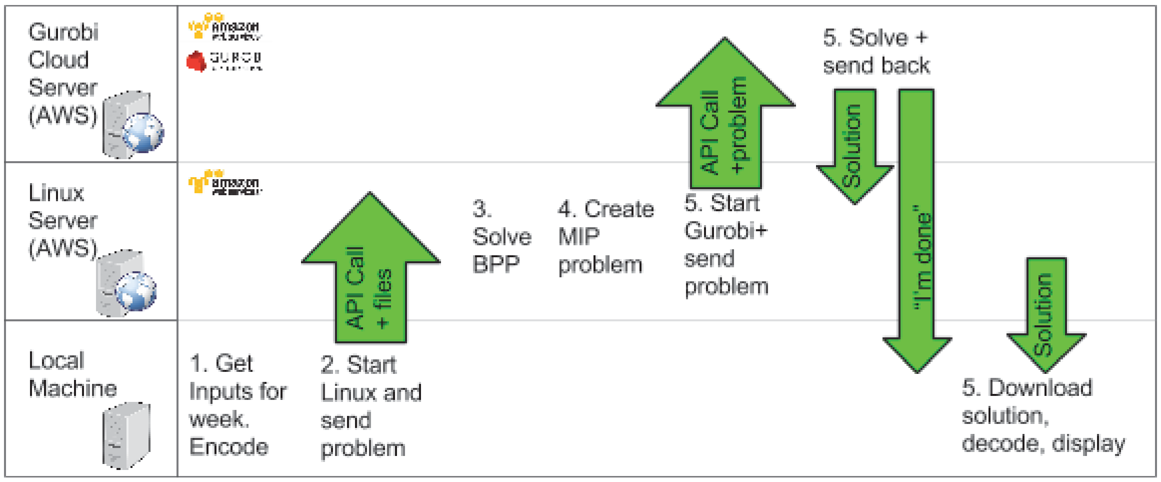

Gurobi Cloud Server (AWS)
Linux Server (AWS)
Local Machine
1. Get Inputs for week. Encode
2. Start Linux and send problem
API Call + files
3. Solve BPP
4. Create MIP problem
5. Start Gurobi+ send problem
API Call +problem
5. Solve + send back
Solution
"I'm done"
Solution
5. Download solution, decode, display

Figure 15: System Workflow View

In order to transfer the problem data between machines and store it between changes the pickle file previously discussed was used (Python Software Foundation). This file structure can store most of the data pertaining to a specific schedule. Other files like Areas.xlsx, Autoclaves.xlsx, and Calendar.xlsx hold data which changes less often or must be manually changed in Excel by the user and are also passed to the AWS server when needed.

4.4 Data Inputs

The major and ongoing task in implementing this model at TIP was to obtain the correct data. Given the amount of model inputs, the team's familiarity with Microsoft Excel, and the ease of programming between Python and Excel it was decided to hold all data in Excel. Standard relational tables were used to maintain data integrity. The following sheets were created and populated with TIP's information. Given the sensitive nature of this information all examples show made up data. The different data sheets and how they interact are shown below in Figure 16 in an entity-relationship diagram. The Recipes List and Recipes-Autoclaves entities shown in the diagram below are described in one Excel sheet only, the Recipes List. Every other entity uses is described in its own Excel sheet.

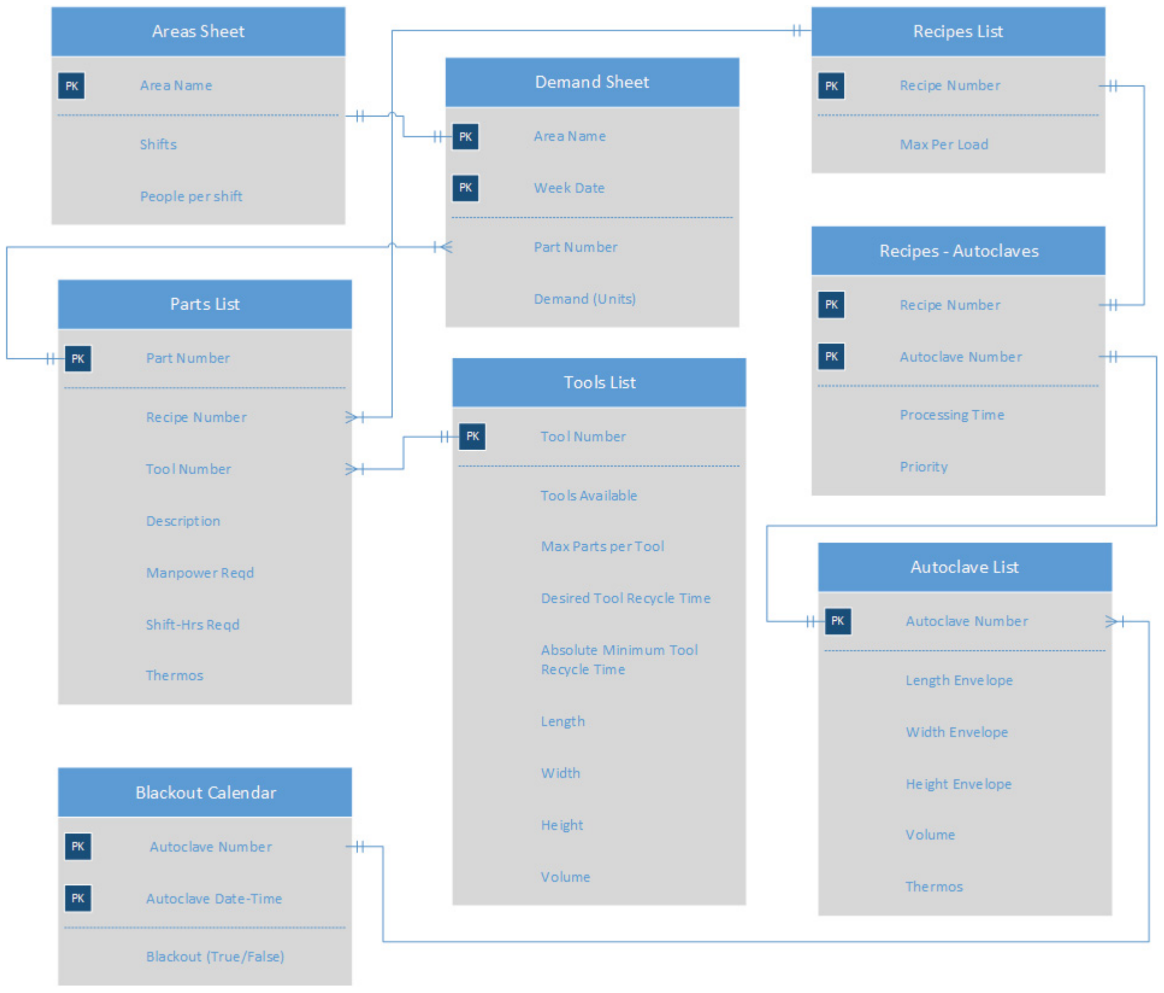
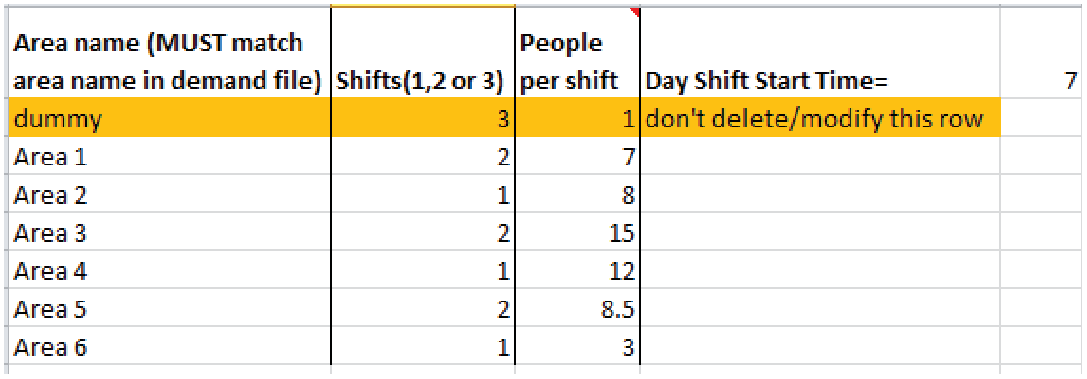


Figure 16: Entity-Relationship Diagram for Data Input Sheets

4.4.1 Areas Sheet

A sheet containing the names of each of the layup areas is required, this sheet stores parameters that are specific to each area. An example of this sheet is shown in Figure 17. It also describes how many shifts per day each area is working given different areas work on different shift schedules. The final piece of information is the number of people per shift that each area has. Although these two parameters can change from week to week, it is not expected that they will change often and therefore are applied to the entire planning horizon. Ideally however they could be changed on a weekly basis to allow transition periods to be smoother. Finally this file also contains the start time of the day shift, default is 7AM. It also has the same area information for a "dummy" area which is used to schedule usable gaps and is used in constraint 2 of the proposed model.

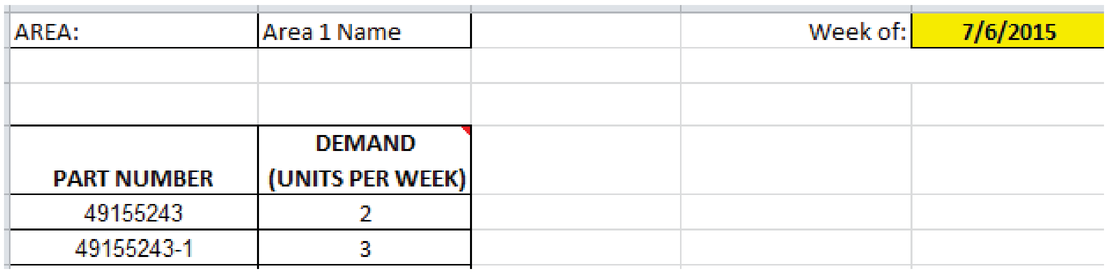


Area name (MUST match area name in demand file)	Shifts(1,2 or 3)	People per shift	Day Shift Start Time=	7
dummy	3	1	don't delete/modify this row	
Area 1	2	7		
Area 2	1	8		
Area 3	2	15		
Area 4	1	12		
Area 5	2	8.5		
Area 6	1	3		

Figure 17: Areas Sheet Example

4.4.2 Demand by Area

Demand sheets need to be provided by each of the Program Production Leaders (PPLs) for their area for each week of the planning horizon. The sheet specifies the area name, which is matched with the Areas Sheet, it then lists each of the part numbers to be produced that week as well as the quantity required. For the PPL's sake and to provide organization it also lists the date of the Monday that starts that week. The program will scan the part number list row by row until it finds an empty row and then stops, given this method it is necessary that there are no gaps in this list. An example of one of these demand sheets is shown in Figure 18 with made up part numbers. Demand sheets are organized in folders by week, all demand sheets in a folder for a week are read into the program. Each folder is named using the 'YYYY month DD' naming scheme where the month name is fully spelled out.

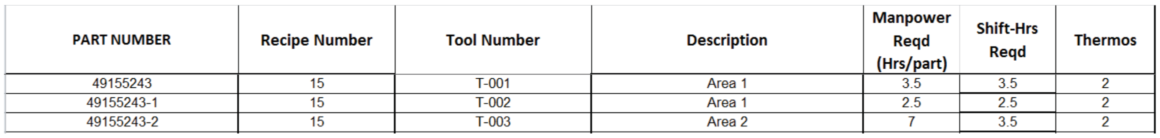


AREA:	Area 1 Name		Week of:	7/6/2015
PART NUMBER	DEMAND (UNITS PER WEEK)			
49155243	2			
49155243-1	3			

Figure 18: Demand Sheet Example

4.4.3 Parts List

The different part numbers that can be built are stored in the parts list sheet along with several attributes which can be seen in Figure 19. The part numbers in the demand sheets must match one in the parts list in order to be built. If they do not match an error showing the missing part number in the parts list is shown. The recipe number and tool numbers are linked to the recipe sheet and tools sheets which will be explained in the following sections. The description typically shows the area name but at times is also used to describe the part, this section is not used by the model and only there for data entry and manual query purposes. The manpower required per part is described in this sheet. This information refers to total amount of manpower which may include several operators. The shift-hours required per part are also stated here and these are used in conjunction with the Early variable in constraint 8. Shift hours can vary from manpower hours required due to number of operators, resting time required while building the part, and other factors. Finally the number of thermocouples required by the part are also provided.

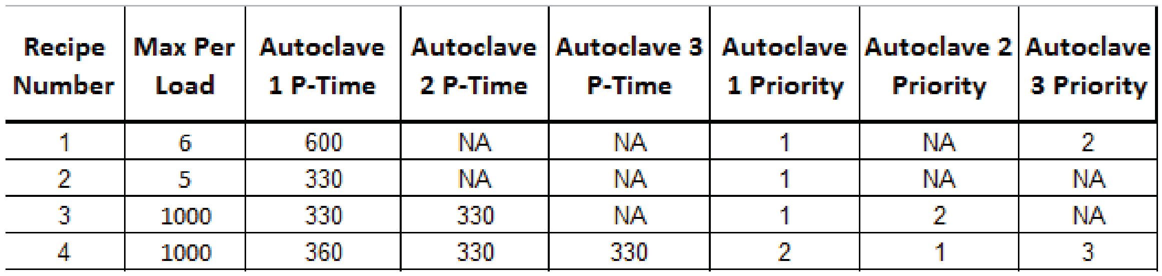


PART NUMBER	Recipe Number	Tool Number	Description	Manpower Reqd (Hrs/part)	Shift-Hrs Reqd	Thermos
49155243	15	T-001	Area 1	3.5	3.5	2
49155243-1	15	T-002	Area 1	2.5	2.5	2
49155243-2	15	T-003	Area 2	7	3.5	2

Figure 19: Parts List Sheet Example

4.3.4 Recipes List

The recipes list contains the same recipe numbers as those provided in the parts list. These link recipes to priority levels on autoclaves, autoclave runtimes, as well as max parts per load for that recipe. An example with made up data of these is shown in Figure 20. The Max per Load parameter is usually used to indicate customer specification constraints, when this constraint is not applicable the value of 1000 is placed instead given that every load TIP cures has a much lower number of parts than 1000 in it. This constraint is used in the APP. The times shown for the autoclaves are in minutes. There are sometimes differences in times between autoclaves as can be seen in the example below for recipe 4. These differences in cure times are due to different autoclave heating and pressurizing rates. Finally the priority for autoclaves is dictated by the preferences of the cure center PPL and are based on several factors including their experience and cost of running these recipes on each autoclave.

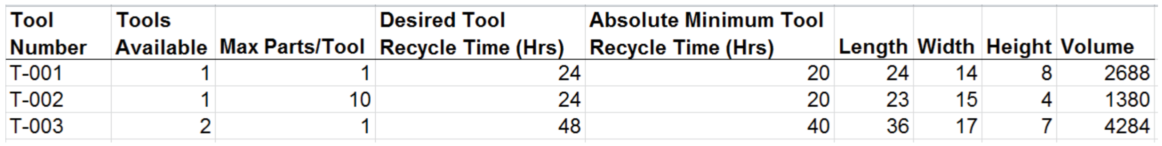


Recipe Number	Max Per Load	Autoclave 1 P-Time	Autoclave 2 P-Time	Autoclave 3 P-Time	Autoclave 1 Priority	Autoclave 2 Priority	Autoclave 3 Priority
1	6	600	NA	NA	1	NA	2
2	5	330	NA	NA	1	NA	NA
3	1000	330	330	NA	1	2	NA
4	1000	360	330	330	2	1	3

Figure 20: Recipes List Sheet Example

4.4.5 Tools List

The tools list contains the same tool numbers as those provided in the parts list, an example of this list is shown in Figure 21. Each tool number may have several physical copies of the tool. Typically at TIP each copy is marked by a -1,-2,-3, etc on the tool number. This system was modified so that the number of tools available showed the number of copies, not different tool numbers followed by a dash and a number. Some tools can make several parts at once, much in the same way that a single egg carton can carry a dozen eggs. This ability is shown in the 'Max Parts/Tool' parameter and is used in the APP. The desired and absolute minimum tool recycle times are also parameters specified in this sheet. Finally the length, width, and height of each tool as well as the volume are specified for use in the APP.

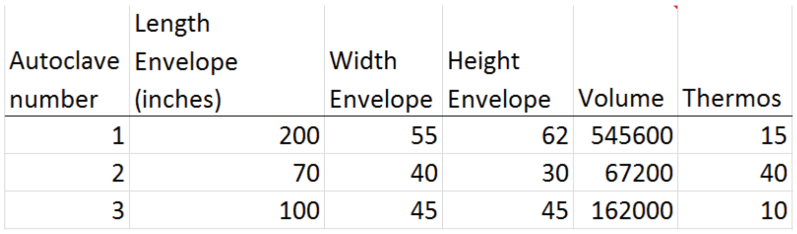


Tool Number	Tools Available	Max Parts/Tool	Desired Tool Recycle Time (Hrs)	Absolute Minimum Tool Recycle Time (Hrs)	Length	Width	Height	Volume
T-001	1	1	24	20	24	14	8	2688
T-002	1	10	24	20	23	15	4	1380
T-003	2	1	48	40	36	17	7	4284

Figure 21: Tools List Example

4.4.6 Autoclave List

The autoclave list stores the relevant parameters specific to each autoclave, an example of this can be seen in Figure 22. The length, width, and height of the inside of the autoclaves is provided here. The volume provided is the product of these three attributes reduced by 20%. This reduction is used to account for space that cannot be possibly used given the different part geometries, racking systems required, and the autoclave shape. The amount of the reduction was originally set by Devine and Kyle at 25% but was later reduced to 20% while fine-tuning the APP (Devine & Milne, 2014).

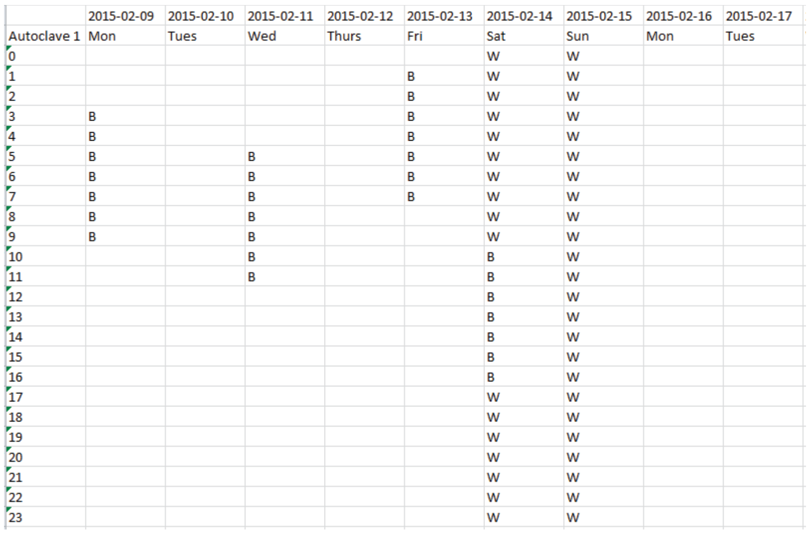


Autoclave number	Length Envelope (inches)	Width Envelope	Height Envelope	Volume	Thermos
1	200	55	62	545600	15
2	70	40	30	67200	40
3	100	45	45	162000	10

Figure 22: Autoclave List Example

4.4.7 Blackout Calendar

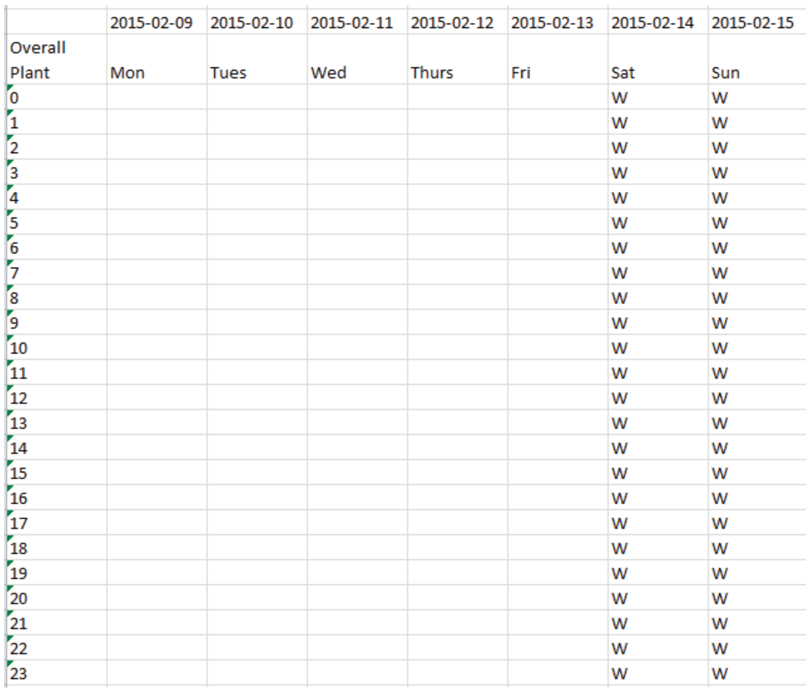
Due to maintenance, cures that are not scheduled by the model, and holidays it became necessary to input times when autoclave loads could not be scheduled by the model. These are shown by the blackout calendar shown in Figure 23. For each week of the planning horizon a blank calendar is created for every autoclave as well as for the plant in general. Times can be designated as "blackout" hours by placing a "B" in the corresponding cells. These blackout times are used in constraint 6.



