The Dissertation Committee for Yumin Deng Certifies that this is the approved version of the following dissertation:

Combining Mathematical Programming and Enhanced GRASP Metaheuristics: An Application to Semiconductor Manufacturing

Committee:

Jonathan F. Bard, Supervisor

J. Wesley Barnes

Leon Lasdon

David P. Morton

Erhan Kutanoglu

Combining Mathematical Programming and Enhanced GRASP Metaheuristics: An Application to Semiconductor Manufacturing

by

Yumin Deng, B. Eng; M. S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

THE UNIVERSITY OF TEXAS AT AUSTIN

December, 2009

This dissertation is dedicated to my family.

Acknowledgement

There are many people who have been invaluable to me throughout my life and especially during the years of my PhD. I wish they all could be here to share this great moment. First, I would like to send my warmest appreciation to my grandmother for everything she has done for me. She raised me and taught me many things to make me become a responsible person. She is the kindest and most caring person that I have ever met, and has suffered much criticism for defending me in difficult times. I really wish she could be here to attend my graduation celebration. I would also like to thank my parents, uncles and aunts for their support of my college studies. I love my little sisters, Yulin, Yunan, Yuling, Jingfang, Chuyi, Xiaoyi and Ruobing (Xiao Bao) as well as my little brothers Jingwen, Jingbo and Jie. Especially for Yulin, I know she has contributed a lot to the family while I have been living in the United States attending graduate school. I really appreciate what she has done and I wish her and my other family members long and happy lives.

I would like to thank my honorable advisor Dr. Jonathan F. Bard for his solid support of my PhD research. He is a successful professor with great achievements and has been an excellent role model for me. He has shared his extensive knowledge with me, inspiring my research ideas and guiding me through my work over the last four years. My experience under his supervision has transformed me from an eager student to a full-fledged researcher. I also want to say thanks to many of my others the professors at UT: Dr. Wesley Barnes, Dr. Leon Lasdon, Dr. David Morton, Dr. John Hasenbein, Dr. Elmira Popova and Dr. Erhan Kutanoglu. They are excellent professors and researchers and have always been helpful and patient with me and my fellow students.

I would also like to take this moment to express my gratitude to Texas Instruments for supporting me while I was doing my dissertation. Dr. Rodolfo Chacon and Dr. John Stuber at TI were invaluable in helping to define my research, providing data, giving me feedback on my reports, and improving my results. I would not be here today without their extensive involvement.

Last but not least, I am heartily grateful to my friends for their understanding and support. We have shared great moments over the last few years and I will always be

there to return their favors. I would like to send my best wishes to them and hope that each will have a prosperous live while fulfilling all their dreams.

Combining Mathematical Programming and Enhanced GRASP Metaheuristics: An Application to Semiconductor Manufacturing

Yumin Deng, PhD

The University of Texas at Austin, 2009

Supervisor: Jonathan F. Bard

Planning and scheduling in semiconductor manufacturing is a difficult problem due to long cycle times, a large number of operational steps, diversified product types, and low-volume high-mix customer demand. This research addresses several problems that arise in the semiconductor industry related to front-end wafer fabrication operations and back-end assembly and test operations. The mathematical models built for these problems turn out to be large-scale mixed integer programs and hard to solve with exact methods. The major contribution of this research is to combine mathematical programming with metaheuristics to find high quality solutions within the time limits imposed by the industrial engineers who oversee the fabrication and test facilities.

In order to reduce the size of problems that arise in practice, it is common to cluster similar product types into groups that reflect their underlying technology. The first part of the research is aimed at developing a greedy randomized adaptive search procedure (GRASP) coupled with path relinking (PR) to solve the capacitated clustering problem. The model is generic and can be applied in many different situations. The objective is to maximize a similarity measure within each cluster such that the sum of the weights associated with the product types does not exceed the cluster capacity in each case. In phase I, both a heaviest weight edge (HWE) algorithm and a constrained minimum cut (CMC) algorithm are used to select seeds for initializing the clusters. Feasible solutions are obtained with the help of a self-adjusting restricted candidate list. In phase II, three neighborhoods are defined and explored using the following strategies:

cyclic neighborhood search, variable neighborhood descent, and randomized variable neighborhood descent (RVND). The best solutions found are stored in an elite pool. In a post-processing step, PR coupled with local search is applied to the pool members to cyclically generate paths between each pair. The elite pool is updated after each iteration and the procedure ends when no further improvement is possible.

After grouping the product types into technologies, a new model is presented for production planning in a high volume fab that uses quarterly commitments to define daily target outputs. Rather than relying on due dates and priority rules to schedule lot starts and move work in process through the shop, the objective is to minimize the sum of the deviations between the target outputs and finished goods inventory. The model takes the form of a large-scale linear program that is intractable for planning horizons beyond a few days. Both Lagrangian relaxation and Benders decomposition were investigated but each proved ineffective. As a consequence, a methodology was developed which was more tailored to the problem's structure. This involved creating weekly subproblems that were myopic but could be solved to optimality within a few minutes, and then post-processing the results with a decomposition algorithm to fully utilize the excessive machine time. The heart of the post-processor consists of a rescheduling algorithm and a dispatching heuristic.

The third part of the research focuses on the combinatorial problem of machine-tooling setup and lot assignments for performing back-end operations. A new model and solution methodology are presented aimed at maximizing the weighted throughput of lots undergoing assembly and test, while ensuring that critical lots are given priority. The problem is formulated as a mixed-integer program and solved again with a GRASP that makes use of linear programming. In phase I of the GRASP, machine-tooling combinations are tentatively fixed and lot assignments are made iteratively to arrive at a feasible solution. This process is repeated many times. In phase II, a novel neighborhood search is performed on a subset of good solutions found in phase I. Using a linear programming-Monte Carlo simulation-based algorithm, new machine-tooling combinations are identified within the neighborhood of the solutions carried over, and improvements are sought by optimizing the corresponding lot assignments.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The process of manufacturing integrated circuits consists of four basic stages: wafer fabrication, wafer probe, assembly or packaging, and final testing. Wafer fabrication is the most technologically complex and capital intensive, involving a precise sequence of processing steps that must be performed in a clean-room environment to reduce the threat of particle contamination (Leachman 2002, Uzoy et al. 1992). Upon completion, functionality tests are performed on the wafers using electrical probes before packaging and final testing (Chiang et al. 2008a). The vast majority of the planning and scheduling research in the industry has focused on wafer fabrication, generally referred to as the *front-end* operation. The remaining three stages comprise the *back-end* operations.

Due to increasing customer requirements and breakthrough advances in technology, the complement of devices produced by a semiconductor manufacturer has increased exponentially over the last decade and often includes hundreds of product types at a single fabrication facility (fab). Developing midterm plans, collecting data and controlling processes for each individual device requires a large amount of resources. Fortunately, similar production routes allow us to categorize many devices by product type and plan for an entire family at a time rather than for its individual members. Nevertheless, during fabrication a device may visit a process many times and must be treated differently each time since processing requirements are a function of the operational step. In addition, the status of a device may change considerably if it fails to pass an inspection step and is sent back to a previous step for rework. After completion, the reworked device is newly categorized to distinguish it from identical devices that were deemed to be defect-free. Similar situation arises when sampling for inspection and test. The sampled devices are considered to be different from the un-sampled devices. Also, there are several situations in which the same devices go through slightly different processing steps.

For management purpose, it is beneficial to group the similar devices into so-called technologies to reduce the size of the planning problem. A technology

corresponds to a set of similar devices which have slightly different characteristics. To perform the analysis, a measure is needed to describe the degree to which two devices are similar. It is desired to cluster the devices in a way such that the total similarity within groups (technologies) is maximized. This defines a clustering problem that we model as a mixed integer program. One of the primary purposes of this dissertation is to develop a heuristic for obtaining high quality solutions in reasonable time to the technology clustering problem. Once the devices are grouped into technologies or families, the inputs, demands as well as the routes of the devices are aggregated by family.

Wafer fabrication is vastly complicated by the reentrant nature of its process flow. Accompanying systems have long and unstable cycle times that make production planning and controlling much more difficult than in discrete parts manufacturing (Glassey and Resende 1988, Van Zant 2000). Factors that contribute to unpredictable cycle times include unreliable equipment, long net processing times, and delays due to batch processing and machine setups. In fabs, the total processing time for each wafer may extend up to three months and include over 1000 operations or steps at hundreds of workstations. When machine tools fail to perform within specifications, production flow may be severely disrupted causing cycle times to increase or fluctuate. Similarly, when wafers fail inspection at different points in their routes they may be scrapped or sent back to an earlier operation for rework with a consequent increase in cycle time.

Because fabs use different fixtures and different material-handling systems for different processes, batch sizes vary by wafer type and process. Therefore, it is not uncommon for one batch of wafers to wait for a second batch of the right size to form at the next operation (Lee and Kim 2002).

Since the early-1990s, there has been a growing effort to model the reentrant nature of semiconductor manufacturing using queueing networks. Dai (1995) and Rybko and Stolyar (1992) established that the stability of a multiclass queueing network is implied by its deterministic fluid counterpart. Motivated by this result, researchers began to focus on finding near-optimal scheduling polices using fluid network analysis. Weiss (1995), for example, developed optimal draining policies for a single job class for different objective functions.

More recently, fluid models have been used to represent relaxations of discrete

scheduling networks. The general approach to the scheduling problem has been to construct a fluid model, find a solution, and then heuristically translate the solution into a discrete scheduling policy. Dai and Weiss (2002) developed a fluid heuristic to minimize the makespan in a job shop while deriving a probabilistic bound for the fluid translation error. Their algorithm fully utilizes the bottleneck machines and paces the remaining machines accordingly. Bertsimas and Sethuraman (2002) solved the same job shop problem using the aforementioned approach. In addition, they applied a fluid synchronization algorithm to translate the optimal solution from the fluid relaxation to a discrete schedule and achieved asymptotically optimal solutions as the number of jobs increased. In related work, Bertsimas et al. (2003) solved the job shop scheduling problem to minimize the holding cost. They applied a revised version of the fluid synchronization algorithm to derive discrete solutions, which were similarly proven to be asymptotically optimal.

Although disruptions and uncertainty can play havoc with a schedule, it is still necessary to develop long-term and midterm plans. As part of this dissertation a new model is presented that can be used to plan daily operations in a fab for up to three months at a time by taking into account expected demand, predefined starts, detailed routings, and machine capacity limits. The work was undertaken in collaboration with the Texas Instruments' (TI) 300-mm facility in Dallas, Texas, referred to as DMOS6. Recognizing that a variety of objectives are used in industry to guide production, such as minimizing cycle time, minimizing late orders, or minimizing work in process (WIP), the objective chosen for our project was the minimization of the total deviations from target outputs. This is equivalent to minimizing the sum of the daily deviations from the forecast demand for each product or device in the system over the planning horizon (cf. Fordyce et al. 1992). The principal decision variables in the model are the WIP level and machine time assignment for each device at each step in each time period.

Admittedly, our approach is somewhat at odds with standard practice. As Leachman et al. (2002) point out, most semiconductor companies manage production under the lot-dispatching paradigm (managing the cycle times of production lots). In that approach, priority rules such as the critical-ratio rule or the least-slack rule are used to schedule lots at each workstation. In their SLIM methodology, Lechman et al. rely on a

target fab-out schedule for each device that is continuous in time, rather than focusing on lot due dates. For example, if the schedules are expressed in terms of output quantities per day, then one quarter of the quantity of a particular device is due six hours into that day. The primary scheduling objects in SLIM are device-step combinations instead of individual lots. This is the approach that we take.

Because our interest is in midterm planning rather than in scheduling or sequencing, we were able to significantly reduce the size of our problem without sacrificing accuracy by aggregating devices by technology to create representative families. For similar reasons, it was not necessary to consider the discrete nature of the real-time decision process. Nevertheless, the resultant model was still not tractable for planning horizons greater than a few days so several different decomposition schemes were explored. A major contribution of this research centers on the use of linear programming to obtain initial solutions and the application of a decomposition algorithm to arrive at the final production plans. For a comprehensive review of production planning models for wafer fabrication, see Asmundsson et al. (2006), Leachman (2002) and Lin (1999).

Most of the back-end operations are scheduled manually with database support to keep track of lot status. However, the importance of these operations in meeting customer due dates has sparked an interest in applying more sophisticated analytic techniques at the shop floor level. Part of our work is aimed at improving the efficiency of machine setup and lot processing during assembly and test (AT). The third major contribution of this dissertation is the presentation of a new model and solution methodology developed in conjunction with Texas Instruments in support of their AT facilities.

In the most general sense, the problem in such facilities has the basic characteristics of a job shop with multiple complexities including dynamic job arrivals, machine unavailability, sequence-dependent setup times, alternative machine assignment options, and batch-type processing. Since the classic job-shop problem is already NP hard (Pinedo 2008), operating AT facilities presents an unusually difficult challenge to line supervisors, especially when 1000s of lots have to be scheduled each day on 100s of machines with many different tooling and temperature requirements. In the past, management has primarily concentrated on cycle time-based objectives since production

lots were not typically related to a particular order in the prevailing make-to-stock environment. However, with Application Specific Integrated Circuit (ASIC) and other specialty processors now a major portion of the market and hence production volume, the ability to meet due dates has become critical for profitability (Chiang et al. 2008b). Because AT operations are closer to the customer, due date-based performance measures are more appropriate than cycle time and WIP level objectives. In addition, design creativity and an explosion of products now means that a given die can be packaged in many different ways and can have different test specifications associated with it, making the problem that much more difficult.

High investment at the back-end, pressure to provide good customer service, and tight coupling of AT operations with the front-end manufacturing is driving the need for more effective planning tools. In support of this need, we have developed a mathematical model with the joint objective of maximizing the weighted sum of lots processed and minimizing the weighted shortages of critical devices over the planning horizon. Tooling considerations and capacity constraints are the predominant factors that limit output. For a given set of lots, the available machines have to be set up with the proper tooling to operate at the appropriate temperature. The model takes the form of a large-scale mixed-integer program, which is solvable with any of the leading commercial codes when the number of lots and tooling-temperature combinations is relatively small. For more realistic instances, we devised a solution methodology based on two-level decomposition that first sets up the machines and then assigns the lots. Using a GRASP (Feo and Resende 1995), a set of high quality feasible solutions is obtained in phase I by repeatedly applying the decomposition strategy to randomly selected machine-tooling configurations. This is followed in phase II by a novel linear programming-Monte Carlo-based neighborhood search scheme that makes use of local branching ideas (Fischetti and Lodi 2003) to improve the results. The decomposition strategy is also used in phase II.

One of the primary interests of this dissertation is to combine both mathematical programming the metaheuristics to solve the practical problem arising from semiconductor industries efficiently. Thus it is helpful to investigate the advantages and disadvantages of mathematical programming and metaheuristics. A literature review is provided here for the emerging research area named matheuristics, which intelligently

combine the two to enhance the overall algorithm efficiency.

Mathematical programming has been achieving great success in solving a wide range of problems involving production planning and scheduling, transportation and vehicle routing, logistics and supply chain optimization, and statistical data analysis, etc. Generally a mathematical model is built at first to represent the problem under investigation. In some cases the model can be further enhanced by tighten some of the constraints with techniques like lifting. Solution methodologies are then proposed based on the analysis of the mathematical model. If the size of the problem is small then a basic Branch and Bound (B&B) may be capable to solve the problem to optimality. For larger instances, it is necessary and beneficial to use intelligent techniques to solve the problems. Such techniques in literature usually involve Branch and Cut (B&C), Branch and Price (B&P), Benders decomposition and Lagrangian Relaxation (LR), etc. These techniques involve a master problem (MP) and a subproblem (SP), i.e., separation problem in B&C, pricing problem in B&P, dual problem in Benders decomposition, and subproblems in LR when some of the constraints are relaxed. Solving the subproblems efficiently is usually critical to the overall performance of the algorithm.

In mathematical programming, both upper bound and lower bound are usually provided to indicate the gap between the current solution and the optimal solution. The algorithm can be terminated earlier for a satisfactory tolerance. The quality of the solution obtained can be asserted. In the aforementioned techniques, usually the subproblem can be further decomposed into even smaller size problems which can be solved efficiently. In some cases the decomposed problems are identical (e.g. identical vehicles in the pricing problem when B&P is applied to vehicle routing problem) thus it is not necessary to solve the pricing problem multiple times. However, there are still disadvantages for mathematical programming. The practical problems are usually very complicated and large scale. It can be either very hard to build a mathematical model for these problems, or a large amount of decision variables and constraints are necessary to represent even a medium size problem. The situation becomes more and more difficult to mathematical programming when the size of the problem increases. Furthermore, the subproblem can easily become very hard to solve when the aimed problem is complicated with additional requirements.

Metaheuristics, on the other hand, have been accepting increasing attention from 1980s. Instead of building a mathematical model for the problem at hand, metaheuristics focus on the problem itself without using decision variables and constraints. Various heuristic procedure have been proposed in literature involving GRASP, Tabu Search (TS), Scatter Search (SS), Simulated Annealing (SA), Genetic Algorithm (GA), etc. There are still many other procedures which are essentially the extension and combination of the aforementioned heuristics. The idea of metaheuristics is much diversified. For example In GRASP, feasible solutions are usually constructed in phase I and improved in phase II iteratively. In TS, some of the movements are considered to be tabu to overcome local optima. A tabu list is maintained in TS which is essentially memory related. Such heuristic idea is not rigid and can always be extended for further enhancement. For example, GRASP can be extended to GRAMP when historical information is taken into account to guide the construction of phase I solution. Although the metaheuristics are diversified, they do share something in common, that is, local search. In metaheuristics, local search is an important component which achieves local optimum within the neighborhood specified. Usually a large amount of computational effort is devoted to local search in most of the metaheuristics.

Metaheuristics have many advantages compared to mathematical programming. The procedures are usually relatively easier to implement. High quality solutions can be obtained in reasonable time even for large scale problems that cannot be solved by pure mathematical programming. The procedure can also be enhanced easily, e.g., by introducing a new neighborhood search. In fact, for many practical problems those need to be solved in a specified amount of time, metaheuristics are usually a better option for its effectiveness. However, there are still some downsides in light of its usefulness. Since metaheuristics are problem oriented, the neighborhood definition needs to be changed if some additional requirements (constraints) are added to the problem. The local search procedure needs to be updated as well since some of the previous local movement may not be feasible any more. The solutions from metaheuristics do provide a feasibility bound on the optimal solution but the gap is still unknown. It is usually very hard to assert the quality of the best found solution other than numerically. Furthermore, the efficiency of metaheuristics highly relies on the performance of local search. The

performance of the overall algorithm can be harmed greatly if the local search step is difficult and time consuming.

In light of the advantages and disadvantages of both mathematical programming and metaheuristics, a new research area focusing on combining the two approaches is emerging and thus gains its name as matheuristics. There are a limited number of publications in literature since matheuristics is still in its infancy. However, if we look into matheuristics in detail, it can be found that the idea has been adopted in many aspects both in mathematical programming and metaheuristics. Matheuristics can be mainly divided to two categories. The first type of matheuristics follows a general mathematical programming framework. The metaheuristics serve as an efficient component to solve the embedded subproblems. Conversely, the second type of matheuristics follows the paradigm of metaheuristics, and applies mathematical programming inside the heuristic framework as a problem solver. Both these two approaches involve mathematical programming and metaheuristics. Some other more intrinsic procedure may switch between the two iteratively.

The application of metaheuristics inside a mathematical programming framework can be found in many publications. In the B&P framework, usually initial columns are generated by constructing a high quality solution heuristically and metaheuristics are applied to solve the subproblems for column generation, e.g., see Bard and Rojanasoonthon (2006), Purnomo and Bard (2007) and Bard and Purnomo (2005). In the B&C framework, the separation problems are usually solved by metaheuristics to generate cuts, e.g., see Bard et al. (2002) and Bard et al. (1998).

Metaheuristics can also be applied within a LR and Benders decomposition framework. Bard and Purnomo (2007) provided a way to combine LR and heuristic idea. Recently, Boschetti and Maniezzo (2009) investigate such application and propose a general procedure for implementation. In the LR framework, they suggest to apply metaheuristics when the Lagrangian multipliers are updated. If feasible solutions are obtained, a feasibility bound can be provided. In the Benders decomposition, the authors suggest the application of metaheuristics in two places. Metaheuristics can be applied to solve the master problem to provide feasibility bound. At the same time the feasible solution obtained is delivered to the subproblem to generate additional cuts that going to

be appended to the master problem. If the subproblem is also a mixed integer programming, metaheuristics can then again be applied to solve the problem with an attempt to improve the feasibility bound. The authors have applied their methodologies to the single source capacitated facility location problem, the membership overlay problem and the multi-mode project scheduling problem. Their approaches are quite promising according to the reported numerical results.

One the other hand, some researchers focus their interests on applying mathematical programming inside a metaheuristic paradigm. Fischetti and Luzzi (2009) illustrated a way to embed mathematical programming into a heuristic framework to enhance the overall algorithm performance. They presented an MIP model for the nesting problem, which places two-dimensional polygon into a rectangular container without overlapping. The objective is to maximize the usage of the rectangular container. The model was further enhanced by lifting the constraints and embedding a specialized branching strategy accommodating the problem. Since the initial numerical experiments were not encouraging, a heuristic was then developed to place the big pieces of polygons at first. A MIP model is built and then simplified to place the small pieces of trims into the holes of the container, which was the so-called multiple containment problem. It turned out that the simplified model can be easily solved as an MIP. A post-procedure is then invoked to remove any overlapping area for a feasible solution. The solutions from the proposed algorithm were quite promising comparing to the greedy heuristic. An average percentage improvement around 1~2% over the greedy heuristic was achieved by the presented method.

Hu et al. (2008) provide another way to combine the metaheuristics and mathematical programming for the generalized minimum spanning tree problem. They developed a variable neighborhood search (VNS) framework which sequentially performs local search within the neighborhoods. The first two neighborhoods are generated through either nodes exchange or edges exchange. For the third neighborhood the authors apply MIP to optimize local parts within candidate solution trees. The proposed approach has been tested for random instances with up to 1280 nodes. The performance is quite encouraging according to the solutions obtained.

In the next chapter, semiconductor manufacturing operations are outlined for both front-end (wafer fabrication) and back-end (assembly and test) operations. Chapter 3 provides a literature review for the clustering problem followed by a discussion of the reactive GRASP developed to find solutions. The midterm fab planning problem is discussed in Chapter 4 where a mathematical model is presented. After trying Lagrangian relaxation and Benders decomposition algorithms without success, a problem-specific decomposition approach is provided. Chapter 5 focuses on the back-end AT operations. A two level hierarchical approach coupled with a GRASP is used to find solutions. Numerical results are provided for Chapters 3, 4 and 5. In Chapter 6, an assessment of the research is given and some conclusions are drawn from the computational experience.

Chapter 2

Outline of Semiconductor Manufacturing Operations

2.1 Wafer Fabrication (Front-end Operations)

Wafer fabrication begins with a smooth (typically silicon) wafer of a certain diameter upon which thousands of integrated circuits are layered through successive operations. Circuits contain between 100 and 50,000 chips. During processing, a wafer goes through the following six primary steps multiple times: deposition, photolithography, etching, ion implantation, photoresist strip, and inspection and measurement. The *routing* (process flow) determines the actual path of a lot through the system. For more detail, see, e.g., Uzsoy et al. (1992).

Planning Challenges. The focus of management in the semiconductor industry is on minimizing production costs and increasing productivity while improving both quality and delivery time performance. Major factors affecting cost are yield, labor, materials, inventory, equipment and facility depreciation, and the number of starts per week (Bai and Gershwin 1994, Hughes and Shott 1986). Because of the huge initial investment required to construct a fab, the goal is to keep it loaded at all times. Thus, the driving force to date has been the manufacture of standard products in fairly high volumes. In such operations, it is common to create a buffer against fluctuations in external demand by holding inventories of probed die, referred to as die-bank inventories, between the front-end and back-end operations. Hence wafer fabs have tended to operate in a make-to stock mode, with production lots rarely being associated with a specific customer order or due date. Together with the high capital costs of equipment, this has resulted in a major emphasis on maintaining high throughput and equipment utilization, while reducing both the mean and the variance of cycle times and inventories. This is the situation at the TI fab.

Reentrant Flow. It is instructive to view a fab as a time-dependent multicommodity network where each node corresponds to a buffer in front of a machine group. A separate

buffer, denoted by $W_{mli}(t)$, can be defined for each product i that is active at time t and whose next operation at machine group m is to be performed at level l. The "level" index refers to the visit number of a product to a machine group during the reentrant flow. The path taken by product i during fabrication is determined by its routing, which is part of the input data. The number of buffers associated with a product is equal to the number of steps that are required for its completion. Figure 2.1 illustrates a reentrant line with 3 machine groups and 11 buffers for product 1, which enters the system at buffer W_{111}. Finished products emerge from machine group 3 after their third visit following a wait in W_{331}. Every wafer in the line visits machine group 1 three times, machine group 2 five times, and machine group 3 three times according to the deterministic routing $1 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow 3$.

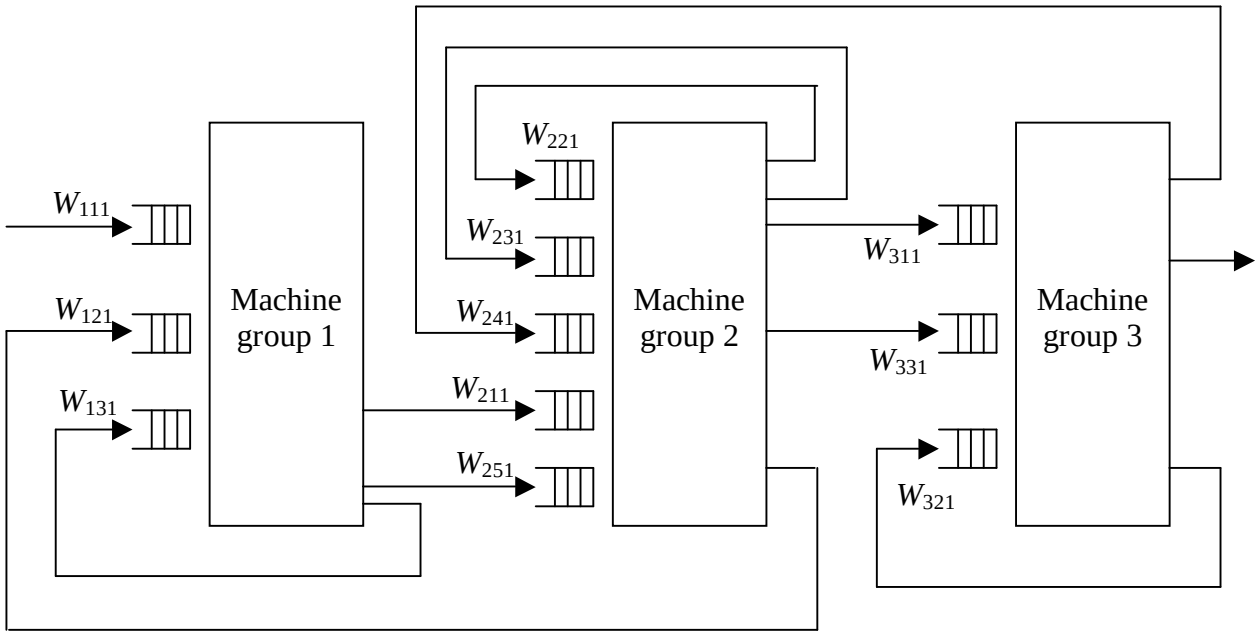

Figure 2.1 Example of a reentrant production line

2.2 Product Assembly and Testing (Back-end Operations)

As yield efficiencies in front-end operations have increased, the need for improving the ability of back-end facilities to handle large volumes of product has increased accordingly. AT outsourcing services represent a growing contribution to total industry revenue. During assembly, integrated circuits are placed in plastic or ceramic packages that protect them from the environment. Examples include dual in-line packages,

leadless chip carriers, and pin-grid arrays. Since it is possible for a given circuit to be packaged in many different ways, there is an explosion of product types at this stage. Once the leads have been attached and the package sealed and tested for leaks and other defects, the product is sent to final test. At that stage automated equipment is used to interrogate each integrated circuit and determine whether it meets the required specifications. The goal is to ensure that customers receive a defect free product.

Figure 2.2 depicts the major steps in back-end operations, which may include anywhere from 20 to 40 processes (Van Zant 2000). Packaged chips are advanced through some or all of these processes before being turned out as finished goods and either shipped to customers or placed in inventory. Because products differ in terms of dimensions, consumables, and process specifications, the process flows differ from product to product.

From Figure 2.2 we see that the test component collectively includes burn-in, electrical testing, marking/branding, baking, programming, mechanical scanning, quality check and packaging, in this order (Freed et al. 2006, Ovacik and Uzsoy 1996). The test-floor can be described as a flexible flow-shop (i.e., the sequence of processing operations is fixed), each lot requires a unique subset of the operations (burn-in, marking, baking, and programming may or may not be required), and multiple machines may be eligible for each operation. In some cases, these machines may not be identical with respect to processing rates or output quality, so there may be lot assignment preferences among the set of eligible machines. Yield and lead-time variability in previous stages of the manufacturing process (i.e., wafer fabrication and probe) result in variable lot sizes and lot priorities at the AT stages. Lot priorities range from low when ample inventory exists, to 'hot' or critical when promise dates are near or orders are past due.

An additional concern is machine failures, which are common and unpredictable despite a heavy emphasis on preventive maintenance. Failures are most likely to occur during a changeover between lots that have noticeably different tooling and temperature requirements. Further complicating the matter is the fact that changeover durations are variable, significant (same order of magnitude as lot processing times), and sequence-dependent (Freed et al. 2006). When planning a changeover, the skill level of the available workers must also be taken into account. Personnel costs can be substantial due

to the need for extensive training, while in some locations, labor shortages bid up wages. In general, the availability of skilled labor constrains throughput. When a single worker is assigned to operate multiple machines, competing demand for his or her services may lead to lost capacity.

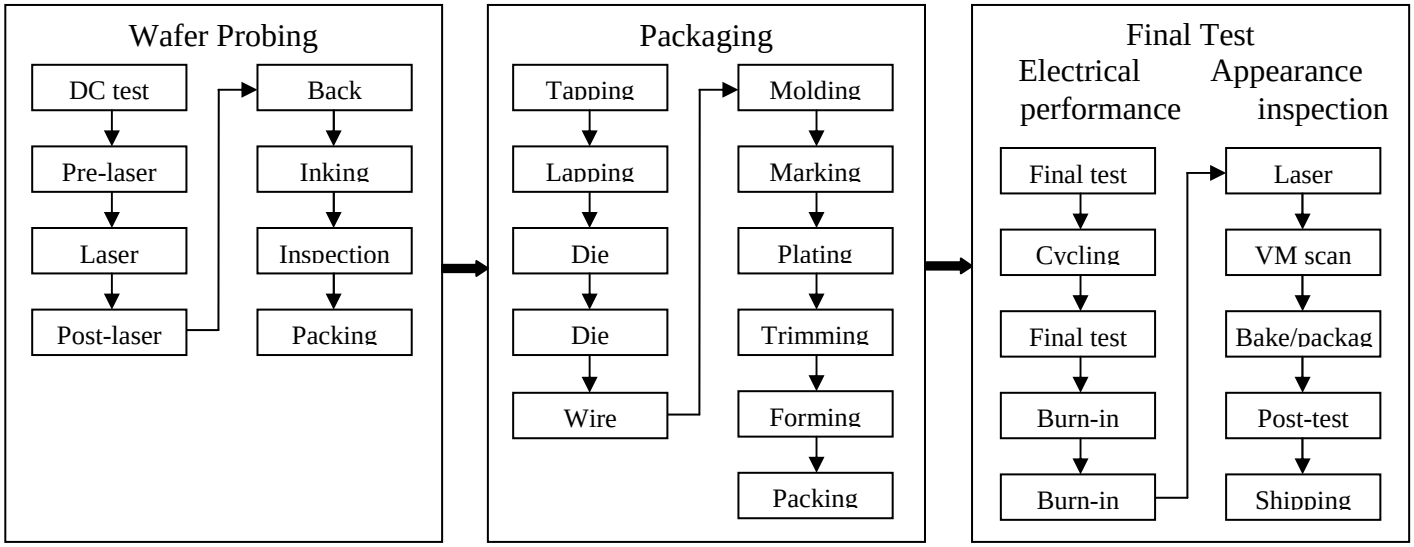

Figure 2.2 High-level back-end process flow

In contrast to the voluminous literature on wafer fabrication, there has been little research on AT operations. Knutson et al. (1999) investigated a problem in which lots in an AT facility were formed to match the size of customer orders. The authors assumed that all lots consisted of the same type of chip and that yield losses were zero. The planning horizon was set to one day and any delivery tardiness or over supply was treated as a penalty. The problem was formulated as a nonlinear integer program with three objectives: maximize the satisfaction of customer demand, minimize the number of die (chips) sent to the warehouse, and minimize delivery tardiness. To find solutions, the model was decomposed into two stages. The first stage took the form of in a knapsack problem with the bi-objective of maximizing facility utilization and minimizing order tardiness. The second stage took the form of a bin (orders) covering problem aimed at minimizing the number of chips sent to the warehouse.

Song et al. (2007) applied ant colony optimization to reduce the conversion time of a bottleneck machine during assembly and test. Three objectives were investigated:

minimization of unfilled customer demand, minimization of total number of machine conversions, and minimization of total conversion time. The authors first constructed a unidirectional graph to represent the machine scheduling problem over the planning horizon. Each node was a triplet describing the machine status in the current time interval. Nodes between adjacent time intervals were connected with edges weighted by transition probabilities. A path from origin to destination represented a valid machine schedule. Hard constraints were addressed by assigning zero probability to some edges. This eliminated an undesirable machine conversion. Soft constraints were addressed by penalizing violations and transition probabilities were updated each time the ants finished their searching. The algorithm was successfully applied at an Intel AT facility and achieved conversion time reductions of up to 20% compared to the manual approach then being used.

Zhang et al. (2007) proposed a two-level hierarchical capacity planning framework to reconfigure kit components in AT operations. The first level focused on midterm planning while the second level created executable plans for individual facilities. The authors also proposed a mixed-integer linear programming model for the first level problem. The methodology was successfully applied at one of Intel's AT sites, resulting in an annual $10 million saving in the purchase of kit components.

Chapter 3

Capacitated Clustering

Clustering primarily involves the partition of objects or data points into different groups to optimize some weighted measure of distance between them. A large variety of applications exist in such areas as manufacturing, network design, pattern recognition, mail delivery, habitat classification, facility location, and statistical data analysis, to name the most prominent (e.g., see Al-Sultan and Khan1996, Bard and Jarrah 2009, Daganzo 2005, Kaufman and Roussweuw 1990, Laporte et al. 1989). In some of these applications, the number of clusters is given while in others the objective is to find the minimum number that satisfies a set of knapsack-type constraints. In this chapter, we address the constrained version of the problem and present a greedy randomized adaptive search procedure (GRASP) to find solutions. Such procedures generally have a construction phase and an improvement phase (Kontoravdis and Bard 1995, Rojanasoonthon and Bard 2005). In developing the methodology, we included several options for each of these phases that markedly improved overall performance. For phase I, we designed both a heaviest weight edge algorithm and a constrained minimum cut scheme for constructing feasible solutions. For phase II, we explored the use of cyclic neighborhood search, variable neighborhood descent (VND) (Hansen and Mladenovic 1997, Hu et al. 2008), and a randomized version of the latter, to achieve local optimality. In the final step, path relinking (Glover et al. 2000) was performed on the top candidates to see if any better solutions could be uncovered on the paths between them. The design and integration of these features within a GRASP framework represents the major contribution of this research.

Although the solution methodology can be directly applied to create families of semiconductor devices, the specific motivation arose from our collaboration with facility planners at mail processing and distribution centers within the U.S. Postal Service (USPS). One of their recurrent tasks is to design zones to help rationalize the bulk movement of mail by powered industrial vehicles (PIVs). Over the course of the day, PIVs are used to transfer mail to and from the docks and between the various workcenters.

Each workcenter performs a specific operation such as canceling stamps, barcoding envelops, and sorting letters to carrier routes. The pickup and drop off locations are called control points and can be regarded as fixed nodes in two-dimensional network. The problem of specifying the zones can be formulated as a mixed integer program. The difficulty in finding optimal solutions stems from its combinatorial nature. In our initial testing, we were unable to achieve convergence with CPLEX 11.0 for instances with more than 40 nodes. Therefore, we took a heuristic approach.

The construction of PIV zones falls into the general area of capacitated clustering, which is further discussed in the next section along with the related literature. In Section 3, the mathematical formulation of the problem is given followed by our solution methodology which includes a reactive GRASP, two initiation procedures, our enhanced neighborhood search techniques, and path relinking. In each case, the algorithm is described and a pseudocode is given. To test the methodology, we randomly generated a large number of instances using data provided by the USPS. The results show that high quality solutions can be obtained for these instances, as well as for those solved by Mehrotra and Trick (1998). An assessment of the overall approach is presented in Section 6.

Various versions of the clustering problem have been extensively studied since the 1960s, with virtually all of them being NP-hard in the strong sense (Brucker 1978). Mulvey and Beck (1984) proposed one of the first models for what has become known as the capacity clustering problem (CCP). In the original formulation, the objective was to find up to p capacitated clusters centered at a to-be-determined median such that the collective dissimilarity between each customer and its median is minimized. Their context was sales force territory design. In formulating the CCP, let $y_{ik} = 1$ if data point i is in cluster k and 0 otherwise, and let $z_k = 1$ if data point k is the median of cluster k and 0 otherwise ($i = 1,\ldots,n$; $k = 1,\ldots,n$). The basic model is

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{k=1}^{n} c_{ik} y_{ik} \tag{1a}$$

$$\text{subject to} \quad \sum_{k=1}^{n} y_{ik} = 1, \quad i = 1,\ldots,n \tag{1b}$$

$$\sum_{i=1}^{n} w_i y_{ik} \le C_k z_k \ , \ k = 1,\ldots,n \tag{1c}$$

$$\sum_{k=1}^{n} z_k \le p \tag{1d}$$

$$y_{ik} \in \{0,1\},\ z_k \in \{0,1\},\ i = 1,\ldots,n;\ k = 1,\ldots,\ n \tag{1e}$$

where w_i is the service demand of customer i, C_k is the capacity of cluster k, $p \ge \left\lceil \sum_{i=1}^{n} w_i \Big/ \left(\frac{1}{n} \sum_{k=1}^{n} C_k \right) \right\rceil$ is the number of clusters, and $c_{ik} = \left(\sum_{l=1}^{s} \left(a_{il} - a_{kl} \right)^2 \right)^{1/2}$ is the dissimilarity measure between i and its median k. In the expression for c_{ik}, the vector $\mathbf{a}_i \equiv (a_{i1},\ldots,a_{is})$ represents the s attributes associated with data point i (or median k when appropriate). When points on a plane are being clustered, $\mathbf{a}_i$ is the two-dimensional vector of their X- and Y-coordinates and c_{ik} is the Euclidean norm. Model (1) is known as the p-median capacitated clustering problem (p-CCP) when C_k is homogeneous (Ahmadi and Osman 2005, Lorena and Senne 2004).