	2015-02-09	2015-02-10	2015-02-11	2015-02-12	2015-02-13	2015-02-14	2015-02-15	2015-02-16	2015-02-17
Autoclave 1	Mon	Tues	Wed	Thurs	Fri	Sat	Sun	Mon	Tues
0						W	W		
1					B	W	W		
2					B	W	W		
3	B				B	W	W		
4	B				B	W	W		
5	B		B		B	W	W		
6	B		B		B	W	W		
7	B		B		B	W	W		
8	B		B			W	W		
9	B		B			W	W		
10			B			B	W		
11			B			B	W		
12						B	W		
13						B	W		
14						B	W		
15						B	W		
16						B	W		
17						W	W		
18						W	W		
19						W	W		
20						W	W		
21						W	W		
22						W	W		
23						W	W		

Figure 23: Blackout Calendar for Autoclave 1

The weekends are also pre-specified in this calendar for overtime. However, additional days can also be labelled as "W" such that there are penalties for scheduling loads during those times. This feature can be used for holidays and other days where it is undesirable to schedule a load but not impossible. This is displayed in the overall plant section of the sheet as shown in Figure 24, the 'W's in the individual autoclave sheets are for reference only and do nothing in the model.



	2015-02-09	2015-02-10	2015-02-11	2015-02-12	2015-02-13	2015-02-14	2015-02-15
Overall Plant	Mon	Tues	Wed	Thurs	Fri	Sat	Sun
0						W	W
1						W	W
2						W	W
3						W	W
4						W	W
5						W	W
6						W	W
7						W	W
8						W	W
9						W	W
10						W	W
11						W	W
12						W	W
13						W	W
14						W	W
15						W	W
16						W	W
17						W	W
18						W	W
19						W	W
20						W	W
21						W	W
22						W	W
23						W	W

Figure 24: Overall Plant Calendar Sheet for Weekends

4.4.8 AWS_d pickle file

The program is designed to be updated and each new week must be added to the existing schedule. Given this design it is required to store the data of the schedule between runs in a form that Python can easily read. The 'AWS_d' pickle file does just that by acting as the program's memory. The pickle file format that Python creates can store in a file any object, in this case the file stores a dictionary which contains several other objects within it. At the start of each run the pickle file is read and the program knows what the previous schedule looked like, then any changes like adding or removing weeks of demand to the schedule can be completed without recomputing the entire schedule. The structure within the dictionary that is stored in the pickle file is shown below in Figure 25.

```
AWS_d['Weeks']=range(0,5)
AWS_d['Tools']=Dictionary of all tools, updated every time new demand is added
AWS_d['Feasibility']=True/False depending on model feasibility
AWS_d['FeasibilityData']=List of infeasibilities if model infeasible
AWS_d['StartTime']=Datetime object with year, month, and day of first Monday
AWS_d['BNum']=Bin number counter, ensures all bins created have unique numbers
    for w in AWS_d['Weeks']:
        AWS_d[w]=Dictionary containing following:
        AWS_d[w]['Start']=Date of week's Monday
        AWS_d[w]['Action']= Could be 'Empty','BPP_Done' or 'Parts'
        AWS_d[w]['BPP']=Store the APP file
        AWS_d[w]['Parts']=Stores demand when 'Action' is 'Parts'
        AWS_d[w]['DummyArea']='dummy'
```

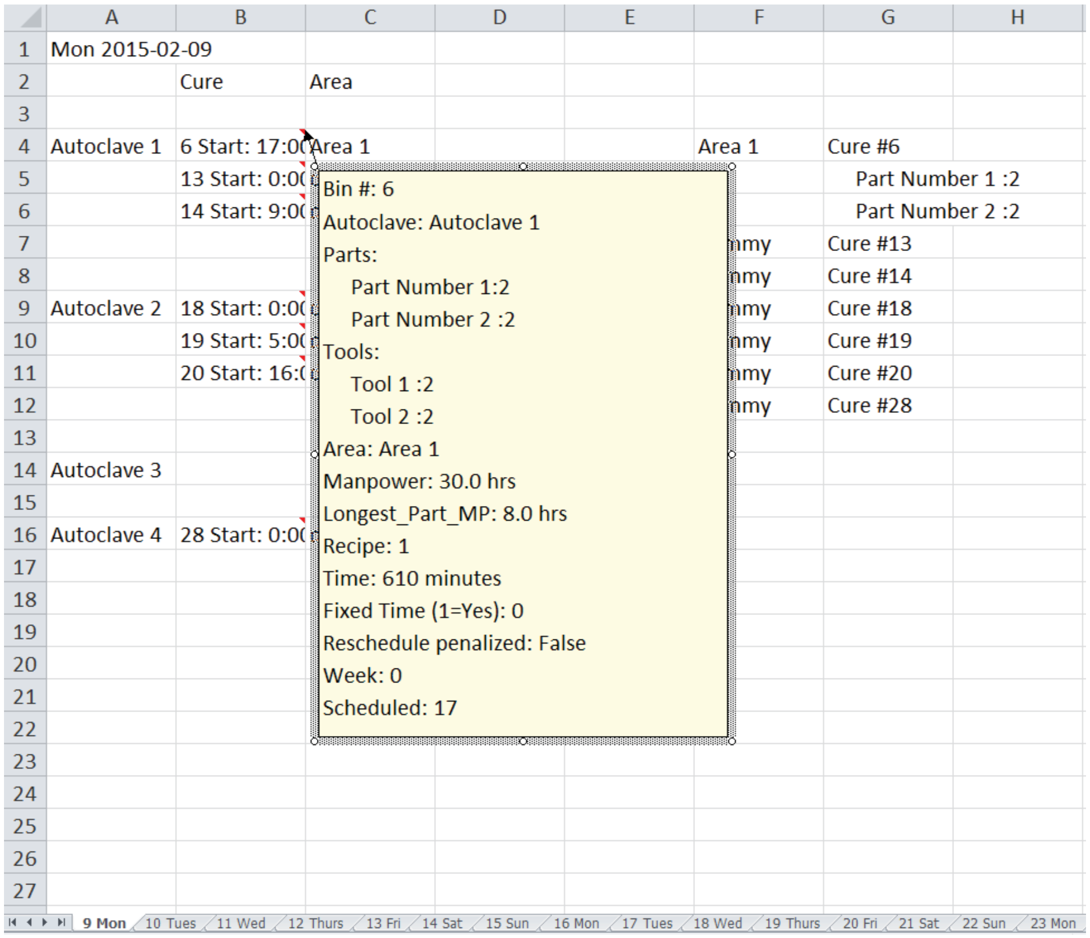
Figure 25: AWS_d Pickle File Structure

As can be seen the file stores more than just the previous schedule, but also information during the optimization. It is also used to pass information back and forth from the Amazon Web Services (AWS) servers, this will be further explained in the SSH Calls section.

4.5 Output

The output optimization model has several outputs, mainly an updated version of the AWS_d pickle file and an Excel sheet to provide schedule information to users. Additionally a visualization of the schedule is possible, that will be covered in section 4.7.8 under Software Features.

The Excel schedule is automatically created when the program finishes running and displays the data of what each bin contains and when it is scheduled for. As shown in Figure 26, a different sheet is created for each of the days in the planning horizon. Within each sheet the cures are organized by autoclave number. Comment boxes for each cure display the information relating to each cure such as parts, tools, and recipe numbers.



	A	B	C	D	E	F	G	H
1	Mon 2015-02-09							
2		Cure	Area					
3								
4	Autoclave 1	6 Start: 17:00	Area 1			Area 1	Cure #6	
5		13 Start: 0:00					Part Number 1 :2	
6		14 Start: 9:00					Part Number 2 :2	
7						mmy	Cure #13	
8						mmy	Cure #14	
9	Autoclave 2	18 Start: 0:00				mmy	Cure #18	
10		19 Start: 5:00				mmy	Cure #19	
11		20 Start: 16:0				mmy	Cure #20	
12						mmy	Cure #28	
13								
14	Autoclave 3							
15								
16	Autoclave 4	28 Start: 0:00						
17								
18								
19								
20								
21								
22								
23								
24								
25								
26								
27								

Bin #: 6
Autoclave: Autoclave 1
Parts:
    Part Number 1:2
    Part Number 2 :2
Tools:
    Tool 1 :2
    Tool 2 :2
Area: Area 1
Manpower: 30.0 hrs
Longest_Part_MP: 8.0 hrs
Recipe: 1
Time: 610 minutes
Fixed Time (1=Yes): 0
Reschedule penalized: False
Week: 0
Scheduled: 17

Figure 26: Example Production Schedule Output

4.6 Data Modification

Given the limits of the data inputs some modification of how data was presented was necessary. This was necessary in three areas: manpower levelling, tool-part interactions, and recipes. These modifications in data input allow the software to more accurately model the system at TIP while keeping the number of data inputs and parameters low.

4.6.1 Manpower Levelling Modifications

Every layup area provides a demand sheet which is turned into autoclave packings. Then the scheduling model attempts to level out the manpower usage of bins that are cured each day throughout the planning horizon for bins that came from one demand sheet. This process assumes that all layup personnel within an area are cross-trained to build all parts. However this is not always the case, sometimes layup rooms will have segments within them that are dedicated to build only certain parts. If the layup room's demand is pooled together then individual segments of the room may have drastically uneven workload throughout the week. Due to this some layup rooms have been segmented into two or more areas for the purposes of the optimization. They submit two or more demand sheets based on segments of the layup room operate. This allows the optimization to better level out manpower requirements throughout the week given layup personnel's ability to build parts.

4.6.2 Tool-Part Interaction Modifications

Using the part and tool sheets provided, only certain part-tool relationships are possible to describe. Figure 27 shows the part-tool relationships that are possible. The first one is a standard one to one relationship. The second one a tool number has multiple physical tools (captured in the 'Tools Available' parameter of the Tools List). The third and fourth relationships have multiple part numbers requiring the same tool, they can use the total number of tools available together as a shared resource.

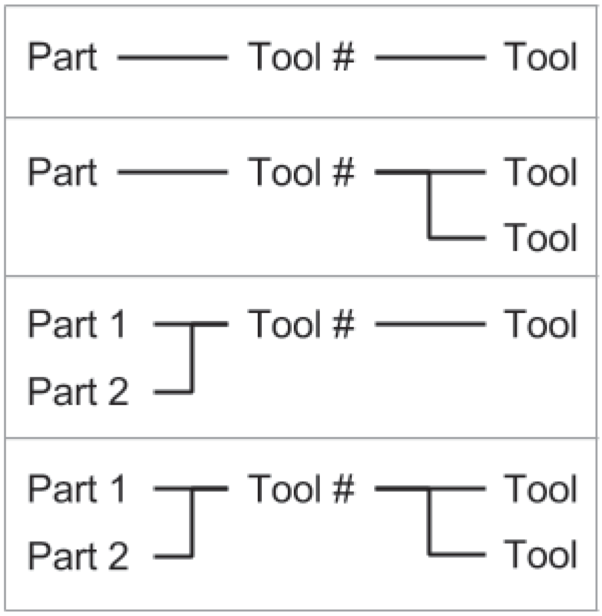


Figure 27: Diagram of Possible Part-Tool Relationships

Figure 28 below shows the part-tool relationships that are not possible to describe using the part and tool sheets described previously. There are modifications required to the data to make relationships of this type possible.

In the first case, two parts share a tool. For this case, it is necessary to merge the part numbers into a single part number. This means the ability to create only one of the two parts is lost in terms of the system, but the only effect this has is in terms of manpower levelling for the area. For most cases where this was done, both part numbers were regularly produced simultaneously on the same tool.

In the second case, a single part number uses two tool numbers. This is easily fixed by merging the two tool numbers. This appeared infrequently and was due to an additional small piece that was required to be placed on the tool. If only one of the two tools was also to be used by another part number that could create issues but this case was never seen at TIP.

The third case shows a part number having the ability to use one tool number and also the option to use a second tool number . Another part number is only able to use the second tool number but not the first. This second part prevents the merging of the tool numbers to create something like the second case from the possible part-tool relationships in Figure 27 above. At TIP this case was very rare and usually the use of the second tool by the first part number never actually happened. Therefore it was decided to lose the capability to schedule the first part number on either tool number and not describe the "OR" relationship shown.

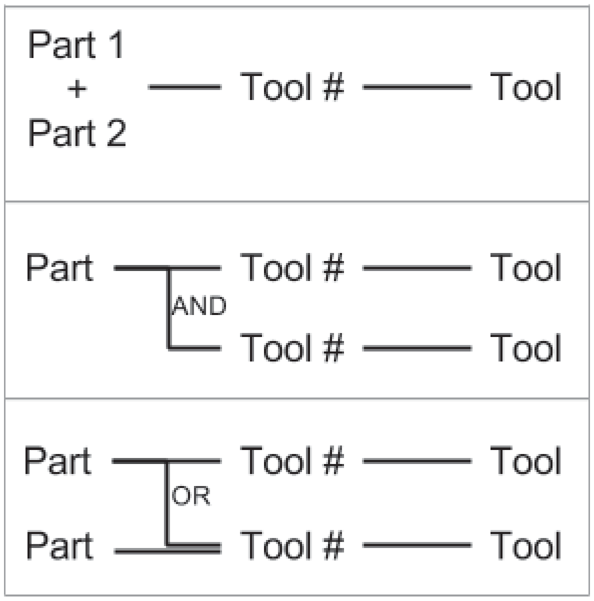


Figure 28: Diagram of Indescribable Part-Tool Relationships

4.6.3 Recipe Modifications

Recipes designate a particular temperature and pressure profile that must be followed by parts. However, when some parts have a much higher mass than others they get up to temperature slower. Therefore although the same recipe is applied there may be defects in parts if certain large and small parts are cured together. To create this break between large and small parts recipes some recipes were divided. For example recipe "1" may be changed to recipes "1a" for large parts and "1b" for small parts. This prevents large and small parts from mixing in a load. This recipe modification is necessary to carry out this segregation of parts effectively and in a manner that is easy for users to understand and change when needed.

4.7 Software Features

There are major differences between a mathematical model that can solve the mathematical program and a software that can solve the industry problem. Apart from a user interface and program robustness there exist other features which are necessary in the software to make it useful. For TIP it was required that the program could handle a rolling time horizon, where weeks were scheduled at a time and it could plan up to four weeks ahead. Given stakeholder preferences and possibly constraints not included in the model, manual scheduling and rescheduling of cures was also a necessary feature. Other features include the ability to penalize rescheduling of cures, modification of cure contents, manually adding cures, fixing the scheduled time of cures, and version control. These features will be discussed in more detail in the following sections.

4.7.1 Rolling Time Horizon

The TIP problem will be solved at a minimum once per week as new demand is added. The AWS_d structure (found in section 4.4.8) can hold a maximum of five weeks. This is so that a new week can be added to the model and then the first week can be removed. When a new week is added the process is simple, the w=4 section of the AWS_d file is updated to reflect the new demand being added. However the optimization is only built to handle at most four weeks, thus the first week (w=0) must be removed. Removing this week is done through the user interface and it causes the AWS_d['StartTime'] value to be updated to that of week 1. Then all the remaining weeks are cascaded down to fill in the void left by week 0. The process of managing the rolling time horizon is almost without work by the user, the user must simply request to either add a new week or remove the first week. This aids data integrity through minimal input by the user.

4.7.2 Penalize Rescheduling

The program is designed to be run on a weekly basis with each time adding a week of demand and removing a week since it is now in the past. Knowing the previous schedule allows the optimization to make the minimal number of changes necessary when modifying the schedule. Penalizing rescheduling is accomplished through constraint 7. Every autoclave packing has an Rpenalize attribute which is set to either True or False. If a schedule is being re optimized all autoclave packings with the Rpenalize set to True incur a penalty if they are not scheduled at the time of the original schedule, as per constraint 7. This ensures that changes to the schedule are minimal, only those required to maintain feasibility or to ensure no large penalty costs are created.

This rescheduling penalty can also be applied to any week currently scheduled and is very useful for when a new week is added to the model but the existing schedule should not be changed. A feature to set the Rpenalize attribute for all autoclave packings in a week is available through the user interface.

4.7.3 Fix Schedule

In the same way that changes to an existing schedule can be penalized, they can also be prevented. A specific autoclave packing or an entire week can be fixed such that no changes to this are done when running the optimization again. This helps when a week, or part of a week, is in the past but changes must be made to the rest of the schedule which is yet to take place. In the User Interface options to automatically fix a certain week, bin, or just whatever portion of the schedule that is in the past are available.

4.7.4 Manual Scheduling

No system is perfect and thus requires manual intervention every so often. When deadlines are missed, priorities rearranged, or materials are unavailable, scheduled autoclave packings must be changed. A feature in the user interface allows for manually scheduling or rescheduling of autoclave packings. This feature is meant to be used when cure times are missed, aircraft-on-ground (AOG) situations are encountered, or preferences differ from that which the optimization model deems best. The manual rescheduling does not verify feasibility of the schedule and must go through the again optimization to be verified but can be used without verification if desired. It is recommended that rescheduling of the changed bins be penalized or the bin schedule fixed prior to optimizing again.

4.7.5 Modify Cure

Another needed feature was to have the ability to manually modify a cure. Modifications here include adding and removing part numbers from a cure as well as changing the autoclave the bin is scheduled for. When a part number is added or removed the associated tools are automatically updated as well. Changing cure autoclaves is also available through the user interface. Changing autoclaves provides no guarantee that all the parts in the bin can fit in the autoclave, a manual scheduling or reoptimization to ensure no other constraints are violated should also take place.

4.7.6 Add Cure

Adding a completely new cure is a key feature that is also supported by the program. This feature is mainly focused at adding demand was not foreseen, such as last minute changes or AOG situations. Adding a cure must be followed by either manual scheduling or running the optimization model again.

4.7.7 Version Control

Given the complex nature of the program due to its various options and how it deals with time it was deemed necessary to include a version control feature. This feature tracks any change done to the AWS_d.p file and saves a copy of that file and the Calendar.xlsx file along with a log of the changes done. With this log it is possible to undo changes and bring the model back to a prior position in time if any mistakes were made. Also if the optimization returns an infeasible solution no schedule is printed out, instead a message that the model is infeasible is stated on the screen and the user is prompted to return the model to a previous position using the version control feature.

4.7.8 Visualization of Schedule

The ability to visualize the schedule is needed in order to be able to get a general sense of what the schedule looks like without needing to go into the details. Using the matplotlib module within Python it was possible to create a visualization of the schedule where every autoclave has a row and all cures within that autoclave are plotted in the row. An example of the visualization if shown in Figure 29.

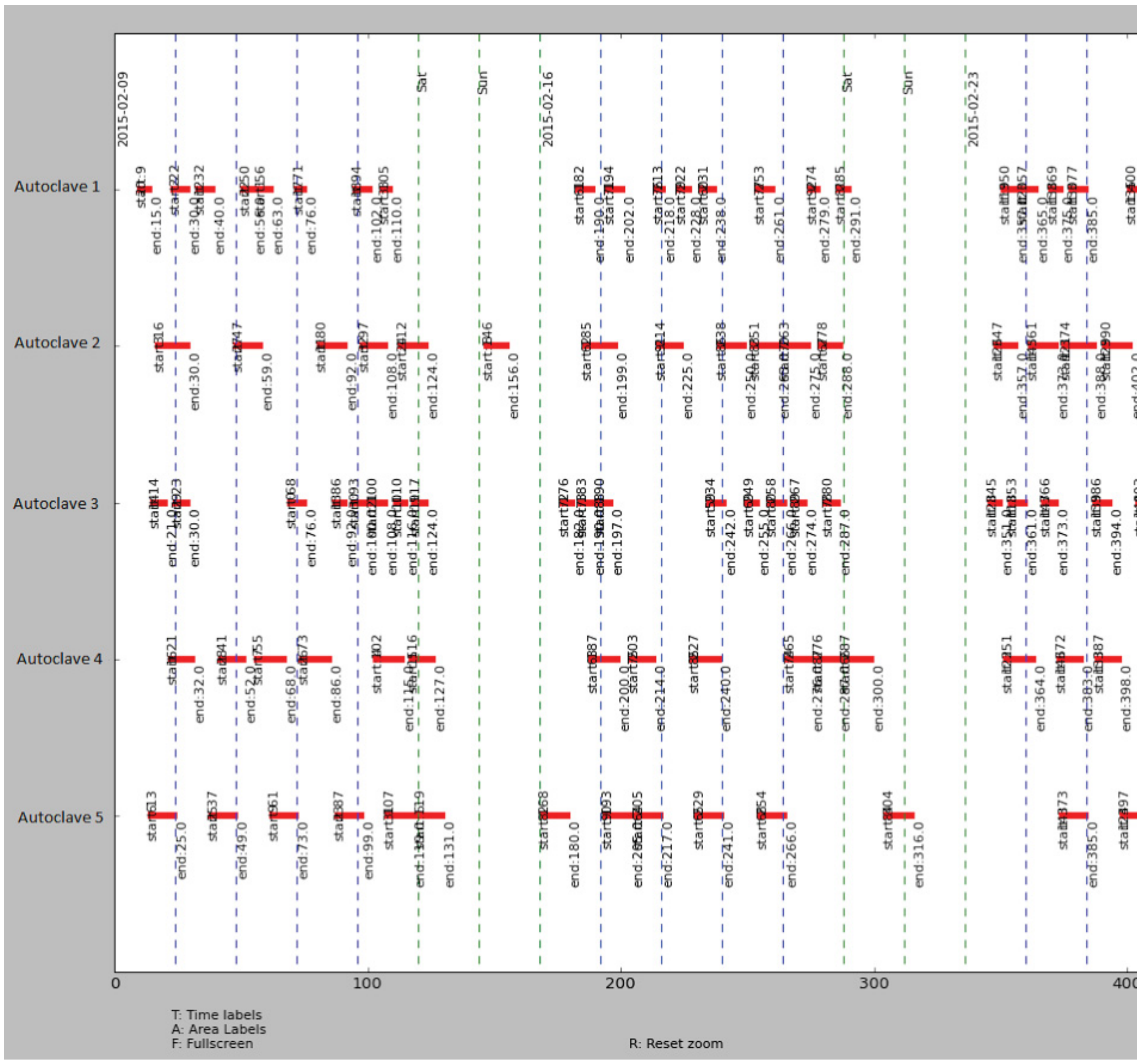
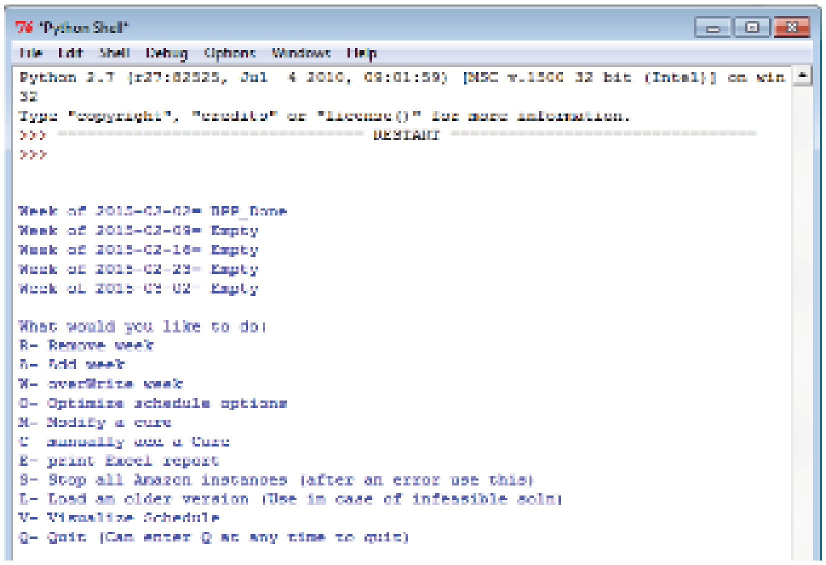


Figure 29: Schedule Visualization Example

4.7.9 User Interface

Every program must have a user interface which is easy to understand, robust, and provides easy access to the features which a user needs. The user interface for the program provided to TIP is very simple but fulfills all these requirements. The interface is a command-line interface, where options are presented to the user and they must choose one based on the letter-based menu. For example, in Figure 30 the top portion outlines the current status of the program, detailing which weeks' demand is loaded in, scheduled, or empty. Then the user is prompted to select one of the options to continue, for example the user may type "A" followed by the "Enter" key to add a week to the schedule. Each of these initial options available may lead the user to sub-menus or complete tasks immediately if no further instructions are required.