The objective function (1a) in effect minimizes the sum of the "distance" between each pair of data points in a cluster. In the formulation, all n data points are candidates for one of the p medians. Constraints (1b) ensure that each data point is assigned to exactly one cluster, and constraints (1c) limit the demand of each cluster k to its capacity C_k. Constraint (1d) restricts the number of clusters created to p and is written as an inequality because it may not be economical to use the full capacity of the system. Logical restrictions are placed on the variables in (1e). If the redundant constraints $y_{ik} \le z_k$, (i, $k = 1,\ldots,n$) are added to the model, then a stronger relaxed formulation is obtained. In that case, when y_{ik} is integral z_k will be integral as well so the binary restriction on those variables can be replaced by $z_k \in [0,1]$, $k = 1,\ldots,\ n$.

A variant of model (1) known as the p-centered capacitated clustering problem (p-CCCP) arises when the median is replaced by the centroid (Negreiros and Palhano 2006). This results in a nonlinear objective function because the dissimilarity weight is now $c_{ik} = \left\| \mathbf{a}_i - \boldsymbol{\zeta}_k \right\|^2$, where $\boldsymbol{\zeta}_k \in \Re^s$ is a free variable that locates the geometric center of the cluster. In either case, a wide variety of solution strategies and techniques have been developed,

from neural networks and genetic algorithms, to fuzzy sets, GRASP, and alternative *c*-means; e.g., see Chiou and Lan (2001), and Osman and Ahmadi (2007).

Cano et al. (2002) proposed a GRASP to solve the *p*-centroid uncapacitated clustering problem. Since the performance of GRASP is affected by the quality of the partial initial solution, their first step was to generate good seed candidates, which is also our first step. They then applied a probabilistic greedy Kaufman initialization in the construction phase (Kaufman and Roussweuw 1990). The Kaufman procedure identifies *p* dispersed points as the cluster centroids. In the improvement phase, the *k*-means method was used for local search. Testing was done on eight real-world benchmark data sets, the largest involving 2310 data points, 19 attributes and 7 clusters. The results showed that Kaufman-based procedure outperformed its counterparts such as random selection, Forgy's method and MacQueen's method [for a discussion of the aforementioned methods, see Hansen and Mladenovic (2001) and Kaufman and Roussweuw (1990)].

Ahmadi and Osman (2005) combined GRASP and adaptive memory programming to solve the *p*-CCCP. The possible centers were ranked and placed on a fixed length restricted candidate list (RCL). At each phase I iteration, one was selected randomly using a probability measure that was updated to reflect the performance of the elite (improving) solutions. The updating procedure was aimed at balancing the so-called density and intensity of the centers. A similar idea is applied in this chapter except that we use the probability measure to control the RCL length rather than to select nodes in phase I (cf. Prais and Ribeiro 1999). In the improvement phase, the authors applied a restricted 1-interchange. Intensification, diversification and aspiration were also considered by setting criteria to determine whether an improved solution should be placed in the elite solution pool. Five randomly generated data sets were used to test the algorithm. The largest instances contained 150 data points and 15 clusters.

Mehrotra and Trick (1998) used column generation and a specialized bounding technique to solve the maximization version of CCP. Their pricing subproblem took the form of what they called a *maximum weight cluster problem*; a tight upper bound was obtained by solving a transportation problem. Branching was governed by the Ryan-Foster rule but it was often unnecessary to go beyond the root node due to the integrality

of the linear programming solution. Testing was done on a DEC Alpha Model 300 using the same data sets as Johnson et al. (1993) who investigated a compiler design problem. The largest instance solved contained 61 nodes and 187 edges, and consumed 352 sec for a right-hand-side value in (1c) of $C_k = 450$ for all k and 394 sec for $C_k = 512$.

Barreto et al. (2006) used a sequential heuristic to find solutions to the capacitated location routing problem, a combination of a facility location problem and a capacitated vehicle routing problem. The proposed algorithm first solves a clustering problem to group the customers (nodes), and then a vehicle routing problem (VRP) to obtain the routes that were subsequently improved by local search. Two kinds of algorithms (hierarchical and non-hierarchical) and six proximity metrics (single linkage, complete linkage, group average, centroid measure, ward measure, saving measure) were proposed and tested for the clustering problem. Optimality gaps of less than 5% were obtained, on average, for instances as large as 318 customers and 4 distribution centers, 150 customers and 10 distribution centers, and 117 customers and 14 distribution centers.

In the development of algorithms for the VRP, it is common to follow the logic of cluster first, route second. Newell and Daganzo (1986) approached the capacitated VRP with a single depot by grouping the customers (nodes) into zones and visiting the nodes within zones in order of their longitude coordinates. For the problems studied, the nodes were distributed randomly with a density function δ and the zones were constructed as wedge-shaped sectors elongated toward the depot. The overall objective was to minimize the expected total travel distance.

Ouyang (2007) extended the work of Newell and Daganzo (1986) by developing a systematic approach to obtain an optimal zone design. The problem studied focused on the construction of vehicle routing zones (VRZ) for given shape and size requirements, as described by Newell and Daganzo. Initially, a set of wedge-shaped zones was created satisfying these requirements. The wedges were then conformally mapped into square zones and a disk model was applied to obtain an approximately optimal partition. Further refinements were carried out by the weighted centroidal Voronoi tessellation algorithm to balance the delivery loads within the zones. From the reported computations, the proposed methodology was seen to outperform an adaptation of the Clarke-Wright heuristic with significant advantage being evidenced for large instances.

3.1 Mathematical Formulation

For a given set of nodes V and connecting edges E, we wish to partition V into p clusters such that the sum of the "benefits" associated with the edges within each cluster is maximized and the sum of the node weights in each cluster falls with the interval $[C^{\min}, C^{\max}]$. For the PIV application with n control points, the problem can be modeled on a graph $G = (V, E)$, where $i \in V$ must appear in exactly one cluster and edge $e = (i,j) \in E$ exists in G only if there is some flow between its endpoints i and j over the week. In creating the model, we make use of the following notation.

Indices and sets

- k index for clusters
- i,j indices for nodes; $i,j \in V$
- e index for edges in G; $e \in E$

Parameters

- c_e weight of edge $e \in G$; $c_e \equiv c_{ij}$, where $i, j \in V$ such that $(i,j) = e \in E$
- w_i weight of node $i \in G$
- p number of clusters to be created
- $C^{\max}$ maximum permitted weight of nodes in each cluster
- $C^{\min}$ minimum required weight of nodes in each cluster

Variables

- x_{ek} 1 if edge e has both its endpoints in cluster k, 0 otherwise
- y_{ik} 1 if node i is included in cluster k, 0 otherwise

Model

$$\phi_{\text{IP}} = \text{Maximize} \sum_{k=1}^{p} \sum_{e \in E} c_e x_{ek} \tag{2a}$$

$$\text{subject to} \sum_{k=1}^{p} y_{ik} = 1, \ \forall\, i \in V \tag{2b}$$

$$x_{ek} \le y_{ik},\ x_{ek} \le y_{jk},\ \forall\, e = (i,j) \in E,\ k = 1,\ldots,p \tag{2c}$$

$$x_{ek} \ge y_{ik} + y_{jk} - 1,\ \forall\, e = (i,j) \in E,\ k = 1,\ldots,p \tag{2d}$$

$$C^{\min} \leq \sum_{i \in V} w_i y_{ik} \leq C^{\max}, \ \forall\ k = 1,\ldots,p \qquad (2e)$$

$$x_{ek} \in \{0,1\},\ y_{ik} \in \{0,1\},\ \ \forall\ i \in V,\ \ e = (i,j) \in E,\ \ k = 1,\ldots,p \qquad (2f)$$

The objective in (2a) is to maximize the sum of the edge weights within clusters, which is equivalent to minimizing the sum of the weights of edges between clusters. If the endpoints of edge e are not in the same cluster, then the corresponding weight c_e is not counted. Constraints (2b) ensure that each node i is included in exactly one cluster, while constraints (2c) and (2d) specify that edge $e = (i, j)$ is in cluster k if and only if both endpoints i and j are in cluster k. Constraints (2e) limit the total weight of the nodes in cluster k to be between $C^{\min}$ and $C^{\max}$. If the node weight $w_i = 1$ for all $i \in V$, then the summation $\sum_{i \in V} w_i y_{ik}$ is the total number of nodes assigned to cluster k. If (2e) is omitted, it is optimal to create a single cluster; i.e., all the nodes and hence edges would be in one cluster. Binary restrictions are placed on all the variables in (2f).

Model (2) can be reduced by observing that constraints (2d) are redundant when the objective function is taken into account and hence can be omitted. That is, when either y_{ik} or y_{jk} is 0, x_{ek} is 0, which gives a feasible solution to (2d); when both y_{ik} and y_{jk} are 1, x_{ek} will be 1 as well since the objective is to maximize the total weight. A secondary consequence of this result is that, x_{ek} can be treated as a continuous variable in the range [0, 1]. Finally, the two-sided inequality (2e) can be simplified by introducing additional slack variables s_k, $k = 1,\ldots, p$. With some algebra, we can rewrite (2e) as follows:

$$\sum_{i \in V} w_i y_{ik} - s_k = C^{\min},\ \ \forall\ k = 1,\ldots, p \qquad (2e')$$

$$0 \leq s_k \leq C^{\max} - C^{\min},\ \ \forall\ k = 1,\ldots, p \qquad (2e'')$$

where constraints (2e″) specify the bounds on the slack variables s_k. For other formulations of p-CCP, see Ferreira et al. (1998) or Mehrotra and Trick (1998).

Strength of LP relaxation. When solving an integer program, the tightness of the bound obtained from the LP relaxation often determines the efficiency of the overall solution procedure. With respect to model (2) the following results states for the nontrivial cases where $\theta \geq 2$ and $n \geq \theta$ that this bound is arbitrarily bad.

Proposition 3.1 Given a completely connected graph $G = (V, E)$ with the objective of partitioning the $|V| = n$ nodes into p clusters, assume that $n \bmod p = 0$ and let $\bar{n} = n / p$ be integral for $n > p \geq 2$. If each cluster, must contain at least 1 node, then the minimum number of edges in a solution is $m_1^{\min} = \binom{\bar{n}}{2} \times p$.

Proof. Consider a solution in which the first $p - 1$ clusters each contains 1 node and cluster p contains the remaining $n - p + 1$ nodes. In this case, there are $\binom{n-p+1}{2}$ edges in cluster p and 0 edges in clusters 1,…p–1. If one node is removed from cluster p and placed in cluster 1, then the total number of edges in the corresponding solution is $\binom{n-p}{2} + 1 < \binom{n-p+1}{2}$. Repeating this process until all the nodes are evenly distributed among the p clusters gives the stated results. ■

Corollary 3.1 Let $\bar{p} = n \bmod p$ such that $0 < \bar{p} < p$. For G completely connected, the minimum number of edges in a clustering solution is $m_1^{\min} = \binom{\bar{n}}{2} \times p + \bar{p} \times \bar{n}$.

Proof. Evenly divide the first $\left\lfloor \frac{n}{p} \right\rfloor \times p$ nodes into p clusters. From Proposition 1, this gives $\binom{\bar{n}}{2} \times p$ edges. Assign the remaining $\bar{p}$ nodes to the first $\bar{p}$ clusters. This introduces $\bar{n}$ additional edges in each of those clusters. ■

Proposition 3.2 In model (2) the bounds satisfy $\mathrm{C}^{\min} \leq \frac{1}{p}\left(\sum_{i \in V} w_i\right) \leq \mathrm{C}^{\max}$.

Proof. Summing up the constraints (2e) for all clusters gives $p\mathrm{C}^{\min} \leq \sum_{k=1}^{p} \sum_{i \in V} w_i y_{ik} \leq p\mathrm{C}^{\max}$. The term $\sum_{k=1}^{p} \sum_{i \in V} w_i y_{ik} = \sum_{i \in V} \left(w_i \sum_{k=1}^{p} y_{ik} \right) = \sum_{i \in V} w_i$ since $\sum_{k=1}^{p} y_{ik} = 1 \; \forall \; i \in V$. Dividing both sides by p gives $\mathrm{C}^{\min} \leq \frac{1}{p}\left(\sum_{i \in V} w_i\right) \leq \mathrm{C}^{\max}$. ■

Corollary 3.2 Let ϕ_{LP} be the objective function obtained by solving (2) after relaxing the integrality requirements on the x and y variables in (2f). Then $\phi_{\text{LP}} = \sum_{e \in E} c_e$.

Proof. Assume that $x_{ek} = 1/p$, $y_{ik} = 1/p$, $\forall\ e \in E$, $i \in V$, $k = 1, 2, \ldots, p$. It can be verified that such LP solution is feasible to model (2) with objective function value $\sum_{e \in E} c_e$.

Since this is the maximum can be obtained, the LP solution is optimal and $\phi_{\text{LP}} = \sum_{e \in E} c_e$.

■

3.2 Solution Methodology

Model (2) is a 0-1 integer linear program of size $O(pn^2)$. For 60 data points and 5 clusters, this translates into a problem with approximately 18,000 variables and constraints in the worst case, which is likely to be beyond the capability of commercial solvers. Real instances are often much larger. Our experience with CPLEX 11.0 showed that some instances with $|V| = 40$ can be solved in a matter of minutes but when $|V| = 50$, runtimes exceed 10 hours. This is not surprising since the linear programming relaxation of (2) is arbitrarily bad. By setting $x_{ek} = y_{ik} = 1/p$ for all e, i and k, the objective function value in (2a) is $\sum_{e \in E} c_e$. In addition, symmetry plays havoc during branch and bound because many equivalent solutions can be obtained by exchanging the cluster numbers of any two clusters. This situation implies the existence of at least $(p - 1)!$ alternative optima. Also, fixing $y_{i1} = 0$ at a particular node in the search tree has very little effect since y_{ik}, $k = 2, \ldots, p$, can still be nonzero.

In light of these observations, we developed a reactive GRASP with the objective of finding high quality solutions to (2). In phase I, good initial solutions are constructed in a greedy manner; in phase II, they are improved by local search. In a post-processing step, a subset of the phase II solutions are assembled in what is called an *elite pool* and subject to further investigation using path relinking (PR). When no better solutions can be found along the paths connecting any pair of pool members, the procedure terminates and the best available solution is output.

3.2.1 GRASP phase I

During construction, our first step is to initialize the p clusters. One of two approaches is used: the heaviest weight edges algorithm (HWE) or the constrained minimum cut

algorithm (CMC). With HWE, we identify the p nodes with the largest weights and assign them in turn to the p clusters. The heaviest unassigned edges incident to these nodes are then sequentially assigned to the corresponding clusters along with their endpoints. The CMC approach makes use of a minimum cut algorithm to partition the graph into p clusters that satisfy the capacity lower bound $C^{\min}$. In either case, the partial solutions associated with the p clusters serve as seeds. The underlying motivation is to identify nodes and edges that are not likely to be in the same cluster in an optimal partition. After initialization, a reactive GRASP is called to construct feasible solutions. The details are given below.

HWE approach to the selection of *p* seeds

The HWE algorithm is illustrated in Figure 3.1. At Step 1, sets and counters are initialized. At Step 2, the nodes are ordered from largest to smallest, that is, $w_{i_1} \geq w_{i_2} \geq \mathrm{L} \geq w_{i_{n-1}} \geq w_{i_n}$, and the heaviest node is assigned to cluster 1, the next heaviest to cluster 2 and so on until p clusters have been initialized or until $w_{i_s} = w_{i_n}$, where $s < p$. The objective is to disperse the heaviest nodes to different clusters in order to increase the chance of getting a feasible solution when capacity is tight. When $w_{i_s} = w_{i_{s+1}} = \cdots = w_{i_n}$, it becomes more effective to assign heavy edges rather than nodes as seeds.

At Step 3, an additional node is assigned to those clusters that have been initialized, or two nodes are assigned if the cluster is empty. In the former case, a free node that is the most heavily connected to the existing node is selected; in the latter case, the endpoint node of the heaviest free edge is assigned. At termination, each cluster will contain exactly two nodes that represent a partial initial solution to the problem. The complexity of the procedure is $O(p \cdot |V|^2)$.

Parameters

E_0 set of unassigned edges

V_0 set of unassigned nodes

V_k set of nodes assigned to cluster k

Procedure: Phase_I_Initialize_HWE(V, E, c, w, p, $C^{\min}$, $C^{\max}$, x')

Input: Set of nodes V, set of edges E, number of clusters p, edge weights matrix c, node weights vector w, and capacity bounds $C^{\min}$ and $C^{\max}$

Output: Partial initial solution x'

Step 1: $E_0 = E$; $V_0 = V$; $V_k = \emptyset$, $k = 1,\ldots,p$; $x'_{ik} = 0$, $\forall\, i \in V$, $k = 1,\ldots,p$;

Step 2: $k = 1$;

while($k \leq p$ and $\max\{w_i, i \in V_0\} \neq \min\{w_i, i \in V_0\}$){

$i^* \in \text{argmax}\{w_i : i \in V_0\}$;

$V_k \leftarrow V_k \cup \{i^*\}$; $V_0 \leftarrow V_0 \setminus \{i^*\}$; $x'_{i^*k} = 1$;

$k \leftarrow k + 1$;

}

Step 3: for($k = 1,\ldots,p$){

if($|V_k| = 0$){

$(i^*, j^*) \in \text{argmax}\{c_{ij} : w_i + w_j \leq C^{\max}, (i, j) \in E_0\}$;

$V_0 \leftarrow V_0 \setminus \{i^*, j^*\}$; $V_k \leftarrow V_k \cup \{i^*, j^*\}$; $x'_{i^*k} = 1$; $x'_{j^*k} = 1$;

}else{//one node already exists in cluster V_k

$j^* \in \text{argmax}\{c_{ij} :\ w_i + w_j \leq C^{\max}, i \in V_k, j \in V_0\}$;

$V_0 \leftarrow V_0 \setminus \{j^*\}$; $V_k \leftarrow V_k \cup \{j^*\}$; $x'_{j^*k} = 1$;

}

}

Figure 3.1 Pseudocode for seed selection with HWE algorithm in phase I of GRASP

Figure 3.2 depicts the partial initial solution provided by HWE for a 9-node, 3-cluster problem with bounds $C^{\min} = 3$ and $C^{\max} = 5$. Assume that the node weights are $w_4 = 3$, $w_3 = 2$, and $w_i = 1$, $\forall\, i \in \{1,2,5,6,7,8,9\}$, and that the edge weights are as shown in the figure. Initially, $V_1 = V_2 = V_3 = \emptyset$. The heaviest node (node 4) is assigned to cluster V_1, while the second heaviest (node 3) is assigned to cluster V_2. Since the remaining nodes all have the same weight, cluster V_3 is left empty, giving $V_1 = \{4\}$, $V_2 = \{3\}$ and $V_3 = \emptyset$. In the next step, node 5 is placed into V_1 since it is the most heavily connected to node 4, and node 2 is placed into V_2 for the same reason. Finally, edge (1,7) is assigned

to V_3 since it has the highest edge weight among the free edges. The algorithm ends and the initial partial solution is $V_1 = \{4,5\}$, $V_2 = \{2,3\}$ and $V_3 = \{1,7\}$.

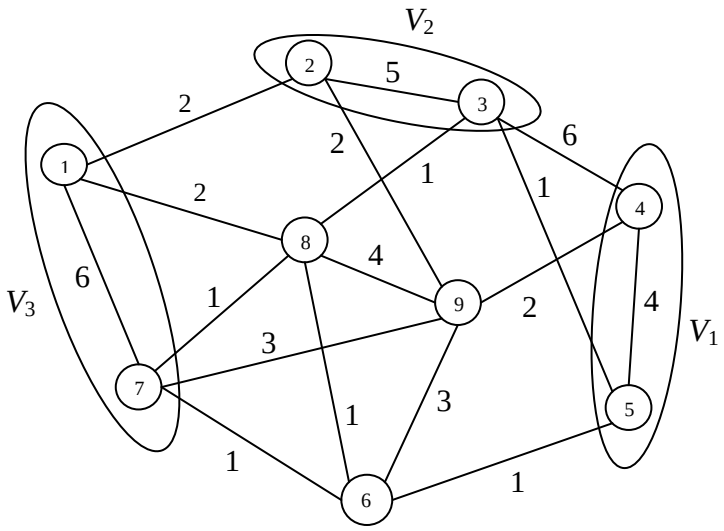

Figure 3.2 Example for identifying seeds with HWE

Minimum cut approach to the selection of *p* seeds

In this approach, we apply a constrained minimum cut scheme to partition the nodes in $G = (V,E)$ into p subsets such that the sum of the node weights in each subset V_k , $k = 1,\ldots,p$, is at least $C^{\min}$, where $V = \cup_{k=1}^{p} V_k$ and $V_k \cap V_s = \emptyset$ for $k \neq s$. The heaviest edge in each cluster will serve as a seed while the remaining edges are removed.

The algorithm is outlined in Figure 3.3. The bulk of the work is done at Step 2 with the call to CMC(V_k, E_k, $C^{\min}$, w, S_1, S_2), which is a heuristic that divides G into two subsets, S_1 and S_2, such that the total weight of the edges between them is minimized while the sum of their individual node weights is at least $C^{\min}$. The problem of finding the global minimum cut in a graph is a special case of model (2), that is, when $p = 2$, $C^{\min} = 1$ in (2e), and the upper bound $C^{\max} \rightarrow +\infty$. The problem becomes NP-hard when $C^{\min} \geq 2$ and $C^{\max}$ is finite. The Frank's (1994) polynomial-time algorithm, a slight improvement on Nagamochi and Ibaraki's (1992) algorithm, is applied to solve the min-cut problem [its complexity is $O(|V|\cdot|E|)$] even though other lower polynomial-time algorithms exist [e.g., Karger and Stein (1996) developed a heuristic for the min-cut problem with $O(|V|^2(\log |V|)^3)$ complexity]. Our choice was based on the fact that in

previous work we found Frank's algorithm easy to implement and extremely efficient on similar size graphs.

Procedure: Phase_I_Initialize_CMC(V, E, c, w, p, $C^{\min}$, $C^{\max}$, x')

Input: Set of nodes V, set of edges E, number of clusters p, edge weights matrix c, node weights vector w, and capacity bounds $C^{\min}$ and $C^{\max}$

Output: partial initial solution x'

Step 1: $V_1 = V$, $V_k = \emptyset$, $k = 2,\ldots,p$; $x'_{ik} = 0$, $\forall\, i \in V$, $k = 1,\ldots,p$;

Step 2: while ($\min\{|V_k| : k = 1,\ldots,p\} = 0$){

$k^* \in \text{argmax}\{|V_k| : k = 1,\ldots,p\}$;

//Apply constrained minimum cut heuristic to V_{k^*}

call CMC(V_{k^*}, E_{k^*}, $C^{\min}$, w, S_1, S_2); // see Figure 3.4

$V_{k^*} = S_1$;

$k^* = \min\{k : |V_k| = 0, k = 1,\ldots,p\}$; //pick the first empty cluster

$V_{k^*} = S_2$;

}

Step 3: for ($k = 1,\ldots,p$) {

//Only keeps the heaviest edge in the cluster

$(i^*, j^*) \in \text{argmax}\{c_{ij} : i \in V_k, j \in V_k\}$;

$V_k = \emptyset$; $V_k \leftarrow V_k \cup \{i^*, j^*\}$; $x'_{i^*k} = 1$; $x'_{j^*k} = 1$;

}

Figure 3.3 Pseudocode for seed selection with CMC algorithm in phase I of GRASP

CMC works by first partitioning the subgraph $G_k = (V_k, E_k)$ into S_1 and S_2 for the unconstrained case, call it UMC, and checking each subset for feasibility. If the lower bound capacity constraint associated with, say S_1, is violated, a node is selected from S_2 and placed in S_1. At this step, the node that is most connected with the nodes in S_1, as measured by the sum of the weights of the incident edges whose endpoints are in S_1, is selected as long as the transfer does not violate the lower bound capacity constraint of S_2.

The process is repeated until a feasible partition is obtained. The procedure is outlined in Figure 3.4.

The same graph used to illustrate HWE will be used to illustrate Phase_I_Initialize_CMC for $p = 3$. At Step 1, $V_1 = V$ and CMC is called. At Step 1 of CMC, applying Frank's UMC algorithm to V_1 returns a minimum cut of 6 with $S_1 = \{6\}$, $S_2 = \{1, 2, 3, 4, 5, 7, 8, 9\}$ and corresponding weights $W(S_1) = 1$ and $W(S_2) = 11$. At Step 2 of CMC, we have $W(S_1) < C^{\min} = 2$ so a node must be transferred from S_2 to S_1. The calculations indicate that node 9 in S_2 is the most heavily connected to S_1 so it is selected. The updated clusters are {6, 9} and {1, 2, 3, 4, 5, 7, 8}. The operations at Step 2 stop since both S_1 and S_2 satisfy the lower bound $C^{\min}$. Let $V_1 \leftarrow S_1$, $V_2 \leftarrow S_2$ and set $S_1 = \emptyset$, $S_2 = \emptyset$. The larger set, V_2, is selected for partitioning at Step 2 of Phase_I_Initialize_CMC. Applying the UMC algorithm at Step 1 of CMC returns a minimum cut of 3 with $S_1 = \{1, 7, 8\}$ and $S_2 = \{2, 3, 4, 5\}$, both of which satisfy the lower bound constraints. Therefore, we put $V_2 \leftarrow S_1$, $V_3 \leftarrow S_2$ and set $S_1 = \emptyset$, $S_2 = \emptyset$. The operations at Step 2 of Phase_I_Initialize_CMC terminate since all three clusters are filled. The heaviest edge in each is retained and the others are removed. The final seeds for the three clusters are $V_1 = \{6, 9\}$, $V_2 = \{1, 7\}$ and $V_3 = \{3, 4\}$, as shown in Figure 3.5.

Procedure: CMC(V_k, E_k, $C^{\min}$, w, S_1, S_2)

Input: Node set V_k, edge set E_k; lower bound on capacity $C^{\min}$; node weights vector w; edge weights matrix c

Output: Partition of nodes into subset S_1 and S_2

Step 1: Apply UMC procedure of Frank to V_k

call UMC(V_k, E_k, S_1, S_2);

let $W(S) = \sum_{i \in S} w_i$;

Step 2: while ($\min\{W(S_1), W(S_2)\} < C^{\min}$) {

$k_1 = \text{argmin}\{W(S_k) : k = 1, 2\}$;

$k_2 = \text{argmax}\{W(S_k) : k = 1, 2\}$;

//select the most beneficial move

$$i^* = \underset{i}{\text{argmax}} \left\{ \sum_{j \in S_{k_1}} c_{ij}, \forall i \in S_{k_2}, W(S_{k_2}) - w_i \geq C^{\min}, W(S_{k_1}) + w_i \geq C^{\min} \right\};$$

Put $S_{k_2} \leftarrow S_{k_2} \setminus \{i^*\}$; $S_{k_1} \leftarrow S_{k_1} \cup \{i^*\}$;

}

Figure 3.4 Pseudocode of CMC scheme

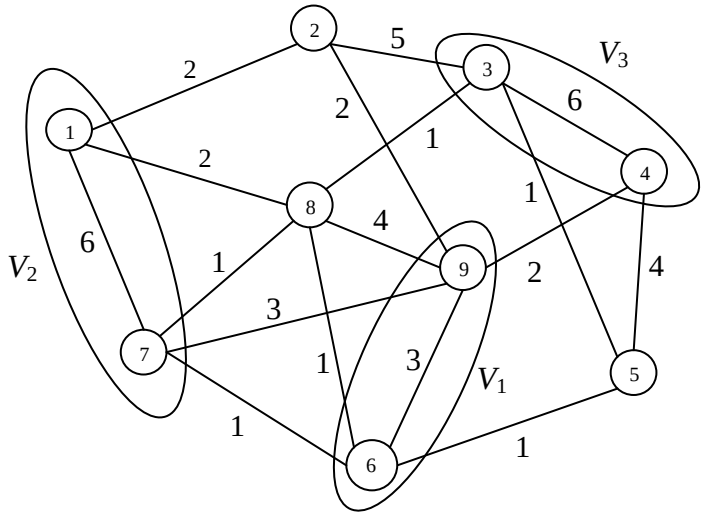

Figure 3.5 Example used to illustrate CMC scheme

Building the candidate list

Two kinds of insertions are considered when building the candidate list (CL), the structure used in GRASP to guide the construction of feasible solutions. The first corresponds to an unassigned node and the second to an unassigned edge. All feasible insertions are included in CL and sorted according to their contribution to the objective value, as measured by total edge weight that would result if the node or edge were actually added to a particular cluster. Candidates that violate the upper bound $C^{\max}$ are discarded.

Let $I(i,k)$ be the increase in the objective function value realized by inserting node i into cluster k and let $I(e,k)$ be the increase realized by inserting edge e to cluster k. Starting with the partial initial solution shown in Figure 3.2 for $C^{\min} = 3$ and $C^{\max} = 5$, the full CL is given in Table 3.1. To see how these values were calculated, consider, for example, edge (8,9). If this edge were included in cluster 3, the objective value would be 10 (that is, $c_{18} + c_{78} + c_{79} + c_{89} = 2 + 1 + 3 + 4 = 10$); if included in cluster 2, the objective value would be 7, and if included in cluster 1, the objective value would be $-\infty$ since

this would lead to a violation of the upper bound C^{max}. Hence, cluster 3 is the first choice for (8,9). This insertion would increase the cluster weight from 2 to 4, which is less than C^{max}.

Table 3.1 Example of CL

CL index	Edge e or node i	Cluster index k	$I(e,k)$ or $I(i,k)$
1	(8,9)	3	10
2	(6,9)	3	7
3	(8,9)	2	7
4	(6,8)	3	5
5	(6,9)	2	5
6	8	3	3
7	9	3	3
8	9	1	2
9	9	2	2
10	(6,8)	2	2
11	6	1	1
12	6	3	1
13	8	2	1
14	6	2	0
15	8	1	0
16	(6,8)	1	$-\infty$
17	(6,9)	1	$-\infty$
18	(8,9)	1	$-\infty$

Self-adjusting RCL

A fraction α of the top candidates in CL, up to some parameterized maximum number denoted by $\bar{l}_{RCL}$, are used to build RCL from which the next construction step is taken. The length of RCL, l_{RCL}, is determined as follows:

$$l_{RCL} = \min \{\max \{\alpha l_{CL}, 1\}, \bar{l}_{RCL}\}$$

where l_{CL} is the current length of CL. The value of α in this equation is adjusted during the GRASP iterations according to the quality of observed solutions. Prais and Ribeiro (1999) indicate that α should be within the range of $(0,1]$.

Let $A = \{\alpha_1, \alpha_{2,\ldots,}\alpha_m\}$ be the finite set of possible values for α and let p_i be the corresponding probability of selecting α_i, $i = 1,\ldots,m$. Initially, p_i is uniformly distributed:

$$p_i = 1/m, \quad i = 1,\ldots,m$$

To see how these probabilities are adjusted, let ϕ^* be the best solution found in all previous GRASP iterations and let A_i be the average value of solutions obtained for $\alpha = \alpha_i$. Initially, each A_i is set to the total edge weight of the graph and 20 experiments are run by sampling α from the above uniform distribution to get 20 additional objective function values. Updating begins at this point by calculating the relative performance of the algorithm under α_i as follows:

$$q_i = \left(\frac{A_i}{\phi^*} \right)^{\delta}, \quad i = 1,\ldots,m$$

where δ is a shape parameter. For higher values of δ, q_i will be lower since $A_i \leq \phi^*$. Normalizing gives

$$p_i = q_i / \sum_l^m q_l \, , \; i = 1,\ldots,m$$

When α_i yields relatively high average solutions A_i, it will have a high probably p_i of being selected as the iterations progress. In the implementation, we followed the suggestions of Prais and Ribeiro and set δ =10, m = 10, and A = {0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}.

Phase I initial solution construction

The partial initial solution constructed with either HWE or CMC is extended to obtain a feasible solution by sequentially adding nodes or edges to each of the p clusters. Assume that RCL is built with length l_{RCL} in accordance with above procedure. Exactly one element is randomly selected from RCL with uniformly distributed probability. The insertion corresponding to the selected element is performed to extend the current partial solution.

Again starting with the partial solution shown in Figure 3.2 and with CL given in Table 3.1, assume that l_{RCL} is determined to be 6. The corresponding RCL is given in Table 3.2 and is seen to contain the top 6 candidates in CL.

If the third element is chosen, for example, then edge (8,9) is placed in cluster 2 and the partial solution is updated. Now, CL is cleared and rebuilt along with RCL. The procedure is repeated until all nodes are assigned to one of the p clusters.

Table 3.2 Example of RCL when $l_{RCL} = 5$

RCL index	Edge e or node i	Cluster index j	$I(i,j)$ or $I(e,j)$
1	(8,9)	3	10
2	(6,9)	3	7
3	(8,9)	2	7
4	(6,8)	3	5
5	(6,9)	2	5
6	8	3	3

3.2.2 GRASP phase II

Three types of neighborhoods are explored in phase II. For current solution x, call them $N_1(x)$, $N_2(x)$ and $N_3(x)$, let $V_k(x)$ be the nodes in cluster k, and let $W_k(x)$ be the corresponding total node weight, $k = 1,\ldots,p$. A description of the neighborhoods follows.

$N_1(x)$ (Extended node insertion) Pick a node $i \in V_k(x)$ with $W_k(x) - w_i \geq C^{\min}$. Choose a cluster $V_s(x)$, $k \neq s$. If $W_s(x) + w_i \leq C^{\max}$, assign i to $V_s(x)$; otherwise, cluster s will exceed the upper bound. For the later situation, pick another node $j \in V_s(x)$, $i \neq j$, and cluster $s_1 \neq s$, such that $C^{\min} \leq W_s(x) + w_i - w_j \leq C^{\max}$ and $W_{s_1}(x) + w_j \leq C^{\max}$. Shift j from $V_s(x)$ to $V_{s_1}(x)$.

$N_2(x)$ (Extended edge insertion) Pick an edge $e \in E$ with endpoints i and j. Two cases may arise; either e is in some cluster $V_k(x)$ or it spans two clusters.

(1) If $i \in V_k(x)$, $j \in V_k(x)$ and $W_k(x) - w_i - w_j \geq C^{\min}$, then find a cluster $s \neq k$ with $W_s(x) + w_i + w_j \leq C^{\max}$ and shift (i,j) from $V_k(x)$ to $V_s(x)$. If no such s exists, then go to next $e \in E$. If $W_k(x) - w_i - w_j < C^{\min}$, removing e from $V_k(x)$ would violate $C^{\min}$. In this situation stop investigating the current edge and go to next $e \in E$.

(2) If e is not an edge within a cluster, let $i \in V_{k_1}(x)$ and $j \in V_{k_2}(x)$. When $W_{k_1}(x) - w_i \geq C^{\min}$ and $W_{k_2}(x) - w_j \geq C^{\min}$, one of the following three methods is used to extend the neighborhood: (i) find a set $s \neq k_1$, $s \neq k_2$ with $W_s(x) + w_i + w_j \leq C^{\max}$ and shift nodes i and j to $V_s(x)$; (ii) if $W_{k_1}(x) + w_j \leq C^{\max}$, shift j from cluster k_2 to k_1; (iii) if $W_{k_2}(x) + w_i \leq C^{\max}$, shift i

from cluster k_1 to k_2. If $W_{k_1}(x) - w_i < C^{\min}$ or $W_{k_2}(x) - w_j < C^{\min}$, stop and go to next $e \in E$.

$N_3(x)$ (Node exchange) For nodes $i \in V_k(x)$ and $j \in V_s(x)$, $k \neq s$, if $C^{\min} \leq W_k(x) - w_i + w_j \leq C^{\max}$ and $C^{\min} \leq W_s(x) - w_j + w_i \leq C^{\max}$, swap i and j. Otherwise, go to next pair of nodes.

The complexity of constructing these neighborhoods is a function of p, $|V|$ and $|E|$. For each case we respectively have:

$N_1(x) \sim O(p^2 \cdot |V|^2)$

$N_2(x) \sim O(p \cdot |E|)$

$N_3(x) \sim O(\sum_{1 \leq k < s \leq p} |V_k(x)| \cdot |V_s(x)|)$

The corresponding pseudocodes are given from Figure A.69 to Figure A.71.

Continuing with the example in Figure 3.2, assume that the current solution is $V_1 = \{4,5,9\}$, $V_2 = \{2,3,8\}$ and $V_3 = \{1,6,7\}$ with capacity bounds $C^{\min} = 3$ and $C^{\max} = 5$, and node weights $w_4 = 3$, $w_3 = 2$, and $w_i = 1$, $\forall\, i \in \{1,2,5,6,7,8,9\}$. For neighborhood N_1, the consequences of reassigning node 8 from cluster 2 to either cluster 1 or 3 are shown in Table 3.3. If cluster 1 is the target, then one of the nodes in cluster 1 must be removed to avoid a violation of the capacity upper bound.

Table 3.3 N_1 neighborhood generated by shifting node 8

Cluster to which node 8 is moved (s)	Node to be shifted (j')	Cluster to which node is shifted (s_1')	Total benefit gained
1	4	2	$-\infty$
	4	3	–3
	5	2	0
	5	3	0
	9	2	–1
	9	3	3
3	--	--	3

For neighborhood N_2, assume that the algorithm is investigating edge (8,9) which crosses clusters 1 and 2. However, node 8 cannot be inserted into cluster 1 due to $C^{\max}$. Alternatively, if node 9 along is shifted into cluster 2, the total benefit gained will be 4 and 11 if edge (8, 9) is inserted to cluster 3.

For neighborhood N_3, consider a swap between node 8 and some other node. After a simple set of calculations, we find that the best swap is between nodes 8 and 6 with benefit 1.

Capacity bounds are maintained during local search to ensure feasibility. Although it is possible to allow infeasible solutions as a strategy to overcome local optimality, such an approach would greatly increase the computational effort of phase II. In general, the GRASP philosophy is to focus the effort on phase I, not phase II. With this in mind, diversification is introduced by accepting inferior solutions that are within some tolerance β, a parameter that is reduced dynamically by Δ after searching each of the three neighborhoods. In the basic implementation, N_1, N_2 and N_3 are explored sequentially with β starting at 1% and decreased to 0 in steps of size $\Delta = 0.2\%$. As β is reduced, the effort shifts from diversification to intensification, and when β reaches 0, no inferior solutions are accepted. A summary of phase II is given in Figure 3.6. At Step 2, we cycle through the neighborhoods, terminating when no improvement is possible. This is called cyclic neighborhood search (CNS).

Procedure: GRASP_Phase_II(x, w, c, β, Δ, $C^{\min}$, $C^{\max}$, x^*)

Input: current solution x, node weights vector w, edge weights matrix c, capacity bounds $C^{\min}$ and $C^{\max}$, tolerance β and stepsize Δ

Output: local solution x^* with respect to neighborhoods $N_1(x)$, $N_2(x)$ and $N_3(x)$.

Step 1: $x^* = x$;

Step 2: while ($\beta > 0$) {

Improve the current solution by local search

call $N_1(x^*, w, c, \beta, C^{\min}, C^{\max}, x_1)$;

call $N_2(x_1, w, c, \beta, C^{\min}, C^{\max}, x_2)$;

call $N_3(x_2, w, c, \beta, C^{\min}, C^{\max}, x_3)$;

$\beta = \beta - \Delta$;

$x^* = x_3$;

}

Step 3: $TEW(x^*) = -\infty; TEW(x_3) = \sum_{k=1}^{p} \sum_{i,j \in V_k(x^3)} c_{ij}$; //*TEW* = total edge weight

Step 4: while ($TEW(x_3) > TEW(x^*)$) {

$x^* = x_3$;

call $N_1(x^*, w, c, 0, C^{\min}, C^{\max}, x_1)$;

call $N_2(x_1, w, c, 0, C^{\min}, C^{\max}, x_2)$;

call $N_3(x_2, w, c, 0, C^{\min}, C^{\max}, x_3)$;

$TEW(x^*) = \sum_{k=1}^{p} \sum_{i,j \in V_k(x^*)} c_{ij}$; $TEW(x_3) = \sum_{k=1}^{p} \sum_{i,j \in V_k(x^3)} c_{ij}$

}

Figure 3.6 Pseudocode for phase II of GRASP

3.2.3 Basic GRASP

Given a graph $G = (V, E)$, a partial initial solution is constructed with either HWE or CMC and extended to a feasible solution using the logic surrounding RCL. Phase II is then applied a predetermined number of times to improve the current solution within the three neighborhoods. The best solution found is output as the optimum. The pseudocode for the basic reactive GRASP is given in Figure 3.7 with the help of the following and aforementioned definitions.

Parameters

N^{GRASP}	number of iterations for GRASP
I^{init}	indicator of approach to build partial initial solution: $I^{\text{init}} = 0$ for HWE approach; $I^{\text{init}} = 1$ for CMC approach
iter	iteration counter

Algorithm: GRASP

Input: set of nodes V, set of edges E, node weights vector w, edge weights matrix c, number of clusters p, capacity bounds $C^{\min}$ and $C^{\max}$, indicator I^{init}, number of iterations N^{GRASP}

Output: heuristic solution x^{best}

Step 1: obtained partial initial solution x'

if (I^{init} equals 0) {

call Phase_I_Initialize_HWE($V, E, c, w, p, C^{\min}, C^{\max}, x'$);

}else{

call Phase_I_Initialize_CMC($V, E, c, w, p, C^{\min}, C^{\max}, x'$);

}

Step 2: $TEW(x^{\text{best}}) = -\infty$;//*TEW* = total edge weight

Step 3: for (*iter* =1,…, N^{GRASP}){
construct CL and RCL, complete x' to initial solution x randomly;
call GRASP_Phase_II(x, w, c, β, Δ, $C^{\min}$, $C^{\max}$, x^*);
if ($TEW(x^{\text{best}}) < TEW(x^*)$){
$x^{\text{best}} = x^*$;
$TEW(x^{\text{best}}) = TEW(x^*)$;
}
}

Figure 3.7 Pseudocode for basic reactive GRASP

3.2.4 Variable neighborhood descent

VND is a systematic approach to exploring the various neighborhoods that define the local search (Hansen and Mladenovic 1997). Say there are $u_{\max}$ of them indexed by u and $N_u \subseteq N_{u+1}$, $\forall\ u = 1,2,\ldots,u_{\max}-1$. VND starts by searching the first neighborhood N_1 (u = 1) and, in general, switches from the current neighborhood N_u to the next neighborhood N_{u+1} when N_u fails to provide an improved solution. If a better solution is obtained from N_u, then VND switches back to N_1. The procedure terminates when VND reaches the final neighborhood $N_{u_{\max}}$ and no improvement is possible. The last solution uncovered is locally optimal for all $u_{\max}$ neighborhoods.

VND has been shown to be efficient in various applications (e.g., see Hu et al. 2008). In our case, it serves as an option in phase II to increase the performance of local search even though the three neighborhoods defined above are not a subset of each other – the usual situation in which VND is applied.

3.2.5 Randomized VND

According to our initial experiments, standard VND did not lead to a balanced exploration of the three neighborhoods. Most of the effort was spent searching N_1. Even though local optimality is guaranteed, such a bias might delay convergence. To address this issue, we adopted a probabilistic weighting scheme similar to the one used for constructing RCL.

Let p_u be the probability of selecting neighborhood N_u after exploring the current neighborhood. A uniform distribution for these values is assumed initially:

$$p_u = 1/u_{\max},\ u = 1,\ldots,u_{\max}$$

Also, let B_u be the total benefit gained by searching neighborhood N_u so far, with initial values set as follows:

$$B_u = \Sigma_{e \in E} c_e,\ u = 1,\ldots,u_{\max}$$

The neighborhood to be searched in the next iteration is randomly determined by the probabilities p_u. Let u^* be the current neighborhood and let b be the improvement realized from the search. The total benefit for N_{u^*} is updated by putting

$$B_{u^*} \leftarrow B_{u^*} + b$$

Note that it is possible for $b < 0$, which would indicate that a nonimproving solution was selected in the diversification step of phase II. In that case, B_{u^*} would decrease and make N_{u^*} a less interesting option to explore. When $b > 0$, B_{u^*} will increase, suggesting that more effort should be placed on searching N_{u^*}. The probabilities p_u are hence updated to take into account the relative quality of solutions found in each neighborhood. In particular,

$$p_u = \left(B_u / \sum_{v=1}^{u_{\max}} B_v \right),\ u = 1,\ldots,u_{\max}$$

The randomized version of VND is called RVND and is run for a predetermined number of iterations, Max_Iter. In the implementation, B is set to 1000, which is large enough to ensure that $B_u > 0$, $u = 1,\ldots,u_{\max}$, for the data used in the testing, and Max_Iter is set to 10.

3.2.6 Path relinking

During phase II, solutions that are unique are saved in a pool and sorted in descending order of their objective function values. The top N^{elite} members of the pool are selected to form the elite solution set S^{elite}. The general idea of PR is to construct a path between pairs of elements in S^{elite} to see if better solutions can be found. As described presently, feasibility is maintained at each iteration, and for a problem with p clusters, at most $p - 2$ distinct solutions will be uncovered along each path. Those that are superior to their generators are stored temporarily and, after all original pairs are examined, are inserted into S^{elite}. At the same time, the bottom elements in S^{elite} are removed to keep $| S^{\text{elite}} |$ constant. The procedure ends when the maximum number of iterations, N_{PR}, is reached

or S^{elite} becomes stable, that is, the elements in S^{elite} do not change between two successive iterations.

PR was first proposed by Glover et al. (2000) and is usually combined with other metaheuristics (e.g., see Boudia et al. 2006). Given the set S^{elite} at the end of phase II, the first step is to select a pair of elements, say x_A and x_B, to serve as path generators. In this context, x_A is known as the initiating solution and x_B as the guiding solution. In attempting to construct a path that links x_A to x_B, let $V_k(x_A)$ be the node set for cluster k associated with x_A and let $V_s(x_B)$ be the node set for cluster s associated with x_B. Now, define a *similarity* measure $S(k, s)$ for $V_k(x_A)$ and $V_s(x_B)$ as follows.

$$S(k, s) = \sum_{e \in V_k(x_A) \cap V_s(x_B)} c_e$$

The value of $S(k, s)$ is the total weight of the common edges in $V_k(x_A)$ and $V_s(x_B)$. The two most similar clusters, call them k_A and s_B, associated with the elite solutions x_A and x_B, are determined by

$$(k_A, s_B) = \text{argmax}\{S(k, s) : k, s \in \{1,\ldots,p\}\}$$

where ties are broken by selecting the clusters with the smallest indices.

Given x_A and x_B, define C_A and C_B as the sets of clusters that are fixed at some iteration in the procedure. Initially, $C_A = \emptyset$ and $C_B = \emptyset$. A path from x_A to x_B is generated in the following manner. First, the most similar clusters k_A and s_B are identified according to the aforementioned logic. Cluster k_A is then modified to be exactly the same as cluster s_B by inserting and removing nodes. The cluster to which a node is moved is determined by a simple local search to minimize the decrease in objective function value. After this operation is performed a new solution x_1 emerges from x_A. The two sets C_A and C_B are updated by putting $C_A \leftarrow C_A \cup \{k_A\}$ and $C_B \leftarrow C_B \cup \{s_B\}$. Solution x_1 is then improved to be x_1^* by local search subject to the restriction that the clusters in C_A are kept constant. Now, starting from x_1^* the process is repeated to get x_2^*, and so on. Termination occurs after $p - 2$ iterations at which time all p clusters are fixed in the sets C_A and C_B; the resulting solution is exactly the same as x_B. The path generated from x_A to x_B is as follows.

$$x_A \rightarrow x_1^* \rightarrow x_2^* \rightarrow \cdots x_{p-2}^* \rightarrow x_B$$

Finally, let $x_{C^*} = \text{argmax}\left\{TEW(x_k^*) : k = 1, ..., p-2\right\}$ be the best solution found along this path. If $TEW(x_{C^*}) > \max\{TEW(x_A), TEW(x_B)\}$, then it is stored and after all pairs of elements in S^{elite} are examined, it is inserted into S^{elite}. If the capacity bounds are tight, it is possible that no feasible solution will be discovered between x_A to x_B. In that case, PR fails for x_A and x_B, and the next pair is examined.

An example of path generation based on the graph in Figure 3.2 is given in Figure 3.8. The nodes are to be partitioned into three clusters with $C^{\min} = 3$, $C^{\max} = 5$ and $w_4 = 3$, $w_3 = 2$ and $w_i = 1$, $i \in \{1,2,5,6,7,8,9\}$. Assume that x_A is {{1, 7, 8}, {2, 9, 4}, {3, 5, 6}} with objective function value $\phi_A = 15$ and that x_B is {{1, 4, 7}, {2, 5, 6}, {3, 8, 9}} with $\phi_B = 12$. Starting with $C_A = \emptyset$ and $C_B = \emptyset$, the goal is to generate a path from x_A to x_B. At Step 1 the clusters most similar with respect to solutions x_A and x_B are $k_A = 1$ and $s_B = 1$ with $S(k_A, s_B) = 6$. Node 8 in $V_1(x_A)$ is removed and inserted into $V_2(x_A)$ while node 4 in $V_2(x_A)$ is shifted to $V_1(x_A)$. Call the transformed solution x_C, and note that cluster 1 in x_C is exactly the same as cluster 1 in x_B; that is, $V_1(x_C) = V_1(x_B)$.

Next, the constant sets are updated giving $C_A = \{1\}$ and $C_B = \{1\}$, and a local search is performed on x_C, which results in an improved solution x_{C^*}. At the next step, the most similar clusters with respect to x_{C^*} and x_B are determined to be $k_{C^*} = 2$ and $s_B = 3$. The sets C_A and C_B are now {1, 2} and {1, 3}, respectively. To make cluster 2 in x_{C^*} the same as cluster 3 in x_B, node 6 is selected and placed in $V_3(x_{C^*})$ while and node 3 is shifted to $V_2(x_{C^*})$. The resulting solution is exactly the same as x_B.

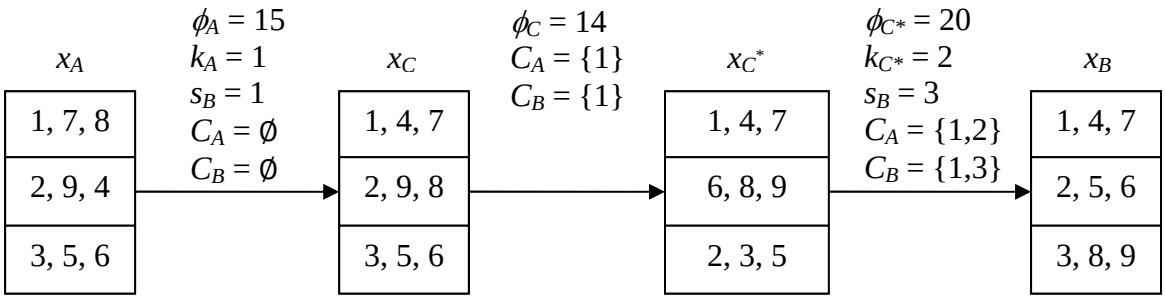

Figure 3.8 An example of path generation

The best solution found along the path is x_{C^*} with $\phi_{C*} = TEW = 20$. Since $\phi_{C*} > \phi_A$ and $\phi_{C*} > \phi_B$, x_{C^*} is outputted and stored for possible insertion into S^{elite}. The pseudocode for path generation is shown in Figure 3.9. In our implementation, $| S^{\text{elite}} |$ is set to 20. For each pair of solutions $x_A \in S^{\text{elite}}$ and $x_B \in S^{\text{elite}}$, two paths are generated, the first starting from x_A and approaching x_B and the second taking the reverse course. For a given S^{elite}, the total number of paths is $O(|S^{\text{elite}}|^2)$. When S^{elite} becomes stable the best solution found up to that point is output.

A potentially inefficient aspect of PR is the application of local search to each solution encountered along a path. Empirically, we found that a complete local search (CLS) strategy may affect the solution quality only locally within the same basin of attraction. In addition, the current solution may have been uncovered previously so applying local search a second time is wasteful. One way to reduce the computational effort is to apply local search only after encountering n^{PR} solutions along a path, where n^{PR} is a parameter adjusted according to the solution quality. This strategy is referred to as *partial local search* (PLS) to distinguish from CLS. For p clusters, $n^{\text{PR}} \in \{1, 2,\ldots, p-2\}$, where $n^{\text{PR}} = 1$ indicates that local search is applied to each solution in the path, $n^{\text{PR}} = 2$ indicates that it is applied to every second solution, and so on. Note that there are at most p solutions along a path including the initiating solution and the guiding solution. Because both of these are already locally optimal, $n^{\text{PR}} \leq p-2$.

The value of n^{PR} is randomly selected at the beginning of each path. The probability function used for this purpose is based on a performance measure $P_i(n^{\text{PR}})$, which is defined as the average objective function value over a path when the frequency of applying local search was n^{PR}; that is,

$$PF_i(n^{\text{PR}}) = \sum_{j=1}^{N_i} I\left\{ j \bmod n^{\text{PR}} \right\} A_{ij} \Big/ \sum_{j=1}^{N_i} I\left\{ j \bmod n^{\text{PR}} \right\}$$

where i is the index for path, N_i is the number of solutions discovered along path i, j is the index for the solutions discovered on a path, A_{ij} is the objective value of the j^{th} solution discovered on path i and $I\left\{ j \bmod n^{\text{PR}} \right\}$ is a Boolean indicator function equals to 1 when (j mod n^{PR}) = 0 (i.e., <true>) and 0 otherwise. The summation $\sum_{j=1}^{N_i} I\left\{ j \bmod n^{\text{PR}} \right\}$

counts the total number of times local search is applied while exploring path i. If n^{PR} is not selected for path i, then $P_i(n^{PR}) = 0$.

Next, we compute the accumulated performance, denoted by $AP(n^{PR})$, for a particular value of n^{PR} by summing over all the paths already generated.

$$AP(n^{PR}) = \sum_{i=1}^{n^{cur}} P_i(n^{PR})$$

Here, n^{cur} is the number of paths that have been explored up to and including the current path.

At the beginning of PR, the probability $p(n^{PR})$ of selecting a particular value of $n^{PR} \in \{1, 2,\ldots, p-2\}$ is assigned a uniform distribution; that is,

$$p(n^{PR}) = 1/(p-2)$$

After exploring a path, this function is updated as follows.

$$p(n^{PR}) = AP(n^{PR}) \Big/ \sum_{n^{PR}=1}^{p-2} AP(n^{PR})$$

Thus, values of n^{PR} corresponding to higher accumulated performance will have a higher probability of being selected.

Procedure: path_generation(w, c, C^{min}, C^{max}, n^{PR}, x_A, x_B, x^*)
Input: node weights vector w, edge weights matrix c, capacity bounds C^{min} and C^{max}, PLS parameter n^{PR}, initiating solution x_A, guiding solution x_B
Output: best solution x^* found along the path from x_A to x_B
Step 1: $C_A = \emptyset$; $C_B = \emptyset$; $TEW^* = -\infty$; $r = 1$;
Step 2: while (x_A is not the same as x_B) {
(k_A, s_B) = argmax{$S(k, s) : k, s \in \{1,\ldots,p\}$};
make $V_{k_A}(x_A)$ the same as $V_{s_B}(x_B)$ by inserting and removing nodes so that x_A becomes x_r;
// keeping the clusters $k \in C_A$ constant, if the PLS condition is satisfied then apply local search to x_r in an attempt to obtain a better solution x_r^*
if (r mode n^{PR} + 1 equals to 0) {
call $N_1(x_r, w, c, 0, C^{min}, C^{max}, x)$; $x_r = x$;
call $N_2(x_r, w, c, 0, C^{min}, C^{max}, x)$; $x_r = x$;

```
            call N3(xr, w, c, 0, C^min, C^max, x_r^*);
        }
        if (TEW(x_r^*) > TEW^*) {
            TEW^* = TEW(x_r^*);  x^* = x_r^*;
        }
        x_A = x_r^*;  r ← r + 1;
    }
```

Figure 3.9 Pseudocode for path generation

3.3 Computational Results

The proposed methodology was implemented in C++ and run under Ubuntu Linux on a Dell Poweredge 2950 workstation with 2 dual core hyperthreading 3.73 GHz Xeon processors and 8 GB memory. In the testing, model (2a) – (2f) was solved, both heuristically with the reactive GRASP and directly with CPLEX 11.0 when possible. A comparison of the results gives insight in the quality of the GRASP solutions as well as the limits of CPLEX.

The following settings were used for the GRASP.

- Both HWE and CMC schemes were applied in phase I to construct partial initial solutions but in separate runs to allow for comparison
- Initial value of diversification parameter: $\beta = 0.01$ with $\Delta = 0.002$ in phase II
- Three options were examined for local search: (1) CNS; (2) VND; (3) RVND
- Number of GRASP iterations: $N^{\text{GRASP}} = 5 \times n^{\text{test}}$ with n^{test} being the number of nodes in the tests (experience has shown that the best solution is most often uncovered within the first 150 GRASP iterations; e.g., see Rojanasoonthon and Bard 2005)
- In PR, the maximum number of iterations is $N_{\text{PR}} = 50$, the number of elite solutions maintained is $| S^{\text{elite}} | = 20$

After some experimentation, the following settings were used for CPLEX.

- Cut generators off
- Emphasis of feasibility over optimality
- Optimality tolerance EpOpt = 1E−04.

- Default frequency for MIP heuristics

In the experiments, the methodology was tested on three data sets. The first contained relatively small instances that were randomly generated based on data provided by the USPS. A single seed was used for each instance. The second contained instances that reflected the full USPS problem. The third were obtained from Mehrotra and Trick (1998). In the next section, we outline the USPS application and describe how the node and edge weights were specified.

3.3.1 USPS application related to clustering control points

The cost of running a mail processing and distribution center (P&DC) is determined in part by the size and composition of the workforce. One of management's goals is to use as few powered industrial vehicles (PIVs) or drivers as possible to move the mail between workcenters, so restricting the number of control points (workcenters) that a driver can service would be suboptimal. However, to facilitate supervision and to avoid violating union rules, control points are first clustered into zones and then the minimum number of PIVs required to service each zone is determined. In the clustering step, it is necessary to take into account such factors as distance between nodes and transfer frequencies. Two nodes are likely to be grouped together if they are directly linked in the process flow, are relatively close to each other, and one is a frequent terminal point of the other.

In the clustering model, it is necessary to specify a measure that numerically captures these characteristics. Such a measure can be viewed as the edge weights, c_{ij}, connecting pairs of nodes i and j in a directional graph. For the test cases, we used the following formula to determine these weights.

$$c_{ij} = \frac{f_{ij} + f_{ji}}{d_{ij}} \times d_{\max}$$

The parameter f_{ij} in this equation denotes the frequency of travel from node i to node j over the planning horizon The numerator represents the traffic intensity, the denominator, d_{ij}, the length of edge (i,j), and the parameter $d_{\max}$ the maximum edge length in the graph, that is, $d_{\max} = \max\{d_{ij} : (i,j) \in E\}$. This value is used to normalize the distance d_{ij}. If the demand between two nodes i and j is high and they are close together, then the

corresponding edge weight will have a relatively high value so the two nodes would likely be in the same optimal partition.

P&DCs typically have between 80 and 90 control points, each of equal weight from management's point of view, so we set $w_i = 1, \forall\, i \in V$. In the planning stage, the number of clusters is specified by the facility manager taking into account the daily volume, the building's footprint, the equipment layout, and the various components of the material handling system. In addition to PIVs, which consist of tugs and forklifts, facilities use fixed conveyers, rolling carts, and an assortment of other mechanisms for material handling. Mathematically, the problem is equivalent to model (2).

3.3.2 Random test instances

Instances of practical size cannot be solved optimally with commercial codes so to test our methodology, we randomly generated a series of data sets based on the characteristics of the Chicago P&DC. This involved the following steps.

(1) Let V be the set of control points in the original P&DC data set and let E be the corresponding set of edges. Define the density γ of the underlying graph as

$$\gamma = |E| / |E_C|$$

where $|E_C|$ is the number of edges when the graph is completely connected. For the given data, the density γ is approximately 0.1626. Also, let c_{max} and c_{min} be the maximum and minimum edge weights, respectively; that is, $c_{max} = \max\{c_{ij}, (i, j) \in E\}$ and $c_{min} = \min\{c_{ij}, (i, j) \in E\}$.

(2) Randomly select n^{test} nodes from the original P&DC data set. Let V^{test} be the corresponding set of nodes and fix $w_i = 1, \forall\, i \in V^{test}$. The number of edges in the completely connected test graph is $\left|E_C^{test}\right| = n^{test}(n^{test} - 1)/2$.

(3) Define $m = \left(2 \cdot \gamma \cdot \left|E_C^{test}\right|\right) / n^{test}$. The product $\gamma \cdot \left|E_C^{test}\right|$ is the number of edges in the test graph that should be generated to maintain the same density, and m is the average number of edges incident to each node in the test graph. In our procedure we aim for $\lfloor m \rfloor$ rather than m incident edges. For example, if $n^{test} = 25$ and $\gamma =$

0.1626, then $\left|E_{\mathrm{C}}^{\mathrm{test}}\right| = 300$ and $\lfloor m \rfloor = \lfloor 3.9 \rfloor = 3$, which means that on average each node is connected to 3 other nodes.

(4) Let E^{test} be the edge set of the test instance, where initially, $E^{\mathrm{test}} = \emptyset$. For each node $i \in V^{\mathrm{test}}$, let N_i be the set of nodes already connected to i and to begin, set $N_i = \emptyset$. If $|N_i| \geq \lfloor m \rfloor - 1$, go to next node i in V^{test}. Otherwise, let p_j, $\forall\, j \in V^{\mathrm{test}} \setminus N_i$, $j \neq i$, be the probability for a node j to be connected to node i. This probability is computed as the ratio of the remaining number of nodes to be connected to i divided by the number of unassigned nodes: $p_j = \left(\lfloor m \rfloor - |N_i|\right) \Big/ \left(n^{\mathrm{test}} - |N_i| - 1\right)$. If j is selected, $N_i \leftarrow N_i \cup \{j\}$, $E^{\mathrm{test}} \leftarrow E^{\mathrm{test}} \cup \{(i, j)\}$. The edge weight c_{ij} is uniformly generated from the interval $[c_{\min}, c_{\max}]$.

3.3.3 Comparison of GRASP and PR with CPLEX

In the first experiments, we compared the reactive GRASP to CPLEX using the different phase I and phase II options for instances with $n^{\mathrm{test}} = 30$ nodes and $p = 5$ clusters. Ten instances were randomly generated in accordance with the above scheme with bounds $C^{\min} = 5$ and $C^{\max} = 8$. The model was built with CPLEX 11.0 Concert Technology version 25 and contained 2330 variables and 4386 constraints. The number of GRASP iterations was set to $N^{\mathrm{GRASP}} = 5 \times n^{\mathrm{test}} = 150$, and was followed by PR in all cases with either CLS or PLS. Finally, a 3600 sec time limit was placed on all CPLEX runs.

The results are summarized in Table 3.4. The second column lists the density of the realized graph, $\gamma^{\mathrm{test}} = \left|E^{\mathrm{test}}\right| \Big/ \left|E_{\mathrm{C}}^{\mathrm{test}}\right|$. The third and fourth columns give the results for the two combinations (HWE, CNS) and (CMC, CNS). The upper row values report the best solutions found by GRASP for the corresponding pair. The lower values in parentheses report the iteration number at which the best solutions were first discovered. Columns 5 and 6 give equivalent results for (HWE, VND) and (CMC, VND), while columns 7 and 8 report the results for (HWE, RVND) and (CMC, RVND). With the exception of problem no. 3 for combination (CMC, RVND), GRASP found identical solutions with the various phase I and phase II options. Average runtimes, t_{avg}, over the six scenarios are given in column 9.

The results from PR with CLS are reported in columns 10 and 11 while the results from PR with PLS are reported in columns 12 and 13. It can be seen that PLS achieves the same solutions as CLS but is less time with the exception of problem no. 8. Other than for problem no. 3, PR could not improve the GRASP solutions since they are optimal. This is confirmed by the results from CPLEX given in columns 14 and 15. The last column provides the gap between the PR solution and the CPLEX solution, $[(\phi_{\text{PR}} - \phi_{\text{CPLEX}}) / \phi_{\text{CPLEX}}] \times 100\%$, which is zero for all cases.

Table 3.5, which is derived from Table 3.4, compares the average performance of the six phase I – phase II combinations. For $j = 1,\ldots,6$, let $j = 1$ indicate (HWE, CNS), $j = 2$ indicate (HWE, VND), $j = 3$ indicate (HWE, RVND), $j = 4$ indicate (CMC, CNS), $j = 5$ indicate (CMC, VND), and $j = 6$ indicate (CMC, RVND). Define the average error e_j for combination j as follows

$$e_j = \frac{1}{N^{\text{test}}} \sum_{i=1}^{N^{\text{test}}} \left[\left(\phi_{ij}^{PR} - \phi_i^{GRASP} \right) \Big/ \phi_i^{PR} \times 100\% \right], \ \forall\, j = 1,\ldots,6$$

where $N^{\text{test}} = 10$ is the number of instances, ϕ_{ij}^{GRASP} is the best solution found by GRASP only with combination j for instance i, and ϕ_i^{CPLEX} is the best solution found by CPLEX for the instance. The average error e_j measures the improvement from PR over the GRASP solutions in all instances for combination j. The higher e_j , the more improvement attained with PR.

The second column in Table 3.5 shows the average number of iterations needed to obtain the best solution for (HWE, CNS) and (CMC, CNS), respectively. The values were calculated by averaging the number of iterations inside parentheses in column three for (HWE, CNS) and column four for (CMC, CNS) in Table 3.4. The third column gives the average errors of (HWE, CNS) and (CMC, CNS), respectively. The next columns report the same statistics for the remaining combinations. As mentioned, GRASP found the optimal solutions with $e_j = 0.00$, $\forall\, j = 1,\ldots,5$ except for the last combination (CMC, RVND), $e_6 = 0.02$.

Applying HWE in phase I required a greater number of iterations on average than CMC to find the best solutions no matter which option was applied in phase II. When the

phase I option was fixed, VND required more iterations than its two counterparts, which performed equally well.

The second set of initial experiments was performed on a 40-node graph for $p = 5$ clusters with bounds $C^{\min} = 5$ and $C^{\max} = 9$. Model (2) contained 4105 variables and 7846 constraints for each of the 10 instances investigated, and $N^{\text{GRASP}} = 200$ iterations. Once again, all instances were generated randomly from the USPS data.

The results are summarized in Table 3.6. All the GRASP – PR runs consumed much less time than CPLEX as can be seen in columns 9, 11 and 13. For problem nos. 1, 3, 4, 5, 7 and 8, CPLEX converged to the optimum; for the remaining instances the 1-hour time limit was reached before optimality could be confirmed. For the GRASP, PR improved the phase II solutions in some cases, especially when VND was applied. In all cases, CLS and PLS achieved identical solutions but PLS required slightly less time. In addition, GRASP with PR invariably provided equivalent or better solutions than CPLEX in much less of time.

The average performance of the phase I – phase II combinations for the 40-node instances is reported in Table 3.7. For a given phase I option, RVND required the least number of iterations, followed by CNS and then VND. In addition, the average error was highest for VND, while the errors for CNS and RVND were roughly the same. When either CNS or VND was applied in phase II there was little difference with respect to HWE and CMC. However, when RVND was applied, CMC required 37% fewer iterations than HWE on average.

The third set of initial experiments was performed on a 50-node graph with $p = 5$, $C^{\min} = 5$ and $C^{\max} = 12$. The optimization model contained 6380 variables and 12306 constraints, while the number of GRASP iterations $N^{\text{GRASP}} = 250$. Again, ten instances were randomly generated from the original USPS data following the aforementioned scheme.

The results are reported in Table 3.8. All of the GRASP runs finished within 10 sec while the PR runs finished within 30 sec. In all cases, the PR solutions were better than those provided by CPLEX except for problem no. 5 where they were identical. Note that CPLEX was never able to converge but always found feasible solutions within the allotted time.

Table 3.4 Computational results from GRASP and CPLEX for $n^{test} = 30$, $p = 5$, $C^{min} = 5$ and $C^{max} = 8$

Prob no.	γ^{test}	GRASP											CPLEX		Opt gap (%)
		GRASP solution							PR solution				Best solution found	Time (sec)	
		CNS		VND		RVND		t_{avg} (sec)	PR+CLS best	t_{avg} (sec)	PR+PLS best	t_{avg} (sec)			
		HWE	CMC	HWE	CMC	HWE	CMC								
1	0.1908	618.97 (1)	618.97 (12)	618.97 (7)	618.97 (12)	618.97 (12)	618.97 (11)	1	618.97	5	618.97	3	618.97	10	0.00
2	0.1977	691.49 (27)	691.49 (2)	691.49 (26)	691.49 (15)	691.49 (16)	691.49 (19)	1	691.49	4	691.49	3	691.49	35	0.00
3	0.1747	667.09 (18)	667.09 (3)	667.09 (18)	667.09 (3)	667.09 (3)	665.48 (7)	1	667.09	4	667.09	3	667.09	21	0.00
4	0.1839	680.54 (6)	680.54 (12)	680.54 (37)	680.54 (17)	680.54 (1)	680.54 (14)	1	680.54	4	680.54	4	680.54	13	0.00
5	0.1609	569.67 (3)	569.67 (8)	569.67 (3)	569.67 (1)	569.67 (34)	569.67 (6)	1	569.67	7	569.67	2	569.67	18	0.00
6	0.1793	642.67 (5)	642.67 (1)	642.67 (3)	642.67 (8)	642.67 (4)	642.67 (1)	1	642.67	4	642.67	1	642.67	29	0.00
7	0.1678	618.25 (5)	618.25 (1)	618.25 (15)	618.25 (11)	618.25 (22)	618.25 (8)	1	618.25	3	618.25	3	618.25	6	0.00
8	0.1609	628.91 (1)	628.91 (19)	628.91 (3)	628.91 (10)	628.91 (4)	628.91 (2)	1	628.91	2	628.91	4	628.91	7	0.00
9	0.1540	544.62 (38)	544.62 (13)	544.62 (52)	544.62 (74)	544.62 (21)	544.62 (2)	1	544.62	3	544.62	1	544.62	5	0.00
10	0.1609	598.66 (3)	598.66 (14)	598.66 (6)	598.66 (5)	598.66 (4)	598.66 (2)	1	598.66	4	598.66	2	598.66	4	0.00

Table 3.5 Average performance for different phase I and phase II combinations with n^{test} = 30, p = 5, C^{min} = 5 and C^{max} = 8

	Phase II					
	CNS		VND		RVND	
Phase I	Avg. number of iter.	Avg. error e_j (%)	Avg. number of iter.	Avg. error e_j (%)	Avg. number of iter.	Avg. error e_j (%)
HWE	10.7	0.00	17.0	0.00	12.1	0.00
CMC	8.5	0.00	15.6	0.00	7.2	0.02

The average performance of GRASP is reported in Table 3.9 for the 50-node instances. The average error $e_j > 0$, $j = 1,\ldots,6$, which means that PR found improved solutions for all combinations. When the phase I option was fixed, VND had the highest error, followed by CNS and RVND. For CNS and VND in phase II, the errors from HWE and CMC were nearly identical. When RVND was applied, the error from HWE was less than half of the error from CMC. With respect to the average number of iterations, VND and RVND performed equally well, while CNS required the least number of iterations no matter which option was used in phase I.

In the fourth set of initial experiments we investigated the performance of GRASP and PR as the number of clusters p was varied from 2 to 10 for the same 30-node graph associated with problem no. 1 in Table 3.4. The bounds were set to be C^{min} = 2 and C^{max} = 15 to reduce their effect on the computations. The results were reported in Table 3.10 and were similar to those already discussed. In all cases, the optimal solution was found by GRASP and CPLEX, but runtimes differed markedly. GRASP with PR were quite stable no matter which combination was used but CPLEX had increasing difficulty as the number of clusters increased.

In the fifth set of initial experiments, a parametric analysis was performed on the bounds for p = 5 fixed. The bounds [C^{min}, C^{max}] were initially set to [2, 10] and then modified in even steps to reach [6, 6]. The results were reported in Table 3.11. For all runs, GRASP with PR was able to find the same optimum obtained by CPLEX but in considerably less time. Similar to the fourth set of experiments, the performance of GRASP was insensitive to the bounds while CPLEX had more difficulty as the range shrank.

Table 3.6 Computational results from GRASP and CPLEX for n^{test} = 40, p = 5, C^{min} = 5 and C^{max} = 9

Prob no.	γ^{test}	GRASP											CPLEX		Opt gap (%)
		GRASP solution							PR solution				Best solution found	Time (sec)	
		CNS		VND		RVND		t_{avg} (sec)	PR+CLS best	t_{avg} (sec)	PR+PLS best	t_{avg} (sec)			
		HWE	CMC	HWE	CMC	HWE	CMC								
1	0.1782	1043.77 (2)	1043.77 (13)	1043.77 (41)	1043.77 (4)	1043.77 (17)	1043.77 (15)	1	1043.77	10	1043.77	9	1043.77	693	0.00
2	0.2013	1127.87 (72)	1127.83 (15)	1127.87 (16)	1127.87 (1)	1127.87 (11)	1127.87 (45)	1	1127.87	10	1127.87	7	1127.87	3600	0.00
3	0.1923	1169.84 (14)	1169.84 (29)	1169.84 (182)	1158.21 (54)	1169.84 (44)	1169.84 (8)	1	1169.84	16	1169.84	7	1169.84	3260	0.00
4	0.1910	1148.93 (73)	1148.93 (42)	1148.93 (144)	1148.93 (155)	1148.93 (121)	1148.93 (20)	1	1148.93	9	1148.93	7	1148.93	1073	0.00
5	0.2026	1124.71 (104)	1125.80 (128)	1125.80 (46)	1125.80 (52)	1125.80 (10)	1125.80 (47)	3	1125.80	11	1125.80	8	1125.80	518	0.00
6	0.2090	1154.98 (77)	1158.93 (60)	1158.93 (134)	1158.93 (73)	1158.93 (145)	1158.93 (37)	4	1158.93	13	1158.93	11	1154.98	3600	0.34
7	0.1872	1065.31 (7)	1065.31 (61)	1059.64 (120)	1062.86 (191)	1065.31 (12)	1064.30 (23)	1	1065.31	9	1065.31	9	1065.31	1420	0.00
8	0.1846	1166.05 (11)	1166.05 (41)	1166.05 (11)	1166.05 (79)	1166.05 (2)	1166.05 (31)	1	1166.05	13	1166.05	9	1166.05	702	0.00
9	0.1833	1096.67 (44)	1096.67 (13)	1096.67 (36)	1096.67 (16)	1096.67 (6)	1096.67 (15)	1	1096.67	7	1096.67	5	1096.67	3600	0.00
10	0.1846	1150.02 (6)	1150.02 (30)	1150.02 (6)	1150.02 (9)	1150.02 (14)	1150.02 (7)	1	1150.02	10	1150.02	8	1150.02	3600	0.00

Table 3.7 Average performance for different phase I and phase II combinations with n^{test} = 40, p = 5, C^{min} = 5 and C^{max} = 9

	Phase II					
	CNS		VND		RVND	
Phase I	Avg. number of iter.	Avg. error e_j (%)	Avg. number of iter.	Avg. error e_j (%)	Avg. number of iter.	Avg. error e_j (%)
HWE	41.0	0.01	73.6	0.05	38.2	0.00
CMC	43.2	0.00	63.4	0.12	24.8	0.01

3.3.4 Application of GRASP and PR to the complete USPS dataset

In the second set of tests, we applied GRASP with PR using PLS to the full USPS dataset, which has 82 nodes and 540 edges (i.e., $\left|V^{test}\right|$ = 82 and $\left|E^{test}\right|$ = 540). The bounds were set as follows: C^{min} = 10 and C^{max} = 20. The goal was to investigate the six combinations of phase I and phase II options for a range of p values. In each run, the number of GRASP iterations N^{GRASP} = 410.

The results are reported in Table 3.12 for $p \in \{5, 6, 7, 8\}$. The first column gives the number of clusters p. The second column indicates the options used for GRASP. The third column, ϕ^{best}, is the best solution found by GRASP and PR. The fourth column indicates the improvement achieved by PR. In column 5, the value i^{best} denotes the iteration at which the best solution was first found by GRASP. If $i^{best} = N^{GRASP}$, the best solution was found by PR. The column labeled t^{best} reports the amount of time spent to find the best solutions while the column $t^{overall}$ gives the combined runtime of GRASP and PR. For problems with 5, 6 or 8 clusters, the same objective function values were found for all six options. For p = 7, the best solutions were found under different combinations. In 12 out of the 18 instances associated with the datasets for p = 5, 7, 8 clusters, PR improved the GRASP solutions by up to 1.15%; for p = 6, PR offered no improvement. In all cases, runtimes were well under 200 sec.