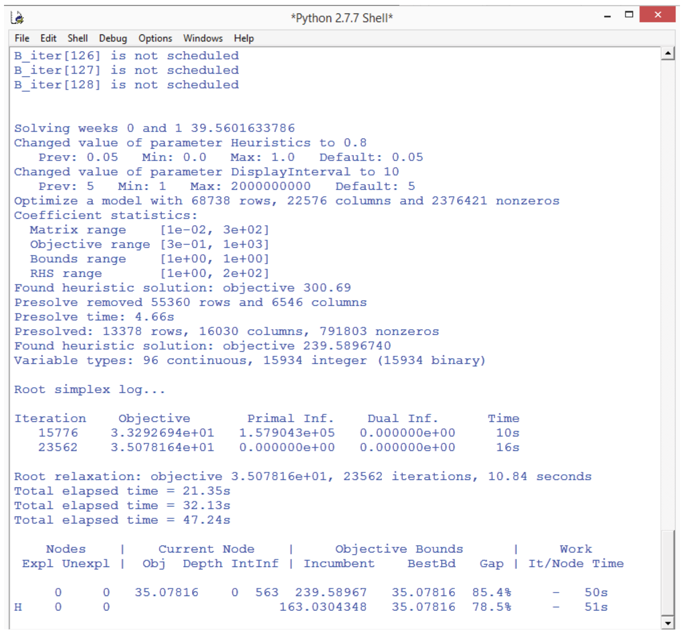
```
Python 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 


Week of 2015-02-02= DPP_Done
Week of 2015-02-09= Empty
Week of 2015-02-16= Empty
Week of 2015-02-23= Empty
Week of 2015-03-02= Empty

What would you like to do:
R- Remove week
A- Add week
W- overWrite week
O- Optimize schedule options
M- Modify a cure
C- manually add a Cure
E- print Excel report
S- Stop all Amazon instances (after an error use this)
L- Load an older version (Use in case of infeasible soln)
V- Visualize Schedule
Q- Quit (Can enter Q at any time to quit)
```

Figure 30: User Interface Main Menu

When the optimization is running the data output of the optimization status or what the program is doing is displayed to the user as can be seen in Figure 31. Here the user can see that weeks 0 and 1 are being optimized as well as the progress completed in that optimization thus far.



```
B_iter[126] is not scheduled
B_iter[127] is not scheduled
B_iter[128] is not scheduled


Solving weeks 0 and 1 39.5601633786
Changed value of parameter Heuristics to 0.8
   Prev: 0.05   Min: 0.0   Max: 1.0   Default: 0.05
Changed value of parameter DisplayInterval to 10
   Prev: 5   Min: 1   Max: 2000000000   Default: 5
Optimize a model with 68738 rows, 22576 columns and 2376421 nonzeros
Coefficient statistics:
  Matrix range    [1e-02, 3e+02]
  Objective range [3e-01, 1e+03]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 2e+02]
Found heuristic solution: objective 300.69
Presolve removed 55360 rows and 6546 columns
Presolve time: 4.66s
Presolved: 13378 rows, 16030 columns, 791803 nonzeros
Found heuristic solution: objective 239.5896740
Variable types: 96 continuous, 15934 integer (15934 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   15776    3.3292694e+01   1.579043e+05   0.000000e+00     10s
   23562    3.5078164e+01   0.000000e+00   0.000000e+00     16s

Root relaxation: objective 3.507816e+01, 23562 iterations, 10.84 seconds
Total elapsed time = 21.35s
Total elapsed time = 32.13s
Total elapsed time = 47.24s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   35.07816    0  563  239.58967   35.07816  85.4%     -   50s
H    0     0                      163.0304348   35.07816  78.5%     -   51s
```

Figure 31: User Interface Displaying Optimization Running

Although this user interface is not graphical and provides limited options it is easy to use, diagnose, and makes it easy to provide error logs. It was well adopted by the users at TIP and given its simple coding any future features of the program can be easily added to the user interface.

4.8 Implemented Models

The two problems, autoclave packing and scheduling, were modelled separately since both are computationally difficult problems to solve as discussed in the Proposed Model section. The APP was modelled using the hill-climbing method previously explained. The stopping condition for the hill-climbing method was determined to be 500 iterations. Less iterations could probably have been used but given the speed of the algorithm in completing these, the potential cost of having more autoclave packings than are needed, and the relative time this takes in comparison to the scheduling portion, a very conservative number of iterations was set. This is further explained in the Computational Results section. This metaheuristic was coded using Python and is solved without the need of a mathematical solver. The output from this is then used as one of the inputs to the scheduling problem, which was also modelled in Python for easy integration and used Pyomo for mathematical modelling. The scheduling problem uses Gurobi Cloud to compute a solution. Due to using Gurobi Cloud and paying by the hour, as well as the possibility of needing to run several optimization runs in a day, a self-imposed time limit on the optimization section of the program was set at 1 hour.

The autoclave packing for each week was completed individually within the program since demand from one week cannot be packed with demand from a different week. Thus this section of code is run for every week that is loaded into the program only once. The output of these autoclave packings serve as the input to the scheduling program.

The scheduling program uses the same code provided in the Proposed Model section, but the model was segmented into sections to improve computational performance. For a four week scheduling model a total of three runs are completed as shown in Table 3. First all of the parameters and constraints for the complete model are created, then most constraints for weeks 2 and 3 are disabled. The disabled constraints are con2, con22, con3, and constraints 8 through 13 for those weeks. They are disabled by calling the ".deactivate()" function on the constraint object. When re-activated the ".activate()" function is called on the constraint object. Once weeks 0 and 1 are scheduled, the schedule for week 0 is fixed leaving week 1 scheduled but unfixed. Then constraints for week 2 are enabled and the optimization is run again, this time having week 0 fixed, weeks 1 and 2 constrained, and week 3 mostly unconstrained. Then the same procedure is repeated, fixing week 1 and enabling the constraints for week 3. This leaves weeks 0 and 1 fixed, and weeks 2 and 3 constrained. This final run is the solution output to the user. The constraints that are disabled and enabled again during this procedure are constraints #2, 22, 3, 8,9,10,11,12 and 13. The rest of the constraints remain enabled throughout the procedure. When less than four weeks are provided this procedure is terminated earlier as required.

Table 3: Progression of Segmented Model Procedure

	Week 0	Week 1	Week 2	Week 3
Run 1	Constrained	Constrained	Unconstraine d	Unconstraine d
Run 2	Fixed	Constrained	Constrained	Unconstraine d
Run 3	Fixed	Fixed	Constrained	Constrained

Computationally the gains are due to limiting of the solution space the solver must consider. Although this procedure decreases the total computational time required to solve the problem it also creates a small potential loss of optimality since potential future options are ruled out as the procedure fixes weeks. The potential loss is small due since the weeks are almost independent given tool recycle times and the fact that scheduling loads on weekends is discouraged so they are further disconnected. Additionally, given the self-imposed time limit, this procedure provides better solutions than leaving the whole model run for the time limit.

Chapter 5 Results

The Proposed Model is scheduled to be implemented at TIP in the form shown in Implementation section. Some small changes were done to Proposed Model which can be seen in the Appendix A3, mainly the removal of infeasibility tracker variables for autoclave and tool infeasibility. These were done to improve computational performance as can be seen in the Computational Results section. Several meetings with the Program Production Leaders (PPLs), Planners, and management have taken place to facilitate the transition from the previous method of running the autoclave centre to the proposed one. At the time of this writing, test schedules have been created and distributed to PPLs for review. If the schedules look satisfactory then the program will be implemented approximately a month after. This month is needed to provide necessary time for kit cutting to provide the right kits for layup and for layup to ensure that the proper tools will be available to meet the oncoming schedule. Time is also needed here to update the parts list, tools list, and recipes list for parts that have not been previously encountered.

5.1 Computational Results

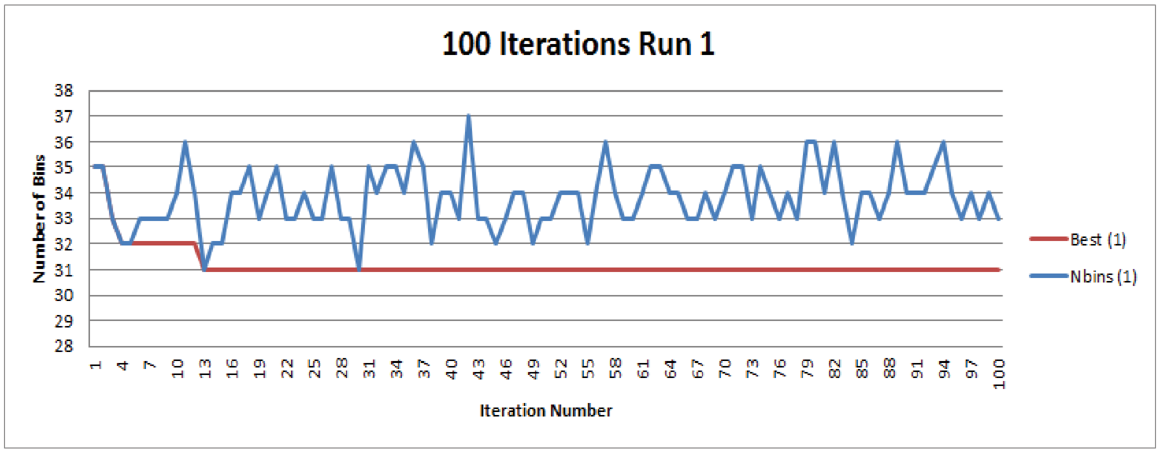
With both models built as discussed in the implementation section, computational experiments were carried out to assess the performance of different sections of the models as well as determine the best set of parameters. For autoclave packing, the number of iterations to be run had to be calculated in a better manner than simply setting it to an arbitrary number. For scheduling several constraints to improve performance by reducing symmetry were tested as well as some parameters for the optimization. The results of these tests are shown in the next two sections for autoclave packing and scheduling respectively.

All tests were run on a computer with an Intel Core i5-3550 CPU 3.7GHz with 8GB RAM running Windows 8.

5.1.1 Autoclave Packing Computational Results

To determine the number of iterations that should be run for the APP as the stopping criteria for the hill-climbing algorithm several runs were tried. The first two runs shown in Figure 32 with 100 iterations show the variability in number of iterations to reach the minimum. Given this a further experiment of 50 runs with 500 iterations each was completed. 40 of the 50 reached the best solution in less than 200 iterations, and only one did not reach the best solution in 500. Computing 500 iterations takes about 100 seconds given the demand mix in the experiment, which was representative of what will be run in practice. Due to the time it takes to run these iterations compared to the cost of obtaining a worse solution, the conservative stopping criteria of 500 iterations was selected.

As a demonstration, the autoclave packing algorithm was run with a sample of 135 unique part numbers, with an aggregated demand of 328 individual parts. This is not a representative sample but rather a small sample of parts used. It was not possible to obtain a representative sample for the autoclave packing due to confidentiality issues.



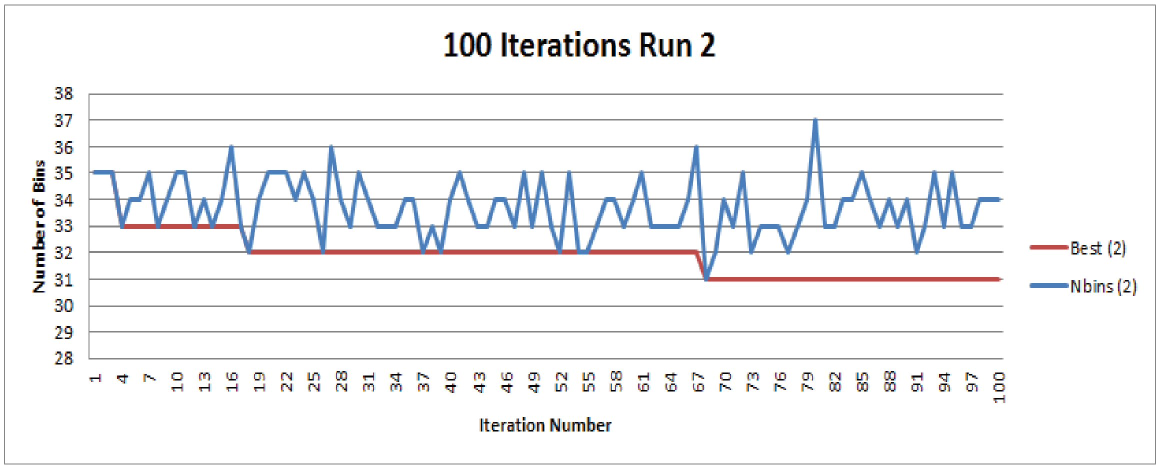


Figure 32: 100 Iteration Runs for APP

5.1.2 Scheduling Computational Results

The following results were obtained with the aforementioned computer and using Gurobi 6.0.4. Two tests were completed, both using a time limit of 3000 seconds and having the MIP Focus parameter set to 1. For the segmented model each of the three runs had a time limit of 1000 seconds. Both runs contain four weeks of demand, however the demand contained and thus the autoclave packings are different so direct comparison between runs is not possible. The purpose of the two runs is to compare how the different run parameters perform under a new schedule and a penalized schedule problem. The penalized schedule problem contains three previously scheduled weeks with penalties for rescheduling those as well as a fourth week that is unscheduled. The new schedule problem has four unscheduled weeks only.

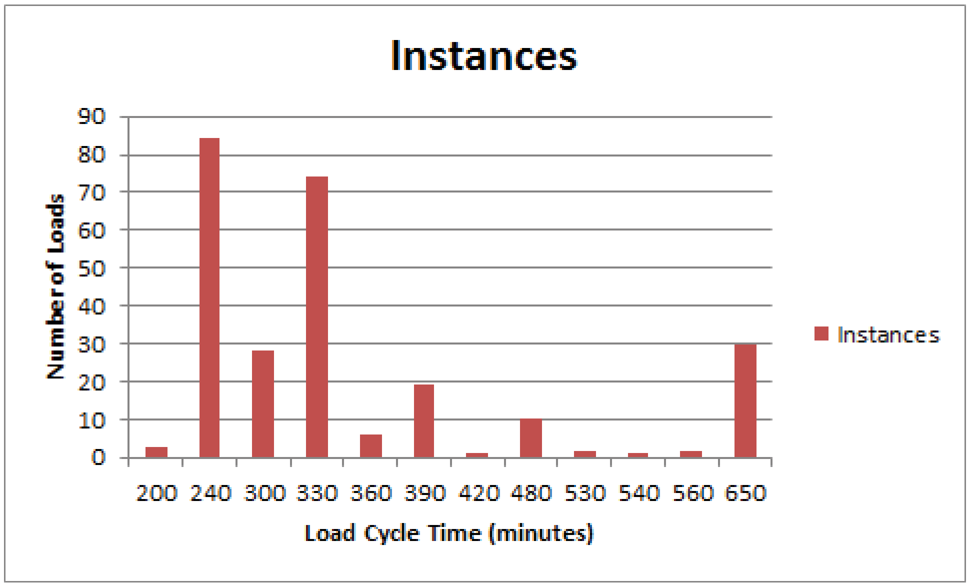


Figure 33: Distribution of Load Cycle Times for Four Week Run

The four week model has a total of 260 loads to be scheduled including dummy bins. The penalized model also has the same 260 loads including dummy bins, of these 198 are already scheduled and the non-dummy bins have penalties associated with rescheduling them. Figure 33 shows the distribution of load cycle times for both the four week run and the penalized run.

Table 4: Computational Results for Scheduling

#	Type	Warm start	Constraints	Inf Vars	Soln Penalized	Gap Penalized	Soln 4 wks	Gap 4wks
1	Segmented	No	FC-12,13,14	No	172.00	2.19%	166.28	1.60 %
2	Segmented	No	FC-12	No	171.86	1.68%	166.41	1.14 %
3	Segmented	No	FC-13,14	No	173.03	3.62%	166.06	1.71 %
4	Segmented	No	FC	No	172.48	2.72%	166.28	2.48 %
5	Segmented	No	FC	Yes	172.59	2.81%	166.43	0.93 %
6	Segmented	Yes	FC	No	172.48	2.72%	166.28	2.48 %
7	Segmented	Yes	FC	Yes	172.78	3.06%	167.78	1.35 %
8	Full	No	FC	No	170.77	11.8%	170.85	30.5 %
9	Full	No	FC	Yes	170.61	13.2%	172.59	30.0 %
10	Full	No	FC-13,14	No	170.78	11.8%	172.61	28.2 %
11	Full	No	FC-12,13,14	No	170.23	11.1%	167.76	26.1 %
12	Full	No	FC-12	No	170.23	10.9%	166.84	27.9 %

As can be seen from Table 4, there are significant benefits to using the segmented model when scheduling all new weeks. The average for the new solution run is 166.50 for the segmented model while 170.13 for the full model. In the penalized model we see the opposite, where the full model has an average of 170.52 while the segmented model's average is 172.46. Although it obtained worse results under the penalized model, those were only slightly higher, while in the new schedule model results were drastically better. The key point to compare across models is the actual solution value, not the gap. The gap should only be used to compare between models of the same type; this is due to the segmented model's fixing of the first two weeks when computing the last iteration and thus reducing the number of possible solutions.

Table 5 below shows the problem size for selected runs. By seeing the presolved instances of the segmented model and full model it is clear that the full model is much larger. This leads to the superior solving capability of the segmented model since Gurobi can focus the same computational capacity on a lesser number of variables and constraints. This is even more evident as the model progresses into the third segment, where the segmented model has nearly half as many variables as the full model. The Gurobi solution logs for the scheduling of the 4 new weeks problem for both the segmented (#2) and full model (#12) are available in Appendix B.

Table 5: Model Size for Segmented/Full Models for Selected Runs for 4 New Weeks

	Constraints	Variables
Segmented #2 before presolve	67,210/ 117,998/ 147,100	44,580/ 33,324 / 22,068
Segmented #2 presolved	16,293 / 14,477/ 12,463	28,740 / 22,275 / 15,668
Full #12 before presolve	194,018	44,580
Full #12 presolved	28,263	32,055

As suggested by these computational results, as well as other runs we have done, the segmented model with parameters in number 2 is the one that will be implemented at TIP. This model is not perfect nor is it guaranteed to perform optimally over all demand-mixes and it should continue to be revisited when major changes to the demand-mix happen.

However, all solutions provided in the tests were feasible solutions which could be implemented at the company. The degree preferences are adhered to, cost, and schedule flexibility are the ones that determine the objective function.

Ideally more examples ranging from small to large models would be presented in computational results, however given the limited amount of data available did not allow for this. Additionally since this project is being implemented at a real facility, it was most important to focus on a problem size that is representative of what TIP will be doing.

Chapter 6 Conclusion

Creating efficient autoclave packings and schedules for composite manufacturing is a difficult task. Many factors must be taken into account, from autoclave load balancing while creating autoclave packings to peak power while scheduling loads across multiple autoclaves. The software provided through this thesis work is proven to solve the packing and scheduling problems at TIP. The proposed model has several benefits over previous work in literature, previous practices at TIP, and provides an automated solution to two very complex problems. Several heuristics and meta-heuristics could have been implemented but none found could deal with all the intricacies of the system at TIP.

The proposed autoclave packing heuristic improves in several ways upon the methods examined in the literature review section. The methods examined in this section either used simple heuristics or used other methods which would have proven infeasible for the problem at TIP. A new method was developed, based on the hill-climbing method by Lewis (2009). The new method takes into account the multitude of constraints surrounding autoclave packing (volume, thermocouples, recipes, etc.), the autoclave priority for each recipe, and levels out the load between autoclaves while packing. Computational results shown in section 5.1.1 show that the method works quickly and effectively.