Table 3.8 Computational results from GRASP and CPLEX for $n^{test} = 50$, $p = 5$, $C^{min} = 5$ and $C^{max} = 12$

Prob no.	γ^{test}	GRASP											CPLEX		Opt gap (%)
		GRASP solution							PR solution				Best solution found	Time (sec)	
		CNS		VND		RVND		t_{avg} (sec)	PR+CLS best	t_{avg} (sec)	PR+PLS best	t_{avg} (sec)			
		HWE	CMC	HWE	CMC	HWE	CMC								
1	0.1845	1639.22 (59)	1634.33 (5)	1631.77 (244)	1633.35 (155)	1639.22 (39)	1639.22 (30)	1	1639.22	20	1639.22	19	1602.56	3600	2.29
2	0.1869	1685.68 (176)	1685.68 (98)	1685.68 (27)	1685.68 (40)	1685.68 (139)	1685.68 (11)	5	1685.68	11	1658.68	18	1646.34	3600	2.39
3	0.1714	1560.56 (155)	1550.17 (80)	1549.55 (143)	1550.17 (37)	1560.56 (43)	1556.19 (238)	5	1560.56	25	1560.56	12	1491.93	3600	4.60
4	0.1706	1677.50 (4)	1679.90 (29)	1678.02 (20)	1678.02 (125)	1679.90 (67)	1679.90 (7)	7	1679.90	29	1679.90	32	1647.10	3600	1.99
5	0.1665	1637.83 (4)	1640.17 (90)	1640.17 (32)	1640.17 (180)	1640.17 (67)	1640.17 (198)	7	1640.17	23	1640.17	13	1640.17	3600	0.00
6	0.1837	1658.19 (55)	1662.09 (203)	1658.19 (148)	1658.19 (37)	1662.09 (212)	1658.43 (246)	2	1662.09	20	1658.19	10	1615.09	3600	2.91
7	0.1796	1670.81 (47)	1670.81 (182)	1669.95 (161)	1670.81 (103)	1670.81 (56)	1670.81 (169)	1	1670.81	12	1670.81	9	1624.56	3600	2.85
8	0.1682	1640.90 (57)	1646.57 (104)	1644.53 (63)	1646.57 (160)	1646.57 (206)	1646.57 (144)	7	1651.09	25	1651.09	11	1594.65	3600	3.54
9	0.1641	1648.39 (12)	1648.39 (66)	1648.39 (175)	1648.39 (204)	1648.39 (29)	1648.39 (42)	1	1648.39	11	1648.39	12	1643.22	3600	0.31
10	0.1747	1690.22 (60)	1690.22 (22)	1690.22 (72)	1671.89 (67)	1690.22 (207)	1690.22 (25)	2	1690.22	21	1690.22	13	1650.11	3600	2.43

Table 3.9 Average performance for different phase I and phase II combinations with n^{test} = 50, p = 5, C^{min} = 5 and C^{max} = 12

	Phase II					
	CNS		VND		RVND	
Phase I	Avg. number of iter.	Avg. error e_j (%)	Avg. number of iter.	Avg. error e_j (%)	Avg. number of iter.	Avg. error e_j (%)
HWE	62.9	0.11	108.5	0.20	106.5	0.03
CMC	87.9	0.12	110.8	0.27	111.0	0.08

3.3.5 GRASP performance on the benchmark problems

In the final set of experiments, GRASP with PR was applied to a set of six benchmark instances with known optimal values. The datasets were provided by Mehrotra and Trick (1998) who solved each of them on a DEC ALPHA 3000 (Model 300) workstation with 150 MHz Alpha 21064 CPU using CPLEX 2.1 as the linear programming solver. The largest dataset contains 61 nodes and 187 edges. In their runs the number of clusters was not specified, so to duplicate that scenario, we set p = 12, a high enough value to ensure that we would always have a sufficient number of clusters.

The results for option (HWE, RVND) with PR and PLS activated are reported in Table 3.13 along with the optimal solutions for two sets of runs, the first with C^{min} = 0, C^{max} = 450 and the second with C^{min} = 0, C^{max} = 512. The number of iterations, N^{GRASP}, was set to 250 in all cases. Our best solutions ϕ^{best} are given in columns 4 and 10, the iteration number at which the best solution i^{best} was first encountered is given in columns 5 and 11, the corresponding runtimes t^{best} are given in columns 6 and 12, and the total runtimes $t^{overall}$ are given in columns 7 and 13.

From the table, we can see that our methodology finds the exact optimum in all cases except the last in considerably less time than reported by Mehrotra and Trick. This, of course, is not surprising since we are using a much faster machine. Scaling runtimes, however, indicates that their algorithm is competitive with ours and so may be preferred for instances of the size investigated their study since it guarantees optimality. Nevertheless, it is difficult to compare performance of exact and heuristic methodologies, especially across different platforms. To a large extent the computational effort of any metaheuristic such as GRASP is proportional to the number of iterations, N^{GRASP} here,

specified at the outset. A final point about the results is that PR only improved the GRASP solution for the largest instance.

Table 3.10 Computational results from GRASP and CPLEX for $n^{test} = 30$, $C^{min} = 2$ and $C^{max} = 15$

Number of clusters p	GRASP									CPLEX		Optimality gap (%)
	GRASP soln							PR soln		Best soln found	Time (sec)	
	CNS		VND		RVND		Time (sec)	PR optimum	Time (sec)			
	HWE	CMC	HWE	CMC	HWE	CMC						
2	824.26 (5)	824.26 (1)	824.26 (5)	824.26 (1)	824.26 (8)	824.26 (1)	< 1	824.26	< 1	824.26	< 1	0.00
3	797.22 (2)	797.22 (11)	797.22 (5)	797.22 (11)	797.22 (3)	797.22 (6)	< 1	797.22	1	797.22	< 1	0.00
4	769.15 (1)	769.15 (1)	769.15 (12)	769.15 (45)	769.15 (2)	769.15 (1)	< 1	769.15	1	769.16	3	0.00
5	734.15 (12)	734.15 (68)	730.01 (75)	730.71 (111)	734.15 (81)	733.85 (17)	< 1	734.15	2	734.15	9	0.00
6	709.38 (12)	709.38 (30)	709.38 (10)	709.38 (24)	709.38 (31)	709.38 (10)	1	709.38	2	709.38	24	0.00
7	685.51 (1)	685.51 (19)	685.51 (46)	685.51 (2)	685.51 (8)	685.51 (9)	< 1	685.51	1	685.51	48	0.00
8	655.70 (27)	655.70 (1)	655.70 (21)	655.70 (64)	655.70 (7)	655.70 (1)	< 1	655.70	1	655.70	59	0.00
9	611.23 (4)	611.23 (27)	611.23 (1)	603.96 (125)	611.23 (6)	611.23 (31)	< 1	611.23	1	611.23	627	0.00
10	549.02 (65)	549.02 (21)	549.02 (54)	538.07 (32)	549.02 (16)	549.02 (22)	< 1	549.02	3	549.02	700	0.00

Table 3.11 Computational results from GRASP and CPLEX for $n^{test} = 30$, $p = 5$

Capacity lower bound C^{min}	Capacity upper bound C^{max}	GRASP									CPLEX		Optimality gap (%)
		GRASP soln							PR soln		Best soln found	Time (sec)	
		CNS		VND		RVND		Time (sec)	PR optimum	Time (sec)			
		HWE	CMC	HWE	CMC	HWE	CMC						
2	10	679.76 (50)	673.61 (80)	672.34 (22)	679.76 (64)	679.76 (11)	679.76 (20)	1	679.76	4	679.76	11	0.00
3	9	654.50 (126)	654.50 (32)	647.12 (28)	654.50 (150)	654.50 (14)	654.50 (11)	< 1	654.50	5	654.50	12	0.00
4	8	628.73 (1)	628.73 (12)	628.73 (1)	628.73 (8)	628.73 (10)	628.73 (13)	< 1	628.73	4	628.73	31	0.00
5	7	603.34 (28)	603.34 (15)	603.34 (38)	603.34 (15)	603.34 (11)	603.34 (10)	< 1	603.34	5	603.34	66	0.00
6	6	582.37 (113)	582.37 (36)	585.28 (123)	582.37 (99)	582.37 (22)	582.37 (48)	< 1	585.28	4	585.28	70	0.00

Table 3.12 Computational results from GRASP and PR with PLS for complete USPS dataset with $n^{test} = 82$, $C^{min} = 10$ and $C^{max} = 20$

No. of clusters p	Phase I	Phase II	GRASP & PR+PLS				
			ϕ^{best}	PR Impr. (%)	i^{best}	t^{best} (sec)	$t^{overall}$ (sec)
5	HWE	CNS	1577.03	0.00	116	22	99
	HWE	VND	1577.03	0.02	410*	69	127
	HWE	RVND	1577.03	0.02	264	57	135
	CMC	CNS	1577.03	0.00	41	6	93
	CMC	VND	1577.03	0.00	410*	70	96
	CMC	RVND	1577.03	0.00	18	5	113
6	HWE	CNS	1540.76	0.00	228	34	91
	HWE	VND	1540.76	0.00	185	27	79
	HWE	RVND	1540.76	0.00	228	52	140
	CMC	CNS	1540.76	0.00	44	7	91
	CMC	VND	1540.76	0.00	3	1	83
	CMC	RVND	1540.76	0.00	64	15	112
7	HWE	CNS	1371.57	1.15	410*	91	104
	HWE	VND	1367.32	1.02	410*	64	89
	HWE	RVND	1371.57	1.15	410*	89	158
	CMC	CNS	1371.57	0.99	410*	107	138
	CMC	VND	1371.57	0.31	410*	55	88
	CMC	RVND	1355.97	0.00	92	17	142
8	HWE	CNS	1148.66	0.36	410*	80	121
	HWE	VND	1148.66	0.52	410*	96	120
	HWE	RVND	1148.66	0.64	410*	85	126
	CMC	CNS	1148.66	0.00	227	34	80
	CMC	VND	1148.66	0.26	410*	62	88
	CMC	RVND	1148.66	0.13	410*	74	117

*For these instances, the best solution was found by PR in the post-processing stage after 410 GRASP iterations.

Table 3.13 GRASP performance on benchmark problem

			$C^{min} = 0$, $C^{max} = 450$						$C^{min} = 0$, $C^{max} = 512$					
			GRASP & PR+PLS				Mehrotra & Trick		GRASP & PR+PLS				Mehrotra & Trick	
Graph	$\|V\|$	$\|E\|$	ϕ^{best}	i^{best}	t^{best} (sec)	$t^{overall}$ (sec)	Soln	Time (sec)	ϕ^{best}	i^{best}	t^{best} (sec)	$t^{overall}$ (sec)	Soln	Time (sec)
1	45	98	2928	13	1	21	2928	46.9	3238	5	< 1	29	3238	79.5
2	30	56	1642	7	1	6	1642	3.0	1748	3	< 1	5	1748	3.3
3	47	101	3569	27	1	23	3569	139.5	3960	6	< 1	33	3960	115.8
4	47	99	1837	2	< 1	32	1837	201.7	1993	4	< 1	32	1993	438.4
5	30	47	1099	2	< 1	9	1099	2.5	1174	1	< 1	8	1174	3.0
6	61	187	22,216	250*	26	84	22,216	352.4	23,552	56	6	77	23,564	394.4

*Best solution was found by PR in the post-processing stage after 250 GRASP iterations.

3.4 Further Discussion

The reactive GRASP presented in this chapter was designed to find high quality solutions to the *p*-capacitated clustering problem. In phase I, two efficient approaches (HWE and CMC) are presented for constructing partial initial solutions, and a dynamic restricted candidate list is proposed to then obtain feasible solutions. In the improvement phase, three neighborhoods, i.e., CNS, VND and RVND, are considered for the local search. In a post-processing step, PR is applied to overcome local optimality and to attempt to uncover even better solutions. Both CLS and PLS are implemented with PR. All components of the methodology were extensively tested on a number of instances of practical size. According to the results, HWE and CMC were comparable when combined with CNS in VND, while HWE combined with RVND gave the best overall performance. However, the runtimes of the latter pair where slightly above the runtimes of the other combinations. During PR, PLS and CLS provide similar results with the latter being a bit more efficient.

In all instances tested, GRASP provided the same or better solutions than CPLEX. These results offer some assurance that GRASP can find high quality solutions when optimality cannot be established. As the number of nodes, edges, and clusters increase, the difficulty in solving model (2) optimally increases as well, often exponentially. For the 50-node, 5-cluster instances, CPLEX failed to converge within the imposed 1-hour time limited so we cannot be sure that the solutions found by our methodology are optimal. Nevertheless, that fact that we were able to optimally solve all the benchmark problems, further attests to its effectiveness.

With respect to path relinking, we can confirm the mixed results reported by others such as Boudia et al. (2006) who have performed similar analyses. For relatively small problem instances, PR offered no advantage here since GRASP was able to find the optimum without it. For larger instances, including the USPS application and some of the benchmark datasets, post-processing the GRASP solutions led to slightly better objective function values. However, the improvement was rarely significant, so it is arguable whether the procedure is justified even when using PLS instead of CLS.

Chapter 4

Midterm Planning to Minimize Deviations from Daily Target Outputs in Semiconductor Manufacturing

The TI fab known as DMOS6 is a mixed-signal analog wafer fab and probe facility producing about 2000 active devices grouped into three technologies that run on one integrated 300-mm manufacturing line (Chacon et al. 2005). To clarify terminology, a "device" is a specific term in the Manufacturing Execution System (MES) and is basically the same as a product. However, a few different devices may derive from the manufacture of the same product by using slightly different operations. For example, a product can be split into two devices such that only one has inspection steps for particle detection, but in the end they are the same product.

Wafer starts average 20,000 per month and reflects a highly diverse product mix. Demand can be characterized as high-mix low-volume and very complex. Although only 5% of the demand is explicitly for low volume technologies, the demand for a majority of devices within each high volume technology may also be low volume. As a result, it is necessary to run many different products simultaneously. This adds to the already challenging problem of managing daily operations, which includes ensuring that production targets are met, minimizing WIP, re-scheduling starts, reducing cycle time, reacting to disturbances in real time, and reducing flow variability.

At DMOS6, planning and scheduling is done hierarchically as depicted in Figure 4.1 (this is a common approach throughout the industry; e.g., see Stray et al. 2006). The box labeled Supply Chain represents the outside world from which orders are received. These are passed to Fab Planning where quarterly and monthly production plans are constructed to balance customer priorities with capacity. Manufacturing Planning is responsible for the day-to-day activities, scheduling starts over the month, setting targets by operation for each product being manufactured, and deciding when to move WIP between operations. Decisions at this level are made by shop floor managers and line supervisors. Dispatch and Execution is the recipient of daily target data and is

responsible for ensuring that the scheduled work is carried out. Decisions at this level fall to the equipment operators who sequence lots and form batches at the various machine groups.

Figure 4.1 Hierarchical planning and scheduling at Texas Instruments

By carefully monitoring system progress, line supervisors can determine whether daily targets are being met. Oversight is facilitated with a decision support system that tracks performance measures such as throughput, WIP, and cycle time by operation and product. These statistics are aggregated by log point (a stage in production such as photolithography), loop (a group of sequential operations in the process flow within a technology) and line to allow higher-level managers to better visualize trends. At the operations level, deviations from target values are flagged and appropriate action is taken when a significant gap exists. This may include reconfiguring machines that do not typically process wafers at the bottleneck operations to do so, deprioritizing some of the work that feeds the bottlenecks, and delaying the start of certain lots.

Shop floor data are collected in real time and fed to the information network. Differences between planned and actual WIP are calculated for each operation and passed

to Manufacturing Planning who is responsible for recovering the schedule. Daily and weekly statistics on production volumes, delays and disruptions are passed up the hierarchy to Fab Planning to be used to refine their models. As lots flow off the line, commitments to customers are confirmed at the Supply Chain level. Similar hierarchical approaches have been discussed by Rivera (2003), Zäpfel and Missbauer (1993) and others.

The problem addressed in this chapter falls in the domain of Manufacturing Planning. For a given number of wafer starts per day and a set of output targets by product, the primarily goal is to develop a model that can be used to determine when to process wafers at each operation in their routing to ensure that those targets are met. A related use of the model is to help managers recover from disruptions.

Whether the model is used for daily planning or recover, a solution should detail the degree to which individual targets and overall demand can be met and the level of WIP in the system at each machine group by log point and operation. The principle objective is to minimize the weighted sum of the deviations from the daily production targets subject to capacity limits, predetermined inductions, product routings, and material flow conservation. This will tend to smooth production and keep the fab running at an even rate (or fixed percentage of capacity), a cornerstone of the lean philosophy (Yavuz and Akcali 2007). An appropriate mix of products at different stages of completion must be maintained at the bottleneck machines to obtain a consistent level of output from the facility.

In the next section, the optimization model is presented and followed in Section 4.3 by a discussion of the data processing issues necessary for implementation. In Section 4.4 the initial computational experience is highlighted, which implied a need for further algorithmic development. Subsequently, both Lagrangian relaxation and Benders decomposition were tried but neither proved successful. Related experience is summarized in Section 4.5. As an alternative, a decomposition algorithm is developed with the description in Section 4.6. This is followed in Section 4.7 by a comprehensive set of test results based on DMOS6 data. The data has been scaled based on the original data to avoid revealing TI's true production. The effectiveness of the approach is discussed and future work is described in Section 4.8.

4.1 Mathematical Model

Production planning in a fab can be modeled as a multicommodity dynamic network problem. The objective of the corresponding linear program (LP) is to schedule wafer movement so that the total deviations are minimized. The notation used in the developments is as follows.

Indices and sets

i	index for devices; $i \in I$
j	index for steps in the processing of a device; $j \in J(i) \cup \{n(i)+1\}$
m	index for machines; $m \in M$
d	index for days; $d \in D = \{1,\ldots,n^D\}$
t	index for time periods; $t = 1,\ldots,\tau$
I	set of devices
$J(i)$	set of steps for device i
M	set of machines
$G(i, j)$	set of machines that can process device i at step j
D	set of days in planning horizon
T	set of time periods; $T = \{1, 2,\ldots,\tau^D, \tau^D+1, \tau^D+2, \ldots, 2\tau^D,.., \tau\}$
T^D	set of time periods in a day; $T^D = \{1, 2,\ldots,\tau^D\}$

Parameters and data

Δt	time interval (indicates the number of minutes within a time period)
τ	planning horizon (in units of periods); $\tau = \|T\|$
τ^D	last period in a day; $\tau^D = \|T^D\|$
$n(i)$	number of steps in the route of device i [$n(i)+1$ is dummy step for device i associated with holding finished goods inventory]
r_{ijm}	(effective) processing rate for machine m when working on device i at step j (wafers/ min)
$STARTS_{id}$	number of wafer starts for device i on day d
T_OUT_{id}	target number of wafers to produce for device i on day d
$\delta_{ij}^1(t)$	0 if $j = n(i)+1$ and $t = d\tau^D + 1$ for device i; 1 otherwise

δ_{ij}^2 1 if $j = 1$ for device i; 0 otherwise

δ_{ij}^3 0 if $j = n(i)+1$for device i; 1 otherwise

$\delta_{ij}^4(t)$ 1 if $j = 1$ and $t = (d-1)\tau^D + 1$ for device i; 0 otherwise

w_i^+ (w_i^-) relative weight associated with a positive (negative) deviation from the target output for device i

$w_{\max}$ penalty weight for the maximum deviation from the target output over the planning horizon

Decision variables

$W_{ij}(t)$ number of wafers (WIP) corresponding to device i in the jth buffer (step) in its routing at the end of time period t, $\forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, t$; $W_{i,n(i)+1}(t)$ represents finished goods inventory

$R_{id}(t)$ number of wafers input to the fab of device i at the beginning of time period t on day d, $\forall\ i \in I, t \in T$ and $d \in D$

$\beta_{ijm}(t)$ fraction of the time machine m is processing device i at step j in its route in time period t, $\forall\ i \in I, j \in J(i), m \in G(i, j), t \in T$

Δ_{id}^+ (Δ_{id}^-) positive (negative) deviation from the target output for device i on day d, $\forall\ i \in I$, $d \in D$

$\Delta_{\max}$ maximum deviation from target output over the planning horizon

Model

$$\text{Minimize} \quad \sum_{i\in I}\sum_{d\in D}(w_i^+\Delta_{id}^+ + w_i^-\Delta_{id}^-) + w_{\max}\Delta_{\max} \tag{3a}$$

$$\text{subject to} \quad \sum_{t\in T^D} R_{id}(t) = STARTS_{id}, \quad \forall\ i \in I, d \in D \tag{3b}$$

$$\delta_{ij}^1(t)\, W_{ij}(t-1) + (1-\delta_{ij}^2) \sum_{m\in G(i,j-1)} \beta_{i,j-1,m}(t) r_{i,j-1,m}\Delta t + \delta_{ij}^4(t)\, R_{i,\lceil t/\tau^D \rceil}(t)$$

$$-\delta_{ij}^3 \sum_{m\in G(i,j)} \beta_{ijm}(t) r_{ijm}\Delta t = W_{ij}(t), \quad \forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, t \in T \tag{3c}$$

$$\sum_{i\in I}\sum_{j\in J(i)} \beta_{ijm}(t) \le 1, \quad \forall\ m \in M, t \in T \tag{3d}$$

$$W_{i,n(i)+1}(d\tau^D) - \Delta_{id}^+ + \Delta_{id}^- = T\_OUT_{id} + \Delta_{i,d-1}^-, \quad \forall\ i \in I, d \in D \tag{3e}$$

$$\Delta_{\max} \geq \Delta_{id}^{+} + \Delta_{id}^{-}, \quad \forall\, i \in I, d \in D \tag{3f}$$

$W_{ij}(t) \geq 0,\ \beta_{ijm}(t) \geq 0,\ \Delta_{id}^{+} \geq 0,\ \Delta_{id}^{-} \geq 0,\ \Delta_{i0}^{-} = 0,\ \Delta_{\max} \geq 0,\ W_{ij}(0)$ given,

The first term in the objective function (3a) is designed to minimize the weighted sum of the positive and negative deviations from the target outputs over the planning horizon. When the objective is to minimize only the negative gaps, this can be achieved by setting $w_i^{+} = 0$ for all $i \in I$. The second term is aimed at minimizing the maximum deviation, which is a surrogate for maintaining output in proportion to demand. The variable $\Delta_{\max}$ is the maximum deviation from the target outputs and is determined by constraints (3f); $w_{\max}$ is the associated penalty.

The first set of constraints (3b) ensures that the required number of wafers for device i are started on day d. The second set of constraints (3c) represents inventory (material) conservation at each step j at the end of period t. It is necessary to keep track of the inventory levels of each device i separately. Finished goods are held as WIP at the dummy step $n(i) + 1$ and takes the value $W_{i,n(i)+1}(t)$ for device i at time t. They are removed from the system at the end of the day by the term $\delta_{ij}^{1}(t)\, W_{ij}(t-1)$ in (3c) by noting that $\delta_{i,n(i)+1}^{1}(t) = 0$ for $t = d\tau^{D} + 1$, i.e., the first period of each day. With respect to starts, the parameter $\delta_{ij}^{4}(t)$ multiplying $R_{i,\lceil t/\tau^{D} \rceil}(t)$ in (3c) takes the value of 1 only when j corresponds to the first step for device i in its routing and t corresponds to the first time period of the day. The index $\lceil t/\tau^{D} \rceil$ identifies the day d that includes time period t, where $\lceil x \rceil$ is the smallest integer greater than or equal to x.

In fractional terms, constraints (3d) ensure that the sum of the time devoted to each step j of device i does not exceed the available time for each machine m in each time period t. Deviations from production targets are tracked by the equations in (3e). On the right-hand side, T_OUT_{id} specifies the target outputs for device i on day d while $\Delta_{i,d-1}^{-}$ specifies the shortages from the previous day. The first variable $W_{i,n(i)+1}(d\tau^{D})$ on the left-hand side represents the amount of finished goods inventory of device i at the end of day d. The argument $d\tau^{D}$ corresponds to the last period of day d. The remaining two

variables account for the deviation from the target with Δ_{id}^{+} indicating surplus and Δ_{id}^{-} indicating shortage. Logically, only one of these variables can be positive in a solution so $\Delta_{id}^{+} \times \Delta_{id}^{-} = 0$ for all *i* and *d*. As mentioned, constraints (3f) are used to obtain the maximum deviation from the target output of any device over the planning horizon. To conclude the model, we note that all of the variables are nonnegative and continuous, as indicated in (3g), and that $W_{ij}(0)$ must be given for all *i*, *j*.

4.2 Data Processing

A massive quantity of data is needed to initialize and solve model (3). A description of the input and output files can be found in Appendix 1. In the fab, machines operate in one of the following modes: batch wafer (BW), batch lot (BL), continuous wafer (CW), inspection lot (IL), outside inspection (O) and pipeline (PL). Formulas were provided by the semiconductor manufacturer to compute the effective processing rates r_{ijm} in each case.

Computing the effective processing rates

Although a machine typically has a fixed processing rate for a particular device at some step in its routing, several additional factors must be taken into account when specifying r_{ijm} in a planning model. In the fab, machines operate in one of the following modes: batch wafer (BW), batch lot (BL), continuous wafer (CW), inspection lot (IL), outside inspection (O) and pipeline (PL). To calculate r_{ijm}, the "*K*-parameters" given below are used to modify the basic rate.

a. The machine processing mode (*K*803)
b. The capacity multiplier/loading factor (*K*852); this parameter has the effect of amplifying the processing rate.
c. Non-machine processing time (*K*817); although this value is always zero in the table, it is included in the formulation for completeness.
d. Load/unload time. Sometimes this component is included in machine processing time, c.
e. Conditional setup time (*K*830)
f. Average number of wafers/lots, average number of lots/batch processed

g. Rework rate (*K*802)

h. Machine uptime (utilization) percentage (*K*866)

i. Most likely turn-around time (hours) (*K*819); time from start to finish of a lot

(1) BW Processing Mode

$$r_{ijm} = \frac{average\ num\ of\ \ wafers/batch}{K812 + K830 + K817} \times K852 \times (1 - K802) \times K866$$

Because no data were provided on the average number of wafers/batch this value was estimated at 25 based on the fact that the maximum batch size *K*835 is usually 50 while the minimum size *K*836 is usually 1.

(2) BL Processing Mode

$$r_{ijm} = \frac{(average\ num\ of\ \ wafers\ per\ \ lot) \times (average\ num\ of\ \ lots\ per\ \ batch)}{K875 + K830 + K817} \times K852 \times (1 - K802) \times K866$$

Again we use 25 as the average number of wafers per lot and estimate the average number of lots per batch as *K*873 × *batch factor*, where K873 is the maximum lot size *K*873 and the batch factor is currently set to be 0.9. This gives

$$r_{ijm} = \frac{25 \times K873 \times batch\ \ factor}{K875 + K830 + K817} \times K852 \times (1 - K802) \times K866$$

(3) CW Processing Mode

This mode is associated with a machine group which acts as stand-in for a non-processing operation. Because the given base processing rates are disproportionally high, the corresponding steps were eliminated from the scheduling model.

(4) IL Processing Mode

Not all lots are selected for inspection and not all wafers in a selected lot are inspected. Sampling is done at both steps. If the lost selection interval parameter

$K850 < 1$, then this value is the probability that a lot will be selected. In this case, the number of wafers that goes through the inspection I

$$number\ of\ wafers = 1.0/K850 \times average\ number\ of\ wafers\ per\ lot$$

If $K850 \geq 1$, then it means that the $K850^{th}$ lot is selected. In this case, the number of wafers that goes through the inspection is

$$number\ of\ wafers = K850 \times average\ number\ of\ wafers\ per\ lot$$

The parameter $K851$ indicates the number of wafers chosen for inspection within the selected lot. If $K851 < 1$, then $K851$ percent of the wafers in the lot are inspected. In this case, the amount of time for inspection is

$$time = average\ number\ of\ wafers\ per\ lot \times K851 \times K824$$

If $K851 \geq 1$, then this many wafers in the lot are to be inspected. For example, $K851 = 2$ means that two wafers are randomly selected from the lot for inspection. The value of *time* will be

$$time = K851 \times K824$$

giving a processing rate of

$$r_{ijm} = number\ of\ wafers\ /\ time$$

(5) O Processing Mode

The calculation of r_{ijm} is proportional to the inverse of the most likely turn-around time parameter $K819$. Because $K819$ is typically small (10^{-4}), r_{ijm} is large so the corresponding steps are eliminated from the model.

(6) PL Processing Mode

$$r_{ijm} = \frac{average\ num\ of\ \ wafers/lot \times K846}{K816 + K830 + K817} \times K852 \times (1 - K802) \times K866$$

Representative device

A typical scheduling problem may have on the order of 100 devices and 600 machines. The data sets that we are working with contain 76 devices and 571 machines. The number of steps in a route is over 650 with the longest being 1190. For a planning horizon of three months (13 weeks) and Δt = 1 hr in model (1), the total number of time periods τ is

$13 \times 7 \times 24 = 2184$, which leads to a problem instance that is unsolvable. Thus some amount of aggregation is needed to reduce the number of variables and constraints.

The first level of aggregation involved the grouping of devices into families by selecting a set of representative devices. In our case, three representative devices were identified from the families C_1, C_2 and C_3, respectively. Instead of modeling all 76 devices at the same time, only these three are considered. In the remainder of the chapter, C_i is used to identify representative device *i*.

The WIP associated with devices that belong to the same family is aggregated to be the WIP of the representative device. In most cases, the routings of the devices are not exactly the same as the routings of the representative devices so some log points or operations might not be included in the model. If a log point of a device does not exist in the representative routing, then the number of wafers at that log point is added to the first operation of the next log point in the representative routing. If a log point exists in the representative routing but not the operation, then the number of wafers at that operation will be added to the WIP at the next operation of the same log point.

The daily input of blank wafers for each family is also aggregated to be the daily input of the representative devices. For testing purposes, the daily target output was set to be the average daily input over the planning horizon; however, these values can be adjusted to reflect forecasted demand.

Removing steps

Further reductions in problem size were achieved by removing inconsequential steps in the routings, including

(1) Steps with type O processing mode
(2) Steps with an empty machine list
(3) Steps with CW mode but without any specified information on routing
(4) Steps with processing rates higher than some specified threshold

The processing rate of wafers going through the O operation is sufficiently large so that virtually no time is required for this operation. As a consequence, no WIP is built up in any time period; wafers simply pass through this operation "instantly." With respect to cases (2) and (3), no machines are involved in these steps; in the case of (4),

when a machine's processing rate is above the specified threshold, e.g., 1000 wafer/min, the corresponding operation will consume a negligibly amount of time and not affect the end results.

4.3 Initial Computational Experience with Basic Decomposition

Model (1) was implemented in C++ and using concert technology provided by CPLEX 10.1. All computations were performed on a Dell Poweredge 2950 workstation running Ubuntu Linux. The machine has 2 dual-core hyperthreading 3.73 GHz Xeon processors and 8 GB memory. After some experimentation the following settings were used in CPLEX when solving the linear programs.

(1) Primal simplex method (set RootAlg = 1)

(2) Devex pricing (set PPriInd = 1)

As an additional simplifying step, constraints (3b) were removed and the variables $R_{id}(t)$ were set as follows.

$$R_{id}(1) = START_{id}, \quad \forall\, i \in I, d \in D$$

$$R_{id}(t) = 0, \quad \forall\, i \in I, d \in D, t = 2, 3, \ldots, \tau^D$$

That is, the raw materials were input to the line at the beginning of the first time period of every day. Table 4.1 reports the size and the memory usage for problem instances for the three representative devices and Δt = 60 min time interval for different planning horizons. For a 28-day (4 weeks) instance, there are 11,282,240 decision variables and 1,998,780 constraints, which consume 7.2 GB of RAM and 10.6 GB of virtual memory. The memory requirements for a 3-month (13 weeks) instance were beyond the limits of our hardware. Therefore, it is not possible to solve a problem of that size without some form of decomposition. The initial approach was to create weekly instances and solve them in sequence, initiating each with the final WIP of the preceding week. This approach is referred to as basic decomposition to distinguish from the decomposition algorithm proposed later in this chapter.

Table 4.1 Problem size and memory usage for different planning horizon

Planning horizon (days)	No. of variables	No. of constraints	RAM (GB)	Virtual memory (GB)
7	2,822,411	499,695	2.4	2.9
14	5,642,354	999,390	4.6	5.5
21	8,462,297	1,499,085	6.8	8.3
28	11,282,240	1,998,780	7.2	10.6

All data used in the testing reflects the fab environment but, as mentioned, was modified so as not to reveal the true internal production capacity. For a 4-week problem using this basic decomposition approach, the running time was 15,571 sec or 4.33 hours. The initial WIP profiles for C_1, C_2 and C_3 are depicted from Figure 4.2 to Figure 4.4 with the vertical axis for WIP level and horizontal axis for step. The cost coefficients w_i^+ and w_{id}^- in model (3) were set at 0.5 for all $i \in I$. The weight for the maximum deviation $w_{\max}$ was set to 0.05. The daily inputs were specified in Table 4.2 while the daily outputs were $T\_OUT_{1d} = 328$, $T\_OUT_{2d} = 315$ and $T\_OUT_{3d} = 26$ for $d = 1, 2, \ldots, 28$. A $\Delta t = 60$ min time interval was used in the generation of the model, and the number of wafers per lot was assumed to be 25.

The solution is summarized in Table A.10 where the first column lists the day index. The remaining columns are divided into three sections: target output T_OUT_{id}, completed output, and deviations (surplus Δ_{id}^+ or shortage Δ_{id}^-). The number of wafers that were actually completed each day is indicated in the second section. The last section reports the difference between the target output and the actual output. A positive value indicates shortage while a negative value indicates surplus. It can be seen that all demand is satisfied for the first week but the shortages start to appear on day 8. The total shortage $TS = \sum_{i \in I} \sum_{d \in D} \left(\Delta_{id}^+ + \Delta_{id}^- \right)$ is 1070.15 wafers.

The WIP profiles for devices C_1, C_2 and C_3 at the end of each week are depicted in Figure A.6 – Figure A.17. In the first week, the WIP has only a few spikes but is otherwise steady. As the weeks progress, the WIP towards the end of the route begins to disappear while the WIP at the initial steps is seen to be piling up. Finished goods are produced by draining the WIP close to the end of the routes without pulling the WIP from

the beginning. There is no WIP after step 400 at the end of the last week but there are over 8000 wafers at the first step. The profiles for C_2 were similar; for C_3, the WIP profile barely changed from week to week and all demand was met since both the input and target output for C_3 are low. The full set of results can be found in Appendix 4.

Table 4.2 Daily input for the 4-week problem

d	$R_{1d}(1)$	$R_{2d}(1)$	$R_{3d}(1)$	d	$R_{1d}(1)$	$R_{2d}(1)$	$R_{3d}(1)$
1	300	300	24	15	325	300	24
2	300	300	24	16	325	275	6
3	312	300	24	17	325	300	0
4	300	299	24	18	325	350	24
5	275	450	0	19	325	300	24
6	299	349	72	20	300	312	54
7	375	249	48	21	300	324	48
8	325	287	48	22	325	325	0
9	325	300	6	23	325	325	0
10	326	300	24	24	300	325	0
11	325	425	24	25	300	300	24
12	325	300	0	26	300	300	72
13	325	300	24	27	275	387	0
14	325	175	72	28	275	375	42

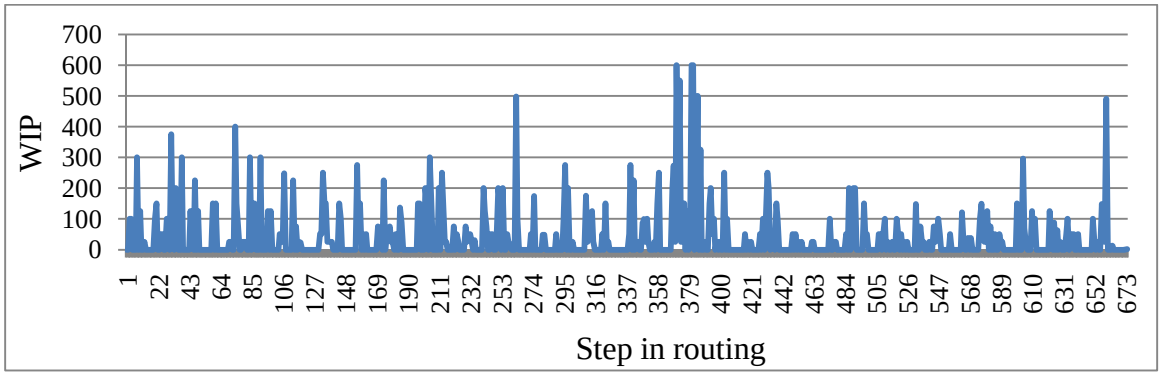

Figure 4.2 Initial WIP of C_1

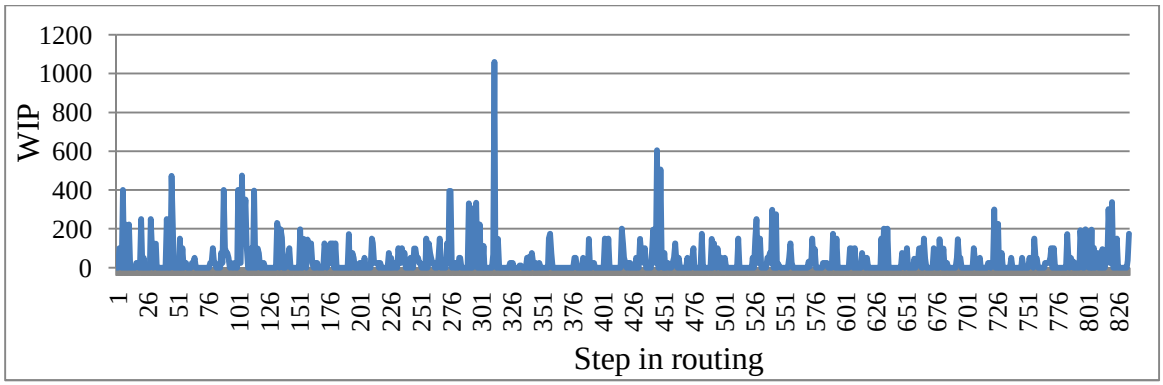

Figure 4.3 Initial WIP of C_2

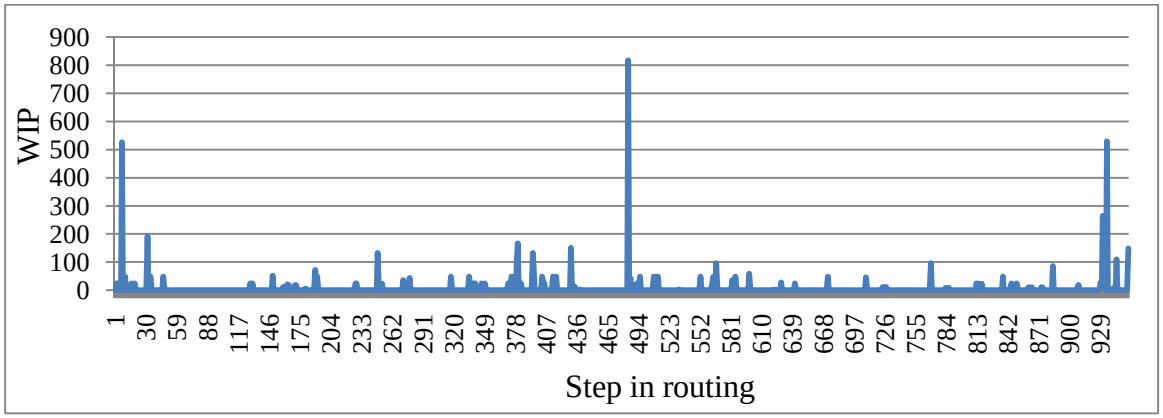

Figure 4.4 Initial WIP of C_3

One explanation for the unbalanced WIP profiles obtained for the first two devices is that the basic decomposed approach is myopic. Because the full problem is broken into 1-week segments, solutions only reflect the demand for that week. The absence of future demand in the decomposed model means that no WIP is processed beyond the amount needed to meet the current demand. Thus, WIP piles up at the front end of the fab as blank wafers are fed into the system in accordance with the given daily start schedule. Once demand is satisfied for the week, the pressure on the fab to continue running is off, even though there is still machine capacity available. The existence of excessive machine time is verified in Table 4.3 for the AP machines, which are used for wet cleaning and experience high usage in practice. The first column identifies the machine index while the second column gives the average usage which is computed as

follows: $\overline{\beta}_m = \sum_{t\in T}\sum_{i\in I}\sum_{j\in J(i)}\beta_{ijm}(t)\big/|T|$, $\forall\, m \in M$. Most of the AP machines operated less than 50% of the time from the basic decomposition approach.