The proposed scheduling model provides several improvements upon the work completed before on autoclave scheduling. The only MIP model found which was designed for autoclave scheduling was the one built for ASTA by Panton and Beaumont in 1997. The proposed model developed in this research has several benefits over the ASTA model. First it uses discrete time intervals of hours instead of complete shifts, this leads to much better solutions as can be extrapolated from Table 1 in section 3.1.3. Secondly it uses the idea of desired tool recycle times to better discern between hard constraints and soft constraints. Thirdly it takes into account manpower levelling at the preceding operations. This helps accommodate them and reduce the risk of parts being late to the cure centre which could lead to rescheduling. Fourthly the proposed model takes into account peak power, a major cost for this energy intensive operation. Fifthly the proposed model penalizes scheduling of loads which would cause overtime, both at the cure centre and layup rooms with the use of the "OT" variable and the "Early" variable respectively. Finally the proposed model uses specially-developed constraints to reduce symmetry in the problem, an idea that could have been implemented to a much further benefit in the ASTA case given that autoclave loads in their case were selected from a pre-built set. However, given the early nature of the work at ASTA the computational capability would not have been available to solve a more complex model like the one proposed in this work.

The model provided is easily adaptable to different situations. In case changes happen to the system, these can be easily dealt with. Most changes like the addition of new areas are easily controlled by the user, however other changes like the addition of new parameters or the addition of an autoclave may require small changes in the code. Additionally, if any changes were to happen where new constraints are necessary in the APP these can be easily modified in the code as required. These changes leave the optimization structure and information workflows intact.

The implementation of the software provided to TIP through the completion of this thesis will continue to provide both operational as well as strategic planning improvements to the company in years to come. This would not have been possible without recent advances in cloud computing, flexible licensing options for optimization software, and the ease provided by the Python programming language in tying all these together. Using cloud computing provided significant benefits in implementation. The computational capacity was not present in-house at TIP. Due to this the purchase of an expensive computer would have been required as well as arrangements for proper maintenance. This is completely avoided by renting computational power on the cloud. Additionally, since the research was conducted by as a collaboration between an outside researcher and TIP, the ability to remotely run and update the program was important. A final important benefit was the ability to license Gurobi by the hour instead of needing to purchase a full license.

This thesis has designed and explored a method for packing and scheduling autoclaves on a scale previously unseen. This thesis has outlined the objectives and constraints regarding the packing of autoclaves and provided a metaheuristic that is capable of meeting the requirements of a real company. Previous works on autoclave scheduling have neglected important scheduling objectives of minimizing peak power and overtime, levelling of layup room manpower, and providing schedules which allow for flexibility and rescheduling. This thesis has explored these new aspects of autoclave packing and scheduling, provided a working model, and has laid the groundwork for implementation of this model in a manufacturing setting. There are limitations of the proposed model and these include include the inability to handle reentrant operations and using one-hour discrete time intervals. Reentrant operations are dealt with using blackout times, while the impact of using one-hour discrete time intervals is shown in section 3.1.3.

With some modifications, the solution approach and models presented in this thesis could be applied to other industries. These industries include semiconductor manufacturing, shoemaking, helicopter-part production, metal-working necessitating the use of furnaces, and ceramic manufacturing. The applications found through the literature review in these areas often had less constraints on the problem than was needed by the autoclave packing and scheduling problem. Therefore, by eliminating some constraints it is possible to modify the models to apply to these other industries.

6.1 Future Work

Several expansions can be done onto the work presented here that were not carried out due to time, data availability, and the interest in first implementing a basic model that was not overly complicated. These expansions may yield great results in terms of financial rewards both through computational improvements, new features, as well as improved visibility throughout the system.

1. Incorporate energy profiles for autoclave-recipe sets into energy calculation

This would improve upon the current constraint #4 that attempts to minimize the number of simultaneous autoclave starts within the same hour. The current constraint fails to account for autoclave power draw as well as the time over which the autoclave draws different amounts of power. Currently for example starting the largest autoclave and the smallest autoclave at the same time is penalized at the same amount that starting the two smallest autoclaves is. Having energy profiles for autoclave-recipe sets would allow the minimization of maximum power draw during a specific period. Additionally another area where this could be used would be to better inform the "Autoclave Priority" parameter while solving the APP to minimize cost.

2. Incorporate capability to deal with re-entrant operations

Some parts must be cured several times with layup steps in between. These parts are currently dealt with using blackout periods which create loads that are not taken into account in the manpower levelling and energy utilization calculations. Additionally, having the ability to move these loads around within the program would allow for a better solution to be provided since there would be less constraints on the optimization.

3. Expand model to include ovens

Ovens were not mentioned during this thesis work since they fall clearly outside the scope of this project. However, their operation is very similar to that of the autoclaves and bringing them into the program would have several benefits. Firstly, loads that are cured in ovens also play a role in manpower levelling and energy utilization, but are not considered at the moment. Secondly, the efficiency of the oven operation could be improved in the same way the autoclaves were improved by this program.

4. Create a complementary model to schedule layup operations

A complementary model that schedules layup operations would work in conjunction with the proposed model to arrive at a schedule this is overall best for the plant. This would reduce friction between the autoclave centre and layup operations since everyone's priorities would be taken into account. However, the autoclave centre's priorities would need to outweigh those of the layup room's given that it is currently the bottleneck.

5. Reformulate objective function in terms of cost

The current objective function weights are based on relative importance of different schedule aspects and have been tweaked into their current form through trial and error. However, a company's goal is to create a profit, not a good schedule. Therefore, given the company's goal and the fact that the autoclave centre does not directly generate revenue, an objective function in terms of minimizing cost is more appropriate. This was not completed during this project due to lack of information.

6. Create capacity analysis tool to estimate cost of taking on new demand

Although more demand is guaranteed to increase the revenues of TIP, it may prove unprofitable if it pushes the production system past a certain point. New demand may require the system to use high levels of peak energy, overtime, and create a much tighter schedule that is not able to respond appropriately to disruptions. A capacity analysis tool that is able to calculate this would prove to be very useful for TIP both in terms of long-term capacity analysis but also in the bidding process for new work. This tool could also be used to inform the specifications needed for a new autoclave given what will work best with current and future demand mixes. This work would need to take place after reformulating the objective function in terms of cost.

7. Keep a history log of both schedules and autoclave packings

Keeping history logs is essential to several aspects of a company's operation so the past can be analyzed to forecast the future, identify mistakes and root causes, as well as better track performance. Modifying the program to save a history log of these would require establishing a database and identifying with the team what is and what is not relevant information that should be saved. Identifying the proper format to best save it in to facilitate future access is also important.

8. Create a feedback loop between the scheduling model and the autoclave packing model

Currently the autoclave packing model runs without regard to how the bins will be scheduled. If there was a feedback loop from the scheduling model to the autoclave packing model then autoclave packing decisions could be made while being informed by the consequences they have on the scheduling model. This could lead to much better overall system performance.

References

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., . . . Stoica, I. (2010). A view of cloud computing. *Communications of the ACM, 53*(4), 50-58.

Balas, E. (1971). Intersection cuts—a new type of cutting planes for integer programming. *Operations Research, 19*(1), 19-39.

Belov, G., & Scheithauer, G. (2006). A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research, 171*(1), 85-106.

Bertsimas, D., & Tsitsiklis, J. (1993). Simulated annealing. *Statistical Science, 8*(1), 10-15.

Bezanson, J., Karpinski, S., Shah, V. & Edelman, A.Julia. Retrieved from http://julialang.org/

Blum, A. L., & Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks,* 5(1), 117-127.

Brooke, A., Kendrick, D., Meeraus, A., Raman, R., & America, U. (1998). The general algebraic modeling system. *GAMS Development Corporation,*

Castro, P. M., & Grossmann, I. E. (2005). New continuous-time MILP model for the short-term scheduling of multistage batch plants. *Industrial & Engineering Chemistry Research, 4*4(24), 9175-9190.

Castro, P. M., & Grossmann, I. E. (2006). An efficient MILP model for the short-term scheduling of single stage batch plants. *Computers \& Chemical Engineering, 3*0(6), 1003-1018.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications, 4*5(1), 41-51.

Coffman Jr, E. G., Csirik, J., Galambos, G., Martello, S., & Vigo, D. (2013). Bin packing approximation algorithms: Survey and classification. *Handbook of combinatorial optimizatio*n (pp. 455-531) Springer.

Cook, L. K., Hinkle, D. A., & Bickmore, T. W. (1990, June). Planning for the manufacturing domain: long-term and reactive scheduling. In *Expert Planning Systems, 1991., First International Conference on* (pp. 6-10). IET.

de Carvalho, J. V. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research, 14*1(2), 253-273.

de Carvalho, J. V. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research, 8*6, 629-659.

Devine, K., & Milne, R. (2014). *Project in industry report: Scheduling of the curing workcentre.*

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2*6(1), 29-41.

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research, 4*4(2), 145-159.

Eiselt, H. A., & Sandblom, C. (2000). *Integer programming and network model*s Springer.

Fanti, M. P., Maione, B., Piscitelli, G., & Turchiano, B. (1996). Heuristic scheduling of jobs on a multi-product batch processing machine. *International Journal of Production Research,* 34(8), 2163-2186.

Flavell, R. B. (1976). A new goal programming formulation. *Omega*, 4(6), 731-732.

Floudas, C. A., & Lin, X. (2004). Continuous-time versus discrete-time approaches for scheduling of chemical processes: A review. *Computers & Chemical Engineering, 2*8(11), 2109-2129.

Fourer, R., Gay, D. M., & Kernighan, B. W. (1990). A modeling language for mathematical programming. *Management Science, 36*(5), 519-554.

Friesen, D. K., & Langston, M. A. (1986). Variable sized bin packing. *SIAM Journal on Computing, 1*5(1), 222-230.

GAMS Development Corporation. (2015). Gams. Retrieved from http://www.gams.com

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. *WH Freeman & Co., San Francisco,*

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research, 9*(6), 849-859.

Glover, F. (1989). Tabu search-part I. *ORSA Journal on Computing,* 1(3), 190-206.

Glover, F. (1990). Tabu search—part II. *ORSA Journal on Computing,* 2(1), 4-32.

Goldratt, E. M., Cox, J., & Whitford, D. (2004). *The goal: A process of ongoing improvemen*t North River Press Great Barrington^ eMA MA.

Gupta, S., & Karimi, I. (2003). An improved MILP formulation for scheduling multiproduct, multistage batch plants. *Industrial \& Engineering Chemistry Research, 4*2(11), 2365-2380.

Gurobi Optimization. Retrieved from http://www.gurobi.com/products/licensing-pricing/commercial-pricing

Gutowski, T. (1997). A brief introduction to composite materials and manufacturing processes. *Advanced Composite Manufacturing,* , 5-41.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research, 13*(2), 311-329.

Hansen, P., & Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing, 44*(4), 279-303.

Haouari, M., & Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers & Operations Research, 36*(10), 2877-2884.

Harper, C. A. Handbook of plastics, elastomers, and composites, 1996. *McGraw-Hill Inc,* 2, 2.41.

Hart, W. E., Laird, C., Watson, J., & Woodruff, D. L. (2012). *Pyomo–optimization modeling in pytho*n Springer Science & Business Media.

Hart, W. E., Watson, J., & Woodruff, D. L. (2011). Pyomo: Modeling and solving mathematical programs in python. *Mathematical Programming Computation,* 3(3), 219-260.

Hennessy, D., & Hinkle, D. (1992). Applying case-based reasoning to autoclave loading. *IEEE Expert*, 7(5), 21-26.

Hindle, K., & Duffin, M. (2006, December). SIMUL8-planner for composites manufacturing. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter* (pp. 1779-1784). IEEE.

Hopp, W. J., & Spearman, M. L. (2011). *Factory physic*s Waveland Press.

IBM.IBM ILOG CPLEX optimization studio. Retrieved from http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud

IBM. (2015). Decision optimization on cloud. Retrieved from http://onboarding-oaas.docloud.ibmcloud.com/software/analytics/docloud/

Ignizio, J. P. (2004). Optimal maintenance headcount allocation: an application of Chebyshev Goal Programming. *International journal of production research*,42(1), 201-210.

Ignizio, J. P., & Cavalier, T. M. (1994). *Linear programming.* Prentice-Hall, Inc.

Johnson, D. S. (1974). Fast algorithms for bin packing. *Journal of Computer and System Sciences,* 8(3), 272-314.

Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., ... &

Wolsey, L. A. (Eds.). (2009). 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-art. Springer Science & Business Media.

Kang, J., & Park, S. (2003). Algorithms for the variable sized Bin Packing Problem. *European Journal of Operational Research, 147*(2), 365-372.

Kirkpatrick, Scott and Gelatt, C Daniel and Vecchi, Mario P and others. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671-680.

Koh, S. G., Koo, P. H., Ha, J. W., & Lee, W. S. (2004). Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. *International Journal of Production Research*, 42(19), 4091-4107.

Lee, I. (1991). A worst-case performance of the shortest-processing-time heuristic for single machine scheduling. *Journal of the Operational Research Society,* , 895-901.

Levine, J., & Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society, 55*(7), 705-716.

Lewis, R. (2009). A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & Operations Research, 36*(7), 2295-2310. doi:http://dx.doi.org/10.1016/j.cor.2008.09.004

Lubin, M., Dunning, I. & Huchette, J. (2014). JuMP — julia for mathematical programming. Retrieved from https://jump.readthedocs.org/en/latest/

Luh, P. B., Wang, J. H., Wang, J. L., Tomastik, R. N., & Howes, T. D. (1997). Near-optimal scheduling of manufacturing systems with presence of batch machines and setup requirements. *CIRP Annals-Manufacturing Technology*,46(1), 397-402.

Maravelias, C. T., & Grossmann, I. E. (2003). Minimization of the makespan with a discrete-time state-task network formulation. *Industrial & Engineering Chemistry Research, 42*(24), 6252-6257.

Margot, F. (2010). Symmetry in integer linear programming. In *50 Years of Integer Programming* 1958-2008 (pp. 647-686). Springer Berlin Heidelberg.

Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional Bin Packing Problem. *Operations Research, 4*8(2), 256-267.

Mathirajan, M., & Sivakumar, A. I. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10), 990-1001.

Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the Bin Packing Problem. *Discrete Applied Mathematics, 2*8(1), 59-70.

Mazumdar, S. (2001). *Composites manufacturing: materials, product, and process engineering.* CrC press.

Meindl, B., & Templ, M. (2012). Analysis of commercial and free and open source solvers for linear optimization problems. *Analysis,*

Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering, 3*0(6), 913-946.

Mitchell, S., O'Sullivan, M., & Dunning, I. (2011). PuLP: A linear programming toolkit for python.

Murgolo, F. D. (1987). An efficient approximation scheme for variable-sized bin packing. *SIAM Journal on Computing, 1*6(1), 149-161.

Nahmias, S., & Olsen, T. L. (2015). *Production and operations analysi*s Waveland Press.

Osman, I. H., & Potts, C. N. (1989). *Simulated annealing for permutation flow-shop scheduling.* Omega, 17(6), 551-557.

Panton, D., & Beaumont, N. (1997). *Optimisation of Work Flow.* Unpublished manuscript.

Pochet, Y., & Wolsey, L. A. (2006). *Production planning by mixed integer programming.* Springer Science & Business Media.

Pokutta, S. (2010). The GNU linear programming kit (GLPK): Resources, tutorials etc. Retrieved from https://spokutta.wordpress.com/the-gnu-linear-programming-kit-glpk/

Ram, B., & Patel, G. (1998, December). Modeling furnace operations using simulation and heuristics. In *Proceedings of the 30th conference on Winter simulation* (pp. 957-964). IEEE Computer Society Press.

Salkin, H. M., & Mathur, K. (1989). *Foundations of integer programmin*g North Holland.

Silver, E., Pyke, D. F., & Peterson, R. (1998). *Inventory management and production planning and* *scheduling.*

Simpson, R., & Abakarov, A. (2009). Optimal scheduling of canned food plants including simultaneous sterilization. *Journal of Food Engineering, 9*0(1), 53-59.

Spearman, M. L., Woodruff, D. L., & Hopp, W. J. (1990). CONWIP: A pull alternative to kanban. *The International Journal of Production Research, 28*(5), 879-894.

Tamiz, M., Jones, D., & Romero, C. (1998). Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of operational research*, 111(3), 569-581.

Van Der Zee, D. J., Van Harten, A., & Schuur, P. C. (1997). Dynamic job assignment heuristics for multi-server batch operations-a cost based approach. *International Journal of Production Research*, 35(11), 3063-3094.

Van Der Zee, D. J., Harten, A. V., & Schuur, P. (2001). On-line scheduling of multi-server batch operations. *IIE Transactions*, 33(7), 569-586.

Van Laarhoven, P. J., Aarts, E. H., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research, 4*0(1), 113-125.

Williams, H. P. (1999). *Model building in mathematical programming.*

Wolsey, L. A., & Nemhauser, G. L. (2014). *Integer and combinatorial optimization*. John Wiley & Sons.

Ye, W. H., Li, J., Chen, W. F., Ma, W. T., & Leng, S. (2014). Study on scheduling method for reentrant autoclave moulding operation of composite materials. Paper presented at the *2013 2nd International Conference on Mechanical Design and Power Engineering, ICMDPE 2013,* *November 29, 2013 - November 30, , 490-49*1 14-18.

doi:10.4028/www.scientific.net/AMM.490-491.14

Appendix A

A1- BPP3.py

The metaheuristic implemented for the autoclave packing problem as programmed in Python.

A2- BinClass.py

The Bin class object within Python is used as a platform for metaheuristic described in A1. This object is also called in the building of the scheduling model to extract data about autoclave loads.

A3-Proposed Scheduling Model as implemented

The proposed MILP scheduling model as programmed in Python using Pyomo, including the segmented approach procedure. Does not include parameters initialization section due to extensive length.

A1- BPP3.py

```python
def Create_BPP(jj,w,bnum,parts,dummy,tools,blkt):
    from random import randint, sample, uniform, seed
    import time
    import numpy as np
    import operator
    import openpyxl
    import string
    import gc
    import pickle
    from Bin_Class import New_Bin
    #This last one imports the Bin_Class: New_Bin


    #seed(50) #Set random number seed

    def print_nice(a): #prints dictionary/list nicely
        if type(a)==dict:
            for item in a:
                print item, a[item] ,"\n"
        else:
            for item in a:
                print item, "\n"

    def get_autoclaves(): #Imports autoclaves from .xlsx file
        wb=openpyxl.load_workbook('Autoclaves.xlsx',data_only=True)
        ws=wb.active

        Columns=[]
        j=0
        for item in list(string.ascii_uppercase)[1:6]: #Creates dict: Columns[0]='A'
            Columns.append(item)
            j+=1
        i=2
        while(1):
            if ws['A%s'%i].value is None:
                break
            au[str(ws['A%s'%i].value)]={}
            for j in Columns:
               au[str(ws['A%s'%i].value)][str(ws['%s1'%j].value)]=\
                      ws['%s%s'%(j,i)].value
            i+=1


    def sort_parts(): #Sorts by Recipe number
        global sorted_parts
        #print parts
        sorted_parts=sorted(parts.iteritems(), key=lambda (x,y):\
                y['Recipe Number'], reverse=False)

    def au_time(autoclave,B_ins): #Returns length of time autoclave is loaded
        t=b_au[autoclave] #blackout time initialized
        #print "Au Time started:"
        for item in B_ins:
            if item.a==autoclave:
                t+=item.time/60
        return t

    #Returns autoclave with highest priority for part
    def au_priority(p,B_ins,nf_list):
        #Load balancing:
        l=[]
        for item in au:
            if item not in nf_list:
                t=item+' Priority'
                if p[1][t]<>0 and au[item]['Volume']>=\
                      tools[p[1]['Tool Number']]['Volume']:
                    l.append(item)#Autoclave number
        if len(l)==1: #If it only fits in one...create that one
            return l[0]
        if len(l)==0: #Part does not fit in any autoclave
            #Print 20 times to get around buffering of data being returned.
            #Makes sure it is displayed
            for x in xrange(0,20):
                print "Part does not fit into any autoclave for which recipe is
allowed"
                print "Check the database for part, tool length, and recipe priorities"
                print p
                print "Did not fit in: ", nf_list
            raise

        l_weight=[]
        for item in l:
                t=item+' Priority'
                #Appends weighted time
                #Weighted time=(1+0.1*priority#)*time
                #Example: Priority 1 time= 1.1*time
                l_weight.append((1+int(p[1][t])*0.1)*au_time(item,B_ins))
                #print p[1][t]
                #print item, " time:", au_time(item,B_ins)
        return l[l_weight.index(min(l_weight))]


    def put_item_in_bin(item, B_ins):
        if len(B_ins)==0:#"0 Bins available"
            return "NoFit"
        for B in B_ins:
            if B.query_fit(parts,tools,item[0]):
```

```
            B.add_part(parts,tools,item[0])
            return "Fit"
    return "NoFit"


def place_in_bins(P_ins,B_ins,instance):
    n=0
    for item in P_ins: #starts from big to small
        if instance==0: #If first instance make duplicates to account for demand
            r=int(item[1]["Demand"])
        else:
            r=1
        for d in xrange(0,r):
            #print item
            #print put_item_in_bin(item)

            if (put_item_in_bin(item,B_ins)=="NoFit"):
                 #Didnt fit in any, create new bin
                nf_list=[]
                a_ins=au_priority(item,B_ins,nf_list)
                vol=au[a_ins]["Volume"]
                ther=au[a_ins]["Thermos"]
                au_len=au[a_ins]["Length Envolope (inches)"]

                B_ins.append(New_Bin(tools,a_ins,vol,au_len,ther,item[1]\
                         ["Recipe Number"],n,w,item[1]['Area']))
                n+=1

                #If it doesn't fit in the bin created...
                while (put_item_in_bin(item,B_ins)=="NoFit"):
                    #Probably doesnt fit due to tool length<
                         #autoclave length constraint.
                    B_ins.pop() #Remove that bin
                    n-=1
                    #Add autoclave that wasnt feasible to list.
                    nf_list.append(a_ins)


                    #Try next autoclave in priority list
                    a_ins=au_priority(item,B_ins,nf_list)
                    vol=au[a_ins]["Volume"]
                    ther=au[a_ins]["Thermos"]
                    au_len=au[a_ins]["Length Envolope (inches)"]
   B_ins.append(New_Bin(tools,a_ins,vol,au_len,\
   ther,item[1]["Recipe Number"],n,w,item[1]['Area']))
   n+=1


def get_order(B_ins):
        #Gets parts for every bin into new_order sequentially, the bins
```

```
            #are in random order though.
            n_order=[]
            for p in sample(range(0,len(B_ins)),len(B_ins)): #For every bin packed
                    for j in B_ins[p].parts: #For every part in bin
                            n_order.append((j,parts[j]))
            return n_order

    def local_improvement(B_ins):
            #print "\nInitial length:",len(B_ins)
            #B_ins has the big chunk of groups
            #B_improv has the small groups taken out
            B_improv=[]
            cutoff=uniform(0.3,0.5)
            for b in B_ins: #Put some items in B_improv
                    if b.vleft > cutoff*b.volume and\
                    b.tleft > cutoff*b.thermos:
                            #If less than 30% full
                            B_ins.remove(b)
                            B_improv.append(b)
            for b in B_improv: #For each bin in improv list
                    while(True): #continues until broken
                            #print "started again"
                            e=0
                            for item in b.parts: #For each part in bins in improv list
                                    if e==1: #Breaks out of for loop
                                            #print "broke out"
                                            break
                                    for B in B_ins:#B in B_ins:
                                             #For each bin in original
                                            try:
                                                  if B.query_fit(parts,tools,item):

B.add_part(parts,tools,item) #Send part to new bin

b.remove_part(parts,tools,item )#Remove part from old bin
                                                            print "Improved locally"
                                                            e=1 #breaks out of next for
loop

                                                            break
                                            except:
                                                    print "out of range...fix sometime,
doesn't REALLY impact performance"
                            if e==0:
                                    break #Breaks the while loop
            B_ins.extend(B_improv)
            #print len(B_ins)
            return B_ins #Returns list with local improvement done


    def print_stats(m,p_binary):
            #m=B_iter[m] to get stats for
            #p_binary= to print or not to print
            v=0 #Total volume
            t=0 #Total time
            i=0 #number >30% empty
            autoclaves_used=[]
            for item in B_iter[m]:
                    v+=item.volume
                    t+=(item.time/60)
                    autoclaves_used.append(item.a) #Gets list of all autoclaves
                    if item.vleft > 0.3*item.volume and item.tleft>0.3*item.thermos:
                            i+=1
            #print "v:",v, "t:",t,"\n"
            uniq=list(set(autoclaves_used)) #Gets list of autoclaves
            autoclave_counter=[]
            autoclave_time=[]
            autoclave_stats=[]
            volume_constrained=0 #Counters for constraints:
            thermo_constrained=0
            for i in xrange(0,len(uniq)):
                    autoclave_counter.append(i)
                    autoclave_time.append(0)
                    for item in B_iter[m]:# in autoclaves_used:
                            if uniq[i]==item.a:
                                    autoclave_counter[i]+=1
                                    autoclave_time[i]+=item.time
                                    volume_constrained+=item.vol_const
                                    thermo_constrained+=item.thermo_const
                    autoclave_stats.append((uniq[i],\
                             autoclave_counter[i], autoclave_time[i]))
            makespan=max(autoclave_time)
            if p_binary==1:
                    print "Volume: %s " %"{:,}".format(v)
                    print "Time: %s" %t
                    print "Number of bins >=30%% empty (Vol and Thermos) : %s" %i
                    print "Vol Const: %s \nThermo Const: %s" %\
                             (volume_constrained, thermo_constrained)
                    for item in autoclave_stats:
                            print "Autoclave: %s, Loads: %s, Time: %s"%\
                             (item[0],item[1],item[2])
                    print "Makespan: ", makespan
                    print "Total # of bins: %s" %len(B_iter[m])
            return v, t, i ,autoclave_stats,makespan, volume_constrained,
thermo_constrained
            #Returns volume, time, # >30% empty, # of part placements restricted by vol
and thermos.


    #Parameters and initialise
    B_iter=[]
```

```
au={} #Autoclaves
LS=[]
l=[]
Volumes=[]
Times=[]
Makespans=[]
NBins=[]
#NBins_Time=[]

#Get data
sort_parts()
new_order=sorted_parts
get_autoclaves()

#Determines number of blackout times in this week for each autoclave
#Used in au_priority function
wk_hrs=xrange(w*168,(w+1)*168)
b_au={}
for item in au:
    b_au[item]=len([k for k in set(blkt[item]) if k in wk_hrs])
print 'B_AU: ', b_au


#Start algorithm=greedy with local improvement
for j in xrange(0,jj):
    print j, time.clock()
    B_iter.append([]) #Create a new iteration in list
    place_in_bins(new_order,B_iter[j],j) #Initial sort
    B_iter[j]=local_improvement(B_iter[j]) #Local improvement, overwrites bins
    new_order=get_order(B_iter[j]) #Randomise order

    stat=print_stats(j,1)
    Volumes.append(stat[0]) #Store volume in list
    Times.append(stat[1]) #Store time in list
    Makespans.append(stat[4]) #Store makespan in list
    NBins.append(len(B_iter[j])) #Store number of bins in list
    #NBins_Time.append(time.clock()) #Get time bin was computed at

    if j==0:
        print "Starting: ", len(B_iter[0])
    if j>0 and len(B_iter[j])>len(B_iter[j-1]):
        print "Got worse somehow!", len(B_iter[j]), "t= %s"%time.clock()
    if j>0 and len(B_iter[j])<len(B_iter[j-1]):
        print "Got better!", len(B_iter[j]), "t= %s"%time.clock()


#RESULTS
print "\n\n\n\n############RESULTS############"
#Print stats for lowest volume
print "\nLowest Volume: ", min(Volumes)
```

```
j=min(xrange(len(Volumes)), key=Volumes.__getitem__)
print "Iter: %s"%j
print_stats(j,1)

#Print stats for lowest time
print "\nLowest Time: ", min(Times)
j=min(xrange(len(Times)), key=Times.__getitem__)
print "Iter: %s"%j
print_stats(j,1)

#Print stats for lowest makespan
print "\nLowest Makespan: ", min(Times)
j=min(xrange(len(Makespans)), key=Makespans.__getitem__)
print "Iter: %s"%j
print_stats(j,1)
iter_to_return=j

#Print stats for lowest number of bins
print "\nLowest number of bins: ", min(NBins)
j=min(xrange(len(NBins)), key=NBins.__getitem__)
print "Iter: %s"%j
print_stats(j,1)
iter_to_return=j

f=open('BPPoutput.txt','a')
f.write("\nj= %s bins=%s"%(j,len(B_iter[j])))

g=open('BPPoutput2.txt','a')
g.write('\nj=%s , %s'%(j,NBins))

#Get list of autoclaves used
au_used=[]
for item in B_iter[j]:
    print "Number: ",item.num
    if item.a not in au_used:
        au_used.append(item.a)
print au_used


#Append 5 dummy bins per autoclave, for "usable gaps"
n=len(B_iter[j]) #reinitialize n to number of bins in run.
dummies=0
for item in au_used:
    for i in xrange(0,3):
        #'dummy' is the rangeset number for the dummy area.
        #(parts_list,tools_list,autoclave,volume,thermos,rec,n,wk,area)
        B_iter[j].append(New_Bin(tools,item,0,0,0,"0",n,w,dummy))
        B_iter[j][n].time=240
        n+=1
        dummies+=1 #number of dummies appended
```

```
#Create a tools list to pass
#Update Bin .num to match bnum to avoid repeats
tlist=[]
for i in B_iter[j]:
    i.num=i.num+bnum
    for t in i.tools:
        tlist.append(t)
tlist=list(set(tlist))
print tlist

bnum+=len(B_iter[j])

gc.collect() #Garbage collector

#Order bin numbers
B_iter[j].sort(key=lambda x: x.num, reverse=False)
#print "j=",j
#print "len:", len(B_iter[j])


return B_iter[j], au_used, bnum
```

A2- BinClass.py

```
class New_Bin():
    def __init__(self,tools_list,autoclave,volume,au_len,thermos,rec,n,wk,area):
        self.num=n
        self.a=autoclave
        self.volume=volume#au[autoclave]["Volume"]
        self.au_len=au_len #au[autoclave]["Length Envolope (inches)"]
        #print "Autoclave: %s, Length: %s"%(self.a,self.au_len)
        self.vleft=self.volume
        self.thermos=thermos#au[autoclave]["Thermos"]
        self.tleft=self.thermos
        self.recipes=[]
        self.recipes.append(rec)
        self.parts=[]
        self.tools=[]
        self.tools_used={}
        self.tools_cap_left={}
        for t in tools_list:
            self.tools_used[t]=0
            self.tools_cap_left[t]=0
        self.manpower=0
        self.area=area
        self.time=0
        self.longest_part=0 #Shift-hrs
        #self.Ashift=1 #Default is one shift
        self.week=wk #No scheduled week
        self.scheduled=None #No scheduled time
        self.fixed=0 #Bin,Def=0, if 1 then scheduled time/week is fixed.
        self.Rpenalize=False
        self.thermo_const=0
        self.vol_const=0
    def Bprint(self):
        sp=''
        uniqp=set(self.parts)
        for item in uniqp:
            sp+="\n     %s :%s"%(item,self.parts.count(item))
        st=''
        uniqt=set(self.tools)
        for item in uniqt:
            st+="\n     %s :%s"%(item,self.tools.count(item))

        print "Bin Number: ", self.num
        print "Autoclave: ", self.a
        print "Volume: ", self.volume
        print "Vleft: ", self.vleft
        print "Thermos: ", self.thermos
        print "Tleft: ", self.tleft
        print "Parts: ", sp
        print "Tools: ", st
        print "Area: ", self.area
        print "Manpower: ", self.manpower
        print "Longest Part MP: ", self.longest_part
        print "Week: ", self.week
        print "Scheduled: ", self.scheduled
        print "Fixed in Opt (1=yes): ", self.fixed
        print "Reschedule penalized: ", self.Rpenalize
        print "Recipes: ", self.recipes
        print "Time: ", self.time
    def Bwrite_parts(self):
        #Returns a list of all parts, along with count
        sp=[]
        uniqp=set(self.parts)
        for item in uniqp:
            sp.append("     %s :%s"%(item,self.parts.count(item)))
        return sp

    def Bwrite(self):
        sp=''
        uniqp=set(self.parts)
        for item in uniqp:
                sp+="\n     %s :%s"%(item,self.parts.count(item))
        st=''
        uniqt=set(self.tools)
        for item in uniqt:
                st+="\n     %s :%s"%(item,self.tools.count(item))
        return "\
    \nBin #: %s\
    \nAutoclave: %s\
    \nParts: %s\
    \nTools: %s\
    \nArea: %s\
    \nManpower: %s hrs\
    \nLongest_Part_MP: %s hrs\
    \nRecipe: %s\
    \nTime: %s minutes\
    \nFixed Time (1=Yes): %s \
    \nReschedule penalized: %s \
    \nWeek: %s\
    \nScheduled: %s \
    "%(self.num, self.a, sp, st,self.area, self.manpower, self.longest_part,\
       self.recipes[0], self.time,self.fixed, self.Rpenalize, self.week,
self.scheduled)
    def query_fit(self,parts_list,tools_list,p):
        #Keep track of tools used in bin
        for t in tools_list:
            if t not in self.tools_used:
                print "added t= %s tool!!!"%t
                self.tools_used[t]=0
                self.tools_cap_left[t]=0
        #Verify constraints
        if self.tleft>=parts_list[p]["Thermos"] and \
           self.au_len>=tools_list[parts_list[p]['Tool Number']]['Length'] and \
            parts_list[p][str(self.a+' Priority')]<>0 and \
            (parts_list[p]["Recipe Number"] in self.recipes) and \
            (self.tools_cap_left[parts_list[p]['Tool Number']]>0 or\
            (self.tools_used[parts_list[p]['Tool Number']]<\
             tools_list[parts_list[p]['Tool Number']]['Tools Available'] and \
             self.vleft>=tools_list[parts_list[p]['Tool Number']]['Volume'])) and\
             len(self.parts)<parts_list[p]['Max Per Load']:
            return True
        else:
            #If designed to fit into autoclave, what restricted it?
            if parts_list[p][str(self.a+' Priority')]<>0:
                if self.vleft<=tools_list[parts_list[p]['Tool Number']]["Volume"]:
                    self.vol_const+=1 #Counts number constrained by Volume
                if self.tleft<=parts_list[p]["Thermos"]:
                    self.thermo_const+=1 #Counts number constrainted by Thermos
            return False
    def add_part(self,parts_list,tools_list,p):
        self.parts.append(p)
        self.manpower+=parts_list[p]['Manpower Reqd (Hrs/part)']
        self.time=max(self.time,parts_list[p][str(self.a+' P-Time')]) #Sets time
        #Determine if a tool must be added with the part:
        if self.tools_cap_left[parts_list[p]['Tool Number']]==0:
            self.tools_used[parts_list[p]['Tool Number']]+=1
            self.tools_cap_left[parts_list[p]['Tool 
Number']]+=tools_list[parts_list[p]['Tool Number']]['Max Parts/Tool']
            self.tools.append(parts_list[p]['Tool Number']) #Add a tool for this part 
to list
            self.vleft-=tools_list[parts_list[p]['Tool Number']]['Volume'] #Reduce 
volume only if tool is added
        self.tleft-=parts_list[p]["Thermos"] #Reduce thermos only if tool  is added
        self.tools_cap_left[parts_list[p]['Tool Number']]-=1 #Reduce the tool capacity 
left

        #Update longest layup time for part in bin
        if self.longest_part<parts_list[p]['Shift-Hrs Reqd']:
            #print p, parts_list[p]['Shift-Hrs Reqd'], parts_list[p]['Manpower Reqd 
(Hrs/part)']
            self.longest_part=parts_list[p]['Shift-Hrs Reqd']

        #print [parts_list[p]['Manpower Reqd (Hrs/part)'] for p in self.parts]
        #print "self.longest= ", self.longest_part

    def remove_part(self,parts_list,tools_list,p):
        try:
            self.parts.remove(p)
            #Add manpower
            self.manpower-=parts_list[p]['Manpower Reqd (Hrs/part)']
            #Add back tool capacity
            self.tools_cap_left[parts_list[p]['Tool Number']]+=1
            #Add back Thermos
            self.tleft+=parts_list[p]['Thermos']
            #print "Thermos left after part removed: ", self.tleft

            if self.tools_cap_left[parts_list[p]['Tool
Number']]>=tools_list[parts_list[p]['Tool Number']]['Max Parts/Tool']:
                #Remove the tool
                self.tools.remove(parts_list[p]['Tool Number'])
                self.tools_used[parts_list[p]['Tool Number']]-=1
                #Reduce capacity
                self.tools_cap_left[parts_list[p]['Tool
Number']]-=tools_list[parts_list[p]['Tool Number']]['Max Parts/Tool']
                #Add back volume
                self.vleft+=tools_list[parts_list[p]['Tool Number']]['Volume']
            ####RECENT CHANGE###
            #print "Remove_part: line 152 change, Bin Class.py"
            #print [parts_list[p]['Manpower Reqd (Hrs/part)'] for p in self.parts]
            if len(self.parts)>0:
                #print [parts_list[p]['Shift-Hrs Reqd'] for p in self.parts]
                self.longest_part=max([parts_list[jj]['Shift-Hrs Reqd'] for jj in
self.parts])
            #print "self.longest= ", self.longest_part
        except:
            print "\nEXCEPT STATEMENT BEGINS"
            print "p: ", p
            print "parts: ",self.parts
            print "tools: ",self.tools
            print "self.tools_used:",self.tools_used
            #print "tools used of p: ",self.tools_used[parts_list[p]['Tool Number']]
            print "ERROR: removing tool"
            print "RELOAD an older version to avoid further errors"
            import time
            time.sleep(10)
            return 0
```