In the first week it is not difficult to meet the target outputs since there are sufficient wafers close to the end of their routes. However, once these wafers are depleted from the line, it becomes increasingly more difficult to satisfy demand in subsequent weeks. This phenomenon is evidenced in the solutions in Table A.10. To reduce the shortages that appear after day 7, all WIP must be processed continually, not only the WIP at the latter steps. The unused machine time should be assigned to different steps to achieve some level of balance.

A second reason for the empty tail phenomenon is that some amount of machine starvation is inevitable when there are shifting bottlenecks, as is the case here (e.g. see Narahari and Khan 1996, Robinson et al. 1995). Wafers cannot be processed fast enough at upstream stations to provide sufficient WIP at downstream stations because of limited machine capacities and the reentrant nature of the flow. A portion of upstream and downstream stations correspond to the same machine. If the demand is much higher than the fab's throughput capacity, then simply inducting wafers into the system in proportion to demand, will necessarily result in long queues. In a fab that is running 24 hours a day, such as DMOS6, the only way to relieve this situation is to add more machines.

4.4 Modified Models

Weekly versions of model (3) only try to satisfy the current demand without looking forward. As suggested by Figure A.9, Figure A.13 and Figure A.17, little of the WIP at the initial steps is processed even though excess machine capacity exists. For example, at step 1 at the end of week 3, the WIP of C_1 is 6662 while at the end of week 4 it is 8762 -- an increase of 1900 wafers. During the week, 7×328 = 2296 wafers were introduced into the system, implying that only about 396 wafers of C_1 were processed at step 1. To remedy this shortsightedness, model (3) needs to be modified.

Table 4.3 Average usage of the AP machines

Machine, m	$\overline{\beta}_m$, %
29	74.7
30	17.92
31	12.6
32	8.53
33	11.67
34	11.21
35	10.88
36	12.17
37	39.56
38	26.92

4.4.1 Pushing the WIP forward

The approach that we developed to overcome the myopic performance of the decomposed model involves pushing the WIP forward. This is achieved by adding a third term to the objective function (3a) that incrementally rewards the presence of WIP at successive steps in a route. The corollary effect is to maximize machine utilization. In this approach, the current demand is considered explicitly while the future demand is considered implicitly.

For device i, let c_{ij} be the "benefit" of a unit of WIP at step j. Then the updated model is

$$\text{Minimize} \quad \sum_{i\in I}\sum_{d\in D}(w_i^+\Delta_{id}^+ + w_i^-\Delta_{id}^-) + w_{\max}\Delta_{\max} - \sum_{i\in I}\sum_{j\in J(i)} c_{ij}W_{ij}(\tau) \qquad (4a)$$

subject to (3c) – (3g)

The third term in (4a), $\sum_{i\in I}\sum_{j\in J(i)} c_{ij}W_{ij}(\tau)$, is designed to provide an incentive for accumulating WIP at downstream stations. The variable $W_{ij}(\tau)$ represents the WIP of device i at step j at the end of the planning horizon τ. For the new term to have the desired effect, it is necessary that $c_{ij} < c_{i,j+1}\ \forall\ i \in I, j \in J(i)$.

As in goal programming, the relative values of the three sets of coefficients in (4a) determines the order in which each term is optimized. Our intent is to first minimize the deviations from the targets, then to hold down the maximum deviation, and finally to

push the WIP. In the implementation, the following scheme was used to fix the reward coefficients:

$$c_{ij} = 0.00001j, \forall\ i \in I, j \in J(i)$$

As j increases, so does the reward making it more profitable to accumulate WIP at the downstream steps in a route. Given the maximum number of steps $J_{\max} = \max\{n(i), \forall\ i \in I\} = 1190$, we have $c_{ij} \leq 0.00001J_{\max} = 0.0119$, $\forall\ i \in I, j \in J(i)$. To some extent, selecting the values for c_{ij} represents a tradeoff between the three objective function terms. Since c_{ij} is defined to be much smaller than the deviation coefficients w_i^+, w_i^- and $w_{\max}$, the new term will not influence the objective of minimizing the deviations from the targets, or minimizing the maximum deviation. If the reward coefficients c_{ij} were set too high, however, then the model would be more inclined to push the WIP rather than reduce the deviations, the primary objective.

Although the modified model is theoretically sound, implementation occasioned a variety of numerical difficulties that could not be resolved. The increase in density of the objective function coefficient vector due to the presence of the additional $O(\sum_{i \in I} |J(i)|)$ terms caused the model to become dual degenerate. Rather than the LP converging within a few hours, the modified model required more than 40 hr for a 1-week problem. To clarify this order of magnitude increase in runtime, we asked ILOG (the CPLEX vendor) to investigate a 2-day instance. Using a Dual Intel Xeon 3.4 GHz processor with 6 GB RAM, and running parallel CPLEX with two threads, it took 2005.69 sec (0.56 hr) to achieve optimality with their barrier method. Crossover to a basis required an additional 831.6 sec. They also investigated a 7-day instance with slightly different parameter settings using a 64bit Linux server with 3 dual core Opteron 275 CPUs (1.8 GHz 1MB cache) and 6GB of RAM. With 4 threads, it took 7422.32 sec (2.06 hr) with the barrier method and an additional 14,459 sec (3.64 hr) to obtain an optimal basic solution. These extremely long runtimes, even with parallel processing, could not justify the use of model (2) directly. As a consequence, several alternative computational schemes were explored.

4.4.2 Lagrangian relaxation

One of the factors that makes model (3) difficult to solve is the need to share machine capacity among the different families of devices. Constraints (3d) tie all the devices together. By removing these constraints and placing them in the objective function as a penalty term, we create a problem whose remaining constraints decompose by representative device and should be much easier to solve than the original. This approach is called Lagrangian relaxation and for linear programs, the optimal objective function values of both the modified problem and the original problem are the same; however, the values of the decision variables might be different and in the case of the former, may not be feasible to the relaxed constraints. If this is the case, then more work has to be done to obtain the optimum.

Let $u_{mt} \geq 0$ for all $m \in \cup_{j \in J(i)} G(i, j)$, $t \in T$ be the Lagrange multipliers associated with constraints (3d), where $\cup_{j \in J(i)} G(i, j)$ is the set of machines that are required to process device i. Then relaxed model i is

$$\text{Minimize} \sum_{d \in D} (w_i^+ \Delta_{id}^+ + w_i^- \Delta_{id}^-) + w_{\max} \Delta_{\max} + \sum_{m \in \bigcup_{j \in J(i)} G(i,j)} \sum_{t \in T} \sum_{j \in J(i)} u_{mt} \beta_{ijm}(t) - \sum_{m \in \bigcup_{j \in J(i)} G(i)} \sum_{t \in T} u_{mt} \tag{5a}$$

subject to

$$\delta_{ij}^1(t)\, W_{ij}(t-1) + (1-\delta_{ij}^2) \sum_{m \in G(i, j-1)} \beta_{i, j-1, m}(t) r_{i, j-1, m} \Delta t + \delta_{ij}^4(t)\, STARTS_{id}$$

$$- \delta_{ij}^3 \sum_{m \in G(i, j)} \beta_{ijm}(t) r_{ijm} \Delta t = W_{ij}(t), \quad \forall\, j \in J(i) \cup \{n(i)+1\},\, t \in T \tag{5b}$$

$$W_{i, n(i)+1}(d\tau^D) - \Delta_{id}^+ + \Delta_{id}^- = T\_OUT_{id} + \Delta_{i, d-1}^-, \quad \forall\, d \in D \tag{5c}$$

$$\Delta_{\max} \geq \Delta_{id}^+ + \Delta_{id}^-, \quad \forall\, d \in D \tag{5d}$$

$$W_{ij}(t) \geq 0,\ \beta_{ijm}(t) \geq 0,\ \Delta_{id}^+ \geq 0,\ \Delta_{id}^- \geq 0,\ \Delta_{i0}^- = 0,\ \Delta_{\max} \geq 0,\ W_{ij}(0) \text{ given},$$

An iterative process is required to find the optimal set of Lagrange multipliers. A typical approach is to fix the multipliers u_{mt} at some value, zero in our case, and then solve model (5) to determine the optimal values of the original problem variables (Wolsey 1998). As mentioned, when u_{mt} is fixed, model (5) can be solved for each device

$i \in I$ separately. These solutions are then used to update u_{mt} with what is known as the subgradient method. The current iteration continues until u_{mt} converges to the optimal multiplier values. As a final step, model (5) is solved with each u_{mt} fixed at its optimal value to determine the corresponding values of the decision variables.

Even though Lagrangian relaxation is frequently used to solve large-scale optimization problems, we found it to be numerical unstable and unable to converge. For a 1-week problem, the multipliers u_{mt} never approached asymptotic values for runtimes of up to 6 hours. This was the case for smaller instances as well.

4.4.3 Benders decomposition

A closer examination of the constraints in the original problem reveals that they exhibit a "staircase" structure in which only the time periods overlap. This is typically the case with inventory balance constraints similar to (3c). Benders decomposition can be used to deal with this situation efficiently. The idea is to divide the original problem into subproblems with each spanning a subset of the planning horizon and then solve them iteratively, checking a set of optimality conditions at each step. The 1-week problem, for example, can be divided into seven 1-day subproblems. At each iteration of the algorithm, a restricted master problem representing the original problem is solved. To populate the master problem, extreme points or extreme rays are obtained by solving the dual of the subproblems. These solutions are then used to generate constraints that are added to the master problem. The iterations continue until no more constraints can be found for the master problem. At that point, optimality has been achieved. The application of Benders decomposition to staircase problem is provided in Appendix 6. Following this approach model (3) for the k^{th} subproblem can be written as

$$\text{Minimize} \sum_{i \in I} \sum_{d \in D^k} \left(w_i^+ \Delta_{id}^+ + w_i^- \Delta_{id}^- \right) + w_{\max} \Delta_{\max} \tag{6a}$$

subject to

$$\delta_{ij}^1(t)\, W_{ij}(t-1) + (1-\delta_{ij}^2) \sum_{m \in G(i,j-1)} \beta_{i,j-1,m}(t) r_{i,j-1,m} \Delta t + \delta_{ij}^4(t)\, START_{i,\lceil t/\tau^D \rceil}$$

$$- \delta_{ij}^3 \sum_{m \in G(i,j)} \beta_{ijm}(t) r_{ijm} \Delta t = W_{ij}(t), \;\; \forall\, i \in I, j \in J(i) \cup \{n(i)+1\}, t \in T^k \setminus \{1, \tau^D |D^k|\} \tag{6b}$$

$$Z_{ij}^{k-1} + (1-\delta_{ij}^2) \sum_{m \in G(i,j-1)} \beta_{i,j-1,m}(1) r_{i,j-1,m} \Delta t + \delta_{ij}^4(1) START_{i,\lceil t/\tau^D \rceil}$$

$$- \delta_{ij}^3 \sum_{m \in G(i,j)} \beta_{ijm}(1) r_{ijm} \Delta t = W_{ij}(1), \ \forall\, i \in I, j \in J(i) \cup \{n(i)+1\} \quad (6c)$$

$$W_{ij}(\tau^D|D^k|-1) + \left(1-\delta_{ij}^2\right) \sum_{m \in G(i,j-1)} \beta_{i,j-1,m}\left(\tau^D \left| D^k \right|\right) r_{i,j-1,m} \Delta t$$

$$- \delta_{ij}^3 \sum_{m \in G(i,j)} \beta_{ijm}\left(\tau^D \left| D^k \right|\right) r_{ijm} \Delta t = Z_{ij}^k, \ \forall\, i \in I, j \in J(i) \cup \{n(i)+1\} \quad (6c)$$

$$\sum_{i \in I} \sum_{j \in J(i)} \beta_{ijm}(t) \le 1, \ \forall\, m \in M, t \in T^k \quad (6d)$$

$$W_{i,n(i)+1}(d\tau^D) - \Delta_{id}^+ + \Delta_{id}^- = T\_OUT_{id} + \Delta_{i,d-1}^-, \ \forall\, i \in I, d \in D^k \quad (6e)$$

$$\Delta_{\max} \ge \Delta_{id}^+ + \Delta_{id}^-, \ \forall\, i \in I, d \in D^k \quad (6f)$$

$$W_{ij}(t) \ge 0, \ \beta_{ijm}(t) \ge 0, \ \Delta_{id}^+ \ge 0, \ \Delta_{id}^- \ge 0, \ \Delta_{i0}^- = 0, \Delta_{\max} \ge 0, \ W_{ij}(0) \text{ given,}$$

where $D^k = \{1, 2, 3, 4, 5, 6, 7\}$ for one week k^{th} subproblem. The initial WIP is specified by variables Z_{ij}^{k-1} while the final WIP is specified by variables Z_{ij}^k. The original inventory constraints (3c) are split into three sets. Constraints (6b) are the same as before for $t \in T \setminus \{1, \tau^D|D^k|\}$. For the first time period of the subproblem ($t = 1$) the inventory constraints are shown in (6c) while the inventory constraints for the last time period ($t = \tau^D|D^k|$) are shown in (6d). The other constraints are the same as in model (3).

Additional parameters and decision variables are required to write the dual of model (6).

Parameters and data

$\xi^1(t)$ 0 if $t = \tau$ for device; 1 otherwise

$\xi_{ij}^2(t)$ 1 if $j = n(i)+1$ and $t = d\tau^D$ for device i; 0 otherwise

ξ_{ij}^3 0 if $j = n(i)+1$ for device i; 1 otherwise

Dual decision variables

v_{ijt} dual variables corresponding to constraints (6b), (6c) and (6d) $\forall\, i \in I, j \in J(i) \cup \{n(i)+1\}, d \in D^k, t \in T^k$

γ_{mt} dual variables corresponding to constraints (6e) $\forall m \in M, t \in T^k$

η_{id} dual variables corresponding to constraints (6f) $\forall\, i \in I, d \in D^k$

ω_{id} dual variables corresponding to constraints (6g) $\forall\ i \in I, d \in D^k$

Model

$$\text{Maximize } \sum_{i\in I}\sum_{d\in D}(\eta_{id}T\_OUT_{id}) - \sum_{i\in I}\sum_{j\in J(i)} v_{ij1}Z_{ij}^{k-1} + \sum_{m\in M}\sum_{t\in T}\gamma_{mt} + \sum_{i\in I}\sum_{j\in J(i)} v_{ij\tau}Z_{ij}^{k} \quad (7a)$$

subject to

$$\delta_{ij}^{1}(t+1)\xi^{1}(t)\ v_{ij,t+1} - v_{ijt} + \xi_{ij}^{2}(t)\,\eta_{id} \le 0,$$

$$\forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, t \in T^k \backslash \{\tau^D | D^k|\}, d = \left\lfloor t/\tau^D \right\rfloor \quad (7b)$$

$$(1-\delta_{ij+1}^{2})\xi_{ij}^{3}\ v_{i,j+1,t}\, r_{ijm}\Delta t - \delta_{ij}{}^{3}\, v_{ijt}\, r_{ijm}\Delta t + \gamma_{mt} \le 0,$$

$$\forall\ i \in I, j \in J(i), m \in G(i,j), t \in T^k \quad (7c)$$

$$\eta_{id} + \omega_{id} \ge -w_i^{+}, \quad \forall\ i \in I, d \in D \quad (7d)$$

$$\eta_{id} - \eta_{i,d+1} - \omega_{id} \le w_i^{-}, \quad \forall\ i \in I, d \in D \quad (7e)$$

v_{ijt}, η_{id} are free, $\gamma_{m\,t} \le 0$, $0 \le \omega_{id} \le w_{\max}$, $\forall\ i \in I, j \in J(i) \cup \{n(\text{i})+1\}, d \in D^k, t \in T^k$ (7f)

The Benders master problem can now be presented by introducing the following definitions.

Sets

E^k set of extreme points of the k^{th} dual subproblem

R^k set of extreme rays of the k^{th} dual subproblem

D^k the set of days of the k^{th} dual subproblem

Parameters and data

NK number of dual subproblems

Decision variables

ϕ_k auxiliary variable for the k^{th} dual subproblem

Z_{ij}^{k} final WIP of device *i* at step *j* at the end of the k^{th} subproblem

Benders Master Problem

$$\text{Minimize } \sum_{k=1}^{NK}\phi_k$$

$$\text{subject to } \phi_1 \ge \sum_{i\in I}\sum_{d\in D^1}(\eta_{id}^{e1}T\_OUT_{id}) - \sum_{i\in I}\sum_{j\in J(i)} v_{ij1}^{e1}W_{ij}(0) +$$

$$\sum_{m\in M}\sum_{t\in T^1}\gamma_{mt}^{e1}+\sum_{i\in I}\sum_{j\in J(i)}v_{ijT}^{e1}Z_{ij}^{1},\ \forall\ e\in E^k,\ k=1 \qquad (8a)$$

$$\phi_k \geq \sum_{i\in I}\sum_{d\in D^k}(\eta_{id}^{ek}T\_OUT_{id})-\sum_{i\in I}\sum_{j\in J(i)}v_{ij1}^{ek}Z_{ij}^{k-1}+\sum_{m\in M}\sum_{t\in T^1}\gamma_{mt}^{ek}+\sum_{i\in I}\sum_{j\in J(i)}v_{ijt}^{ek}Z_{ij}^{k},$$

$$\forall\ e\in E^k,\ k=2,\ldots,NK-1 \qquad (8b)$$

$$\phi_{NK} \geq \sum_{i\in I}\sum_{d\in D^{NK}}(\eta_{id}^{eNK}T\_OUT_{id})-\sum_{i\in I}\sum_{j\in J(i)}v_{ij1}^{eNK}Z_{ij}^{NK-1}+\sum_{m\in M}\sum_{t\in T^1}\gamma_{mt}^{eNK},$$

$$\forall\ e\in E^k,\ k=NK \qquad (8c)$$

$$0 \geq \sum_{i\in I}\sum_{d\in D^1}(\eta_{id}^{r1}T\_OUT_{id})-\sum_{i\in I}\sum_{j\in J(i)}v_{ij1}^{r1}W_{ij}(0)+\sum_{m\in M}\sum_{t\in T^1}\gamma_{mt}^{r1}+\sum_{i\in I}\sum_{j\in J(i)}v_{ijT}^{r1}Z_{ij}^{1},$$

$$\forall\ r\in R^k,\ k=1 \qquad (8d)$$

$$0 \geq \sum_{i\in I}\sum_{d\in D^k}(\eta_{id}^{rk}T\_OUT_{id})-\sum_{i\in I}\sum_{j\in J(i)}v_{ij1}^{rk}Z_{ij}^{k-1}+\sum_{m\in M}\sum_{t\in T^1}\gamma_{mt}^{rk}+\sum_{i\in I}\sum_{j\in J(i)}v_{ijT}^{rk}Z_{ij}^{k},$$

$$\forall\ r\in R^k,\ k=2,\ldots,NK-1 \qquad (8e)$$

$$0 \geq \sum_{i\in I}\sum_{d\in D^{NK}}(\eta_{id}^{rNK}T\_OUT_{id})-\sum_{i\in I}\sum_{j\in J(i)}v_{ij1}^{rNK}Z_{ij}^{NK-1}+\sum_{m\in M}\sum_{t\in T^1}\gamma_{mt}^{rNK},\ \forall\ r\in R^k,\ k=NK \qquad (8f)$$

$$\phi_k \text{ free } \forall\ k=1,2,\ldots,NK,\ Z_{ij}^{k}\geq 0,\ \forall\ i\in I,\ j\in J(i),\ k=1,2,\ldots,NK-1 \qquad (8g)$$

Constraints (8a), (8b) and (8c) are the so-called optimality cuts. They are generated from the extreme points of the dual subproblems. Attention needs to be paid to the first subproblem and the last subproblem. For the first subproblem the initial WIP is fixed to be $W_{ij}(0)$ which can be considered as a fixed end. For the last subproblem the final WIP is not used for another subproblem thus it could be viewed as a free end. The constraints (8d), (8e) and (8f) are the so-called feasibility cuts. They are generated from the extreme rays of the dual subproblems.

Benders decomposition can now be applied to the original problem. The variables Z_{ij}^{k} are initialized at first to specify the WIP information for each subproblem. Once these variables are fixed the dual model (7) is solved for each subproblem. If the subproblem is feasible an extreme point will be generated and hence an optimality cut will be appended to the master problem (8). Otherwise an extreme ray will be generated and a feasibility cut will be added to the master problem (8). When all the subproblems are solved with the generated cuts appended to the master problem, model (8) can then be solved to obtain the updated WIP Z_{ij}^{k}. The solution Z_{ij}^{k} is again used to specify the WIP

information for the subproblems and a new iteration starts. The iteration continues until Z_{ij}^{k} becomes stable and $\sum_{k=1}^{NK} \phi_k$ achieves its maximum.

Unfortunately, numerical difficulties also arose when applying Benders to model (3). In the computations, Benders tries to find optimal machine time assignment with fixed initial and final WIP values for each subproblem. According to the numerical experiments, it was likely that the primal subproblem became infeasible, that is, there was no feasible machine assignment to achieve the targeted final WIP from the given initial WIP. As a consequence, the subproblems only generated extreme rays (infeasibility cuts in the master) so a feasible (never mind optimal) solution to the original problem was never found.

4.5 Decomposition Algorithm

The inability to achieve convergence with either of the aforementioned decomposition techniques led to the development of third approach that is more heuristic in nature. The idea is to take advantage of the fact that model (3) can be solved relatively quickly for short planning horizons and will process as much WIP as possible to meet current demand. In the first step of this approach, the planning horizon is again broken into 1-week segments and model (3) rather than model (4), is solved for each. Because the WIP "pushing" term is not in objective function (3a), many of the machines are not fully occupied and may have extensive idle capacity. To ensure that this capacity is not wasted, two additional components are included in the methodology. In the first component, a rescheduling algorithm is applied to each time period, initialized with the solution of model (3). In the second, a score is assigned to each device-step combination, and all remaining machine time is assigned in proportion to the scores. The final WIP in the current week is then taken as the initial WIP of next week and the computations are repeated. The algorithm continues in this way until all the weekly subproblems are solved. The rescheduling and scoring procedures are now discussed.

4.5.1 Rescheduling each time period

For each time period $t \in T$, a scheduling problem is solved using the solution of model (3) as input in an effort to better utilize machine capacity. The model can be formulated with the help of the following additional definitions.

Parameters

w_{ij}^{+} (w_{ij}^{-})	relative weight associated with a positive (negative) deviation from the target output for device i at step j
$w_{\max}$	weight for the maximum deviation
w_{WIP}	weight for the positive WIP deviation
WIP_limit	target WIP level
$d_{ij}(t)$	target output of device i at step j in time period t (defined below)
$\overline{W}_{ij}(t)$	WIP of device i at step j at the end of time period t, $\forall\, i \in I, j \in J(i) \cup \{n(i)+1\}, t \in T$, as indicated by the solution of model (3)
$\overline{\beta}_{ijm}(t)$	fraction of the time machine m processes device i at step j in time period t, $\forall\, i \in I, j \in J(i), m \in G(i, j), t \in T$, as indicated by the solution of model (3)

Decision variables

$\Delta_{ij}^{+}(t)$	positive deviation from target of device i at step j in time period t, $\forall\, i \in I, j \in J(i)$
$\Delta_{ij}^{-}(t)$	negative deviation from target of device i at step j in time period t, $\forall\, i \in I, j \in J(i)$
$\Delta_{\max}$	maximum output deviation
Δ_{WIP}^{+} (Δ_{WIP}^{-})	positive (negative) deviation of $W_{\max}$ from *WIP_limit*
$W_{\max}$	maximum WIP of all device-step combinations in time period t

Model for time period t

$$\text{Minimize} \quad \sum_{i \in I} \sum_{j \in J(i)} \left(w_{ij}^{+} \Delta_{ij}^{+}(t) + w_{ij}^{-} \Delta_{ij}^{-}(t) \right) + w_{\max} \Delta_{\max} + w_{\text{WIP}} \Delta_{\text{WIP}}^{+} \tag{9a}$$

subject to

$$\overline{W}_{ij}(t) + (1-\delta_{ij}^2) \sum_{m \in G(i,j-1)} \left(\beta_{i,j-1,m}(t) - \overline{\beta}_{i,j-1,m}(t)\right) r_{i,j-1,m} \Delta t$$

$$- \delta_{ij}^3 \sum_{m \in G(i,j)} \left(\beta_{ijm}(t) - \overline{\beta}_{ijm}(t)\right) r_{ijm} \Delta t = W_{ij}(t), \ \forall\ i \in I, j \in J(i) \cup \{n(i)+1\} \qquad (9b)$$

$$\sum_{i \in I} \sum_{j \in J(i)} \beta_{ijm}(t) \le 1, \quad \forall\ m \in M \qquad (9c)$$

$$\sum_{m \in G(i,j)} \beta_{ijm}(t) r_{ijm} \Delta t \ - \ \Delta_{ij}^{+}(t) + \ \Delta_{ij}^{-}(t) = d_{ij}(t) + \Delta_{ij}^{-}(t-1), \ \forall\ i \in I, j \in J(i) \qquad (9d)$$

$$\Delta_{\max} \ge \Delta_{ij}^{+}(t) + \Delta_{ij}^{-}(t), \quad \forall\ i \in I, j \in J(i) \qquad (9e)$$

$$W_{\max} \ge W_{ij}(t), \ \forall\ i \in I, j \in J(i) \qquad (9f)$$

$$W_{\max} - \ \Delta_{\mathrm{WIP}}^{+} + \Delta_{\mathrm{WIP}}^{-} = WIP\_limit \qquad (9g)$$

$$\beta_{ijm}(t) \ge \overline{\beta}_{ijm}(t), \ \forall\ i \in I, j \in J(i) \backslash \{n(i)\}, m \in G(i,j),$$

$$\beta_{ijm}(t) = \overline{\beta}_{ijm}(t), \ \forall\ i \in I, j = n(i), m \in G(i,j)$$

$$W_{ij}(t) \ge 0, \ \beta_{ijm}(t) \ge 0, \ \Delta_{ij}^{-}(0) = 0, \ \overline{W}_{ij}(t) \text{ and } \overline{\beta}_{ijm}(t) \text{ given},$$

$$\forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, m \in M, d \in D \qquad (9h)$$

The parameters $\overline{W}_{ij}(t)$ and $\overline{\beta}_{ijm}(t)$ are given by the solution to model (3) and indicate the current WIP level and machine usage in time period *t* for device *i* at step *j* for machine *m*. Two goals are involved in the objective function. The first is reflected in the first two terms in (9a) which minimize the sum of the total weighted deviations from the targets in the current period *t* plus the weighted maximum deviation. The second goal is to restrict the maximum WIP level to some prescribed value *WIP_limit*, as indicated by the third term in (9a). Currently, *WIP_limit* is set to 1000 to reflect historical levels in DMOS6. Imposing hard bounds on WIP can lead to infeasibilities. In the numerical runs, w_{ij}^{+} and w_{ij}^{-} are set to 1.0, $w_{\max}$ is set to 0.5 and w_{WIP} is set to 0.1 to reflect the priorities of the goals.

Constraints (9b) keep track of the WIP levels; the term $\left(\beta_{ijm}(t) - \overline{\beta}_{ijm}(t)\right)\Delta t$ accounts for additional processing time on machine *m* allocated to device *i* at step *j* in time period *t*. Constraints (9c) ensure that the capacity of the machines is not violated while constraints (9d) and (9e) are used to compute the positive, negative and maximum

deviations, respectively. The term $\sum_{m \in G(i,j)} \beta_{ijm}(t) r_{ijm} \Delta t$ in (9d) indicates the output of device *i* at step *j* in time period *t*; the parameter $d_{ij}(t)$ is the target output in time period *t* and is derived by uniformly apportioning the daily target output as follows.

$$d_{ij}(t) = T\_OUT_{id} / \tau^D, \text{ with } d = \lceil t / \tau^D \rceil$$

The highest WIP level $W_{\max}$ is determined by constraints (9f) while the positive and negative WIP deviations from *WIP_limit* are computed in (9g). Bounds on the decision variables are specified in (9h). For the last step $n(i)$ of the route for device *i*, the values of $\beta_{ijm}(t)$ are fixed to $\overline{\beta}_{ijm}(t)$ in order to maintain the same output specified by the solution to model (3). Allowing these values to change can lead to infeasible solutions.

Updating the WIP

After solving model (9) for each time period $t \in T \setminus \{\tau\}$, the WIP levels and machine usage results need to be updated. This is done by putting

$$\overline{W}_{ij}(t) \leftarrow W^*_{ij}(t),\ \overline{\beta}_{ijm}(t) \leftarrow \beta^*_{ijm}(t),\ \forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, m \in M$$

where $W^*_{ij}(t)$ and $\beta^*_{ijm}(t)$ are obtained from solving model (4). However, these new values may imply that the solution to model (3) for time period *t* + 1, i.e., $\overline{W}_{ij}(t+1)$ and $\overline{\beta}_{ijm}(t+1)$, may no longer be feasible since $\overline{W}_{ij}(t)$ has changed. Using Eq. (3c) to update $\overline{W}_{ij}(t+1)$, when $\overline{W}_{ij}(t+1) \geq 0$ the machine usage $\overline{\beta}_{ijm}(t+1)$ is valid so no adjustments are necessary. When $\overline{W}_{ij}(t+1) < 0$, we set it to 0 and decrease $\overline{\beta}_{ijm}(t+1)$ accordingly. This is done by taking each machine $m \in G(i,j)$ in turn and reducing $\overline{\beta}_{ijm}(t+1)$ until the original WIP $\overline{W}_{ij}(t)$ is depleted. Any remaining machine usage values, $\overline{\beta}_{ijm}(t+1)$, are set to zero. The pseudocode for this procedure is outlined in Figure 4.5. In Step 1, the WIP is updated. In Step 2, $\overline{\beta}_{ijm}(t+1)$ is updated when $\overline{W}_{ij}(t+1) < 0$.

Time period *t* rescheduling algorithm

The pseudocode for the algorithm that updates the schedule incrementally is outlined in Figure 4.6. In Step 1, model (9) is solved for each time period $t \in T \setminus \{\tau\}$ and the resulting solution is then used to update the WIP levels and machine usages in Step 2.

The iterations continue until all time periods $t \in T \setminus \{\tau\}$ are investigated. The new values of $\overline{W}_{ij}(t)$ and $\overline{\beta}_{ijm}(t)$ give the improved WIP profiles and machine assignments.

Procedure: Update_WIP($\beta_{ijm}(t+1)$, $W_{ij}(t)$, $W_{ij}(t+1)$)

Input: Updated WIP at time period t, $W_{ij}(t)$; WIP at time period $t+1$ from solution to model (3), $W_{ij}(t+1)$; machine usage from solution to model (3), $\beta_{ijm}(t+1)$

Output: Updated WIP and machine usage at time period $t+1$, $W_{ij}(t+1)$ and $\beta_{ijm}(t+1)$

Step 1: for($i \in I, j \in J(i)$){

$$W_{ij}(t+1) = \delta_{ij}^1(t+1)\, W_{ij}(t) + (1-\delta_{ij}^2) \sum_{m \in G(i,j-1)} \beta_{i,j-1,m}(t+1) r_{i,j-1.m} \Delta t + \delta_{ij}^4(t+1) STARTS_{i,\lceil (t+1)/\tau^D \rceil} - \delta_{ij}^3 \sum_{m \in G(i,j)} \beta_{ijm}(t+1) r_{ijm} \Delta t ;$$

Step 2: if ($W_{ij}(t+1) < 0$){

$W_{ij}(t+1) = 0$; *flag* = 0;

$$sum = \delta_{ij}^1(t+1)\, W_{ij}(t) + (1-\delta_{ij}^2) \sum_{m \in G(i,j-1)} \beta_{i,j-1,m}(t+1) r_{i,j-1.m} \Delta t + \delta_{ij}^4(t+1) STARTS_{i,\lceil (t+1)/\tau^D \rceil};$$

for ($m \in G(i, j)$){

if (*flag* equals to 0){

$AT_{ijm}(t) = \beta_{ijm}(t+1)\Delta t$; $RT_{ijm}(t) = sum/r_{ijm}$;

if ($AT_{ijm}(t) < RT_{ijm}(t)$) $sum = sum - \beta_{ijm}(t+1) r_{ijm} \Delta t$;

else $\beta_{ijm}(t+1) = sum/(r_{ijm}\Delta t)$; $sum = 0$; $flag = 1$;

}else{

$\beta_{ijm}(t+1) = 0$;

}

}//end *m*

}

}

Figure 4.5 Pseudocode for updating WIP in the next time period

Procedure: Rescheduling ($\overline{W}_{ij}(t), \overline{\beta}_{ijm}(t)$)

Input: Solution to model (1), $\overline{W}_{ij}(t)$ and $\overline{\beta}_{ijm}(t)$

Output: Updated solution $\overline{W}_{ij}(t)$ and $\overline{\beta}_{ijm}(t)$

Step 1: for ($t \in T \setminus \{\tau\}$){

Solve model (4), obtain solution $W^*_{ij}(t)$ and $\beta^*_{ijm}(t)$;

Step 2: $\overline{W}_{ij}(t) \leftarrow W^*_{ij}(t)$, $\overline{\beta}_{ijm}(t) \leftarrow \beta^*_{ijm}(t)$, $\forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, m \in M$;

call update_WIP($\overline{\beta}_{ijm}(t+1), \overline{W}_{ij}(t), \overline{W}_{ij}(t+1)$);

}

Figure 4.6 Pseudocode of rescheduling algorithm

4.5.2 Dispatching heuristic

Idle machine time may still exist after running the aforementioned rescheduling algorithm. To ensure that the machines are fully utilized, a dispatching heuristic is applied to push the WIP forward. It makes use of a scoring scheme to assign processing priorities to all device-step combinations.

Scoring scheme

In the design of the dispatching heuristic, the goal is determine how best to allocate the machine capacity that remains after solving model (3) and then rescheduling each time period. As part of the procedure, a score $S_{ij}(t)$ is calculated for step j of device i in time period t, and is used to set priorities such that the remaining unused machine time is assigned proportionally to the scores. The calculations are performed by the procedure outlined in Figure 4.7.

Step 1 starts the iteration for the indices i and j. In Steps 2 and 3, $S_{ij}(t)$ is dynamically adjusted as a function of the WIP level $W_{ij}(t)$. In Step 2, $S_{ij}(t)$ is set to zero when the amount of WIP at the next step $j + 1$ exceeds *WIP_limit*. The aim is to restrict processing at step j in period t when there is already ample WIP in front of the operation performed at step $j + 1$. This will reduce the WIP at step $j + 1$ in period $t + 1$.

In Step 3, the planning horizon is implicitly divided into segments of four periods each. In the first period of a segment (r = 1), higher scores will be assigned to steps with larger amounts of WIP in the earlier steps of a routing (j small) to ensure that at least some wafers are moved forward for processing at future steps. In the second and fourth periods (r = 2 or 0), scores are calculated in such a way that unused machine time will be allocated to those steps with large WIP. The corresponding objective is to reduce spikes. In the calculations, the normalizing term Max_WIP_i represents the highest WIP level associated to device i at the end of period t. A second parameter δ is used to bias or control the scores throughout a routing. The step with maximum $W_{ij}(t)$ in period t will always have a score of 1; for the remaining steps, the higher the value of δ, the lower the score. After extensive experimentation, δ was set to 10.

Procedure: Score_assignment($W_{ij}(t)$, *WIP_limit*, δ, $S_{ij}(t)$)
Input: Current WIP $W_{ij}(t)$, parameter δ and *WIP_limit*;
Output: Score $S_{ij}(t)$, $\forall\, i \in I, j \in J(i)$ in time period t;
Step 1: for ($i \in I, j \in J(i)$) {
Step 2: if ($W_{i,j+1}(t) \geq WIP\_limit$) $S_{ij}(t) = 0$;
Step 3: else
$r = t \bmod 4$;
if (r = 1) $S_{ij}(t) = 1/j + W_{ij}(t)$;
else if (r = 3) $S_{ij}(t) = j + W_{ij}(t)$;
else $Max\_WIP_i = \max\{W_{ij}(t): \forall\, j \in J(i)\}$;

$$S_{ij}(t) = \left(\frac{W_{ij}(t)}{Max\_WIP_i} \right)^{\delta};$$

}//end loop

Figure 4.7 Pseudocode for score assignment procedure

In the third period (r = 3), higher scores will be assigned to steps with larger WIP at the tail end of the routing to push wafers forward as they near completion. The scores are assigned alternatively in a way such that both the initial and tail parts of a routing are taken into consideration without allowing wafers to accumulate at intermediate steps.

A close look at the details of the calculations in Figure 4.7 reveals two contrary objectives. The first is to push or pull the WIP forward; the second is to smooth the WIP along the routings to avoid excessive build-ups. The procedure reflects the compromise adopted to split the remaining capacity between the two objectives.

The calculations are illustrated in Table 4.4 for a 1-hour time period and two devices. After solving model (3) and the rescheduling algorithm, assume that machine *m* still has 30 min of remaining capacity. Steps 1, 3, 5, 6 of device 1 and steps 2, 4, 6, 7 of device 2 require machine *m* for processing. The corresponding WIP levels are shown in the third column of the table. The parameters δ =10 and *WIP_limit* = 9. The total score $\sum_{i,j} S_{ij}(t)$ is 46.56 when r = 1, 73 when r = 3 and 1.48 when r = 2 or 0. The percentage of time assigned is shown in the 5th, 7th and 9th columns for these cases. It can be seen that relatively high WIP values lead to higher scores regardless of the value of r. For example, in the highlighted line, the WIP for device 1 is 10 at step 6 and the time assignment percentage is roughly 22% for r = 1 or 3 and 67% for r = 2 or 0. One exception is that the score will be zero when the WIP at the next step is higher than *WIP_limit*, e.g., step 5 of device 1. As a consequence, no remaining machine time will be allocated to step 5 until the WIP at step 6 is less than 9.

Table 4.4 Example of applying the scoring scheme to machine *m* at time period *t*

			r = 1		r = 2 or 0		r = 3	
Device i	Step j	WIP $W_{ij}(t)$	Score $S_{ij}(t)$	Time assigned (%)	Score $S_{ij}(t)$	Time assigned (%)	Score $S_{ij}(t)$	Time assigned (%)
1	1	4	5.00	10.74	10^{-4}	0.01	5.00	6.85
1	3	7	7.33	15.74	0.028	1.90	10.00	13.70
1	5	5	0.00	0.00	0.00	0.00	0.00	0.00
1	6	10	10.17	21.84	1	67.32	16.00	21.92
2	2	4	4.50	9.66	10^{-4}	0.01	6.00	8.22
2	4	9	9.25	19.87	0.3487	23.47	13.00	17.81
2	6	2	2.17	4.66	0.00	0.00	8.00	10.96
2	7	8	8.14	17.48	0.1074	7.23	15.00	20.55

As an aside, it should be mentioned that many dispatching rules, such as SPT, priority critical ratio, FIFO, EDD and WINQ, have been developed over the years for

related problems (e.g., see Pfund et al. 2006, Saito 2007, Wein 1988). Several of these rules have been tried but each proved ineffective.