A3-Proposed Scheduling Model as Implemented

```
def CreateSchedule(AWS_d,wknds,blkt):
    import amazon
    #####   Start Gurobi COMPUTESERVER     #####
    Sch_ip=amazon.start_instance('Sch')
    amazon.set_gur_license() #sets gurobi license


    from random import randint
    import time
    import pyomo.environ as cpr
    from pyomo.opt import SolverFactory, SolverStatus, TerminationCondition
    import datetime
    import calendar
    from itertools import chain
    import math
    import pickle

    #Save inputs for troubleshooting
    pickle.dump([AWS_d,wknds,blkt],open("sched.p","wb"))
    #AWS_d,wknds,blkt=pickle.load(open("sched_updated_4wks.p","rb"))
    #decode=pickle.load(open("decode.p","rb"))

    months={'January': 1,'February': 2,'March': 3,
            'April': 4,'May': 5,'June':6,
            'July':7,'August': 8,'September': 9,
            'October': 10,'November':11,'December': 12}

    print "Starting CreateSchedule"

    #####   Get Data     #####
    B_iter=[]
    wks=[] #Wks that contain BPPs
    au_list=[]
    dummies={}
    for w in AWS_d['Weeks']:
        if AWS_d[w]['Action']=='BPP_Done':
            #dummies[w]=AWS_d[w]['DummyArea']
            for b in AWS_d[w]['BPP'][0]:
                B_iter.append(b) #Add Bin to B_iter
                au_check=b.a
                if au_check not in au_list:
                    au_list.append(au_check)
            wks.append(w)
    tlist=AWS_d['Tools']


    #Manpower import
```

```
layup_manpower=AWS_d['Layup_Manpower']
print layup_manpower


#Used to translate tool numbers to rangeset integers
t_dict={}
t_dict2={}
i=0
for item in tlist: #Give each tool number an integer number
        t_dict[item]=i
        t_dict2[i]=item
        i+=1

#Used to translate autoclave numbers to rangeset integers
au_dict={}
#Used to translate rangeset integers to autoclave numbers
au_dict2={}
i=0
for item in au_list:
   au_dict[item]=i
   au_dict2[i]=item
   i+=1


#####   Create Model     #####
model=cpr.ConcreteModel()

au=len(au_list) #number of autoclaves
ls=len(B_iter) #number of logical sets
#minutes=1 #1= in hours, 60=in minutes, 4=in 15 minute intervals
ss=len(wks)*7*24 #Hrs in current model
p=len(tlist)

print "WKS: ", wks
weeks={}
for item in wks:
    weeks[item]=xrange(item*168,(item+1)*168)


#####   Sets     #####
#Creates a list of all tools, z
model.P=cpr.RangeSet(0,p-1)

#Creates a list of all the autoclaves, j
model.Au=cpr.RangeSet(0,au-1)

#Creates a list of all weeks, w
model.W=cpr.RangeSet(0,len(wks)-1)

#Creates a list of all logical sets, i
```

```
model.L=cpr.RangeSet(0,ls-1)

#Creates a list of all available shifts, k
#1...s
model.S=cpr.RangeSet(0,ss-1)

#Creates a list of all available areas, r
areas=[]


for i in AWS_d['AreaShifts']:
    areas.append(i)
model.Ar=cpr.Set(initialize=areas)


#####   Parameters     #####
#Determines which LS are dummies
dum={}
for i in model.L:
    if B_iter[i].area=='dummy':
        dum[i]=1 #Dummy
    else:
        dum[i]=0 #Not a Dummy
model.DL=cpr.Param(model.L,initialize=dum)

#Determines the last slot in each week
wmax={}
for w in model.W:
        wmax[w]=max(weeks[w])
model.MaxInWeek=cpr.Param(model.W, initialize=wmax)

#Determines the first slot in each week
wmin={}
for w in model.W:
        wmin[w]=min(weeks[w])
        #print "Min in week %s is %s"%(w,min(weeks[w]))
model.MinInWeek=cpr.Param(model.W, initialize=wmin)

#Logical sets in each week
lwk={}
for w in model.W: #Create a list for each week
        lwk[w]=[]
for i in model.L: #Add i to appropriate week list
        lwk[B_iter[i].week].append(i)
        #lwk[week_schedule[i]].append(i)
model.LL=cpr.Param(model.W,initialize=lwk)

#Week for each Logical Set
wkl={}
for w in model.W:
        for i in model.LL[w]:
                wkl[i]=w
model.WKL=cpr.Param(model.L,initialize=wkl)

#Lists of shifts k in each week
sinweek={}
for w in model.W:
        sinweek[w]=[]
for k in model.S:
        for w in model.W:
                if k in weeks[w]:
                        #print w, k
                        sinweek[w].append(k)
model.SW=cpr.Param(model.W, initialize=sinweek)
print "SW[0]: ", model.SW[0]

#Determines week relating to time. Opposite of model.SW. k
wins={}
for w in model.W:
        for k in model.SW[w]:
                wins[k]=w
model.WinS=cpr.Param(model.S,initialize=wins)

#Manpower for each area
mpa={}
for i in model.Ar: #for every area
        mpa[i]=layup_manpower[i]
model.MPA=cpr.Param(model.Ar, initialize=mpa)

#Shifts for each area
model.SAR=cpr.Param(model.Ar, initialize=AWS_d['AreaShifts'])

#Start time of day shift
model.S_Time=AWS_d['Day_Start']

#Manpower for each LS
mpl={}
for i in xrange(0,ls): #for every LS
        mpl[i]=B_iter[i].manpower
model.MPL=cpr.Param(model.L,initialize=mpl)

#Area for each LS
als={}
for i in xrange(0,ls):
        als[i]=B_iter[i].area
model.ALS=cpr.Param(model.L,initialize=als)

#Creates a list of which how many tool instances are in each logical set
pl={}
for i in xrange(0,ls): #For every LS
```

```
        for z in xrange(0,p): #initialize pl to 0
                pl[i,z]=0
        for item in B_iter[i].tools:
                #print item
                pl[i,t_dict[item]]=B_iter[i].tools.count(item) #add items to bin
model.PL=cpr.Param(model.L,model.P, initialize=pl)

#Gets list of logical sets that tool Z goes into.
il={}
for z in xrange(0,p):
        il[z]=[]
        for i in xrange(0,ls):
                if model.PL[(i,z)]<>0:
                        il[z].append(i)
model.IL=cpr.Param(model.P, initialize=il)

#Creates a list of lists that indicates which LS's i are in autoclave j
a={}
for j in xrange(0,au):
        a[j]=[]
for i in model.L:
        a[au_dict[B_iter[i].a]].append(i)
model.A=cpr.Param(model.Au,initialize=a)


#Creates a list of tool recycle times
c={}
for item in tlist:
        c[t_dict[item]]=int(tlist[item]\
                         ['Absolute Minimum Tool Recycle Time (Hrs)'])
model.C=cpr.Param(model.P, initialize = c)

#Creates a list of desired tool recycle times
cd={}
for item in tlist:
        cd[t_dict[item]]=int(tlist[item]['Desired Tool Recycle Time (Hrs)'])
model.CD=cpr.Param(model.P, initialize = cd)

#Creates a tools list, identifying number of tools for each part
tools_list={}
for item in tlist:
        tools_list[t_dict[item]]=int(tlist[item]['Tools Available'])
model.tools=cpr.Param(model.P, initialize=tools_list)

#Creates a list of cycle times
n={}
for i in xrange(0,ls):
    n[i]= int(cpr.ceil(B_iter[i].time/(60)))#originally in minutes

#Weekends- from Main2_AWS_side
```

```
model.weekends=wknds#cpr.Param(initialize=wknds)

#Blackout- from Main2_AWS_side
blkt_trans={}
for item in au_list:
        blkt_trans[au_dict[item]]=blkt[item]
model.blkt=cpr.Param(model.Au, initialize=blkt_trans)
print "Blackout: "
print blkt_trans


#Get list of bin packings that are in same week
#that are exactly the same in terms of tools
#Must also not be fixed
wsame={}
for w in model.W:
    #for i in lwk[w]:
    wsame[w]=[(i,ii) for i in lwk[w] for ii in lwk[w] if \
     model.ALS[i]==model.ALS[ii] and\
     set(B_iter[i].tools)==set(B_iter[ii].tools) and\
      (B_iter[i].a==B_iter[ii].a) and\
    B_iter[i].fixed<>1 and B_iter[ii].fixed<>1 and\
     i<ii]

model.Wsame=cpr.Param(model.W, initialize=wsame)

#Get minimum time units apart that two bins that are exactly the same
#must be scheduled apart. Based on tool recycle times.
wsame_time={}
for w in model.W:
    for i,ii in model.Wsame[w]:
        #check only i since both are same
        #For tools that are the same and bin uses maximum amount of tools available
        try:
            t=max([tlist[z]['Absolute Minimum Tool Recycle Time (Hrs)'] \
             for z in B_iter[i].tools \
                          if sum(1 for zz in B_iter[i].tools if zz==z)/\
                          (model.tools[t_dict[z]])==1])
            wsame_time[(i,ii)]=t
            print wsame_time[(i,ii)]
        except:
            wsame_time[(i,ii)]=0


#Get list of bin packings that share one tool and use all of that tool
#If in same area
        #If not in wsame[w] already
#For every tool in there, if both are the same tool # and both use all of that tool
        #If at least one of the circumstances in the above line happens,
        #add to list.
    wsomesame={}
    for w in model.W:
        wsomesame[w]=[(i,ii) for i in lwk[w] for ii in lwk[w] \
                      if model.ALS[i]==model.ALS[ii] and \
                      i<>ii and\
                      ((i,ii) not in wsame[w]) and ((ii,i) not in wsame[w]) and\
                      sum(1 for z in B_iter[i].tools\
                          for zz in B_iter[ii].tools \
                          if z==zz and ((sum(1 for zc in B_iter[i].tools if zc==z)+ \
                                         sum(1 for zzc in B_iter[ii].tools if zzc==z))\
                                        (2*model.tools[t_dict[z]]))==1\
                          )>=1]
    model.Wsomesame=cpr.Param(model.W, initialize=wsomesame)

    #Get minimum distance apart for bins in wsomesame
    #For every tool in both bins
        #If tools are the same and both bins use maximum number of tool available
    wsomesame_time={}
    for w in model.W:
        for i,ii in model.Wsomesame[w]:
            if i<ii:
                wsomesame_time[(i,ii)]=max([tlist[z]\
                                    ['Absolute Minimum Tool Recycle Time (Hrs)'] \
                           for z in B_iter[i].tools for zz in B_iter[ii].tools \
                           if z==zz and \
                           (((sum(1 for zc in B_iter[i].tools if zc==z)+ \
                             sum(1 for zzc in B_iter[ii].tools if zzc==z))/ \
                            (2*model.tools[t_dict[z]]))==1)])
                #Dont do computation for other side, just copy
                wsomesame_time[(ii,i)]=wsomesame_time[(i,ii)]

#####   Variables     #####

    #Create basic Load[w,i,k] variable
    def set_init(model):
            """Initializes auxiliary set."""
            return [(w, i, k) for w in model.W for i in model.LL[w] \
                      for k in model.SW[w]]
    model.AUX = cpr.Set(dimen=3, initialize=set_init)
    model.Load = cpr.Var(model.AUX, within=cpr.Binary, name='vLoad')


    #Create precedence binary variable, Before[w, i,ii]
    def set_precendence(model):
        return [(i,ii) for w in model.W for i,ii in model.Wsomesame[w]]
    model.AUX_Before=cpr.Set(dimen=2, initialize=set_precendence)
    #Goes both ways. (i,ii) and (ii,i) are both in there.
    model.Before=cpr.Var(model.AUX_Before, within=cpr.Binary, name='vBefore')
```

```
#Variable identifies number of starts on weekend.
model.OT=cpr.Var(within=cpr.NonNegativeReals,name='vOT')

#Weekend penalize per day (Sat, Sun)
model.SatWeekend=cpr.Var(model.W, within=cpr.Binary, name='vSatWeekend')
model.SunWeekend=cpr.Var(model.W, within=cpr.Binary, name='vSunWeekend')

#Variable used to penalize peak power usage
model.Energy=cpr.Var(model.W, within=cpr.NonNegativeReals,\
                     name='vEnergy')

#Infeasibility variables
model.Tool_Inf=cpr.Var(model.P, within=cpr.NonNegativeReals, \
                       name='vTool_Inf')
model.Au_Inf=cpr.Var(model.Au, within=cpr.NonNegativeReals, \
                     name='vAutoclave_Inf')

#Reschedule penalized
model.Resched=cpr.Var(within=cpr.NonNegativeReals,name='vResched')

#Long items to layup at start of week penalized
model.Early=cpr.Var(model.L,within=cpr.NonNegativeReals,name='vEarly')

#Manpower leveler
model.MP_lvl=cpr.Var(model.W, model.Ar,within=cpr.NonNegativeReals,\
                   name='ManPower_Lvl')

#Desired Tool Recycle Time
model.Desired_Tool_Rec=cpr.Var(model.P, within=cpr.NonNegativeReals,\
                                 name='vDesiredRecycleT')



#####   Objective     #####

model.obj = cpr.Objective(expr=4*model.OT\
                          -0.3*sum(model.Load[w,i,k]\
                               for w in model.W\
                               for i in set(model.DL).intersection(model.LL[w])\
                               for k in model.SW[w])\
                          +5*sum(model.Energy[w]\
                                 for w in model.W)
                          +1000*sum(model.Tool_Inf[z]\
                                    for z in model.P)\
                          +1000*sum(model.Au_Inf[j]\
                                    for j in model.Au)\
                          +4*sum(model.MP_lvl[w,r]\
                                 for w in model.W\
                                 for r in model.Ar)\
                          +2*sum(model.Desired_Tool_Rec[z]\
                                       for z in model.P)\
                                +1.5*model.Resched\
                                +1.4*sum(model.Early[i]\
                                          for i in model.L)\
                                +8*sum(model.SatWeekend[w]\
                                       for w in model.W)\
                                +20*sum(model.SunWeekend[w]\
                                       for w in model.W),sense=cpr.minimize)

print "Adding constraints to model"
print "--------------------------"


#####   Contraints     #####

#Create set for constraint 1
def set_init1(model):
        return [(w, i) for w in model.W for i in set(model.LL[w])\
                    if model.DL[i]<>1]
model.AUX1 = cpr.Set(dimen=2, initialize=set_init1)

#1 Each LS is scheduled
print "Adding #1 constraints"
def con1_rule(model,w,i):
        return sum(model.Load[w,i,k]\
                   for k in model.SW[w])== 1
model.con1= cpr.Constraint(model.AUX1,rule=con1_rule, name='Con1')
print "Constraint 1 added", time.clock()


#Auxillary Set: for constraint 101
def set_init101(model):
        return [(w, i) for w in model.W for i in set(model.LL[w])\
                    if model.DL[i]==1]
model.AUX101 = cpr.Set(dimen=2, initialize=set_init101)

#101 Dummy LS are optional to schedule
print "Adding #101 constraints"
def con101_rule(model,w,i):
        return sum(model.Load[w,i,k]\
                   for k in model.SW[w]) <=1
model.con101= cpr.Constraint(model.AUX101, rule=con101_rule, name='Con101')
print "Constraint 101 added", time.clock()



#Auxillary Sets: for constraint 2
def set_init2_0(model):
        return [(k) for k in set(model.SW[0])]
def set_init2_01(model):
        return [(k) for k in set(model.SW[0]).union(model.SW[1])]
def set_init2_12(model):
        return [(k) for k in set(model.SW[1]).union(model.SW[2])]
def set_init2_23(model):
        return [(k) for k in set(model.SW[2]).union(model.SW[3])]



#2 Sufficient TOOL recycle time is assigned
print "Adding #2 and #22 constraints"
def con2_rule(model, k,z):
    a=[model.Load[model.WinS[k+t],i,min(k+t,ss-1)]*model.PL[i,z] \
                   for t in xrange(0,min(model.C[z],ss-k))\
                   for i in set(model.IL[z]).intersection(\
                                model.LL[model.WinS[min(k+t,ss-1)]])]
    if len(a)<=model.tools[z]:
        #print "Constraint 2: k= %s, z=%s feasible"%(k,z)
        return cpr.Constraint.Feasible
    else:
        #print "Constraint 2 added, k=%s, z=%s"%(k,z)
        return sum(a)/model.tools[z] <=1
#+ model.Tool_Inf[z]

#22 Desired Tool recycle time is considered
#print "Adding #22 constraints"
def con22_rule(model, k,z):
    a=[model.Load[model.WinS[k+t],i,min(k+t,ss-1)]*model.PL[i,z] \
                   for t in xrange(0,min(model.CD[z],ss-k))\
                   for i in set(model.IL[z]).intersection(\
                                model.LL[model.WinS[min(k+t,ss-1)]])]
    if len(a)<=model.tools[z]:
        return cpr.Constraint.Feasible
    else:
        return sum(a)/model.tools[z] <=1 + model.Desired_Tool_Rec[z]

if len(wks)==1:
    model.AUX2_0 = cpr.Set(dimen=1, initialize=set_init2_0)
    model.con2_0= cpr.Constraint(model.AUX2_0, model.P, \
                                rule=con2_rule, name='Con2_0')
    model.con22_0= cpr.Constraint(model.AUX2_0, model.P,\
                                 rule=con22_rule, name='Con22_0')
    print "Added #2,#22, constraints for week 0"
if len(wks)>=2:
    model.AUX2_01 = cpr.Set(dimen=1, initialize=set_init2_01)
    model.con2_01= cpr.Constraint(model.AUX2_01, model.P,\
                                 rule=con2_rule, name='Con2_01')
    model.con22_01= cpr.Constraint(model.AUX2_01, model.P,\
                                 rule=con22_rule, name='Con22_01')
    print "Added #2,#22constraints for weeks 0 and 1"
if len(wks)>=3:
    model.AUX2_12 = cpr.Set(dimen=1, initialize=set_init2_12)
    model.con2_12= cpr.Constraint(model.AUX2_12, model.P, \
                            rule=con2_rule, name='Con2_12')
    model.con22_12= cpr.Constraint(model.AUX2_12, model.P,\
                             rule=con22_rule, name='Con22_12')
    print "Added #2,#22 constraints for weeks 1 and 2"
if len(wks)>=4:
    model.AUX2_23 = cpr.Set(dimen=1, initialize=set_init2_23)
    model.con2_23= cpr.Constraint(model.AUX2_23, model.P,\
                             rule=con2_rule, name='Con2_23')
    model.con22_23= cpr.Constraint(model.AUX2_23, model.P,\
                             rule=con22_rule, name='Con22_23')
    print "Added #2,#22 constraints for weeks 2 and 3"
print "Constraint #2,#22 added", time.clock()




#Create set for constraint 3
#List of possible Load[w,i,k]'s running at k.
def set_init3_1(model,j,k):
        return [(i,kk) for i in set(model.L).intersection(model.A[j]) \
                for kk in model.SW[model.WKL[i]]\
                if n[i]+kk>=k and kk<=k]
model.AUX3_1 = cpr.Set(model.Au, model.S, dimen=2, initialize=set_init3_1)

#3 Ensures autoclave CYCLE TIME is respected
print "Adding #3 constraints"
def con3_rule(model, j, k):
        if len(model.AUX3_1[j,k])<=1:
                return cpr.Constraint.Feasible
        else:
                return sum(model.Load[model.WKL[i], i, kk]\
                           for i,kk in model.AUX3_1[j,k])\
                           <= 1 #+ model.Au_Inf[j]
if len(wks)==1:
     model.con3_0= cpr.Constraint(model.Au, model.AUX2_0,\
                             rule= con3_rule, name='Con3_1')
     print "Added #3 constraints for week 0"
if len(wks)>=2:
    model.con3_01= cpr.Constraint(model.Au, model.AUX2_01,\
                             rule= con3_rule, name='Con3_2')
    print "Added #3 constraints for weeks 0 and 1"
if len(wks)>=3:
    model.con3_12= cpr.Constraint(model.Au, model.AUX2_12,\
                             rule= con3_rule, name='Con3_3')
    print "Added #3 constraints for weeks 1 and 2"
if len(wks)>=4:
    model.con3_23= cpr.Constraint(model.Au, model.AUX2_23,\
                             rule= con3_rule, name='Con3_4')
    print "Added #3 constraints for weeks 2 and 3"
print "Constraint 3 added", time.clock()


#Auxillary Set: set for constraint 4
def set_init4(model):
        return [(w, k) for w in model.W for k in model.SW[w]]
model.AUX4 = cpr.Set(dimen=2, initialize=set_init4)

#4 Penalizes that start times to autoclaves are not different
print "Adding #4 constraints"
def con4_rule(model,w,k):
        return sum(model.Load[w,i,k] \
                   for i in set(model.L).intersection(model.LL[w])
                        if model.DL[i]==0)<=model.Energy[w]
model.con4= cpr.Constraint(model.AUX4, rule= con4_rule, name='Con4')
print "Constraint 4 added", time.clock()

#5 Penalizes starts on the weekends
print "Adding constraint #5"
def con5_rule(model):
        return sum(model.Load[w,i,k] \
                   for w in model.W\
                   for i in set(model.L).intersection(model.LL[w]) \
                   for k in set(model.weekends).intersection(\
                                 model.SW[w])) <=model.OT
model.con5=cpr.Constraint(rule= con5_rule, name='Con5')
print "Constraint 5 added", time.clock()


#Auxillary Set: all i's in violation of blocked off times
#at each timepoint for each autoclave
#Could return an empty list
def set_init6(model,j):
        return [(i,k) for i in set(model.L).intersection(model.A[j]) \
                for k in model.S.intersection(model.SW[model.WKL[i]])\
                for kk in model.blkt[j]\
                if n[i]+k+1>=kk and k<=kk]
print "Warning following is normal"
print "It's due to multiple violations of same constraint in preprocessing"
model.AUX6 = cpr.Set(model.Au, dimen=2, initialize=set_init6)

#6 Makes no run/starts on completely blocked off times.
print "Adding constraint #6"
def con6_rule(model,j):
    if len(model.AUX6[j])>0:
        return sum(model.Load[(model.WKL[i],i,k)] \
                for i,k in model.AUX6[j])==0
    else:
        return cpr.Constraint.Feasible
```

```
model.con6=cpr.Constraint(model.Au, rule=con6_rule, name='Con6')
print "Constraint 6 added", time.clock()


#7 Penalizes if k has changed.
print "Adding constraint #7"
def con7_rule(model):
    return sum((1-model.Load[model.WKL[i],i,B_iter[i].scheduled]) \
                for i in model.L\
                if B_iter[i].Rpenalize==True\
                and model.DL[i]<>1)\
                <=\
                model.Resched
model.con7=cpr.Constraint(rule=con7_rule, name='Con7')


#8 Penalizes starts near start of week for large layup items
print "Adding constraint #8"
def con8_rule(model,w, i):
    n=int(math.ceil(B_iter[i].longest_part)) #get number of shift-hrs to complete
    sh=0
    #s_wk=model.WKL[i]*7*24

    #model.SAR[model.ALS[i]] is number of shifts for area
    #Check if have to count number of complete shifts
    if model.SAR[model.ALS[i]]<3:
        #count number of shifts, and leave overflow
        while n>8:
            n-=8
            sh+=1
    #Assumes only day shift works on part
    if model.SAR[model.ALS[i]]==1:
        return sum(model.Load[w,i,k]\
                    for k in xrange(model.MinInWeek[w],\
                                    model.MinInWeek[w]+int(sh*24)+n+model.S_Time))\
                    <=model.Early[i]
    #Assumes day and evening shifts work on parts
    if model.SAR[model.ALS[i]]==2:
        return sum(model.Load[w,i,k]\
                    for k in xrange(model.MinInWeek[w],\
                                    model.MinInWeek[w]+int(math.floor(sh/2))*24\
                                    +sh%2*8+n+model.S_Time))\
                    <=model.Early[i]
    #Since shifts are continuous, dont worry about shift times
    if model.SAR[model.ALS[i]]==3:
        return sum(model.Load[w,i,k]\
                    for k in xrange(model.MinInWeek[w],model.MinInWeek[w]+n))\
                         <=model.Early[i]

#9 Manpower Leveling Constraint
```

```
print "Adding constraint #9"
def con9_rule(model, w, r, d):
    if len([i for i in model.L if model.ALS[i]==r])==0:
        print "Constraint #9- (%s,%s,%s) FEASIBLE"%(w,r,d)
        return cpr.Constraint.Feasible
    else:
        return sum(model.Load[model.WKL[i],i,k]*model.MPL[i]\
                   for i in model.L if model.ALS[i]==r\
                   for k in xrange(model.WKL[i]*7*24+d*24,\
                                   model.WKL[i]*7*24+(d+1)*24))\
                    /(model.MPA[r]*model.SAR[r]*8)\
                    <= model.MP_lvl[w,r]

#10 Overtime on Saturday, daily penalization
def con10_rule(model,w):
    return sum(model.Load[w,i,k]\
               for i in model.LL[w]\
               for k in xrange(w*7*24+5*24,w*7*24+5*24+17))\
               <=\
               model.SatWeekend[w]*100

#11 Overtime from Saturday 5pm to Sunday midnight, daily penalization
def con11_rule(model,w):
    return sum(model.Load[w,i,k]\
               for i in model.LL[w]\
               for k in xrange(w*7*24+5*24+17,w*7*24+7*24))\
               <=\
               model.SunWeekend[w]*100

#12 Symmetry restricting constraint for tool recycle time when bins are identical
#When bins use the maximum number of tools available they are further split
#by minimum
    #tool recycle time
def con12_rule(model, w,i,ii):
    return sum(model.Load[w, i, k]*k for k in model.SW[w])+ wsame_time[(i,ii)] <= \
           sum(model.Load[w, ii, k]*k for k in model.SW[w])+1


#13_most Removes some possibilities in an easier fashion than the #2 constraint
#Does not include any dummy bins
def con13_rule(model, w, i, ii):
    return sum(model.Load[w, i, k]*k for k in model.SW[w])+ \
           wsomesame_time[i,ii]*model.Before[i,ii] - \
           sum(model.Load[w, ii, k]*k for k in model.SW[w]) -1 \
           <= \
           0 + (model.MaxInWeek[0]+1)*(1-model.Before[i,ii])

#Segmented constraint calls
if len(wks)>=1:
    #Constraint 12 and 13 Auxillary Sets
    #model.AUX12_0=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                                           for i,ii in model.Wsame[w] if w==0])
    model.AUX13_0=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W
                        for i,ii in model.Wsomesame[w] if w==0])

    model.con8_0=cpr.Constraint(set([0]), model.LL[0],\
                               rule=con8_rule, name='Con8_0')
    model.con9_0=cpr.Constraint(set([0]), model.Ar, xrange(0,7),\
                               rule=con9_rule, name='Con9_0')
    model.con10_0=cpr.Constraint(set([0]), rule=con10_rule, name='Con10_0')
    model.con11_0=cpr.Constraint(set([0]), rule=con11_rule, name='Con11_0')
    model.con12_0=cpr.Constraint(model.AUX12_0, rule=con12_rule, name='Con12_0')
    model.con13_0=cpr.Constraint(model.AUX13_0, rule=con13_rule,name='Con13_0')
    print "Added #9-13 constraints for week 0"

if len(wks)>=2:
    #Constraint 12 and 13 Auxillary Sets
    #model.AUX12_1=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                                           for i,ii in model.Wsame[w] if w==1])
    model.AUX13_1=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                        for i,ii in model.Wsomesame[w] if w==1])

    model.con8_1=cpr.Constraint(set([1]), model.LL[1],\
                               rule=con8_rule, name='Con8_1')
    model.con9_1=cpr.Constraint(set([1]), model.Ar, xrange(0,7),\
                               rule=con9_rule, name='Con9_1')
    model.con10_1=cpr.Constraint(set([1]), rule=con10_rule, name='Con10_1')
    model.con11_1=cpr.Constraint(set([1]), rule=con11_rule, name='Con11_1')
    #model.con12_1=cpr.Constraint(model.AUX12_1, rule=con12_rule, name='Con12_1')
    model.con13_1=cpr.Constraint(model.AUX13_1, rule=con13_rule,name='Con13_1')
    print "Added #9-13 constraints for week 1"

if len(wks)>=3:
    #Constraint 12 and 13 Auxillary Sets
    #model.AUX12_2=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                                           for i,ii in model.Wsame[w] if w==2])
    model.AUX13_2=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                        for i,ii in model.Wsomesame[w] if w==2])


    model.con8_2=cpr.Constraint(set([2]), model.LL[2],\
                        rule=con8_rule, name='Con8_2')
    model.con9_2=cpr.Constraint(set([2]), model.Ar, xrange(0,7),\
                        rule=con9_rule, name='Con9_2')
    model.con10_2=cpr.Constraint(set([2]), rule=con10_rule, name='Con10_2')
    model.con11_2=cpr.Constraint(set([2]), rule=con11_rule, name='Con11_2')
    #model.con12_2=cpr.Constraint(model.AUX12_2, rule=con12_rule, name='Con12_2')
    model.con13_2=cpr.Constraint(model.AUX13_2,rule=con13_rule,name='Con13_2')
    print "Added #9-13 constraints for week 2"
```

```
if len(wks)>=4:
    #Constraint 12 and 13 Auxillary Sets
    #model.AUX12_3=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                                           for i,ii in model.Wsame[w] if w==3])
    model.AUX13_3=cpr.Set(dimen=3, initialize=[(w,i,ii) for w in model.W\
                    for i,ii in model.Wsomesame[w] if w==3])

    model.con8_3=cpr.Constraint(set([3]), model.LL[3],\
                             rule=con8_rule, name='Con8_3')
    model.con9_3=cpr.Constraint(set([3]), model.Ar, xrange(0,7),\
                             rule=con9_rule, name='Con9_3')
    model.con10_3=cpr.Constraint(set([3]), rule=con10_rule, name='Con10_3')
    model.con11_3=cpr.Constraint(set([3]), rule=con11_rule, name='Con11_3')
    #model.con12_3=cpr.Constraint(model.AUX12_3, rule=con12_rule, name='Con12_3')
    model.con13_3=cpr.Constraint(model.AUX13_3, rule=con13_rule,name='Con13_3')
    print "Added #9-13 constraints for week 3"
print "Constraints #9-13 added", time.clock()


#14 One must be before the other.
def con14_rule(model, i,ii):
    return model.Before[i,ii]+model.Before[ii,i]==1

model.con14=cpr.Constraint(model.AUX_Before, \
                                  rule=con14_rule, \
                                  name='Con14_rule')

#Load initial solution
#Fix anything that should be fixed.
for i in model.L:
    if B_iter[i].scheduled<>None:
        for k in model.SW[B_iter[i].week]:
            if k<>B_iter[i].scheduled:
                #print "Load[%s,%s,%s]"%(B_iter[i].week,i,k)
                model.Load[B_iter[i].week,i,k]=0
            else:
                if B_iter[i].fixed==1:
                    print B_iter[i].num, "scheduled+fixed: ", k
                else:
                    print B_iter[i].num, "scheduled: ", k

                model.Load[B_iter[i].week,i,k]=1
            if B_iter[i].fixed==1:
                model.Load[B_iter[i].week,i,k].fixed=True
    else:
        print "B_iter[%s] is not scheduled"%i

def fix_week(w):
        for i in model.LL[w]:
                for k in model.SW[w]:
                        model.Load[w,i,k].fixed=True

#opt.symbolic_solver_labels = True
#model.write('problem_whole.lp')
#Deactivate constraints except for the first week (week 0)
if len(wks)>=3:
    model.con2_12.deactivate()
    model.con22_12.deactivate()
    model.con3_12.deactivate()
    model.con8_2.deactivate()
    model.con9_2.deactivate()
    model.con10_2.deactivate()
    model.con11_2.deactivate()
    #model.con12_2.deactivate()
    model.con13_2.deactivate()

if len(wks)>=4:
    model.con2_23.deactivate()
    model.con22_23.deactivate()
    model.con3_23.deactivate()
    model.con8_3.deactivate()
    model.con9_3.deactivate()
    model.con10_3.deactivate()
    model.con11_3.deactivate()
    #model.con12_3.deactivate()
    model.con13_3.deactivate()

for w in model.W:
    if w==3:
        fix_week(0) #Fix values for W0
        fix_week(1) #Fix values for W1
        model.con2_23.activate()
        model.con22_23.activate()
        model.con3_23.activate()
        model.con8_3.activate()
        model.con9_3.activate()
        model.con10_3.activate()
        model.con11_3.activate()
        #model.con12_3.activate()
        model.con13_3.activate()


    if w==0 and len(wks)>1:
        continue

    if w==2:
        fix_week(0) #Fix values for W0
        model.con2_12.activate()
        model.con22_12.activate()
        model.con3_12.activate()
        model.con8_2.activate()
```

```
            model.con9_2.activate()
            model.con10_2.activate()
            model.con11_2.activate()
            #model.con12_2.activate()
            model.con13_2.activate()

        print "\n\nSolving weeks %s and %s"%(w-1,w), time.clock()

        opt = SolverFactory('gurobi', solver_io='python')
        #opt.symbolic_solver_labels = False #Make them symbolic
        #opt.options['ResultFile']='problem_weeks%s_%s.lp'%(w-1,w)
        opt.options['MIPGap']=0.005
        opt.options['MIPGapAbs']=0.2
        #opt.options['Heuristics']=0.80
        opt.options['MIPFocus']=1
        opt.options['DisplayInterval']=10
        opt.options['TimeLimit']=800

        startsolve=time.clock()
        results = opt.solve(model, tee=True) #, warmstart=True)
        #print results.Solution.Status
        print "Solved in: ", time.clock()-startsolve, " seconds"
        print "Retrieving results", time.clock()
        #model.solutions.load_from(results)
        #instance.load(results)

        print "Tool Infeasibility= %s"%sum(model.Tool_Inf[z].value for z in model.P)
        for z in model.P:
            if model.Tool_Inf[z].value >0:
                print "Infeasible tool: ", z, model.Tool_Inf[z].value
                #Remove next line, works only with decode
                #print "Tool ID:", decode[z]
                #print tlist[z]'''


    if (results.solver.termination_condition == TerminationCondition.infeasible):
        print "\nModel is infeasible\n"
        AWS_d['Feasibility']=False
        pickle.dump(AWS_d,open('AWS_d.p','wb'))
        print "Wrote report" #Ends the connection

    else:
        #Model is feasible
        #####   Report     #####
        f=open('output.txt','w')
        f.write("Bin ,Autoclave, Hour\n")
        sch_results={}

        for w in model.W:
            for k in model.SW[w]:
```

```
        for j in model.Au:
            for i in set(model.A[j]).intersection(model.LL[w]):
                if model.Load[w,i,k]==1:
                    sch_results[i]=k #set time for load i
                    f.write("\n%s,%s,%s, n[i]= %s, Au ready again: %s,",\"
                             Tool ready again (if 24 hrs): %s"\
                            %(i,au_dict2[j],k,n[i],str(k+n[i]),str(k+c[0])))
                    f.write(B_iter[i].Bwrite())
                    f.write("\n")
f.close()

print "model.Early[i]"
for i in model.L:
    if model.Early[i].value<>0:
        print i, model.Early[i].value

print "model.MP_lvl[w,r]"
for r in model.Ar:
    for w in model.W:
        print r,w, ": ", model.MP_lvl[w,r].value

#Inf=[]
'''print "Tool Infeasibility= %s"%sum(model.Tool_Inf[z].value for z in model.P)
for z in model.P:
        if model.Tool_Inf[z].value >0:
                print "Infeasible tool: ", t_dict2[z], model.Tool_Inf[z].value
                Inf.append(('Tool Infeasibility',t_dict2[z]))

print "Au Infeasibility= %s"%sum(model.Au_Inf[j].value for j in model.Au)
for j in model.Au:
        if model.Au_Inf[j].value >0:
                print "Infeasible Au: ", j, model.Au_Inf[j].value
                Inf.append(('Autoclave Infeasibility',au_dict2[j]))'''

AWS_d['Feasibility']=True

#Count number of desired recycle times unmet
drec=0
for z in model.P:
    if model.Desired_Tool_Rec[z].value<>0:
        print "Drec: ",z,", : ",model.Desired_Tool_Rec[z].value
        drec+=1

#Count number of changes to initial schedule
resch=0
for i in model.L:
    if B_iter[i].scheduled<>None:
        for k in model.SW[model.WKL[i]]:
            if model.Load[model.WKL[i],i,k]==1 and \
               B_iter[i].scheduled<>k:
                    resch+=1


#Count number of dummy slots scheduled
n=0
for w in model.W:
    for i in model.L.intersection(model.LL[w]):
        if model.DL[i]==1:
            for k in model.SW[w]:
                if model.Load[w,i,k]==1:
                    n+=1

#Count instances where 2+ autoclaves started
en=0
for w in model.W:
    for k in model.SW[w]:
        j=0
        for i in model.L.intersection(model.LL[w]):
            if model.Load[w,i,k]==1 and model.DL[i]==0:
                j+=1
        if j>1:
            en+=1
        if j>=2:
            print "Started %s autoclaves at once at time %s!"%(j,k)

#Count number of Saturdays and Sundays scheduled
wkends=[0,0] #Sat, Sun
for w in model.W:
    print model.SatWeekend[w].value
    print model.SunWeekend[w].value
    wkends[0]+=model.SatWeekend[w].value
    wkends[1]+=model.SunWeekend[w].value

print "Scheduled %s dummy slots"%n
print "Rescheduled %s bins from original schedule"%resch
print "Scheduled %s OT starts"%model.OT.value
print "Saturday OT Weekends: %s"%wkends[0]
print "Sunday OT Weekends: %s"%wkends[1]
print "Number of 2+ concurrent autoclave starts: %s"%en
print "Number of desired tool recycle times not met: %s"%drec
for w in model.W:
    print "Energy usage value for week %s: %s"%(w,model.Energy[w].value)

#B_iter=pickle.load(open('BPP_Pickle.p', "rb" ))
for item in sch_results:
    B_iter[item].scheduled=sch_results[item]

for item in B_iter:
    for b in AWS_d[item.week]['BPP'][0]:
        if item.num==b.num:
            b.scheduled=item.scheduled
            #print b.Bprint()


pickle.dump(AWS_d,open('AWS_d.p','wb'))
print "solution decoded"

#Print line below is essential. The local computer scans for it to identify when
#solving is completed
print "Wrote report"
```

Appendix B

B1- Computational Results from Segmented Scheduling Model

Shows the Gurobi log created when running the segmented scheduling model with parameters in #2 from Table 5for the 4 new weeks trial.

B2- Computational Results of Full Model

Shows the Gurobi log created when running the full scheduling model with parameters in #12 from Table 5for the 4 new weeks trial.

B1- Computational Results from Segmented Scheduling Model

The segmented model runs three 1000 second runs, the first run is solving weeks 0 and 1 while leaving the rest unconstrained. This can also be seen as the best bound of the objective function at the end of this run is 35, and after constraints are added to week 2 the best bound jumps up to 97. This process is also evident in the third run. The gap in contrast gets progressively smaller with each run, this is due to a number of variables being fixed every time. In the second run, week 0 is completely fixed and therefore the variables are taken out (also observable in the decreased number of variables in each run). As variables get fixed, potential solutions are eliminated and the problem becomes easier to solve and therefore reducing the gap is considerably easier.

```
Gurobi 6.0.4 (win64) logging started 07/30/15 15:57:43

Optimize a model with 67210 rows, 44580 columns and 2530018 nonzeros
Coefficient statistics:
  Matrix range     [1e-02, 3e+02]
  Objective range  [3e-01, 1e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+02]
Presolve removed 50917 rows and 15840 columns
Presolve time: 4.19s
Presolved: 16293 rows, 28740 columns, 988247 nonzeros
Variable types: 114 continuous, 28626 integer (28626 binary)
Found heuristic solution: objective 733.0240942

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   14603    1.9472006e+01   5.897403e+05   0.000000e+00     10s
   27875    2.1569016e+01   1.966042e+04   0.000000e+00     20s
   37276    2.2188764e+01   1.495826e+05   0.000000e+00     30s
   46914    2.2286251e+01   1.919308e+03   0.000000e+00     40s
   48591    2.2347470e+01   0.000000e+00   0.000000e+00     42s

Root relaxation: objective 2.234747e+01, 48591 iterations, 37.50 seconds
Total elapsed time = 53.05s
Total elapsed time = 65.22s
Total elapsed time = 72.24s
```

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   22.34747    0 1082  733.02409   22.34747  97.0%     -   81s
H    0     0                     65.4893116   22.34747  65.9%     -   84s
H    0     0                     64.5706522   22.34747  65.4%     -  102s
     0     0   33.65110    0  729   64.57065   33.65110  47.9%     -  125s
H    0     0                     60.6987319   33.65110  44.6%     -  149s
     0     0   34.30798    0  766   60.69873   34.30798  43.5%     -  155s
     0     0   34.54441    0  832   60.69873   34.54441  43.1%     -  181s
     0     0   34.56729    0  771   60.69873   34.56729  43.1%     -  195s
     0     0   34.56729    0  485   60.69873   34.56729  43.1%     -  226s
H    0     0                     53.1862319   34.56729  35.0%     -  238s
H    0     2                     53.0112319   34.56729  34.8%     -  266s
     0     2   34.56729    0  485   53.01123   34.56729  34.8%     -  266s
     1     3   34.73561    1  572   53.01123   34.73561  34.5% 10140  274s
     5     6   34.95337    5  392   53.01123   34.95337  34.1%  2376  316s
     7     6   34.95337    6  403   53.01123   34.95337  34.1%  7947  320s
H   11     7                     52.9278986   34.95337  34.0%  5413  321s
    33    32   34.95337   18  397   52.92790   34.95337  34.0%  2028  331s
    48    41   34.95337   22  413   52.92790   34.95337  34.0%  1463  341s
    52    44   34.95337   23  404   52.92790   34.95337  34.0%  1363  351s
    76    66   34.95337   29  391   52.92790   34.95337  34.0%   983  363s
H  146   138                     52.8445652   34.95337  33.9%   585  366s
   282   272   36.41104   80  390   52.84457   34.95337  33.9%   367  372s
   446   438   41.05270  127  358   52.84457   34.95337  33.9%   286  380s
   769   757   41.09470  199  430   52.84457   34.95337  33.9%   222  392s
   914   902   41.13670  240  405   52.84457   34.95337  33.9%   217  401s
  1046  1028   41.18570  264  295   52.84457   34.95337  33.9%   218  410s
  1374  1355   42.98170  334  134   52.84457   34.95337  33.9%   197  421s
  1877  1839   45.84420  447   18   52.84457   34.95337  33.9%   170  432s
  2142  2094   35.94437   59  485   52.84457   34.95337  33.9%   163  476s
  2144  2095   45.54420  416  840   52.84457   34.95337  33.9%   162  564s
  2145  2096   52.75254  284  732   52.84457   34.95337  33.9%   162  623s
  2146  2097   51.66920  132  766   52.84457   34.95337  33.9%   162  631s
  2147  2097   41.05270  134  753   52.84457   34.95337  33.9%   162  642s
  2148  2098   45.54420  383  807   52.84457   34.95337  33.9%   162  654s
  2149  2099   41.10520  232  511   52.84457   34.95337  33.9%   162  687s
H 2149  1993                     52.8373188   34.95337  33.8%   162  719s
H 2149  1894                     52.8373188   34.95337  33.8%   162  722s
  2151  1897   35.08670   15  412   52.83732   35.08670  33.6%   249  768s
  2153  1896   35.08670   16  484   52.83732   35.08670  33.6%   250  770s
  2163  1912   35.08670   20  392   52.83732   35.08670  33.6%   252  790s
  2203  1935   35.08670   25  372   52.83732   35.08670  33.6%   262  804s
  2299  1989   35.55337   38  411   52.83732   35.08670  33.6%   271  818s
H 2326  1911                    52.7539855   35.08670  33.5%   273  818s
  2435  1995   37.81004   51  387   52.75399   35.08670  33.5%   278  833s
  2599  2096   38.56476   64  424   52.75399   35.08670  33.5%   283  857s
  2984  2282   41.27504   88  313   52.75399   35.08670  33.5%   273  884s
H 3143  2260                    52.7123188   35.08670  33.4%   276  884s
  3565  2546   45.07504  152  138   52.71232   35.08670  33.4%   248  900s
  4564  3133   45.71504  361   35   52.71232   35.08670  33.4%   211  928s
  5579  3698   38.37354   85  400   52.71232   35.08670  33.4%   192  946s
H 5600  3605                    52.7123188   35.08670  33.4%   192  947s
H 6123  3630                    52.6706522   35.08670  33.4%   183  947s
  6558  3980   52.20704  120  385   52.67065   35.08670  33.4%   175  965s
  7335  4480   52.20704  182  258   52.67065   35.08670  33.4%   168  986s
  7684  4787     cutoff   28        52.67065   35.08670  33.4%   173 1000s

Cutting planes:
  Gomory: 7
  Cover: 2
  Clique: 74
  MIR: 20
  Flow cover: 13
  GUB cover: 15
  Zero half: 12

Explored 7963 nodes (1555323 simplex iterations) in 1000.10 seconds
Thread count was 4 (of 4 available processors)

Time limit reached
Best objective 5.267065217391e+01, best bound 3.508670289855e+01, gap 33.3847%

Gurobi 6.0.4 (win64) logging started 07/30/15 16:14:45

Optimize a model with 117998 rows, 33324 columns and 3621349 nonzeros
Coefficient statistics:
  Matrix range    [1e-02, 5e+02]
  Objective range [3e-01, 1e+03]
  Bounds range    [1e+00, 1e+00]
  RHS range       [2e-01, 3e+02]
Found heuristic solution: objective 467.393
Presolve removed 103521 rows and 11049 columns
Presolve time: 5.72s
Presolved: 14477 rows, 22275 columns, 894944 nonzeros
Found heuristic solution: objective 338.5926659
Variable types: 110 continuous, 22165 integer (22165 binary)
```

```
Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   10048    9.4283671e+01   4.900670e+05   0.000000e+00     10s
   25273    9.5895745e+01   2.879208e+03   0.000000e+00     20s
   37948    9.6515779e+01   9.683992e+03   0.000000e+00     30s
   51503    9.6607817e+01   6.937094e+03   0.000000e+00     40s
   62344    9.6629781e+01   0.000000e+00   0.000000e+00     49s

Root relaxation: objective 9.662978e+01, 62344 iterations, 42.56 seconds
Total elapsed time = 50.96s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   96.62978    0  914  338.59267   96.62978  71.5%     -   55s
H    0     0                    259.0961957   96.62978  62.7%     -   56s
H    0     0                    193.6548913   96.62978  50.1%     -   56s
H    0     0                    114.8152174   96.62978  15.8%     -   77s
     0     0   97.13396    0 1044  114.81522   97.13396  15.4%     -   88s
H    0     0                    112.1152174   97.13396  13.4%     -  113s
     0     0   97.28285    0 1083  112.11522   97.28285  13.2%     -  124s
     0     0   97.39927    0 1105  112.11522   97.39927  13.1%     -  143s
     0     0   97.40992    0 1091  112.11522   97.40992  13.1%     -  149s
H    0     0                    111.8152174   97.40992  12.9%     -  187s
     0     0   97.43914    0 1031  111.81522   97.43914  12.9%     -  206s

Cutting planes:
  Gomory: 2
  Cover: 9
  Clique: 89
  MIR: 25
  GUB cover: 6
  Zero half: 33

Explored 0 nodes (125082 simplex iterations) in 249.69 seconds
Thread count was 4 (of 4 available processors)

Solve interrupted
Best objective 1.118152173913e+02, best bound 9.743913576190e+01, gap 12.8570%

Gurobi 6.0.4 (win64) logging started 07/30/15 16:19:25

Optimize a model with 147100 rows, 22068 columns and 3411503 nonzeros
Coefficient statistics:
  Matrix range     [1e-02, 7e+02]
  Objective range [3e-01, 1e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e-01, 3e+02]
Found heuristic solution: objective 411.425
Presolve removed 134637 rows and 6400 columns
Presolve time: 5.60s
Presolved: 12463 rows, 15668 columns, 767049 nonzeros
Found heuristic solution: objective 303.0920293
Variable types: 92 continuous, 15576 integer (15576 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   13601    1.6114028e+02   4.432704e+04   0.000000e+00     10s
   30949    1.6260294e+02   0.000000e+00   0.000000e+00     19s

Root relaxation: objective 1.626029e+02, 30949 iterations, 13.23 seconds
Total elapsed time = 20.86s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  162.60294    0  651  303.09203  162.60294  46.4%     -   25s
H    0     0                    249.2173913  162.60294  34.8%     -   25s
H    0     0                    173.1387681  162.60294  6.09%     -   42s
     0     0  163.10051    0  785  173.13877  163.10051  5.80%     -   55s
     0     0          -    0       173.13877  163.10051  5.80%     -   62s

Cutting planes:
  Gomory: 4
  Implied bound: 1
  Clique: 99
  MIR: 15
  GUB cover: 8
  Zero half: 23

Explored 0 nodes (65126 simplex iterations) in 62.61 seconds
Thread count was 4 (of 4 available processors)

Solve interrupted
Best objective 1.731387681159e+02, best bound 1.631005073207e+02, gap 5.7978%

Gurobi 6.0.4 (win64) logging started 07/30/15 18:36:05
```

```
Optimize a model with 67210 rows, 44580 columns and 2530018 nonzeros
Coefficient statistics:
  Matrix range     [1e-02, 3e+02]
  Objective range [3e-01, 1e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+02]
Presolve removed 50917 rows and 15840 columns
Presolve time: 4.20s
Presolved: 16293 rows, 28740 columns, 988247 nonzeros
Variable types: 114 continuous, 28626 integer (28626 binary)
Found heuristic solution: objective 733.0240942