Dispatching heuristic procedure

Figure 4.8 displays the pseudocode for the dispatching heuristic. The aforementioned definitions along with the following symbols are used in the construction of the pseudocodes.

$f_m(t)$ fraction of total machine time indicated in the solution of model (3) that is assigned to machine m in period t

$S_{ij}(t)$ score assigned to step j of device i in time period t

$SS_m(t)$ summation of scores for the steps processed by machine m in time period t

p_{ijmt} proportion of remaining unused machine time of machine m assigned to step j of device i in time period t

$AT_m(t)$ remaining unused machine time associated with machine m in time period t

$RT_{ijm}(t)$ time required to process all the wafers at step j of device i by machine m in time period t

In Step 1, the total assigned machine time $f_m(t)$ for each machine m is calculated to determine the machine usage. If $f_m(t) < 1$, then excess machine time exists and the scores $S_{ij}(t)$ are computed in Step 2. In Step 3, the excess machine time is then divided proportionally to p_{ijmt} and the WIP $W_{ij}(t)$ at the end of the current time period is updated accordingly in Step 4. The subroutine for updating the machine time assignments β_{ijmt} and the WIP $W_{ij}(t)$ in time period t is given in Figure 4.9. If $t \in T \setminus \{\tau\}$, the WIP at the end of the next time period $t + 1$ needs to be updated as well using the pseudocode displayed in Figure 4.5.

Procedure: Dispatching_heuristic($W_{ij}(t)$, $\beta_{ijm}(t)$, *WIP_limit*, δ)

Input: Current WIP movement $W_{ij}(t)$ and machine assignment $\beta_{ijm}(t)$, $\forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, m \in M$; parameters *WIP_limit* and δ;

Output: updated $W_{ij}(t)$ and $\beta_{ijm}(t)$, $\forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, m \in M$;

```
for (t ∈ T){
    for (m ∈ M){
```

Step 1: //Compute the total assigned time as a fraction

$f_m(t) = \sum_{i \in I} \sum_{j \in J(i)} \beta_{ijm}(t)$

if ($f_m(t) < 1$){

//Compute the total score of machine m in time period t

Step 2: call Score_assignment($W_{ij}(t)$, *WIP_limit*, δ, $S_{ij}(t)$);

$SS_m(t) = \sum_{i \in I} \sum_{j \in J(i)} S_{ij}(t)$;

//Assign the remaining machine time proportionally to the score

Step 3: for ($i \in I, j \in J(i), m \in G(i, j)$){

$p_{ijmt} = S_{ij}(t) / SS_m(t)$;

call Update_machine_time_assignment($\beta_{ijm}(t)$, p_{ijmt}, $W_{ij}(t)$, r_{ijm});

}

}//end if

}//end m loop

Step 4: if ($t \in T \setminus \{\tau\}$) {

//Update the WIP according the updated machine time assignment.

call Update_WIP($\beta_{ijm}(t+1)$, $W_{ij}(t)$, $W_{ij}(t+1)$);

}

}//end t loop

Figure 4.8 Pseudocode of dispatching heuristic

Procedure: Update_machine_time_assignment($\beta_{ijm}(t)$, p_{ijmt}, $W_{ij}(t)$, r_{ijm})

$AT_m(t) = (1 - f_m(t))\Delta t$; $RT_{ijm}(t) = W_{ij}(t)/r_{ijm}$;

if($RT_{ijm}(t) > AT_m(t)p_{ijmt}$){

$W_{ij}(t) = W_{ij}(t) - AT_m(t)p_{ijmt}\, r_{ijm}$; $W_{i,j+1}(t) = W_{i,j+1}(t) + AT_m(t)p_{ijmt}\, r_{ijm}$;

$\beta_{ijm}(t) = \beta_{ijm}(t) + AT_m(t)p_{ijmt}/\Delta t$;

}else{

//WIP $W_{ij}(t)$ can be drained

$W_{i,j+1}(t) = W_{i,j+1}(t) + W_{ij}(t)$; $W_{ij}(t) = 0$; $\beta_{ijm}(t) = \beta_{ijm}(t) + RT_{ijm}(t)/\Delta t$;

}

Figure 4.9 Pseudocode for updating machine time assignment and current WIP

4.5.3 Integration of algorithmic components

In summary, the decomposition algorithm works as follows. After reading in all parameters, the problem is decomposed into a set of weekly subproblems for a given planning horizon. Model (3) is solved first and any unused machine time is allocated heuristically with the help of the rescheduling algorithm and followed by the dispatching heuristic to push WIP forward and simultaneously spread it evenly along the routes. The WIP associated with the current solution at the end of the 7th day is taken as the initial WIP of the next subproblem. The computations are repeated until all subproblems are solved. The following notation is used to explain the procedure outlined in Figure 4.10.

w index for subproblems

n^{sub} number of subproblems (each subproblem is a week but, in general, a subproblem can be any number of days or any division of time)

n^{days} number of days in a subproblem

The WIP is initialized in Step 1. Step 2 contains iteration for the subproblems. After model (3) is solved, the excessive machine time is allocated by the dispatching scheduling and dispatching heuristic in sequence. In Step 3, the initial WIP of each subproblem is updated. The final solution is outputted when all the subproblems are solved. The complexity of the composite algorithm is $O\left(n^{sub} \cdot |T| \cdot |M| \cdot \Sigma_{i \in I} J(i)\right)$, which can be seen by counting the nested *for* loops.

Procedure: Decomposition_algorithm($Init\_WIP_{ij}$, WIP_limit, δ)

Input: Initial WIP at $t = 0$, $Init\_WIP_{ij}$, $\forall\, i \in I, j \in J(i)$; parameters WIP_limit and δ;

Output: WIP levels $W_{ij}^{*}(t)$ and machine assignments $\beta_{ijm}^{*}(t)$ found by the heuristics;

Step 1: //read the given initial WIP

$W_{ij}(0) = Init\_WIP_{ij}$, $\forall\, i \in I, j \in J(i)$;

Step 2: for($w = 1, \ldots, n^{sub}$){

Solve model (1) and obtain the solutions $W_{ij}(t)$ and $\beta_{ijm}(t)$;

Rescheduling ($W_{ij}(t)$, $\beta_{ijm}(t)$);

Dispatching_heuristic($W_{ij}(t)$, $\beta_{ijm}(t)$, *WIP_limit*, δ);

Step 3: //Set the final WIP to be the initial WIP for the next subproblem

$W_{ij}(0) = W_{ij}(n^{days}\tau^{D}), \forall\ i \in I, j \in J(i)$;

}

$W^{*}_{ij}(t) = W_{ij}(t),\ \beta^{*}_{ijm}(t) = \beta_{ijm}(t), \forall\ i \in I, j \in J(i) \cup \{n(i)+1\}, m \in M, t \in T$

Figure 4.10 Pseudocode of the decomposition algorithm

4.5.4 Bottleneck machines

For our purposes, a bottleneck step is one at which the number of wafers in queue is consistently above a prescribed threshold value (see Lozinski and Glassey (1988) for a discussion of detection mechanisms). The machines that perform the operations at such steps are called *bottleneck machines* and consistently evidence high utilization, averaging over 97% in the TI environment. For the current problem instance, the threshold values for devices 1 and 2 are set to 1000 wafers. For device 3 the threshold value is 500 wafers.

It should be mentioned that in the initial runs, the metrology tools rather than photolithography tools became bottlenecks, which is not true in practice. There are at least two possible explanations for this discrepancy. The first is that the company's data records are not completely accurate; the second is that the processing rates were underestimated due to faulty assumptions related to average lot sizes, uptime, or sampling procedures. To reflect the true situation, a list of operations that should not be bottlenecks is provided by the company. For the machines associated with the operations in the list, their processing rates are increased by a factor called *ProcRateInc*, which varied between 2 and 20.

4.6 Computational Results

The results obtained from running the decomposition algorithm for a 4-week (28-day) problem are presented first and then extended out to 3 months (13 weeks). The representative devices in the three families C_1, C_2 and C_3 have 674, 835 and 956 steps, respectively, in their routes. The threshold value for step reduction was set to a large value (10^6) such that no steps will be removed due to high processing rates. A time

interval Δt = 60 min was used in the generation of the model, and the number of wafers per lot was assumed to be 25.

The objective function coefficients were not changed from the settings used in the initial runs discussed in Section 6. The daily input, denoted by the parameter $STARTS_{id}$, was generated by the program from the lot starts data provided by TI and spanned the 3-month period from September 1 through December 1, 2007. The daily output demand, T_OUT_{id} , was computed as the average of the total input over the planning horizon; that is,

$$T\_OUT_{id} = \sum_{d \in D} STARTS_{id} / |D|, \ \forall\, i \in I, d \in D$$

where $|D|$ is the number of days in the planning horizon. The initial WIP, $W_{ij}(0)$, was calculated by the program to reflect the state of the fab on September 1, 2007. To avoid unnaturally high spikes in WIP during the LP runs, we set WIP_limit = 1000.

In the computations, model (3) was solved with the primal simplex algorithm in CPLEX 10.1. All other options were tried but none provided comparable performance on the 1-week problem. For example, the barrier method took approximately 270 sec to set up the model and over 1 hr to find a solution. In addition, it required 6.5 GB of RAM whereas the primal simplex required only 2.4 GB. Perturbing right-hand-side values and objective function coefficients did not improve these results.

4.6.1 Problem with 4-week planning horizon

The initial WIP $W_{ij}(0)$ of the 4-week problem is depicted in Figure 4.2. The daily inputs for the three devices are given in Table 4.2, with $|D|$ = 28. The daily target output is T_OUT_{1d} = 312 wafers for C_1, T_OUT_{2d} = 315 wafers for C_2 and T_OUT_{3d} = 26 wafers for C_3, $\forall\, d \in D$.

The 4-week problem was solved in 2,199 sec (36.65 min). Each 1-week subproblem contains 2,822,411 variables and 499,695 constraints. The whole set of results are included in Appendix 5. Table 4.5 provides the output statistics by week. The second column reports the corresponding total shortages $TS = \sum_{i \in I} \sum_{d \in D} \left(\Delta_{id}^{+} + \Delta_{id}^{-} \right)$. The third column t^{LP} gives the time required by CPLEX to solve the model (1) LP, and indicates an upward trend. The fourth column t^{rs} reports the time to run the rescheduling

algorithm, while the last column t^{dh} reports the time to run the dispatching heuristic. The statistics in the last three columns do not include the overhead time which required 2199 – 654 – 296 – 452 = 797 sec.

As seen in Table 4.5, *TS* = 0 for the four weeks, which implies that the solution is optimal. The WIP profiles of C_1 at the end of each week are include in Appendix 5 and repeated here in Figure 4.11 – Figure 4.14. From these profiles we can identify where the bottlenecks occurred along the routes. For C_1, wafers accumulated at steps 57, 106, 128, 311, 368, 431 and 650 with WIP over 1000 wafers. Steps 57, 106, 128 and 311 are associated with the AP machines (wet etching). Step 368 is associated to the VF machines (furnace for annealing); step 431 is associated to the MP machines (electroplating) while step 650 is associated to the ET machines (dry etching).

Table 4.5 Output statistics for the 4-week problem

Week no.	*TS*	t^{LP} (sec)	t^{rs} (sec)	t^{dh} (sec)
1	0	73	47	129
2	0	86	78	109
3	0	265	83	107
4	0	230	88	107
Total	0	654	296	452

For C_2, the bottleneck steps are 21, 66, 79, 141, 176, 432, 475, 527, 570, 619, 753, 796 and 835 with WIP over 1000 wafers. All the bottleneck steps except the last two are associated to the VF machines. Step 796 requires the ET machines while step 835 is the last step of the route and should not be counted as a bottleneck since the WIP at this step accumulates due to the pushing logic of the decomposition algorithm. Even if it were possible to process this WIP, the objective function in models (3) and (9), which are designed to restrict output deviations, might discourage it.

For C_3, the bottleneck steps are 511 and 956 with WIP over 500 wafers. Step 511 is associated to the HD machines (wet clean operations) while Step 956, being the last step in the route, is once again constrained by the objective of minimizing output deviations.

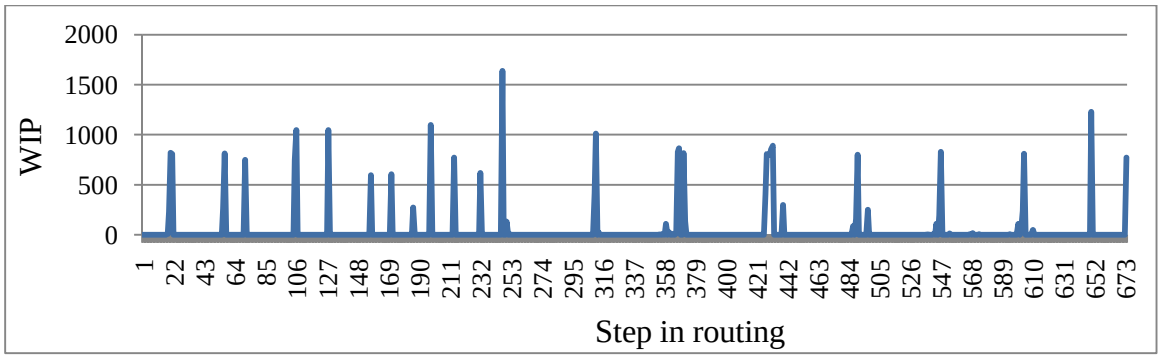

Figure 4.11 WIP profile of C_1 at the end of the 1st week

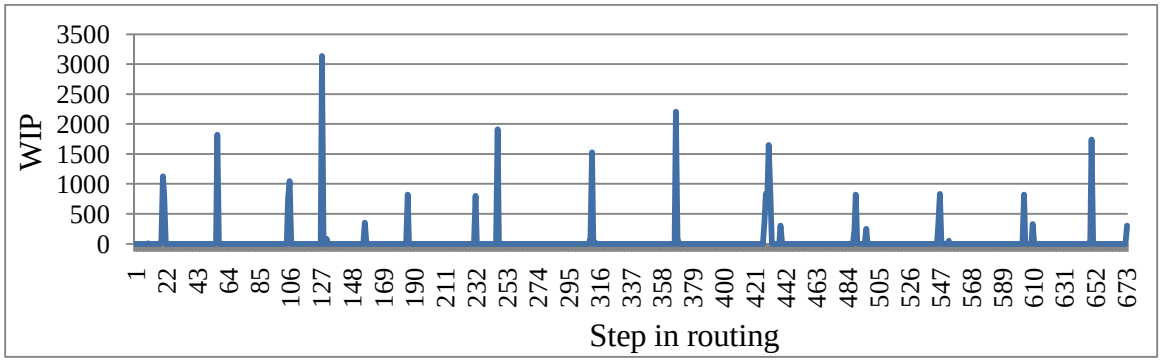

Figure 4.12 WIP profile of C_1 at the end of the 2nd week

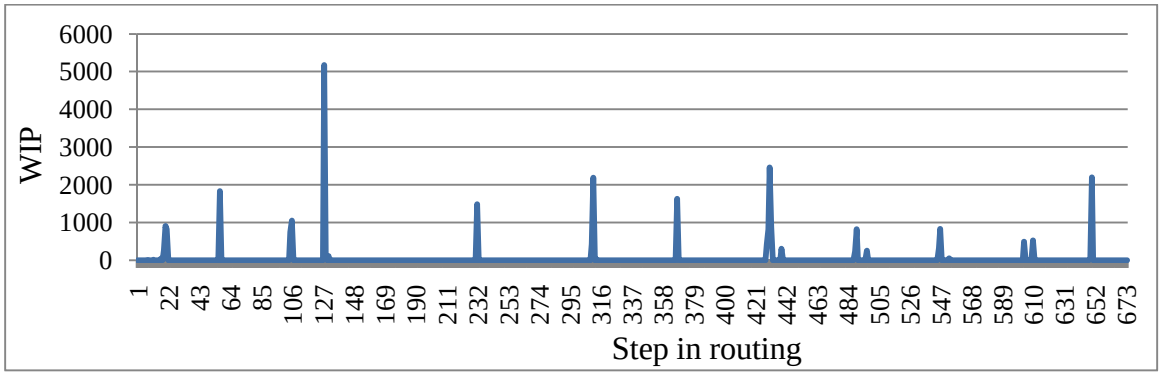

Figure 4.13 WIP profile of C_1 at the end of the 3rd week

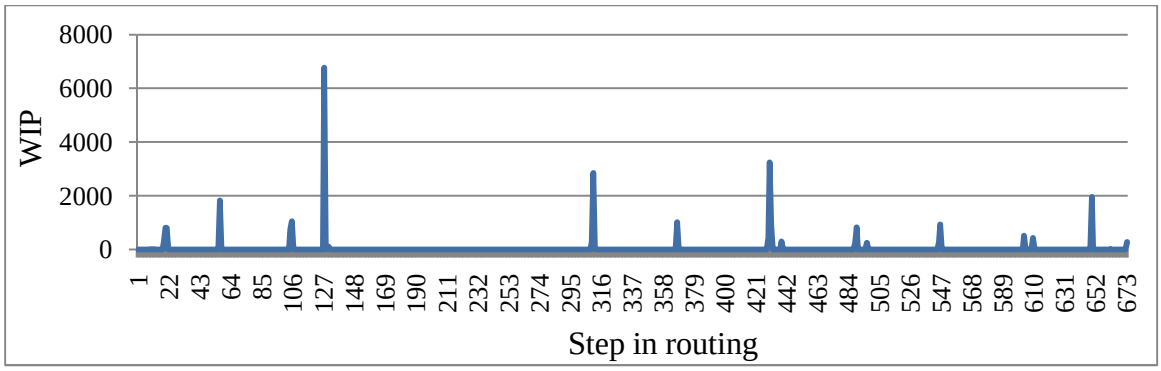

Figure 4.14 WIP profile of C_1 at the end of the 4th week

The current solution is a significant improvement over the initial solution discussed in Section 6 since all shortages have been eliminated. However, as can be seen from Figure 4.11 – Figure 4.14 that there is substantial WIP buildup at various steps which implies some instability in the system. A similar situation was observed for C_2 and C_3.

4.6.2 Problem with 13-week planning horizon

The basic parameters for the 13-week problem are the initial WIP, $W_{ij}(0)$, which is displayed in Figure 4.2 – Figure 4.4 and the daily input which is not listed here but can be found in Appendix 5. The daily output targets are T_OUT_{1d} = 328 wafers for C_1, T_OUT_{2d} = 315 wafers for C_2 and T_OUT_{3d} = 26 wafers for C_3, $\forall\ d \in D$ where $|D|$ = 91.

The solution for the 13-week problem was found in 9,030 sec (2.51 hr) and is detailed in Appendix 6. Table 4.6 reports the output statistics by week. The second column indicates that shortages first appear in week 8 and increase as the weeks progress. From week 10 through week 13 they are steady at 550.20 wafers. The third column indicates that the time to solve model (3) is well under 500 sec for each week. The time spent on rescheduling in each period is given in the fourth column and shows an upward trend. The final column indicates that the time spent on the dispatching heuristic is stable at roughly 120 sec. A total of 9030 – 3027 – 3515 – 1637 = 851 sec was required for the overhead computations.

The total shortage *TS* is 2944.42 wafers. The daily shortages for the three devices are displayed in Figure 4.15 – Figure 4.17, respectively, with positive values on the

vertical axis associated with Δ_{id}^{-} and negative values with Δ_{id}^{+}. Shortages first appear at day 52 and then fluctuate for both C_1 and C_2. From day 53 to day 91, the maximum daily shortages are 60.1 wafers for C_1 on day 62, 52.6 wafers for C_2 on days 57, 64, 71, 78 and 85, and 26 wafers for C_3 from day 60 to 91. In fact, there is no output for C_3 on the last 32 days. Over the 91-day horizon, the average daily deviations for the three devices are 11.55, 11.55 and 9.26, respectively.

Table 4.6 Output statistics for the 13-week problem

Week no.	*TS*	t^{LP} (sec)	t^{rs} (sec)	t^{dh} (sec)
1	0.00	110	191	134
2	0.00	109	234	207
3	0.00	207	239	115
4	0.00	224	251	116
5	0.00	144	260	117
6	0.00	195	263	119
7	0.00	143	281	125
8	260.60	368	291	114
9	483.02	321	289	114
10	550.20	301	283	116
11	550.20	177	303	119
12	550.20	429	302	120
13	550.20	299	328	121
Total	2944.42	3027	3515	1637

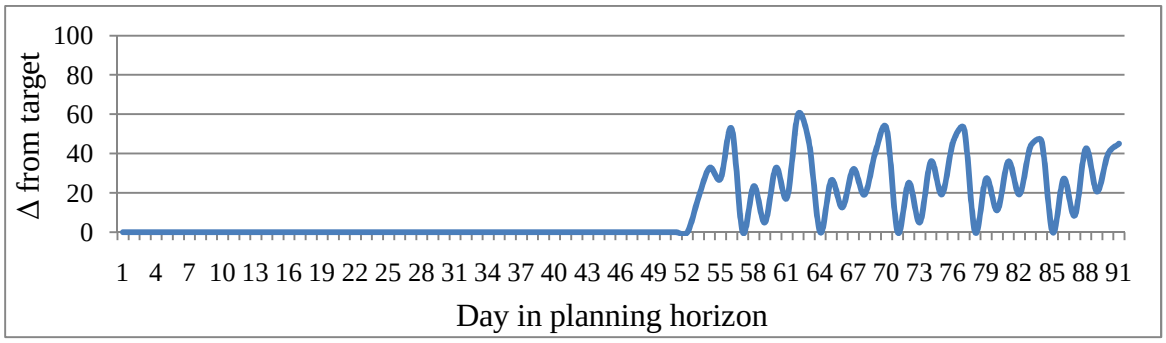

Figure 4.15 Daily shortage of C_1

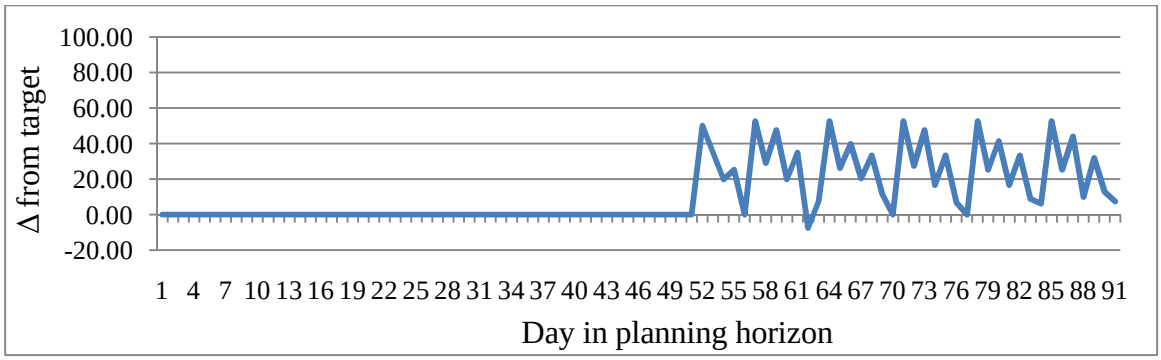

Figure 4.16 Daily shortage of C_2

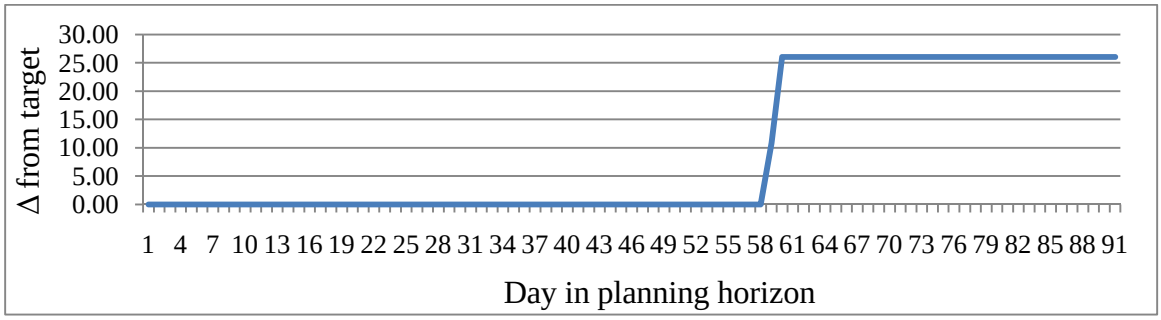

Figure 4.17 Daily shortage of C_3

The three WIP profiles at the end of the 13th week are displayed in Appendix 6 and also repeated here in Figure 4.18 – Figure 4.20. The bottleneck steps and the associated machine information are reported in Table 4.7. The first three columns list the devices, the bottleneck steps, and the corresponding machine tools. The last two columns give the number of available machines and the associated processing rates. As in the 4-week problem, the AP, VF and ET machines are the bottlenecks.

A closer look at the routings reveals that the AP machines are involved in 14 steps for C_1 with an average processing rate of 0.5433 wafers/min, 14 steps for C_2 with an average processing rate of 0.4957 wafers/min and 26 steps in C_3 with an average processing rate of 0.5002 wafers/min. To get a better understanding of fab capacity, assume that the capacities of the AP machines are assigned to the three devices in proportional to the daily target outputs, and that for the same device, these capacities are evenly assigned to the associated steps. The output of the fab can be estimated as follows.

$$C_1: \quad 0.5433 \times \frac{328}{328+315+26} \times \frac{1}{14} \times 9 \times 60 \times 24 = 246.58 \text{ wafers/day}$$

$$C_2: \quad 0.4957 \times \frac{315}{328+315+26} \times \frac{1}{14} \times 9 \times 60 \times 24 = 216.06 \text{ wafers/day}$$

$$C_3: \quad 0.5002 \times \frac{26}{328+315+26} \times \frac{1}{26} \times 9 \times 60 \times 24 = 9.96 \text{ wafers/day}$$

These daily outputs are lower than the daily target outputs for the three devices. The estimated shortage is (328 + 315 + 26) – (246.58 + 216.06 + 9.96) = 196.66 wafers/day. Applying the same analysis to the VF machines, the estimated daily outputs for the three devices are 528.97, 273.91 and 27.30 wafers, respectively. This results in a surplus of (528.97 + 273.91 + 27.30) – (328 + 315 + 26) = 161.18 wafers/day. However, a closer look indicates that the reason why the VF machines become bottlenecks is because the processing rate at the steps immediately preceding the bottleneck steps is much higher: for C_2, the processing rate is 40 wafers/min at step 618 and for C_3 it is 48,000 wafers/min at step 641.

For the ET machines, this analysis is more straightforward since they are only required for step 650 in the route of C_1, step 796 in the route of C_2 and step 922 in the route of C_3. The maximum output for the two ET machines is 0.205 × 2 × 60 × 24 = 590.4 wafers/day. In contrast, the daily shortage is (328 + 315 + 26) – 590.4 = 78.6 wafers while the weekly shortage is 78.6 × 7 = 550.2 wafers. This value coincides exactly with the shortages from week 10 to week 13 as reported in Table 4.5. As such, the daily output requirements are beyond the capacity of the available machines, implying that the fab will eventually become unstable.

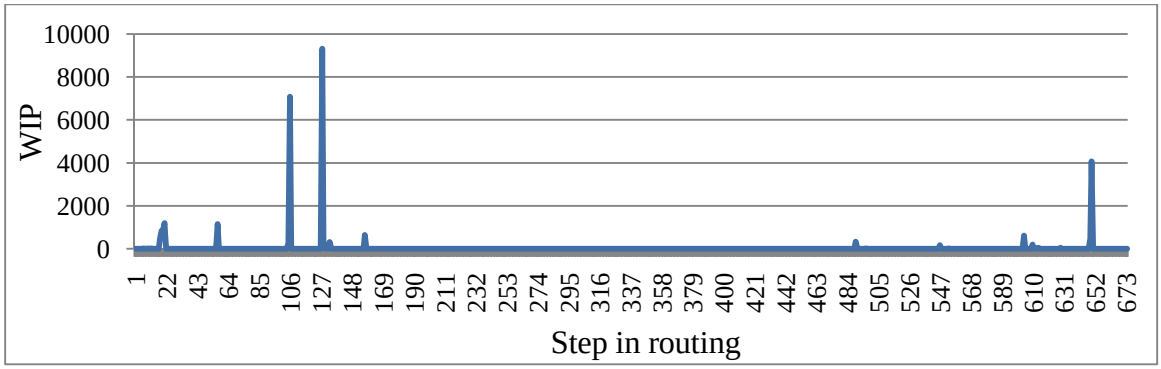

Figure 4.18 WIP profile of C_1 at the end of the 13th week

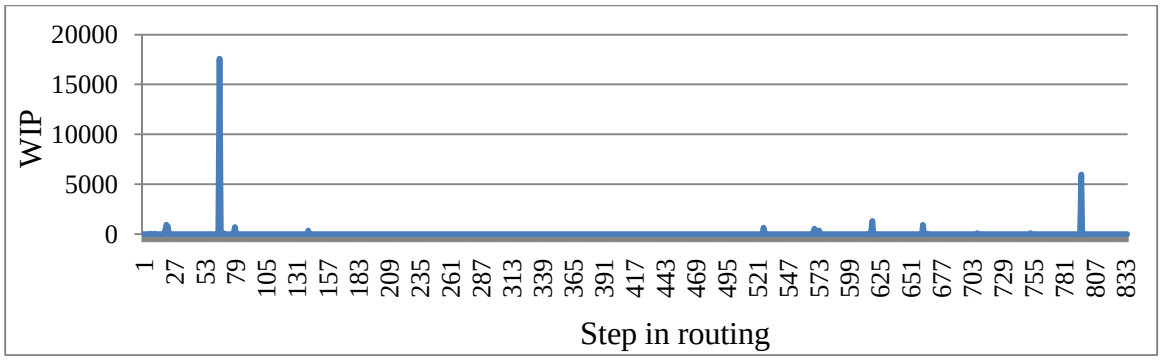

Figure 4.19 WIP profile of C_2 at the end of the 13th week

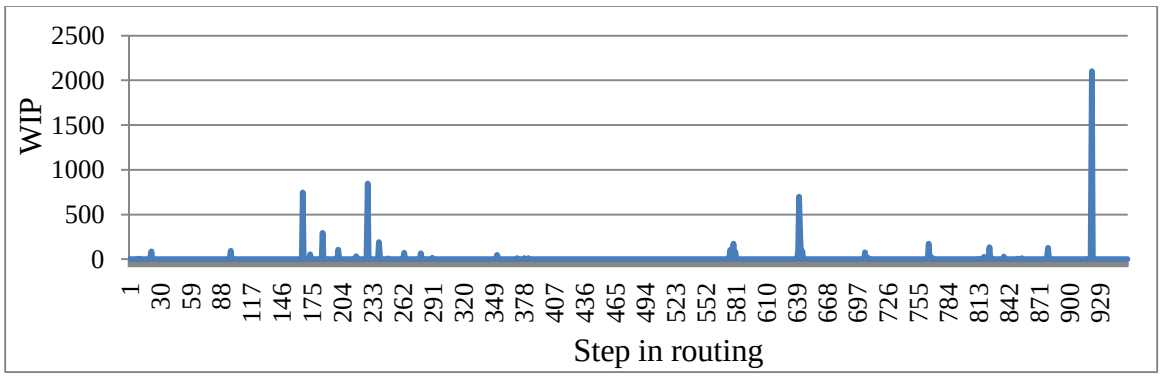

Figure 4.20 WIP profile of C_3 at the end of the 13th week

Table 4.7 Bottleneck information for the 13-week problem

Device (family)	Bottleneck step(s)	Tool	Number of tools	Processing rate (wafers/min)
C_1	21	AP	9	0.4464
	57	AP	9	0.5515
	106	AP	9	0.5515
	128	AP	9	0.4934
	650	ET	2	0.2050
C_2	66	AP	9	0.5515
	619	VF	8	0.3729
	796	ET	2	0.2050
C_3	167	AP	9	0.5859
	229	AP	9	0.4261
	642	VF	8	0.5357
	922	ET	2	0.2050

Intuitively, the bottleneck machines should be busy most of the time. This can be verified from the data in Table 4.8, which reports the average percent utilization for the

AP, VF and ET machines over the 91-day planning horizon. It can be verified that these machines are fully utilized with $\overline{\beta}_m$ = 100%.

Table 4.8 Average usage of bottleneck machines over 91 days

Machine number, *m*	Machine group	$\overline{\beta}_m$ (%)	Machine number, *m*	Machine group	$\overline{\beta}_m$ (%)
29	AP	100	543	VF	100
30	AP	100	544	VF	100
31	AP	100	545	VF	100
32	AP	100	546	VF	100
33	AP	100	547	VF	100
34	AP	100	548	VF	100
35	AP	100	549	VF	100
36	AP	100	550	VF	100
37	AP	100	284	ET	100
38	AP	100	285	ET	100

4.6.3 Rolling horizon for subproblems

Even if no shortages exist after running the decomposition algorithm for several weeks, the optimal solution obtained for a particular week may not be optimal when the full problem is solved as a whole. In fact, an excessively large number of alternate optima exist for both models (3) and (9) due to the underlying network structure of the problem. If only a small subset of the alternate optima are robust with respect to the 13-week problem, then it is likely that as the decomposition algorithm progresses, the subproblem solutions will become less robust. This could lead to suboptimal final WIP values which in turn would produce suboptimal subproblem solutions. If this occurs, the subproblem solutions will deteriorate over time with respect to the full problem. (A similar phenomenon is commonly observed when statistical models are used to make forecasts; i.e., variance increases with the length of the forecast.)

To investigate the implications of this situation, we implemented the decomposition algorithm in a rolling horizon framework. In this approach, an *s*-day problem, say, is solved but only the results for the first *r* days ($r < s$) are actually applied. The WIP at the end of the first *r* days is used to initialize another *s*-day problem whose

solution again is only applied for the next *r* days. This process is outlined in Figure 4.21 and would be repeated indefinitely.

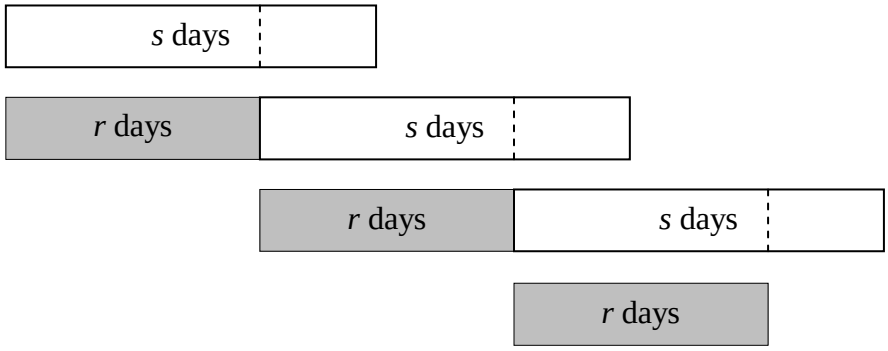

Figure 4.21 Rolling horizon for the subproblems

In the implementation we set $s = 9$ and $r = 7$. For the 4-week problem, the solution was obtained in 2,954 sec compared to 2,199 sec for the original approach and similarly resulted in no shortages. The WIP profiles at the end of each week were almost identical to those shown in Figure 4.11 – Figure 4.14. For the 13-week problem, the computational time was 13,364 sec compared to 9,030 sec; however, a slightly better solution was found with $TS = 2943$ wafers. In these runs, a shortage of 55.8 wafers first appeared in week 7, increased steadily, and then stabilized again at 550.2 wafers in the last three weeks.

The immediate observation from these results is that for the parameter settings used in the analysis, the rolling horizon scheme consumes proportionally more time without noticeably improving the solution. Although a larger *s* would make the subproblems more robust, the shortage in the last few weeks in the 13-week problem will eventually stay at 550.2 as indicated by the analysis of the ET machines in Section 9.2.

4.7 Further Discussion

The 4-week problem can be solved in about 37 min, which is a reasonable amount of time, and provides results without any shortage. For the 13-week problem, the occurrence of shortages began in week 8 and continued to the end of the planning horizon due to the capacity limits of the available machines. Assuming that the processing rates

are correct, to improve throughput, either additional bottleneck machines need to be brought on line or their processing rates need to be increased.

The final issue concerns shift scheduling. Because the results from our model are in aggregate form with respect to the three representative families, and are expressed as continuous rather than discrete values, more work is needed to construct a daily plan that takes into account the actual devices in the system, setup times between lots, tooling, and other factors that are common in discrete manufacturing. A second model would have to be developed for this purpose.

Chapter 5

Scheduling Back-End Operations in Semiconductor Manufacturing

At back-end facilities, finished wafers go through an extensive regimen of inspection and testing that can take up to 3 hours at each step. Over a planning horizon of anywhere from 8 hours to several days, hundreds of thousands of wafers, grouped into thousands of lots must be assembled and tested. Each wafer must go through approximately 32 discrete operations before it enters finished goods inventory. The AT facility has hundreds of machines that are used to perform the required processes. At each operation, a queued lot must be assigned to one of a subset of appropriate machines, and when two successive lots consist of different devices, a setup is incurred between lots. Setups or changeovers are performed by a crew of technicians and typically take 2 hours, although fewer hours may be needed, depending on the tooling. If the current device on a machine must be tested at a high temperature while its successor requires testing at room temperature, and both use the same fixtures, then the setup time is equal to the amount of time it takes for cool down, usually an hour. Labor is generally not a constraining factor.

Each lot contains a number of chips of the same device, ranging from a few hundred to several thousand. Two lots may contain the same device but a different number of chips. A lot remains in the facility until it undergoes all 32 operations. All lots are associated with customers and have delivery due dates. When a delivery is late, a penalty is incurred which is a function of lateness and volume. Because setups are so time consuming, it is critical for the planners to assign lots to machines and tooling to machines in such a way that as few setups as possible are required and due dates are taken into account.

The age of a lot is the current time minus the time it entered the facility. For each operation, each lot is assigned to a particular machine for processing. To be eligible, the machine must be set up with the appropriate tooling pieces, as specified by the lot's *routing table* and must be able to operate at the required temperature. Machines are divided into families. In most cases, two machines from the same family are identical; however, it is possible that "identical" machines operate under different temperatures and

hence are not interchangeable. The limiting resource at most operations is the number of tooling pieces. As with machines, tooling pieces are divided into tooling families and only operate at a limited number of temperatures.

Each AT operation can be viewed as independent of the others so the corresponding problems are separable. As such, the discussion in the remainder of the chapter relates to an individual operation rather than the AT facility as a whole. For an incoming lot, a particular route must be selected, if there is more than one option. A route specifies the eligible machine family, the tooling requirements, the processing rate, and the operating temperature. Once a route is selected, the lot is assigned to one of the machines in the specified family and the required tooling pieces are installed. Each assigned lot is processed completely without preemption and each machine can be set up at most once during the planning horizon to operate at only one temperature. That is, if machine m is set up with tooling configuration λ_1 under temperature τ_1, then it cannot run with another tooling setup λ_2 or under another temperature τ_2 later in the planning horizon, even when τ_2 is feasible for configuration λ_1.