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   14366    1.9418261e+01   1.435839e+05   0.000000e+00     10s
   27480    2.1569001e+01   8.783346e+03   0.000000e+00     20s
   36644    2.2188723e+01   2.128976e+05   0.000000e+00     30s
   46282    2.2265352e+01   3.670737e+03   0.000000e+00     40s
   48591    2.2347470e+01   0.000000e+00   0.000000e+00     43s

Root relaxation: objective 2.234747e+01, 48591 iterations, 38.22 seconds
Total elapsed time = 54.03s
Total elapsed time = 66.35s
Total elapsed time = 73.45s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   22.34747    0 1082  733.02409   22.34747  97.0%     -   83s
H    0     0                       65.4893116   22.34747  65.9%     -   86s
H    0     0                       64.5706522   22.34747  65.4%     -  104s
     0     0   33.65110    0  729   64.57065   33.65110  47.9%     -  127s
H    0     0                       60.6987319   33.65110  44.6%     -  150s
     0     0   34.30798    0  766   60.69873   34.30798  43.5%     -  156s
     0     0   34.54441    0  832   60.69873   34.54441  43.1%     -  182s
     0     0   34.56729    0  771   60.69873   34.56729  43.1%     -  196s
     0     0   34.56729    0  485   60.69873   34.56729  43.1%     -  228s
H    0     0                       53.1862319   34.56729  35.0%     -  240s
H    0     2                       53.0112319   34.56729  34.8%     -  269s
     0     2   34.56729    0  485   53.01123   34.56729  34.8%     -  269s
     1     3   34.73561    1  572   53.01123   34.73561  34.5% 10140  278s
     2     4   34.84633    2  480   53.01123   34.84633  34.3%  5509  280s
     5     6   34.95337    5  392   53.01123   34.95337  34.1%  2376  323s
H   11     7                        52.9278986   34.95337  34.0%  5413  329s
```

```
    13     7   34.95337    8  397    52.92790   34.95337  34.0%  4678  332s
    48    41   34.95337   22  413    52.92790   34.95337  34.0%  1463  349s
    52    44   34.95337   23  404    52.92790   34.95337  34.0%  1363  359s
    72    65   34.95337   28  399    52.92790   34.95337  34.0%  1030  368s
    76    66   34.95337   29  391    52.92790   34.95337  34.0%   983  372s
H  146   138                       52.8445652   34.95337  33.9%   585  375s
   282   272   36.41104   80  390    52.84457   34.95337  33.9%   367  382s
   553   546   41.05270  151  350    52.84457   34.95337  33.9%   257  393s
   769   757   41.09470  199  430    52.84457   34.95337  33.9%   222  402s
   914   902   41.13670  240  405    52.84457   34.95337  33.9%   217  411s
  1046  1028   41.18570  264  295    52.84457   34.95337  33.9%   218  421s
  1374  1355   42.98170  334  134    52.84457   34.95337  33.9%   197  432s
  1877  1839   45.84420  447   18    52.84457   34.95337  33.9%   170  443s
  2142  2094   35.94437   59  485    52.84457   34.95337  33.9%   163  490s
  2144  2095   45.54420  416  840    52.84457   34.95337  33.9%   162  583s
  2145  2096   52.75254  284  732    52.84457   34.95337  33.9%   162  645s
  2146  2097   51.66920  132  766    52.84457   34.95337  33.9%   162  653s
  2147  2097   41.05270  134  753    52.84457   34.95337  33.9%   162  665s
  2148  2098   45.54420  383  807    52.84457   34.95337  33.9%   162  677s
  2149  2099   41.10520  232  511    52.84457   34.95337  33.9%   162  713s
H 2149  1993                       52.8373188   34.95337  33.8%   162  746s
H 2149  1894                       52.8373188   34.95337  33.8%   162  749s
  2151  1897   35.08670   15  412    52.83732   35.08670  33.6%   249  798s
  2154  1897   35.08670   16  413    52.83732   35.08670  33.6%   250  800s
  2163  1912   35.08670   20  392    52.83732   35.08670  33.6%   252  821s
  2203  1935   35.08670   25  372    52.83732   35.08670  33.6%   262  836s
  2299  1989   35.55337   38  411    52.83732   35.08670  33.6%   271  851s
H 2326  1911                       52.7539855   35.08670  33.5%   273  851s
  2435  1995   37.81004   51  387    52.75399   35.08670  33.5%   278  865s
  2599  2096   38.56476   64  424    52.75399   35.08670  33.5%   283  891s
  2984  2282   41.27504   88  313    52.75399   35.08670  33.5%   273  919s
H 3143  2260                       52.7123188   35.08670  33.4%   276  919s
  3227  2232   44.01504  303   23    52.71232   35.08670  33.4%   270  920s
  3565  2546   45.07504  152  138    52.71232   35.08670  33.4%   248  936s
  4564  3133   45.71504  361   35    52.71232   35.08670  33.4%   211  966s
  5579  3698   38.37354   85  400    52.71232   35.08670  33.4%   192  985s
H 5600  3605                       52.7123188   35.08670  33.4%   192  985s
H 6123  3630                       52.6706522   35.08670  33.4%   183  985s
  6558  3974   52.20704  120  385    52.67065   35.08670  33.4%   175 1000s

Cutting planes:
  Gomory: 7
  Cover: 2
  Clique: 74
  MIR: 20
  Flow cover: 13
  GUB cover: 15
  Zero half: 12

Explored 7201 nodes (1373014 simplex iterations) in 1000.16 seconds
Thread count was 4 (of 4 available processors)

Time limit reached
Best objective 5.267065217391e+01, best bound 3.508670289855e+01, gap 33.3847%

Gurobi 6.0.4 (win64) logging started 07/30/15 18:53:10

Optimize a model with 117998 rows, 33324 columns and 3621349 nonzeros
Coefficient statistics:
  Matrix range    [1e-02, 5e+02]
  Objective range [3e-01, 1e+03]
  Bounds range    [1e+00, 1e+00]
  RHS range       [2e-01, 3e+02]
Found heuristic solution: objective 467.393
Presolve removed 103521 rows and 11049 columns
Presolve time: 6.32s
Presolved: 14477 rows, 22275 columns, 894944 nonzeros
Found heuristic solution: objective 338.5926659
Variable types: 110 continuous, 22165 integer (22165 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    8473    9.4241051e+01   5.921382e+03   0.000000e+00     10s
   22573    9.5843709e+01   3.413751e+04   0.000000e+00     20s
   33598    9.6413379e+01   7.471527e+04   0.000000e+00     30s
   45358    9.6546499e+01   4.150227e+03   0.000000e+00     40s
   57052    9.6627473e+01   3.559046e+03   0.000000e+00     50s
   62344    9.6629781e+01   0.000000e+00   0.000000e+00     55s

Root relaxation: objective 9.662978e+01, 62344 iterations, 48.40 seconds
Total elapsed time = 60.58s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   96.62978    0  914  338.59267   96.62978  71.5%     -   63s
H    0     0                     259.0961957   96.62978  62.7%     -   64s
H    0     0                     193.6548913   96.62978  50.1%     -   64s
H    0     0                     114.8152174   96.62978  15.8%     -   87s
```

```
     0     0   97.13396    0 1044  114.81522   97.13396  15.4%     -  100s
H    0     0                    112.1152174   97.13396  13.4%     -  128s
     0     0   97.28285    0 1083  112.11522   97.28285  13.2%     -  140s
     0     0   97.39927    0 1105  112.11522   97.39927  13.1%     -  161s
     0     0   97.40992    0 1091  112.11522   97.40992  13.1%     -  168s
H    0     0                    111.8152174   97.40992  12.9%     -  211s
     0     0   97.43914    0 1031  111.81522   97.43914  12.9%     -  233s
     0     2   97.43914    0 1007  111.81522   97.43914  12.9%     -  308s
     1     3   99.21618    1  973  111.81522   97.44823  12.8% 48610  357s
     3     3  101.63613    2  799  111.81522   97.44823  12.8% 24845  376s
     5     3  104.37728    3  634  111.81522   97.44823  12.8% 18239  389s
     7     4  104.37728    5  552  111.81522   97.44823  12.8% 13158  390s
    48    43  104.82705   20  494  111.81522  104.37728  6.65%  2216  400s
   142   140  105.55505   52  455  111.81522  104.37728  6.65%   940  411s
   220   215  105.55505   71  460  111.81522  104.37728  6.65%   719  421s
   302   297  105.55505   99  462  111.81522  104.37728  6.65%   621  437s
H  303   298                    110.9152174  104.37728  5.89%   619  437s
H  304   299                    110.0228261  104.37728  5.13%   618  437s
   306   303  105.55505  100  444  110.02283  104.37728  5.13%   617  441s
   374   374  109.76505  118  413  110.02283  104.37728  5.13%   573  450s
   495   485  109.81505  156  335  110.02283  104.37728  5.13%   520  464s
H  529   512                    109.8665761  104.37728  5.00%   511  464s
   534   526  109.81505  167  231  109.86658  104.37728  5.00%   509  470s
   588   569  109.81505  179  265  109.86658  104.37728  5.00%   499  483s
   669   640  109.81505  180  272  109.86658  104.37728  5.00%   505  497s
   729   679     cutoff  181       109.86658  104.37728  5.00%   503  507s
   790   733  109.81505  182  314  109.86658  104.37728  5.00%   508  514s
   855   795     cutoff   10       109.86658  104.37728  5.00%   506  521s
   935   856  104.51439   11  540  109.86658  104.37728  5.00%   493  534s
  1043   940  109.76505  140 1031  109.86658  104.37728  5.00%   479  636s
H 1044   893                    109.2353261  104.37728  4.45%   479  705s
  1046   894  109.23533  109 1003  109.23533  104.37728  4.45%   478  775s
  1047   895  105.25505   52 1024  109.23533  104.37728  4.45%   477  838s
  1048   895  109.23533  128 1055  109.23533  104.37728  4.45%   477  872s
H 1048   851                    108.4339674  104.37728  3.74%   477  887s
  1050   853  105.55505   76  977  108.43397  104.37728  3.74%   476  899s
H 1050   810                    108.3402174  104.37728  3.66%   476  931s
  1052   813  105.54255   14  626  108.34022  104.37728  3.66%   704  981s
  1058   815  105.54255   17  526  108.34022  105.54255  2.58%   702  990s
  1064   820  105.93898   19  542  108.34022  105.54255  2.58%   699 1000s

Cutting planes:
  Gomory: 4
  Cover: 5
  Clique: 121
  MIR: 18
  Flow cover: 1
  GUB cover: 8
  Zero half: 38

Explored 1084 nodes (881499 simplex iterations) in 1000.05 seconds
Thread count was 4 (of 4 available processors)

Time limit reached
Best objective 1.083402173913e+02, best bound 1.055425543478e+02, gap 2.5823%

Gurobi 6.0.4 (win64) logging started 07/30/15 19:10:21

Optimize a model with 147100 rows, 22068 columns and 3411503 nonzeros
Coefficient statistics:
  Matrix range    [1e-02, 7e+02]
  Objective range [3e-01, 1e+03]
  Bounds range    [1e+00, 1e+00]
  RHS range       [2e-01, 3e+02]
Found heuristic solution: objective 424.292
Presolve removed 134637 rows and 6400 columns
Presolve time: 5.87s
Presolved: 12463 rows, 15668 columns, 767049 nonzeros
Found heuristic solution: objective 309.5253627
Variable types: 92 continuous, 15576 integer (15576 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   12255    1.5812718e+02   4.815001e+04   0.000000e+00     10s
   31925    1.5970742e+02   9.223177e+03   0.000000e+00     20s
   35020    1.5971972e+02   0.000000e+00   0.000000e+00     22s

Root relaxation: objective 1.597197e+02, 35020 iterations, 16.09 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  159.71972    0  610  309.52536  159.71972  48.4%     -   30s
H    0     0                    221.9840580  159.71972  28.0%     -   31s
H    0     0                    170.6615942  159.71972  6.41%     -   46s
     0     0  160.48036    0  774  170.66159  160.48036  5.97%     -   77s
     0     0  161.74203    0  795  170.66159  161.74203  5.23%     -  112s
H    0     0                    170.5782609  161.74203  5.18%     -  124s
     0     0  161.91972    0  830  170.57826  161.91972  5.08%     -  132s
     0     0  161.97024    0  903  170.57826  161.97024  5.05%     -  139s
     0     0  161.97800    0  659  170.57826  161.97800  5.04%     -  164s
H    0     0                    170.1927536  161.97800  4.83%     -  186s
     0     0  161.97800    0  619  170.19275  161.97800  4.83%     -  221s
H    0     0                    168.9449275  161.97800  4.12%     -  228s
     0     0  161.97800    0  763  168.94493  161.97800  4.12%     -  249s
     0     0  162.11007    0  808  168.94493  162.11007  4.05%     -  263s
     0     0  162.16016    0  820  168.94493  162.16016  4.02%     -  280s
     0     0  162.16537    0  846  168.94493  162.16537  4.01%     -  283s
     0     0  162.16925    0  684  168.94493  162.16925  4.01%     -  295s
H    0     0                    168.6449275  162.16925  3.84%     -  318s
     0     0  162.16925    0  627  168.64493  162.16925  3.84%     -  347s
     0     0  162.16925    0  767  168.64493  162.16925  3.84%     -  365s
     0     0  162.22148    0  813  168.64493  162.22148  3.81%     -  416s
     0     0  162.30662    0  832  168.64493  162.30662  3.76%     -  438s
     0     0  162.32384    0  827  168.64493  162.32384  3.75%     -  446s
     0     0  162.33994    0  683  168.64493  162.33994  3.74%     -  460s
     0     2  162.33994    0  662  168.64493  162.33994  3.74%     -  488s
     2     3  163.69836    1  611  168.64493  163.69836  2.93%  8739  494s
     3     3  164.48159    2  491  168.64493  163.70603  2.93%  8823  500s
    12    15  164.52686    8  427  168.64493  164.49918  2.46%  2635  513s
    27    28  164.53460   13  444  168.64493  164.52153  2.45%  1452  520s
   173   179  164.63119   62  479  168.64493  164.52193  2.44%   484  532s
   345   338  164.91205   98  479  168.64493  164.52193  2.44%   449  546s
H  409   347                    167.3137681  164.52193  1.67%   413  546s
H  508   461                    167.0137681  164.52193  1.49%   407  554s
   588   539  165.17769  112  439  167.01377  164.52193  1.49%   432  561s
H  594   529                    166.4137681  164.52193  1.14%   430  561s
   704   622  165.36961  132  393  166.41377  164.52193  1.14%   474  578s
   774   679  165.49266  145  427  166.41377  164.52193  1.14%   501  588s
   832   711  165.58756  158  431  166.41377  164.52193  1.14%   540  601s
  1036   839  164.97518   81  659  166.41377  164.52193  1.14%   548  678s
  1038   840  165.21892   76  627  166.41377  164.52193  1.14%   547  705s
  1039   841  164.63119   39  777  166.41377  164.52193  1.14%   546  728s
  1040   842  165.86618  126  785  166.41377  164.52193  1.14%   546  755s
  1041   842  164.61330   18  830  166.41377  164.52193  1.14%   545  760s
  1043   844  165.52386   93  689  166.41377  164.52193  1.14%   544  774s
  1044   844  164.82982   14  689  166.41377  164.52193  1.14%   544  791s
  1045   847  164.52193   12  507  166.41377  164.52193  1.14%   693  816s
  1048   847  164.53998   13  442  166.41377  164.52193  1.14%   694  822s
  1057   864  164.61368   17  390  166.41377  164.52193  1.14%   692  836s
  1109   884  164.83330   32  224  166.41377  164.52193  1.14%   708  848s
  1270   975  164.59006   20  510  166.41377  164.52193  1.14%   684  863s
  1517  1105  165.33295   41  510  166.41377  164.52193  1.14%   641  878s
  1659  1192  165.50791   62  432  166.41377  164.52193  1.14%   628  894s
```

```
 1818  1290  166.33355  103  374  166.41377  164.52193  1.14%   624  910s
 1964  1329  166.40141  125  307  166.41377  164.52193  1.14%   635  930s
 2152  1381  164.86091   22  412  166.41377  164.52193  1.14%   629  947s
 2356  1470  164.61040   23  508  166.41377  164.52193  1.14%   609  967s
 2525  1574  164.96977   44  475  166.41377  164.52193  1.14%   608  985s
 2723  1667  165.12803   71  444  166.41377  164.52193  1.14%   602 1000s

Cutting planes:
  Gomory: 9
  Cover: 4
  Clique: 162
  MIR: 26
  GUB cover: 12
  Zero half: 50

Explored 2858 nodes (2132040 simplex iterations) in 1000.05 seconds
Thread count was 4 (of 4 available processors)
Time limit reached
Best objective 1.664137681159e+02, best bound 1.645219341601e+02, gap 1.1368%
```

B2- Computational Results of Full Model

The full model runs for a time limit for 3000 seconds, and arrives at larger gap than the segmented model does for the same problem. This is due to more solutions being present since variables are not fixed as they are in the segmented model as Gurobi progresses through the problem. This also yields a much better lower bound, as can be seen at the end of the solution at 120.3 in the full model but only 164.5 in the segmented model. If both models were allowed to finish the full model would obtain a better solution since it can make changes to all weeks when needed, as opposed to the segmented model which cannot.

```
Gurobi 6.0.4 (win64) logging started 07/31/15 00:00:52

Optimize a model with 194018 rows, 44580 columns and 6855772 nonzeros
Coefficient statistics:
  Matrix range    [1e-02, 7e+02]
  Objective range [3e-01, 1e+03]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 2e+02]
Found heuristic solution: objective 623.652
Presolve removed 165755 rows and 12525 columns (presolve time = 11s) ...
Presolve removed 165755 rows and 12525 columns
Presolve time: 11.27s
Presolved: 28263 rows, 32055 columns, 1734108 nonzeros
Found heuristic solution: objective 478.4952886
Variable types: 141 continuous, 31914 integer (31914 binary)

Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0   -6.5331000e+03   9.800253e+05   0.000000e+00     12s
   12321    1.0061086e+02   1.843730e+05   0.000000e+00     20s
   21007    1.0338512e+02   4.828302e+03   0.000000e+00     30s
   27774    1.0403499e+02   9.074958e+04   0.000000e+00     40s
   34036    1.0403518e+02   2.719124e+05   0.000000e+00     50s
   40399    1.0411472e+02   3.235931e+05   0.000000e+00     60s
   46257    1.0422668e+02   2.365104e+03   0.000000e+00     70s
   51812    1.0427327e+02   2.646165e+03   0.000000e+00     80s
   53179    1.0427352e+02   0.000000e+00   0.000000e+00     83s

Root relaxation: objective 1.042735e+02, 53179 iterations, 71.02 seconds
Total elapsed time = 101.27s
Total elapsed time = 131.79s
Total elapsed time = 156.57s
Total elapsed time = 171.52s
```

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  104.27352    0  967  478.49529  104.27352  78.2%     -  178s
H    0     0                    286.6721014  104.27352  63.6%     -  180s
H    0     0                    185.6260870  104.27352  43.8%     -  277s
     0     0  110.50366    0 1570  185.62609  110.50366  40.5%     -  320s
     0     0  115.76288    0 1628  185.62609  115.76288  37.6%     -  409s
     0     0  117.42651    0 1715  185.62609  117.42651  36.7%     -  531s
H    0     0                    171.0786232  117.42651  31.4%     -  629s
     0     0  118.24564    0 1607  171.07862  118.24564  30.9%     -  673s
     0     0  118.61951    0 1230  171.07862  118.61951  30.7%     -  837s
     0     2  118.61951    0 1230  171.07862  118.61951  30.7%     - 1162s
H    1     3                    169.8952899  118.62139  30.2% 20119 1244s
     2     4  120.60347    2 1389  169.89529  118.62786  30.2% 65071 1417s
     4     4  121.85082    3 1188  169.89529  118.62786  30.2% 37164 1445s
     6     5  123.92240    4 1084  169.89529  118.62786  30.2% 27336 1562s
     8     4  123.92240    5 1037  169.89529  118.77955  30.1% 27797 1646s
    10     8  123.92240    6 1017  169.89529  118.77955  30.1% 22345 1766s
    13    10  124.01948    7  975  169.89529  120.37622  29.1% 24073 1825s
    17    11  124.01948    8  945  169.89529  120.37622  29.1% 20772 1855s
H   25    19                    167.1474638  120.37622  28.0% 14911 1945s
    29    24  124.02133   14  959  167.14746  120.37622  28.0% 13977 2038s
    41    33  124.02480   18 1008  167.14746  120.37622  28.0% 10642 2104s
    63    54  124.02480   19  952  167.14746  120.37622  28.0%  7176 2245s
    71    62  124.02480   21  956  167.14746  120.37622  28.0%  6443 2292s
    75    66  124.02480   22  958  167.14746  120.37622  28.0%  6132 2370s
H  116    82                    166.8474638  120.37622  27.9%  4201 2371s
   137   129  124.07595   38  989  166.84746  120.37622  27.9%  4016 2441s
   172   163  124.13970   40 1017  166.84746  120.37622  27.9%  3600 2525s
   203   194  124.35778   44 1024  166.84746  120.37622  27.9%  3411 2573s
   260   252  124.37296   53 1005  166.84746  120.37622  27.9%  2896 2625s
   332   324  124.37486   94  910  166.84746  120.37622  27.9%  2459 2668s
   396   387  124.37515  120  965  166.84746  120.37622  27.9%  2189 2860s
   432   425  125.00065  123  980  166.84746  120.37622  27.9%  2305 2934s
   530   522  125.78343  154  949  166.84746  120.37622  27.9%  2011 2997s
   610   600  124.73068  115  832  166.84746  120.37622  27.9%  1877 3000s

Cutting planes:
  Gomory: 10
  Cover: 14
  Clique: 207
  MIR: 45
  GUB cover: 31
  Zero half: 51

Explored 611 nodes (1502158 simplex iterations) in 3000.13 seconds
Thread count was 4 (of 4 available processors)
Time limit reached
Best objective 1.668474637681e+02, best bound 1.203762215380e+02, gap 27.8525%
```