At the beginning of each planning horizon, typically a shift or a day, a finite number of lots are available for processing. A subset of these lots may contain what are called *key* and *package* devices, and are singled out for special treatment. Any demand that cannot be satisfied for these two types of devices occasion a large penalty for the company. It is thus desirable to ensure that as many of these "hot" lots as possible are processed over the planning horizon to avoid or reduce penalties. Regular lots are assigned a value that depends on their age and remaining cycle time in the facility.

Problem statement. For a given planning horizon, AT operation and set of lots, determine how each available machine should be configured with tooling to operate at a specified temperature so that the weighted sum of the lots processed is maximized without violating the system's capacity. The solution should also minimize the number of key and package devices falling short of their demand.

In the literature, this problem is generally referred to as a parallel machine scheduling problem with setups, due dates, and a lateness objective. Other objectives, such as minimizing the time to complete all lots (i.e., minimize makespan), minimizing

the number of setups, or minimizing the number of late jobs, have similar characteristics. In all cases, current technology limits the size of an instance that can be solved optimally to less than a dozen machines and several hundred lots (e.g., see Bard and Rojanasoonthon 2006).

The optimization model is presented in the next section. The details of the decomposition strategy and GRASP are presented in Section 5.2. Test results using data provided by Texas Instruments reported in Section 5.3. An assessment of the model and several suggestions for improving the methodology are provided in Section 5.4.

5.1 Mathematical Formulation

The AT facility planning problem can be modeled as a mixed-integer problem (MIP) using the notation given in Figure A.72. Although the formulation includes only a handful of constraints, a disproportionate amount of notation is required to correctly account for all the machine-tooling-temperature combinations.

$$\text{Maximize} \quad \sum_{i\in M}\sum_{l\in L(i)}\sum_{s\in S(i,l)}\left(w_l-\varepsilon_s\right)x_{ils}-\sum_{k\in K}w_1^k\Delta_1^k-\sum_{p\in P}w_2^p\Delta_2^p \tag{10a}$$

$$\text{subject to} \quad \sum_{i\in M(l)}\sum_{s\in S(i,l)}x_{ils}\le 1,\ \forall\, l\in L \tag{10b}$$

$$\sum_{\lambda\in\Lambda(i)}y_{i\lambda}\le 1,\ \forall\, i\in M \tag{10c}$$

$$\sum_{i\in M}\sum_{\tau\in\bar{T}(n)}\sum_{\lambda\in\Lambda(i,t,\tau)}b_{\lambda t}y_{i\lambda}\le\sum_{m\in\bar{N}(n)}n_{t,m}^{\text{tooling}},\ \forall\, t\in T, n\in N \tag{10d}$$

$$\sum_{l\in L(i,\lambda)}\sum_{s\in S(i,l,\lambda)}\frac{n_l^{\text{chips}}}{r_{ils}}x_{ils}\le H_i y_{i\lambda},\ \forall\, i\in M,\lambda\in\Lambda(i) \tag{10e}$$

$$\sum_{i\in M}\sum_{l\in L(i,k)}\sum_{s\in S(i,l)}n_l^{\text{chips}}x_{ils}+C\Delta_1^k\ge n_k^{\text{min\_chips}},\ \forall\, k\in K \tag{10f}$$

$$\sum_{i\in M}\sum_{l\in L(i,p)}\sum_{s\in S(i,l)}n_l^{\text{chips}}x_{ils}+C\Delta_2^p\ge n_p^{\text{min\_chips}},\ \forall\, p\in P \tag{10g}$$

$$x_{ils}\in\{0,1\},\forall\, i\in M, l\in L(i), s\in S(i,l),\ y_{i\lambda}\in\{0,1\},\ \forall\, i\in M,\lambda\in\Lambda(i)$$

$$\Delta_1^k\ge 0,\ \Delta_2^p\ge 0,\ \forall\, k\in K, p\in P \tag{10h}$$

The objective function (10a) is designed to maximize the total weighted number of lots processed over the planning horizon and to minimize the total weighted shortages for the key and package devices. The weight of lot l, w_l = lot age + remaining cycle time

planned, and the weights w_1^k and w_2^p are the penalties for shortages for all $k \in K$ and $p \in P$. The latter are both set to values larger than $\max\{w_l : l \in L\}$, implying that priority in the optimization is given to minimizing shortages over maximizing the weighted sum of lots processed. When all the weights w_l have the same value and $w_1^k = w_2^p = 0$, the problem is equivalent to maximizing the throughput. The parameter ε_s in the first term of (10a) is the penalty incurred when route s is chosen. Both prime and alternate routes exist for some lots. To encourage the selection of prime routes when at all possible, we use the following settings: $\varepsilon_s = 0$ for s a prime route; $\varepsilon_s \in (0, \min\{w_l : l \in L\})$ for s an alternate route

Constraints (10b) require that if lot l is assigned to machine $i \in M(l)$, then the tooling associated with one of the routes $s \in S(i,l)$ must be set up on that machine. Lot l cannot be assigned to more than one machine or be given more than one route. These constraints do not require that each lot be processed but the objective function ensures that the as many lots as possible are processed when there are a sufficient number of machines, tooling pieces, and time available.

Constraints (10c) limit each machine i to at most one tooling configuration from the set $\Lambda(i)$. When the number of lots $|L|$ is small, or when the available tooling is limited, it may not be desirable or feasible to set up all machines. Also, once the tooling configuration λ is selected for a particular machine, changeovers are not permitted during the planning horizon.

Constraints (10d) restrict the total number of tooling pieces assigned to machines from family t to the number of pieces available under temperature combination n. The left-hand side of these constraints counts the number of tooling pieces from family t associated with the choice of $y_{i\lambda}$ over all machines, temperatures in n, and corresponding tooling setups. The right-hand side counts the total available number of tooling pieces in family t under temperature combination n by summing n_{tm}^{tooling} over all combinations $m \in \overline{N}(n)$. For each $t \in T$, there are n_{tm}^{tooling} tooling pieces that can be used under the n^{th} combination if m shares some temperatures with n. Assume that there are three discrete temperatures, that is, $\overline{T} = \{1,2,3\}$, and $n_{tm}^{\text{tooling}} = 1$, $\forall\, t \in T$, $m \in N$, and let the set of possible temperature combinations $N = \{\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$. For

$n = 4$, for example, the temperature set $\overline{T}(4) = \{1,2\}$ and $\overline{N}(4) = \overline{N}(\{1,2\}) = \{\{1\}, \{2\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\} = N \setminus \{3\}$. The right-hand side of (10d) under combination n is then $\left|\overline{N}(4)\right| = 6$ for all tooling families $t \in T$.

Constraints (10e) impose a processing time limit on each machine $i \in M$ when it is set up under configuration $\lambda \in \Lambda(i)$. The left-hand side tracks the amount of time required to process each lot l assigned to machine i following route s. The time available for machine i depends on its status. If a machine i is active, then exactly one of the variables $y_{i\lambda} = 1$, for $\lambda \in \Lambda(i)$, as required by constraints (10c). Thus, the available processing time is H_i when machine i is active and 0 when it is idle. These constraints also impose a logical relationship between x_{ils} and $y_{i\lambda}$. When the setup variable $y_{i\lambda} = 0$, the lot assignment variables $x_{ils} = 0$, $\forall\, l \in L(i,\lambda)$, $s \in S(i,l,\lambda)$ so any lot l requiring setup configuration λ cannot be processed on machine i.

Constraints (10f) ensure that as many lots as possible containing key device k are processed, at least until demand $n_k^{\text{min\_chips}}$ is satisfied. The shortage $C\Delta_1^k$ will be positive if some of the demand cannot be met due to limited resources. In that case, a penalty equal to $w_1^k \Delta_1^k$ is incurred, where $C = \max\{w_l : l \in L\} + 0.1\Sigma_{l \in L} w_l$ is a normalizing constant used to ensure that the left-hand-side coefficients in (1f) are all the same order of magnitude. A similar set of constraints (10g) is included for package devices $p \in P$. In (10h), binary restrictions are placed on the x_{ils} and $y_{i\lambda}$ variables, and nonnegative restrictions are placed on the shortage variables Δ_1^k and Δ_2^p.

Although it is possible to tighten the linear programming relaxation of (10) by adding logic constraints $x_{ils} \leq y_{i\lambda}$, $\forall\, i \in M$, $\lambda \in \Lambda(i)$, $l \in L(i,\lambda)$, $s \in S(i,l,\lambda)$ and removing $y_{i\lambda}$ from the right-hand side of (10e), doing so would increase the problem size by $O(|M| \cdot |\Lambda| \cdot |L|)$ constraints, making it more than an order of magnitude larger. In our initial testing with CPLEX 11, this modification vastly increased runtimes and so was not adopted.

Proposition 5.1. The assembly and test scheduling problem (ATP) represented by model (10) is NP-complete in the strong sense.

Proof. We will show that a restricted version of ATP is an instance of the bin packing problem (BPP), which is known to be NP-complete in the strong sense (Garey and Johnson 1979). For BPP, we have the following definition.

INSTANCE: Finite set U of items, a size $s_u \in Z^+$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K.

QUESTION: Is there a partition of U into disjoint sets $U_1,U_2,\ldots,U_K$ such that the sum of the sizes of items in each U_i is B or less?

To see how an instance of BPP can be reduced to an instance of ATP, we create a simplified version of ATP where all $|M|$ machines are identical, the available processing time on each machine is identical, i.e., $H_i = H$ for all $i \in M$, there are no key or package devices, there are no tooling or setup requirements, the processing time of lot l is r_l is machine-independent, and the objective function weights $w_l = 1$ for all $l \in L$. For BPP, we let $U = L$, $s_u = r_l$ when $u = l$, $B = H$, $K = |M|$, which gives rise to the simplified ATP. If we can find a solution such that the set U can be partitioned into K subsets such that $\sum_{u \in U_i} s_u \le B$ for $i = 1,\ldots,K$, then we can find a solution such that all the lots in WIP can be processed on the $|M|$ machines in H hours, and vice versa. The fact that BPP can be transformed into the simplified instance of ATP in polynomial time completes the proof. ■

5.2 Solution Methodology

Model (10) contains $n^{\text{var}} \cong \sum_{i \in M} \sum_{l \in L(i)} \left|S(i,l)\right| + \sum_{i \in M} \left|\Lambda(i)\right| + |K| + |P| = O(|M|\cdot|L|\cdot|S| + |M|\cdot|\Lambda| + |D|)$ variables and $n^{\text{con}} \cong |L| + |M| + |T|\cdot|N| + \sum_{i \in M} \left|\Lambda(i)\right| + |K| + |P| = O(|L| + |T|\cdot|N| + |M|\cdot|\Lambda| + |D|)$ constraints. Problem size is dominated by the number of lot assignment variables x_{ils} and the number of lot assignment constraints (10b), both of which grow linearly with $|L|$. For a small case with 300 lots, 5 machine families and a total of 20 machines, 10 tooling families and a total of 50 tooling pieces, 3 operating temperatures, and 50 devices with 40 being key or package devices, the model contains approximately $n^{\text{var}} = 2220$ variables and $n^{\text{con}} = 650$ constraints. Such instances exhibit optimality gaps at the root node of the search tree that average 3% and solve quickly.

Real instances with, say, 2000 lots contain roughly 84,000 binary variables and 3300 constraints and are much slower to converge, if they do at all.

To ensure reasonable runtimes, we developed a heuristic, two-level decomposition scheme and embedded it in a reactive GRASP. Our approach is based on the observation that model (10) becomes much easier to solve when the machines setups are given, that is, when the $y_{i\lambda}$ variables are fixed, leaving what we term the *lower level problem* (LLP) in the x_{ils} variables. At the upper level, a strategic decision is made concerning machine-tooling pairings.

Phase I of the GRASP is designed to uncover a diversity of high quality feasible solutions by randomly selecting the *y* variables in accordance with an adaptive greedy measure and then solving the resultant LLP to obtain the optimal lot assignments, *x*. This process is repeated many times. In phase II, an attempt is made to improve a subset of the candidates uncovered in phase I using a high-level neighborhood search. Before presenting the overall analytic framework, our approach to solving the lower and upper level problems is described.

5.2.1 Lower level problem

When the tooling setup variables $y_{i\lambda}$ are fixed at, say, $\bar{y}_{i\lambda}$, constraints (10c) and (10d) can be dropped from model (10) and the right-hand side of (10e) becomes a constant. The reduced model, denoted by LLP, is as follows.

$$\text{Maximize} \quad \sum_{i\in M}\sum_{l\in L(i)}\sum_{s\in S(i,l)} \left(w_l - \varepsilon_s\right) x_{ils} - \sum_{k\in K} w_1^k \Delta_1^k - \sum_{p\in P} w_2^p \Delta_2^p \tag{11a}$$

$$\text{subject to} \quad \sum_{i\in M(l)}\sum_{s\in S(i,l)} x_{ils} \le 1, \ \forall\, l \in L \tag{11b}$$

$$\sum_{l\in L(i,\lambda)}\sum_{s\in S(i,l,\lambda)} \frac{n_l^{\text{chips}}}{r_{ils}} x_{ils} \le H_i \bar{y}_{i\lambda}, \ \forall\, i \in M, \lambda \in \Lambda(i) \tag{11c}$$

$$\sum_{i\in M}\sum_{l\in L(i,k)}\sum_{s\in S(i,l)} n_l^{\text{chips}} x_{ils} + C\Delta_1^k \ge n_k^{\text{min\_chips}}, \ \forall\, k \in K \tag{11d}$$

$$\sum_{i\in M}\sum_{l\in L(i,p)}\sum_{s\in S(i,l)} n_l^{\text{chips}} x_{ils} + C\Delta_2^p \ge n_p^{\text{min\_chips}}, \ \forall\, p \in P \tag{11e}$$

$$x_{ils} \in \{0,1\}, \forall\, i \in M, l \in L(i), s \in S(i,l), \ \Delta_1^k \ge 0, \ \Delta_2^p \ge 0, \quad \forall\, k \in K, p \in P \tag{11f}$$

For $\bar{y}_{i\lambda} = 1$, let $\bar{\lambda}_i$ be the corresponding tooling configuration for machine *i*. By implication $\bar{y}_{i\lambda} = 0$ for all $\lambda \in \Lambda(i) \backslash \{\bar{\lambda}_i\}$ and constraints (2c) can be written as

$$\sum_{l \in L(i,\bar{\lambda}_i)} \sum_{s \in S(i,l,\bar{\lambda}_i)} \frac{n_l^{\text{chips}}}{r_{ils}} x_{ils} \le H_i, \quad \forall\, i \in M \tag{11c'}$$

with the additional restriction $x_{ils} = 0$, $\forall\, i \in M$, $\lambda \in \Lambda(i) \backslash \{\bar{\lambda}_i\}$, $l \in L(i,\lambda)$, $s \in S(i,l,\lambda)$. Model (11) contains approximately $\sum_{i \in M} \sum_{l \in L(i,\bar{\lambda}_i)} \left| S(i,l,\bar{\lambda}_i) \right| + |K| + |P|$ variables and $|L| + |M| + |K| + |P|$ constraints, which is a sizable reduction from model (10), but still a difficult IP. Since our algorithm requires that (11) be solved repeatedly for different values of the $y_{i\lambda}$ variables in both phases of the GRASP, we propose solving the LP relaxation of (11) and then constructing feasible solutions guided by the results, rather than applying an IP solver directly. In addition, we were motivated by the desire to eliminate any dependency on a commercial product.

LP based heuristic for LLP

Let x^{LP} be the solution to the LP relaxation of model (11). To transform x^{LP} into an integral solution x^{IP}, we start by truncating the fractional lot assignments to get

$$x_{ils}^{\text{IP}} = x_{ils}^{\text{LP}} \quad \text{for } x_{ils}^{\text{LP}} \text{ integral}$$

$$x_{ils}^{\text{IP}} = 0 \quad \text{for } x_{ils}^{\text{LP}} \text{ fractional}$$

which empirically turns out to be a good starting point. Let L_0 be the set of unassigned lots and let L_i be the set of lots assigned to machine *i* in accordance with x_{ils}^{IP}. For device *j*, the output *out*(*j*) and the shortage *sh*(*j*) are defined as follows.

$$out(j) = \sum_{i \in M} \sum_{l \in L_i \cap L(j)} n_l^{\text{chips}}, \quad \forall\, j \in D, \quad sh(j) = n(j) - out(j), \quad \forall\, j \in D$$

The term *n*(*j*) is the target output for device *j* in constraints (11d) and (11e), that is, $n_j^{\text{min\_chips}}$ for $j \in \{K \cup P\}$, and $-\infty$ for regular device $j \in D \backslash \{K \cup P\}$. There is no output requirement for regular devices.

The benefit of an unassigned lot $l \in L_0$ is defined by the function

$$ben(l) = w_l + \left(w_{d_l} / C \right) \cdot \min \left\{ n_l^{\text{chips}}, sh(d_l) \right\} \cdot I\left\{ sh(d_l) > 0 \right\} \cdot I\left\{ d_l \in \left\{ K \cup P \right\} \right\} \tag{12}$$

where d_l is the device contained in lot l and $I\{\alpha\}$ is an indicator function equal to 1 if the phrase α is "true" and 0 otherwise. The first right-hand-side term in (12) is the lot weight; the second term measures the penalty reduction [potential gain in the objective function (2a)] that would result if the lot contains a key or package device. In the second term, $\left(w_{d_l}/C\right)$ is the unit shortage penalty associated with the chips in lot l. The weight $w_{d_l} = w_1^k$ if $d_l \in K$ and $w_{d_l} = w_2^p$ if $d_l \in P$. For the indicator function $I\{\alpha\}$, when $d_l \in \{K \cup P\}$ and $sh(d_l) > 0$, $I\left\{sh(d_l) > 0\right\} = I\left\{d_l \in \left\{K \cup P\right\}\right\} = 1$. The magnitude of the penalty reduction depends on $\min\left\{n_l^{\text{chips}}, sh(d_l)\right\}$. If $n_l^{\text{chips}} < sh(d_l)$, then all n_l^{chips} chips contained in lot l go towards reducing the penalty. Otherwise, only $sh(d_l)$ of them contribute.

Using the benefit function calculated in (12), a feasible assignment of lots to machines is given by x^{IP}, and then is improved locally. For each machine $i \in M$, let t_i^{IP} be the time consumed by the lots assigned to it in partial solution x^{IP}. With these values in mind, procedure $N_1(x^{\text{IP}})$ is applied to assign as many lots as possible to the available machines in an expedient manner, and then procedure $N_2(x^{\text{IP}})$ is used to perform a neighborhood search giving solution LLP_heur(y). The pseudocodes of the two procedures are provided in Figure A.73 and Figure A.74.

$N_1(x^{\text{IP}})$ (Greedy lot insertion) Sort the unassigned lots $l \in L_0$ in nonincreasing order according to $ben(l)$. Pick the next lot $l \in L_0$ and a machine $i \in M$. If $l \in L(i, \bar{\lambda}_i)$ and $t_i^{\text{IP}} + n_l^{\text{chips}}/r_{ils} \le H_i$, where route $s = \text{argmax}\{r_{ils}, s \in S(i,l,\bar{\lambda}_i)\}$, assign l to machine i and go to the next unassigned lot; otherwise, go to next machine. If l cannot be assigned to any machine, go to the next unassigned lot.

$N_2(x^{\text{IP}})$ (Lot swap) Sort the unassigned lots $l \in L_0$ in nonincreasing order according to $ben(l)$. Pick the next lot $l \in L_0$ and a machine $i \in M$. If $l \in L(i, \bar{\lambda}_i)$, pick a lot $l' \in L_i$. Let $s = \text{argmax}\{r_{ils} : s \in S(i,l,\bar{\lambda}_i)\}$ and $s' = \text{argmax}\{r_{il's} : s \in S(i,l', \bar{\lambda}_i)\}$. If $t_i^{\text{IP}} + n_l^{\text{chips}}/r_{ils} - n_{l'}^{\text{chips}}/r_{il's} \le H_i$ and $ben(l) > ben(l')$, swap lots l and l' and go to next unassigned lot $l \in L_0$; otherwise, go to next lot $l' \in L_i$. If $l \notin L(i, \bar{\lambda}_i)$, lot l cannot be assigned to machine i, go to next machine.

5.2.2 Upper level problem

The solution provided by LLP is a function of the machine setup variables y. The upper level problem (ULP) aims to identify the optimal machine setups such that the overall objective (10a) is maximized. The following mathematical model is used for this purpose.

$$\text{Maximize} \quad \text{LLP\_heur}(y) \tag{13a}$$

$$\text{subject to} \quad \sum_{\lambda \in \Lambda(i)} y_{i\lambda} \leq 1, \ \forall\, i \in M \tag{13b}$$

$$\sum_{i \in M} \sum_{\tau \in \bar{T}(n)} \sum_{\lambda \in \Lambda(i,t,\tau)} b_{\lambda t} y_{i\lambda} \leq \sum_{m \in \bar{N}(n)} n_{t,m}^{\text{tooling}}, \ \forall\, t \in T, n \in N \tag{13c}$$

$$y_{i\lambda} \in \{0,1\}, \ \forall\, i \in M, \lambda \in \Lambda(i) \tag{13d}$$

Constraints (13b) – (13c) repeat (10c) – (10d) and along with (13d) define the feasible machine-tooling pairings. However, model (13) cannot be solved directly since the objective function (13a) is not an explicit function of y. GRASP is proposed as follows to generate solutions. In each phase I iteration, a candidate list (CL) is built from machine-tooling combinations and sorted according to the benefit associated with each. A restricted candidate list (RCL) is then constructed from CL whose length is adjusted based on the quality of solutions obtained from previous iterations. Letting *SIM* be the set of identical machines, a scoring list is also maintained to grade each $(j, \lambda) \in SIM \times \Lambda(i)$. Feasible solutions are constructed by randomly selecting (j, λ) combinations from RCL until all available capacity is used. As mentioned, a subset of solutions generated in phase I is passed to phase II for improvement using neighborhood search.

Building the CL

Each element in CL is a triplet consisting of some $j \in SIM$, a tooling setup $\lambda \in \Lambda(j)$, and the corresponding benefit $ben(j, \lambda, L_0)$, where L_0 is the set of unassigned lots. The benefit is computed by solving the following knapsack problem

$$ben(j, \lambda, L_0) = \max \left\{ \sum_{l \in L(j,\lambda) \cap L_0} ben(l) z_l : \sum_{l \in L(j,\lambda) \cap L_0} \frac{n_l^{\text{chips}}}{r_{ils}} z_l \leq H_i, \ z_l \in \{0,1\}, \ \forall\, l \in L(j,\lambda) \cap L_0 \right\}$$

where $ben(l)$ is the value calculated in (12) when lot $l \in L_0$ is assigned to machine $i \in SIM_j$. The term $n_l^{\text{chips}} / r_{ils}$ is the time required to process lot l on machine i with tooling

setup λ, and route $s = \text{argmax}\{r_{ils} : s \in S(i, l, \lambda)\}$. The decision variables z_l, $\forall\, l \in L(j\ \lambda) \cap L_0$ are binary such that $z_l = 1$ when lot l is assigned to the machine $i \in SIM_j$ and 0 otherwise. The elements in CL are sorted in nonincreasing order of $ben(j, \lambda, L_0)$.

Instead of solving the knapsack problem exactly, a heuristic is used to reduce runtimes. The pseudocode of the heuristic is shown in Figure A.75. In Step 1, the unassigned lots $l \in L_0$ are sorted according to rate of benefit $ben(l) \Big/ \left(n_l^{\text{chips}} / r_{ils} \right)$ in nonincreasing order. In Step 2, the lots are assigned to the machine in a greedy way until there is no more lots can be assigned due to the time limit constraint.

An example of a CL is shown in Table 5.1. The first two columns identify the feasible machine-tooling combinations (j,λ) while the third column gives the benefit $ben(j, \lambda, L_0)$ associated with the knapsack solution. CL is sorted in nonincreasing order of the benefit.

Table 5.1 An example of CL

SIM, j	Tooling setup, λ	$ben(j, \lambda, L_0)$
2	1	100
2	3	90
3	2	80
1	3	70
2	2	60
1	1	50

Self-adjusted RCL

RCL is derived from CL by taking only the top elements. Since RCL guides the construction process in phase I, its length, l_{RCL}, must strike a balance between solution quality and diversity. If l_{RCL} is large then it is likely to produce many inferior initial solutions; if it is small, many good solutions may be missed. Therefore, instead of setting l_{RCL} to a fixed value, it is restricted within the following range: $l_{RCL} \in \{2, 3, \ldots, \bar{l}_{RCL}\}$, where $\bar{l}_{RCL}$ is a (predetermined) maximum length. The value of l_{RCL} is adjusted during the GRASP iterations according to the quality of observed solutions, as described by Paris and Ribeiro (1999).

Let $A = \{\alpha_1, \alpha_{2,\ldots,}\alpha_m\}$ be the set of considered values for l_{RCL} and let p_i be the corresponding probability of selecting α_i, i =1,…,m. Initially, p_i is uniformly distributed; that is,

$$p_i = 1/m,\ \ i = 1,\ldots,m$$

To see how these probabilities are adjusted, let ϕ^* be the best solution found in all previous GRASP iterations and let A_i be the average value of solutions obtained for l_{RCL} = α_i. Now, define

$$q_i = \left(\frac{A_i}{\phi^*}\right)^{\delta},\ \ i = 1,\ldots,m \tag{14a}$$

to be the relative performance of the algorithm under α_i, where δ is a shape parameter. For higher values of δ, q_i will be lower since $A_i \le \phi^*$. Normalizing gives

$$p_i = q_i / \sum_{\gamma=1}^{m} q_\gamma\ ,\ i = 1,\ldots,m \tag{14b}$$

When α_i yields relatively high average solutions, A_i it will have a high probably p_i of being selected as the iterations progress. In the implementation, we set δ =50 and A = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.

Grading the (SIM, Λ) combinations

The quality of the phase I solutions strongly depends on the associated machine-tooling pairings selected. A good (j, λ) combination is one that appears frequently in good solutions. To identify such pairs we devised a scoring list (SL) to grade each (j, λ) combination that arises in phase I. Each element of SL consists of the *SIM* index j, the tooling setup index λ, and the score $S_{j\lambda}$ of the corresponding (j, λ) combination. Let ϕ_k be the objective function value in (4a) found at iteration k of phase I, let $\phi^*_{j\lambda}$ be the best solution found so far using (j, λ), and let $\bar{y}^k_{i\lambda}$ [for all $i \in SIM_j$ and $\lambda \in \Lambda(i)$] be the corresponding machine-tooling pairings. The score $S_{i\lambda}$ is defined as a function of the average objective function value and the best objective function value over all phase I iterations; that is,

$$S_{j\lambda} = \frac{\sum_{1\le k\le n^{\text{Phasel}}} \sum_{i\in SIM(j)} \phi_k I\{\bar{y}^k_{i\lambda} = 1\}}{\sum_{1\le k\le n^{\text{Phasel}}} \sum_{i\in SIM(j)} I\{\bar{y}^k_{i\lambda} = 1\}} \cdot \left(\phi^*_{j\lambda}\right)^2 + c\ ,\ \forall\ j \in SIM,\ \lambda \in \Lambda(j)$$

where n^{PhaseI} is the total number of iterations in phase I and $I\{\cdot\}$ is an indicator function. The numerator of the fraction calculates the total objective value over all iterations when combination (j, λ) was present while the denominator counts the total number of times the combination was applied. The grading scheme emphasizes $\phi^*_{j\lambda}$ and corresponds to our intensification strategy. The constant c is included to avoid setting $S_{i\lambda} = 0$ when some (j, λ) combination is never selected. In the implementation $c = 1000$.

Let (j^i, λ^i, b^i) be the i^{th} element in RCL, $i = 1, 2,\ldots,l_{RCL}$. To determine the probability that this element will be selected we use the same procedure used to determine the length of RCL. The relevant formulas are

$$q_i = \left[\frac{S_{j^i \lambda^i}}{\left(\phi^*\right)^3} \right]^{\delta}, \ \forall\, i = 1, 2,\ldots,l_{RCL} \tag{15a}$$

$$p_i = q_i \Big/ \sum\nolimits_{\gamma=1}^{l_{RCL}} q_\gamma, \ \forall\, i = 1, 2,\ldots,l_{RCL} \tag{15b}$$

where δ is again a shape parameter and is set to 50 to emphasize intensification. According to Eqs. (15a) – (15b), setups with higher scores have a higher probability of being part of a solution; larger values of δ increase the corresponding probabilities. To avoid overemphasizing intensification, a lower limit of ε is imposed on the probability of p_i; that is,

$$\overline{p}_i = \varepsilon, \ \forall\, p_i < \varepsilon, \ i = 1, 2,\ldots,l_{RCL} \tag{15c}$$

$$\overline{p}_i = \left(1 - \varepsilon \sum\nolimits_{r=1}^{l_{RCL}} I\left\{p_r < \varepsilon\right\}\right) \cdot \frac{p_i}{\sum_{r=1}^{l_{RCL}} I\left\{p_r \geq \varepsilon\right\}}, \ \forall\, p_i \geq \varepsilon, \ i = 1, 2,\ldots,l_{RCL} \tag{15d}$$

Equation (15c) eliminates the situation where some elements in RCL cannot be selected due to very lower probability while equation (15d) distributes the remaining probability proportionally to the original value p_i. In the implementation, $\varepsilon = 0.01$ is used to allow appropriate diversification.

Construct initial solutions in phase I

Each phase I iteration produces a feasible solution to model (10) in conjunction with CL, RCL, SL and the aforementioned heuristic solvers. The pseudocode is given in Figure A.77. Initialization is done in Step 1 followed by n^{PhaseI} iterations in Step 2. At the

beginning of each outer iteration, we reset $L_0 \leftarrow L$ and $L_i \leftarrow \emptyset$, $\forall\, i \in M$. The benefit function *ben*(*l*), for all $l \in L$, is then calculated according to Eq. (12) based on which lots have already been assigned. At each inner iteration, CL is built and then truncated according to the probability distribution in Eq. (14b) to get RCL. Exactly one element of RCL is selected with probability based on SL and Eq. (15). The machine corresponding to the selected element is configured with the specified tooling, and all sets, data structures and functions are updated. An inner iteration is halted when there are no more machines or tooling pieces available. At this point the *y* variables are fixed at y^* and LLP is solved to get x^*. The solution (x^*, y^*) is appended to the set S^{PhaseI} and the next outer iteration is performed.

Phase II: LP-based local neighborhood search

Local branching is a technique for embedding metaheuristic concepts such as neighborhood search, intensification and diversification, into branch and bound. The objective is to achieve high quality solutions in reasonable time without necessarily verifying optimality. Given a feasible reference solution $\bar{y}$, let $\bar{Y} = \{(i,\lambda) : \bar{y}_{i\lambda} = 1, i \in M, \lambda \in \Lambda(i)\}$. As proposed by Fischetti and Lodi (2003), we define a local branch cut as follows:

$$\Delta(y, \bar{y}) = \sum_{(i,\lambda)\in\bar{Y}} (1 - y_{i\lambda}) \le K \qquad (16)$$

Constraint (16) generates a neighborhood of radius *K* around the current solution $\bar{y}$. For example, assume that there are 3 machines with solution $\lambda_1 = 1$, $\lambda_2 = 3$ and $\lambda_3 = 2$ or $\bar{y}_{11} = 1$, $\bar{y}_{23} = 1$, $\bar{y}_{32} = 1$ and all other $\bar{y}_{i\lambda} = 0$. The branch cut is $(1 - y_{11}) + (1 - y_{23}) + (1 - y_{32}) \le K$.

Note that instead of including all the binary variables in the cut, as Fischetti and Lodi do, we only include the ULP variables, *y*.

Now, at each phase II outer iteration, model (10) is solved as an LP with constraint (16) added. Let $y^{\text{LP}} = (y^{\text{LP}}_{i\lambda},\ i \in M, \lambda \in \Lambda(i))$ be the relaxed solution for the machine setup variables in the extended model. In light of (10c), the $y^{\text{LP}}_{i\lambda}$ values can be viewed as the probability of setting up machine *i* with tooling configuration λ. These

values are used at each phase II inner iteration to perform a Monte Carlo simulation. That is, for each machine i, a tooling configuration λ is drawn from $\Lambda(i)$ using "probabilities" $y_{i\lambda}^{\mathrm{LP}}$. Once the tooling is selected for the machines, the resulting LLP is solved by the heuristic described in Section 5.1 to obtain the best lot assignments.

The phase II LP-based local branch (LPLB) neighborhood search can be interpreted as a destruction-construction algorithm that works on the upper level decision variables y. The pseudocode of the LPLB algorithm is provided in Figure A.76. The algorithm consists of a series of n^{LPLB} Monte Carlo replications. In replication s, the machine setups are simulated using y^{LP} as probabilities to get y^s and, if feasible to (10d), the resultant LLP is solved to get a feasible integer solution (x^s, y^s). The objective function value, denoted by $sim\_obj(x^s, y^s)$, is compared to the incumbent, and when an improvement is identified, (x^*, y^*) is updated. When y^s is infeasible it is discarded and the next replication performed. In the implementation $n^{\mathrm{LPLB}} = 10$.

5.2.3 Summary of GRASP

In phase I, good feasible solutions are constructed in a greedy way using a benefit function for each lot and RCL. A subset of these solutions is improved by LPLB in phase II. The best feasible solution found at the end of phase II is output. A high level pseudocode is given in Figure A.78.

5.3 Computational Results

The proposed GRASP was implemented in C++ and tested under Ubuntu Linux on a Dell Poweredge 2950 workstation with 2 dual core hyperthreading 3.73 GHz Xeon processors and 8 GB physical memory. In the numerical experiments, the test cases were randomly generated from a dataset provided by TI. Model (10a) – (10h) was solved heuristically with GRASP and directly with CPLEX 11.0. The following parameter settings were used for GRASP,

- $n^{\mathrm{PhaseI}} = 1000$
- For RCL, $\bar{l}_{RCL} = 11$, $\delta = 50$
- For the grading scheme, $\delta = 50$, $c = 1000$, $\varepsilon = 0.01$
- In phase II, $K = 1$, $n^{\mathrm{LPLB}} = 10$

- COIN CLP was used as the LP solver (http://www.coin-or.org/projects/Clp.xml)

For CPLEX, a limit of 3600 sec was imposed on all runs.

5.3.1 Random test instances

The basic data set consisted of 84 machines divided into 8 families, 2078 lots of which 80% were either key or package devices, 106 tooling pieces divided into 28 families, and 1 temperature.

CPLEX was able to solve this problem in negligible time due to the excess capacity relative to the number of lots. In addition, many of the routes did not require any tooling. To create more difficult instances that better reflect the operational environment, a series of representative cases are constructed using the following random case generator. The pseudocode of the random cases generator is provided in Figure A.79.

5.3.2 Comparison of GRASP with CPLEX

Two datasets were generated for testing purposes. The first has $|L^{\text{test}}| = 1000$ lots, $|M^{\text{test}}| = 80$ machines, $m_g = 10$ machine groups, $|T^{\text{test}}| = 30$ tooling families, $n_D = 100$ devices with $n_K = 40$ key devices and $n_P = 40$ package devices, and $n_{\text{temp}} = 3$ operating temperatures. The number of tooling pieces t^{test} was selected from the set {100, 300, 500}. Ten random cases were generated and solved by CPLEX and GRASP for each t^{test} value. Table 5.2 and Table 5.3 report the results for $|L^{\text{test}}| = 1000$ and $t^{\text{test}} = 100$. In Table 5.2, the columns denoted by ϕ^{CPX}, ϕ^{I} and ϕ^{II} are the best solutions found by CPLEX, GRASP phase I and GRASP phase II, respectively. The columns TW and TS give the total lot weights and the total shortages of key and package devices for the best solutions. The gap Δ^{CPX} is the optimality gap given by CPLEX, while Δ^{GRASP} is the percentage gap between the GRASP and CPLEX solutions: $[|\phi^{\text{CPX}} - \phi^{\text{II}}| / |\phi^{\text{CPX}}|] \times 100$.

According to the results, none of the CPLEX runs converged within 3600 sec (the average optimality gap was 17.35 %). GRASP required much less time, averaging 536 sec, and producing solutions that were 3.41% on average, as indicated by Δ^{GRASP}. For problem nos. 1, 2 and 7, the negative gap indicates that GRASP outperformed CPLEX. As seen in Table 5.3, no improvement was obtained in phase II. This was due to the fact that the tooling pieces are very limited so there are few if any good options within a neighborhood.

Table 5.3 also provides some individual performance measures for CPLEX and GRASP. Column 2 reports the size of the search tree and column 3 indicates the node at which the best solution was found. For all but problem no. 10, CPLEX uncovered the best solution late in the search tree. Column 4 reports the iteration at which GRASP found the best solution, which was similarly towards the end of the process, except for problem no. 4. The last two columns indicate the number of phase I solutions that were carried over to phase II and the corresponding percentage improvement. For smaller instances than those investigated here, we generally found improvements averaging 5%, but for the reasons previously mentioned, phase II was not successful on instances with 1000 or more lots and compatible numbers of tooling pieces.

Table 5.4 reports the results for CPLEX and GRASP when $|L^{test}| = 1000$ and $t^{test} = 300$. As the number of tooling pieces increase the problems become easier to solve by CPLEX, which now exhibits gaps, Δ^{CPX}, of less than 0.10%. In contrast, GRASP has a harder time executing phase 1 because there are many more (j, λ) combinations to explore. Runtimes for these instances averaged about 1000 sec, and the corresponding solutions are with gaps less than 3% in all cases. The TS values obtained by GRASP are identical to the values obtained by CPLEX except problem nos. 4, 5 and 6. Once again, however, phase II provided no improvement since the tooling pieces are still limited (see Table 5.5).

Table 5.6 and Table 5.7 report the results for GRASP and CPLEX when $|L^{test}| = 1000$ and $t^{test} = 500$. The TS values are identical in all cases and the gap, as measured by Δ^{GRASP}, is well under 0.10%, although GRASP uncovered slightly better solutions for problem nos. 1 and 7. No improvement was obtained during phase II. The average computational time of CPLEX is 1234 sec, which is slightly more than the average computational time of GRASP 1078 sec.

The second set of experiments involved the same parameter settings as the first except that $|L^{test}| = 2000$. For $t^{test} = 100$, the computational results are reported in Table 5.8 and Table 5.9 where CPLEX is seen to have a much more difficulty time then previously. The optimality gap Δ^{CPX} averaged 66.27%. A large gap Δ^{GRASP} around 20% can be observed for problem nos. 3 and 4. However, In 7 out of the remaining 8 cases, GRASP obtained better solutions than CPLEX in less than half the time.

Table 5.10 and Table 5.11 report the results for $|L^{test}| = 2000$ and $t^{test} = 300$. In 6 out of the 10 cases GRASP obtained better solutions than CPLEX with a maximum gap 7.80%. GRASP obtained better solutions in the rest of the cases especially for problem no. 1 with a gap –32.89% Again, phase II failed to provide better results, this time because the phase I solutions are almost optimal. Table 5.12 and Table 5.13 report the results for $|L^{test}| = 2000$ and $t^{test} = 500$. The gaps between the GRASP and CPLEX solutions are well under 1% except that a 7.46% gap was observed in problem no. 3. The remaining statistics and the phase II results parallel those obtained in the other experiments.

According to the reported results, the problem was hard to solve when the number of tooling pieces were very limited. CPLEX always experienced a hard time to solve the problem while the gaps Δ^{GRASP} were big under this situation. As the number of tooling pieces increased, the problem became easier to solve by both CPLEX and GRASP and the gaps Δ^{GRASP} became much smaller.

5.4 Summary

This chapter presented a mathematical model to schedule the back-end operations in semiconductor manufacturing. To solve the problem efficiently without reliance on third-party software, a two-level heuristic was developed within a reactive GRASP framework. The novelty in phase I of the GRASP centered on the dynamic adjustment of RCL in accordance with the solution quality and the use of a grading scheme to guide the machine setups. In phase II, a neighborhood search based on local branching and Monte Carlo sampling was devised to improve phase I solutions. Extensive testing showed that comparable objective function values could be obtained with the GRASP, often in significantly less time than required by CPLEX. These results confirmed that public domain software combined with intelligent heuristics can be competitive with top commercial products. Nevertheless, there is still room for improvement with respect to the phase II algorithm. A post-processor such as path relinking coupled with local branching could be applied to the set of elite solutions to increase the intensity of the neighborhood search.

Table 5.2 Comparison of numerical results from CPLEX and GRASP with $|L^{test}| = 1000$ and $t^{test} = 100$

Prob. No.	CPLEX					GRASP								
	Best solution found			Time (sec)	Δ^{CPX} (%)	Phase I solution				Phase II solution				Δ^{GRASP} (%)
	ϕ^{CPX} ($\times 10^7$)	TW	TS			ϕ^{I} ($\times 10^7$)	TW	TS	Time (sec)	ϕ^{II} ($\times 10^7$)	TW	TS	Time (sec)	
1	–8.5697	71,900	1,695,270	3600	19.68	–8.4714	59,312	1,675,600	682	–8.4714	59,312	1,675,600	54	–1.15
2	–4.0430	59,456	1,450,580	3600	8.49	–3.9861	64,884	1,430,390	475	–3.9861	64,884	1,430,390	17	–1.41
3	–30.1709	60,872	1,494,290	3600	16.87	–30.8340	60,087	1,527,120	399	–30.8340	60,087	1,527,120	51	2.20
4	–4.4618	58,955	1,387,610	3600	20.76	–4.5520	59,972	1,415,650	458	–4.5520	59,972	1,415,650	34	2.02
5	–3.7262	47,423	1,483,700	3600	11.24	–3.9891	42,897	1,588,090	359	–3.9891	42,897	1,588,090	32	7.06
6	–31.3312	82,524	1,397,250	3600	18.20	–33.4955	69,081	1,493,690	959	–33.4955	69,081	1,493,690	52	6.91
7	–3.3032	57,451	1,236,520	3600	27.42	–3.2716	60,196	1,224,840	440	–3.2716	60,196	1,224,840	23	–0.96
8	–7.4827	56,883	1,594,640	3600	17.95	–8.1717	43,477	1,741,090	393	–8.1717	43,477	1,741,090	39	9.21
9	–26.1961	43,562	1,553,010	3600	16.24	–27.4550	44,095	1,627,630	287	–27.4550	44,095	1,627,630	14	4.81
10	–7.6735	54,413	1,439,770	3600	16.67	–8.08714	54,248	1,517,320	552	–8.08714	54,248	1,517,320	37	5.39

Table 5.3 Performance of CPLEX and GRASP for $|L^{test}| = 1000$ and $t^{test} = 100$

Problem no.	CPLEX		GRASP		
	Tree size	Node best soln found	Iteration best found	No. phase I improved	Improve (%)
1	3100	3014	956	0	0
2	3752	2719	931	0	0
3	3750	2590	965	0	0
4	5052	5047	554	0	0
5	8165	6715	780	0	0
6	2625	2604	988	0	0
7	3965	3664	925	0	0
8	15173	14585	650	0	0
9	18382	10224	698	0	0
10	7116	2506	978	0	0

Table 5.4 Comparison of numerical results from CPLEX and GRASP with $|L^{test}| = 1000$ and $t^{test} = 300$

Prob. No.	CPLEX					GRASP								
	Best solution found			Time (sec)	Δ^{CPX} (%)	Phase I solution				Phase II solution				Δ^{GRASP} (%)
	ϕ^{CPX} ($\times 10^7$)	TW	TS			ϕ ($\times 10^7$)	TW	TS	Time (sec)	ϕ ($\times 10^7$)	TW	TS	Time (sec)	
1	–5.4536	95,832	1,079,830	2798	0.01	–5.4538	93,693	1,079,830	939	–5.4538	93,693	1,079,830	24	0.00
2	–2.8628	95,897	1,029,370	942	0.01	–2.8638	94,581	1,029,380	892	–2.8638	94,581	1,029,380	32	0.03
3	–28.1722	93,791	1,395,480	428	0.00	–28.1718	97,532	1,395,480	695	–28.1718	97,532	1,395,480	30	0.00
4	–3.3378	98,460	1,039,720	2046	0.01	–3.3789	98,816	1,052,500	1091	–3.3789	98,816	1,052,500	24	1.23
5	–2.8879	89,933	1,152,030	3600	0.10	–2.8858	92,108	1,151,280	1111	–2.8858	92,108	1,151,280	39	–0.07
6	–21.0521	98,248	939,039	456	0.00	–21.5633	96,662	961,824	972	–21.5633	96,662	961,824	38	2.43
7	–2.2370	96,537	839,557	3600	0.01	–2.2368	98,120	839,557	712	–2.2368	98,120	839,557	25	–0.01
8	–4.5180	99,991	964,227	2359	0.00	–4.5180	99,972	964,227	1358	–4.5180	99,972	964,227	38	0.00
9	–15.4814	79,913	918,117	3600	0.01	–15.4806	87,262	918,117	787	–15.4806	87,262	918,117	20	–0.01
10	–4.1144	96,713	773,243	3600	0.01	–4.1145	95,562	773,243	927	–4.1145	95,562	773,243	45	0.00

Table 5.5 Performance of CPLEX and GRASP for $|L^{test}| = 1000$ and $t^{test} = 300$

Problem so.	CPLEX		GRASP		
	Tree size	Node best soln found	Iteration best found	No. phase I improved	Improve (%)
1	666	666	581	0	0.00
2	522	522	700	0	0.00
3	526	526	136	0	0.00
4	773	773	654	0	0.00
5	600	509	794	0	0.00
6	320	320	773	0	0.00
7	997	790	424	0	0.00
8	891	891	701	0	0.00
9	921	662	967	0	0.00
10	925	823	131	0	0.00

Table 5.6 Comparison of numerical results from CPLEX and GRASP with $|L^{test}| = 1000$ and $t^{test} = 500$

Prob. No.	CPLEX					GRASP								
	Best solution found			Time (sec)	Δ^{CPX} (%)	Phase I solution				Phase II solution				Δ^{GRASP} (%)
	ϕ^{CPX} ($\times 10^7$)	TW	TS			ϕ ($\times 10^7$)	TW	TS	Time (sec)	ϕ ($\times 10^7$)	TW	TS	Time (sec)	
1	–5.8021	97,052	1,148,730	439	0.01	–5.8017	101,127	1,148,730	1106	–5.8017	101,127	1,148,730	116	–0.01
2	–2.3737	96,375	853,843	1510	0.01	–2.3751	82,153	853,843	715	–2.3751	82,153	853,843	30	0.06
3	–16.1078	93,266	798,081	431	0.00	–16.1072	99,410	798,081	709	–16.1072	99,410	798,081	23	0.00
4	–2.7629	100,102	861,206	2341	0.00	–2.7628	100,195	861,206	1045	–2.7628	100,195	861,206	42	0.00
5	–2.4324	99,293	971,250	312	0.01	–2.4323	100,183	971,250	1212	–2.4323	100,183	971,250	91	0.00
6	–15.6539	87,528	698,312	2073	0.01	–15.6538	88,542	698,312	757	–15.6538	88,542	698,312	40	0.00
7	–1.7049	97,360	640,756	2704	0.01	–1.7048	98,745	640,756	1282	–1.7048	98,745	640,756	46	–0.01
8	–5.0706	100,476	1,081,930	1066	0.00	–5.0705	101,331	1,081,930	1098	–5.0705	101,331	1,081,930	29	0.00
9	–15.4451	98,403	916,078	344	0.00	–15.4450	99,634	916,078	1221	–15.4450	99,634	916,078	28	0.00
10	–4.9188	99,011	924,118	1123	0.00	–4.9187	100,863	924,118	1145	–4.9187	100,863	924,118	40	0.00

Table 5.7 Performance of CPLEX and GRASP for $|L^{test}| = 1000$ and $t^{test} = 500$

Problem no.	CPLEX		GRASP		
	Tree size	Node best soln found	Iteration best found	No. phase I improved	Improve (%)
1	280	280	717	0	0.00
2	1277	1277	345	0	0.00
3	529	529	793	0	0.00
4	811	811	477	0	0.00
5	488	488	982	0	0.00
6	848	848	979	0	0.00
7	592	592	214	0	0.00
8	502	502	58	0	0.00
9	500	500	807	0	0.00
10	499	499	44	0	0.00

Table 5.8 Comparison of numerical results from CPLEX and GRASP with $|L^{test}| = 2000$ and $t^{test} = 100$

Prob. No.	CPLEX					GRASP								
	Best solution found			Time (sec)	Δ^{CPX} (%)	Phase I solution				Phase II solution				Δ^{GRASP} (%)
	ϕ^{CPX} ($\times 10^7$)	TW	TS			ϕ ($\times 10^7$)	TW	TS	Time (sec)	ϕ ($\times 10^7$)	TW	TS	Time (sec)	
1	–5.4864	84,540	930,132	3600	77.90	–5.5560	78,141	941,820	375	–5.5560	78,141	941,820	69	1.27
2	–3.1812	105,652	778,564	3600	49.06	–2.8638	107,835	701,186	761	–2.8638	107,835	701,186	40	–9.98
3	–27.0091	94,542	861,342	3600	49.23	–32.9753	109,566	1,051,590	539	–32.9753	109,566	1,051,590	38	22.09
4	–5.5085	125,678	643,495	3600	93.17	–6.5228	108,740	761,519	907	–6.5228	108,740	761,519	93	18.41
5	–3.4350	126,273	602,702	3600	77.25	–3.0824	121,097	540,959	1494	–3.0824	121,097	540,959	69	–10.26
6	–22.9703	108,139	862,916	3600	61.30	–22.3600	107,964	840,001	715	–22.3600	107,964	840,001	117	–2.66
7	–4.2288	108,747	916,185	3600	57.10	–3.7904	105,744	821,379	684	–3.7904	105,744	821,379	76	–10.37
8	–5.2215	91,613	1,216,760	3600	75.01	–4.6449	90,406	1,072,270	603	–4.6449	90,406	1,072,270	64	–11.04
9	–33.9759	90,762	1,523,710	3600	57.15	–32.0446	99,247	1,437,160	1112	–32.0446	99,247	1,437,160	80	–5.68
10	–4.4439	88,925	566,490	3600	65.56	–4.1040	81,656	523,155	538	–4.1040	81,656	523,155	71	–7.65

Table 5.9 Performance of CPLEX and GRASP for $|L^{test}| = 2000$ and $t^{test} = 100$

Problem no.	CPLEX		GRASP		
	Tree size	Node best soln found	Iteration best found	No. phase I improved	Improve (%)
1	1466	1427	985	0	0.00
2	1266	929	882	0	0.00
3	2457	2409	169	0	0.00
4	1295	1181	919	0	0.00
5	1160	1097	807	0	0.00
6	1483	1473	897	0	0.00
7	2370	2361	638	0	0.00
8	1421	895	787	0	0.00
9	2562	2551	196	0	0.00
10	1366	788	932	0	0.00

Table 5.10 Comparison of numerical results from CPLEX and GRASP with $|L^{test}| = 2000$ and $t^{test} = 300$

Prob. No.	CPLEX					GRASP								
	Best solution found			Time (sec)	Δ^{CPX} (%)	Phase I solution				Phase II solution				Δ^{GRASP} (%)
	ϕ^{CPX} ($\times 10^7$)	TW	TS			ϕ ($\times 10^7$)	TW	TS	Time (sec)	ϕ ($\times 10^7$)	TW	TS	Time (sec)	
1	–0.3247	173,051	57,887	3600	42.77	–0.2179	146,519	39,372	1004	–0.2179	146,519	39,372	54	–32.89
2	–2.4612	171,423	79,008	3600	0.04	–2.6532	153,240	85,071	853	–2.6532	153,240	85,071	39	7.80
3	–1.1027	169,721	130,505	3600	0.09	–1.1045	151,884	130,505	1239	–1.1045	151,884	130,505	72	0.16
4	–1.2748	193,849	226,242	3600	1.46	–1.2590	177,348	223,197	1374	–1.2590	177,348	223,197	71	–1.24
5	–4.1694	166,446	157,180	3600	3.19	–4.0393	163,231	152,283	1678	–4.0393	163,231	152,283	126	–3.12
6	–0.4257	196,148	96,233	3600	0.11	–0.4266	187,223	96,233	1892	–0.4266	187,223	96,233	115	0.21
7	–0.4784	178,110	114,333	3600	0.23	–0.4816	146,824	114,333	1121	–0.4816	146,824	114,333	75	0.67
8	–1.1708	185,760	151,310	3600	0.09	–1.1719	174,780	151,310	2682	–1.1719	174,780	151,310	119	0.09
9	–4.2825	183,949	164,267	3600	0.03	–4.2835	174,321	164,267	1533	–4.2835	174,321	164,267	93	0.02
10	–0.4556	176,590	135,321	3600	7.20	–0.4469	154,138	132,206	910	–0.4469	154,138	132,206	81	–1.91

Table 5.11 Performance of CPLEX and GRASP for $|L^{test}| = 2000$ and $t^{test} = 300$

Problem no.	CPLEX		GRASP		
	Tree size	Node best soln found	Iteration best found	No. phase I improved	Improve (%)
1	1179	1178	11	0	0.00
2	913	789	1	0	0.00
3	1082	805	960	0	0.00
4	811	711	736	0	0.00
5	1200	1131	393	0	0.00
6	1473	1471	952	0	0.00
7	900	892	920	0	0.00
8	1193	1111	632	0	0.00
9	1173	571	851	0	0.00
10	591	590	12	0	0.00

Table 5.12 Comparison of numerical results from CPLEX and GRASP with $|L^{test}| = 2000$ and $t^{test} = 500$

Prob. No.	CPLEX					GRASP								
	Best solution found			Time (sec)	Δ^{CPX} (%)	Phase I solution				Phase II solution				Δ^{GRASP} (%)
	ϕ^{CPX} ($\times 10^7$)	TW	TS			ϕ ($\times 10^7$)	TW	TS	Time (sec)	ϕ ($\times 10^7$)	TW	TS	Time (sec)	
1	–0.3732	192,442	66,433	3600	0.32	–0.3740	184,422	66,433	2599	–0.3740	184,422	66,433	140	0.21
2	–0.3094	199,924	80,345	3600	0.10	–0.3099	194,460	80,345	2447	–0.3099	194,460	80,345	456	0.16
3	–4.6566	185,417	149,042	3600	0.03	–5.0040	158,389	160,030	1108	–5.0040	158,389	160,030	69	7.46
4	–1.0001	191,310	118,789	3600	0.04	–1.0009	182,550	118,789	1618	–1.0009	182,550	118,789	222	0.08
5	–0.8423	197,065	150,688	3600	0.10	–0.8447	173,129	150,688	2057	–0.8447	173,129	150,688	287	0.28
6	–2.6890	193,261	101,693	1561	0.01	–2.6888	194.371	101,693	4169	–2.6888	194.371	101,693	227	–0.01
7	–0.9460	191,589	208,567	3600	0.10	–0.9458	193,178	208,567	2436	–0.9458	193,178	208,567	138	–0.02
8	–0.3732	190,726	90,372	3600	0.13	–0.3738	183,961	90,372	1530	–0.3738	183,961	90,372	97	0.16
9	–0.2374	193,224	32,657	3600	0.14	–0.2371	196,093	32,657	2201	–0.2371	196,093	32,657	169	–0.13
10	–2.7069	169,208	104,032	3600	0.04	–2.7083	154,934	104,032	1584	–2.7083	154,934	104,032	120	0.05

Table 5.13 Performance of CPLEX and GRASP for $|L^{test}| = 2000$ and $t^{test} = 500$

Problem so.	CPLEX		GRASP		
	Tree size	Node best soln found	Iteration best found	No. phase I improved	Improve (%)
1	1237	1115	767	0	0.00
2	1656	1359	13	0	0.00
3	947	618	12	0	0.00
4	1529	1027	102	0	0.00
5	1921	1780	18	0	0.00
6	677	677	45	0	0.00
7	1803	1601	977	0	0.00
8	951	771	959	0	0.00
9	1782	1182	691	0	0.00
10	1405	1403	649	0	0.00

Chapter 6

Assessment of Research

The general idea underlying this research was to determine the extent to which exact optimization methods could be combined with metaheuristics to solve practical problems. The work is divided into three separate but related projects corresponding to Chapters 3, 4 and 5, respectively. In this chapter, an assessment is presented for each part of the research. The proposed algorithms are first summarized and their numerical results evaluated. Their weaknesses are then discussed and suggestions are made for possible improvement. Some research experiences are also shared and future work on potential enhancements is outlined.

6.1 Capacitated Clustering

The problem addressed in this research topic is a classical capacitated clustering problem which has been well studied along with many variants. The contribution of this research mainly concerns the development of an efficient and effective heuristic to solve the problem in reasonable time compared to the commercial software package CPLEX. The proposed methodology combines GRASP and PR, and is a mixture of various heuristic ideas. Two approaches, HWE and CMC, are available for phase I while three options, CNS, VND and RVND, are used to perform the phase II neighborhood search. A PR post-processing procedure with either CLS or PLS is also implemented for potential improvement over the GRASP elite solutions.

The algorithm has been tested extensively on different datasets with various parameter settings. According to the results, the performance of GRASP+PR is very stable and always consumes much less time than CPLEX to obtain very high quality, if not optimal, solutions. This allows application of GRASP+PR to practical problems. However, there are still some drawbacks to the algorithm. Just like any other heuristic, it is very difficult or even impossible to guarantee that GRASP+PR can generate good solutions for large-scale problems. As the number of nodes to be clustered increases, the CL, RCL as well as the size of the three neighborhoods increase exponentially. Finding a

good solution becomes more and more difficult as the size of the problem increases. A bounding technique may be necessary in order to assess the performance of the algorithm for larger cases.

The proposed heuristic can be enhanced in several ways. Currently, the GRASP iterations are mutually independent, that is, each subsequent iteration starts from scratch after the previous iteration is finished. The effort spent on previous iterations is simply ignored. In order to utilize the results from previous iterations, a memory based learning technique can be introduced to the current algorithm. This variant of GRASP is also referred to as Greedy Randomized Adaptive Memory Programming (GRAMP). The idea is to synthesize the results from previous iterations to guide the algorithm at the current iteration. As an example, the results can be used to guide the construction of CL. This approach has actually been implemented in Chapter 5 in the form of SL to assign probability to the CL elements. The scheme is promising according to the results. With respect to phase II improvement, an idea common in tabu search can also be applied to the algorithm. By restricting repeated neighborhood movements it should be possible to overcome a local optimum.

In any case, the proposed GRASP demonstrates that a measurable advantage can be achieved by combining different heuristic ideas to solve the clustering problem efficiently. The methodology can serve as a guideline for future algorithm development for solving related optimization problems of practical size.

6.2 Midterm Planning in Semiconductor Manufacturing

The midterm planning problem in semiconductor manufacturing turns out to be a large-scale LP which cannot be solved directly. The contribution of this research relates to the development of a decomposition scheme to fully utilize excess machine time. In the proposed algorithm, the full problem is divided into weekly subproblems. The final WIP of the previous week is treated as the initial WIP of the next week. For each subproblem, the corresponding LP is solved in an attempt to minimize the total weighted deviations from specified targets. In light of those results, an attempt is then made to utilize any machine time remaining by solving a rescheduling problem followed by a heuristic WIP-pushing algorithm. As reported in Chapter 4, great improvement was realized for the 4-

week problem when compared to the results obtained from a pulling decomposition approach. For the 13-week problem, the results indicated that shortages are unavoidable due to the machine capacity constraints. However, it is still hard to evaluate the solution to the 13-week problem since the optimal solution is unknown.

Lagrangian relaxation and Benders decomposition were also investigated but the results were not promising. For Lagrangian relaxation, the machine capacity constraints (3d) were removed and appended to the objective function (3a) as a penalty term using Lagrangian multipliers as weights. This led to individual subproblems for the device families which solved quickly. A standard subgradient algorithm was applied to update the multipliers at each iteration. However, it turned out to be that the multipliers never converged due to numerical difficulties.

Benders decomposition instead breaks the full problem into weekly subproblems. At the beginning of Benders decomposition, the initial and final WIP values are specified for each week. The subproblem duals are then solved one by one for the given WIP levels to generate extreme points and extreme rays, which are appended to the master problem. The master problem is then resolved to generate new WIP profiles for each week. The updated WIP values are again sent to each subproblem to generate new cuts. The algorithm iterates in this way until a stopping criterion is satisfied.

Unfortunately, convergence was never achieved at the subproblem level. According to the algorithm, both the initial WIP and the final WIP are fixed for each subproblem, which is then solved to minimize the sum of the target output deviations. However, given the initial WIP the subproblem will be infeasible when it is not possible to schedule production over the week to meet the final WIP. In this case, only feasibility cuts are generated by the subproblems. Experience showed that after thousands of iterations, no optimality cuts were ever generated so the algorithm never converged.

One way to reduce or eliminate the infeasibility is to introduce deviation variables into the WIP conservation constraints (6b) in the last time period of each week. In the objective function (6a), each WIP deviation variable would be highly penalized to force a solution that zeroed the mismatch whenever possible. In the current formulation, there appears to be insufficient incentive for the model to narrow this gap, although the full

problem is feasible. The updated subproblem will always be feasible and generate optimality cuts only, although there may still be a mismatch in WIP levels at termination. In that situation, a heuristic would be needed to smooth the WIP while trying to maintain the machine assignments that appear in the final solution. This enhancement is considered future work.

6.3 Back-End Operations in Semiconductor Manufacturing

The third part of the research was aimed at determining machine-tooling combinations and corresponding lot assignments to maximize throughput during back-end operations while minimizing shortages of key and package devices. The mathematical model of the problem has the characteristics of a two-level assignment problem. The contribution of the research centered on the development of a GRASP that was shown to be comparable in performance to CPLEX. Extensive numerical testing indicated problem instances arising in two of Texas Instruments' facilities could be solved quickly and provided measurably improvement over current practice. The GRASP has been under tested for implementation in the AT facilities in Asia recently.

In phase I of the GRASP, the parameter δ is set to 50 when constructing both RCL and SL. This was determined after running a set of initial experiments which indicated that it is best to emphasize intensification in building RCL and SL in order to achieve high quality machine-tooling combinations. The disadvantage of this setting is that many similar solutions are generated during the run. In other words, there could be a danger for over intensification, although diversification is built into the algorithm in the design of the calculations in (15c) and (15d). One way to remedy this situation is to adjust δ according to the solution. If similar solutions keep appearing then the value of δ should be reduced. Otherwise, after a few iterations, its value should be increased until it reaches some specified upper bound, say, 50.

As discussed in Chapter 5, there was no improvement in phase II LBLP for the randomly generated test cases. The primary reason can be found in the limited number of free tooling pieces that could be paired with the machine that became available at the given iteration; in other words, there was little room for neighborhood swaps. The value of K in LBLP was varied in preliminary tests but the results were consistently poor. A

second reason for the ineffective of phase II relates to the LP relaxation of the LB problem. Because the solution was so fractional few *y*-variables could be fixed and binary sampling from the corresponding probabilities during the simulation often left machines idle. One way to improve phase II would be to solve the resulting problem directly as an IP rather than using LBLP but the results would depend on the performance of the IP solver from COIN-OR. Recall that the rationale for developing the GRASP was to use open source software.

Branch and price (B&P) is another way to try to solve the AT problem without relying too heavily on a commercial IP code. A master problem could be created from model (10) so that each column represents a machine-tooling combination along with compatible lot assignments. The objective function coefficient associated with a column would be the benefit gained by processing the assigned lots. Initial columns could be generated from the heuristic solution. In B&P, the master problem is solved to provide dual prices that are used to construct the subproblem objective functions. In this approach, the subproblems are simply knapsack problems and can be solved heuristically. Columns that price out negatively are appended to the master problem which is then resolved to yield new dual prices. The algorithm iterates accordingly until no new columns are generated or until some other stopping criteria are satisfied. If the values of the decision variables turn out to be integral then the algorithm stops and the solution found is optimal. Otherwise, the current node is partitioned following the logic of branch and bound and column generation is applied at each descendent node. The procedure terminates when all the nodes in the search tree are fathomed.

A preliminary B&P algorithm has been implemented for the AT problem and the results are promising. The root node was solved after around 40 iterations. The algorithmic remaining components, such as the design of a branching strategy and the use of stabilization, are left for future work.

In building model (10) for the back-end operations a number of assumptions were made, the most critical being that all machines are idle at the beginning of the planning horizon. It was further assumed that machines are to be set up once at time zero and that the tooling allocations and temperature settings cannot be changed during the planning horizon. Moreover, fractional lot processing is not allowed even when a machine is idle.

Finally, the problem is solved for a static WIP profile that is given at time zero so there is no accounting for new arrivals or rework.

In reality, these assumptions may not hold. First, the number of lots available for processing changes periodically as upstream operations are completed and as downstream operations produce reentrant flow. In the latter case, some lots may return to a particular test area for additional testing after completing the preliminary round. Thus, the actual WIP profile changes over time with some lots showing up repeatedly. Moreover, at the beginning of the planning horizon most machines are occupied, and it is often necessary to alter their setup and operating temperature if throughput is to be maximized. Because the planning horizon defines a somewhat arbitrary cutoff point, practical considerations dictate that machines not be left idle when there are no lots that fit completely within the remaining time, but instead that setups be scheduled so that waiting lots can begin processing.

The current algorithm needs to be enhanced to accommodate the aforementioned shortcomings. The first step is to develop an iterative scheme to capture the updated WIP. The initial machine status can be addressed by adjusting the available machine time. If a machine is processing lots at time zero then it is a simple matter to reduce the available time on that machine to by the amount equal to the required processing time of the remaining lots in queue. Factional lot processing can also be taken care by post-processing the solution produced by the AT algorithm. However, it turns out to be difficult to handle multiple machine setups without introducing many more binary variables. Several heuristics are now being considered but the best way to modify the algorithm is left to future research.

Appendix 1: Input Files, Data Structure and Output Files for the DMOS6 Tool

To run the program the user must provide a set of input files described below. Some of these files need to be modified in accordance with the following instructions. This appendix discusses the data objects, how to prepare the input files, the ways to compute processing rates, and the output files.

Required input files

(1) Routing.csv

This file includes the routes of the different devices. Each line corresponds to a step, which is defined as a combination of a *logpoint* and an *operation*. Thus a step is a logpoint-operation level definition. The file also includes various "*K* parameters" which are used to compute the processing rate for each step. The fields in the table are shown in the Appendix 2.

Note:

(a) The "Facility" fields are always "DMOS6".

(b) The table should be sorted in ascending order according to the field "OrigRecNo" before processed by the program.

(c) When this file is read into the program the processing rate for each step is computed. The procedures used to compute the processing rates are also included in this appendix. A data object named "Production" is created when this file is read.

(2) Stations.csv

The file contains information of the machines such as Misti-ID and "*K* Parameters". The fields in the table are shown in the Appendix 2. Some of the "*K* Parameters" are used to compute the processing rates for the steps. A data object named "Machine_Set" is created to store the information for the machines.

(3) Consols.csv and Family.csv

The "Consols.csv" file contains the information on the device family. The file is not read by the program. However, the user needs to manipulate the file to

generate another file named “Family.csv”, which will be used by the program. It is easy to generate the file “Family.csv” from the file “Consols.csv”. The user only needs to choose the family he wants to process and select the devices belonging to this family. For instance, if the user is interested in the family “C035,” then he can select all of the devices belonging to “C035” and copy-paste to the file “Family.csv”. A portion of a “Family.csv” is shown in Table A.1. Note that the user needs to append some character like comma in the last column. The appended character is used to delimitate the fields when it is read by the program.

Table A.1 Example of Family.csv

C035	C027	C021	,
B4JRD15060BPP	B5BJF761924B	EZ/E771676D17	,
B4JRD15060B4P	EZ/E761536Z00	EZ/F771657A02	,
B4JRD15075BPP	EZ/E761536Z02	EZ/F771657A07	,
B4JRD751686J4P	EZ/E761541A00	EZ/F771657A08	,
B4JRF751613D4P	EZ/E761560Z00	EZ/F771657A09	,
B4JRF751625PF	EZ/E761909A16	EZ/F771657A10	,
B4JRF751625PP	EZ/F761503B23	EZ/F771657A11	,
B4JRF7516254F	EZ/F761504A02	EZ/F771657A11L	,
B4JRF7516254P	EZ/F761504B08	EZ/F771657A12	,
B4JRF751672D4P	EZ/F761504B10	EZ/F771657A13	,
B4JRF751989B4P	EZ/F761522A49	EZ/F771657A14	,
B4JSF751613HHT	EZ/F761522A50	EZ/F771657Z11	,
B4JTD15060C	EZ/F761522A51	EZ/F771657Z12	,
…	...	…	,

The file “Family.csv” is read by the program to identify the devices belonging to the same family. For instance, Table A.1 identifies three families with their corresponding devices. For each family a device is selected to represent the entire family. Such a device is called the “representative device” and includes information on the number of daily input blank wafers, the initial inventory, and the daily output. When the file “Family.csv” is read by the program a data object named “Family” is created to store the information.

(4) wip_data.txt

This file contains the information on the initial inventory or WIP in the shop. The fields of the file are given in the Appendix 2.

For each representative device, the initial WIP of the other devices in the same family is aggregated to be the initial WIP of the representative device. The initial WIP is computed as follows.

$$\text{Init\_inventory} = \text{Prime\_inv} + \text{Rework\_inv} + \text{Hold\_inv}$$

Note that there is no title line in the file.

(5) Lotstarts.csv

The file "Lotstarts.csv" describes the quantity of blank wafers to be input to the shop every day as illustrated in Table A.2. The headings are given in the Appendix 2.

Note that an additional column name "day index" is appended to the table to describe the index for days. In the original file provided by TI there was no such column. The date information is described in the column "StartDate." However, the format of the cells in "StartDate" can't be easily processed. The range of the dates in this table starts from 9/1/2007 and ends at 12/01/2007. The column "day index" treats 9/1/2007 as the 1^{st} day and 12/01/2007 as the 91^{th} day consecutively.

Table A.2 Example of Lotstarts3month.csv

Facility	Item	Next logpoint	Estimated ShipDate	Quantity	StartTime	StartDate	Hot-Flag	Hold-Flag	Logpoint	Lotnumber	day index	
DMOS6	P4HTF751992APW	110		25	12/30/1899 0:02:50	9/1/2007 0:00	N	N	110	7244220	1	end
DMOS6	P4ITW751980CWM	110		25	12/30/1899 0:03:14	9/1/2007 0:00	N	N	110	7244221	1	end
DMOS6	P4ITW751980CWM	110		25	12/30/1899 0:03:26	9/1/2007 0:00	N	N	110	7244222	1	end
DMOS6	P4JTF751543ZWM	110		25	12/30/1899 0:03:49	9/1/2007 0:00	N	N	110	7244223	1	end
DMOS6	TEXX/PR18KSER	110		24	12/30/1899 0:37:17	9/1/2007 0:00	N	N	110	7244227	1	end
DMOS6	TEXX/PR18KSER	110		24	12/30/1899 0:37:36	9/1/2007 0:00	N	N	110	7244228	1	end
DMOS6	TEXX/TELSIR	110		24	12/30/1899 0:38:11	9/1/2007 0:00	N	N	110	7244229	1	end
DMOS6	TT4B/CUSEEDA	110		24	12/30/1899 0:38:39	9/1/2007 0:00	N	N	110	7244230	1	end

Note that an addition column is appended to the end of the table. All the elements in this column are set to be string “end”. This makes it easier for the program to read and manipulate the data.

(6) Input.txt

The file “Input.txt” describes the input parameters for running the program and contains nine lines. As an example,

```
3          //number of families to run (also the number of devices to run)
C035,P4JTF751543ZWM,          //Family code and the representative device
C027,P5BJF761503BM,
C021,P6GBX2057R10,
1000       //Threshold value to reduce the number of steps
60         //time interval in minutes
25         //number of wafers per lot
4          //number of subproblems
7          //number of days in each subproblem
```

The first line of the file describes the number of families or representative devices to be scheduled. In the next three lines we have the family code and the name of the representative device. A threshold value is shown in the fifth line.

This value is used to reduce the number of steps before building the model. Its units are wafer/minute. For a value of 1000, for example, any steps with processing rates higher than 1000 wafer/minute will be eliminated from the routes and thus will not be included in the model. The next line gives the time interval, in minutes, that is being simulated by the program. The size of the model is proportional to this value; e.g., reducing it by ½ doubles the size of the model. This parameter can be set to any number of minutes that divides evenly into 60, such as 10 min, 15 min, or is a multiple of 60 such as 120 min. The next line shows the number of wafers per lot. The user can also change this value to reflect the real situation. The next line shows the number of subproblems for the run. The last line shows he number of days included in a subproblem. In the above case 4 subproblems are run with each for 7 days. Thus the planning for 28 days can be obtained by this setting. The user can also modify these two values to obtain solution for other planning horizon.

Note: The sequence of the families should be the same as the sequence in the "Family.csv" file or else there will be a mismatch of data.

(7) NonConstrainMachOpnDesc.txt

This file contains a partial list of operations that appear under the column heading *K*855 in the Stations.csv file. Machines whose *K*855 parameter is included in this file list are considered to be non-bottlenecks. In some instances, the computed effective processing rate deviates from the actual processing rate, and hence may become a bottleneck in the scheduling model. To avoid this situation, the processing rate of all machines that perform an operation listed in this file is multiplied by the constant " Rate_Increase." The default value is Rate_Increase = 20. The current list is shown in Appendix 3 and can be modified by the user to include any operation-machine combination that should not be a bottleneck. Note that there is no title line in the file.

Data Structure

The following data objects are created after reading the input files.

(1) Production

The “Production” data object contains the routing information for all of devices in the fab. The routing information is also stored in the data object named “Route,” which will be described in Appendix 3.

(2) Step

A “Step” data object is defined to hold the information for a device at the logpoint–operation level. It also holds the “*K* Parameters,” which are required to compute the processing rates. Each line in the “Routing.csv” table can be viewed as a step. The machine list information is also stored in the “Step” data objects.

(3) Route

A “Route” data object is a combination of “Step” data objects. A “Route” object holds the information of a route for some device. Each “Route” data object corresponds to a block of information in the “Routing.csv” file which belongs to the same device.

(4) Machine_Set

The “Machine_Set” data object is used to hold the information for the machines. Such information involves “Misti ID” and the “*K* Parameters.”

(5) Family

The “Family” object is used to hold the information for the families. For each family, the object contains the related devices, which are given in the “Family.csv” file.

Output Files

(1) Summary.csv

This file contains the information on finished products and the deviations (shortages) for each time period. The time period (interval) is specified by setting the appropriate parameter in the Input.tex file. An example is given in Table A.3, which is divided into three sections.

The first section shows the target daily output (demand) for the devices. The "day" column gives the day index while the 2nd, 3rd and 4th columns list the demand for the three devices. For instance, the demand for device 2 on the first day is 316 wafers. Note that at present the demand is the same each day for a device. The second section of the table shows the number of wafers completed. The columns are defined in the same way. The last section gives the deviation from the target output for each day and each device. It can be seen that the current production schedule does not lead to any shortages for the week.

Table A.3 Example of Summary.csv

	Target output				No. devices completed				Deviations		
Day	C035	C027	C021	Day	C035	C027	C021	Day	C035	C027	C021
1	311	316	24	1	311	316	24	1	0	0	0
2	311	316	24	2	311	316	24	2	0	0	0
3	311	316	24	3	311	316	24	3	0	0	0
4	311	316	24	4	311	316	24	4	0	0	0
5	311	316	24	5	311	316	24	5	0	0	0
6	311	316	24	6	311	316	24	6	0	0	0
7	311	316	24	7	311	316	24	7	0	0	0

(2) Shop_production.csv

This file indicates the number of wafers to process at each logpoint-operation during each time period. An example of this file is shown in Table A.4; the complete file is about 8 MB in size for 4–week problem.

Table A.4 Example of Shop_production.csv

Family index	Device name	Step	Logpoint num	Operation num	Production at $t = 1$	Production at $t = 2$	Production at $t = 3$	Production at $t = 4$	Production at $t = 5$
1	P4JTF751543ZWM	1	110	301	0.05	0.04	0.06	0.03	0.06
1	P4JTF751543ZWM	2	112	6930	0.00	0.00	0.00	0.00	0.00
1	P4JTF751543ZWM	3	282	1600	0.00	0.00	0.00	0.00	0.00
1	P4JTF751543ZWM	4	282	2000	2.21	2.21	2.21	2.23	2.21
1	P4JTF751543ZWM	5	282	6640	0.00	0.59	0.60	0.66	0.69
1	P4JTF751543ZWM	6	290	2500	0.00	0.00	1.19	0.00	1.29
1	P4JTF751543ZWM	7	290	6500	0.00	0.00	0.00	0.31	0.32
1	P4JTF751543ZWM	8	300	3075	2.68	2.42	1.67	1.08	1.42
1	P4JTF751543ZWM	10	300	3600	0.78	0.86	0.83	0.75	0.54

1	P4JTF751543ZWM	11	300	3740	0.00	0.00	0.00	0.00	0.00
1	P4JTF751543ZWM	12	300	3750	0.00	0.00	0.00	0.00	0.00
1	P4JTF751543ZWM	13	300	6750	2.42	3.77	13.52	4.12	26.17
1	P4JTF751543ZWM	14	300	3800	0.00	0.13	0.07	0.03	0.06

The first column identifies the index of the device. The second column gives the device name. The third column lists the step number. The fourth and fifth columns show the logpoint number and operation number of the step. Each of the remaining columns gives the number of wafers that need to be processed during the corresponding time interval -- 60 min in this case. For example, the sixth column indicates that for device P4JTF751543ZWM, 0.0532 wafers should be produced at step 1 in the first hour, 0.042824 in the second hour, and so on. All values are fractional since the problem is modeled with continuous variables. A more informative value can be obtained by rolling up production to the logpoint level. To obtain a daily plan, either those values would have to be rounded to the nearest integer, or a second model would have to be solved for each day or shift that included more detail and insisted on integer production quantities.

Remark:

Some steps are not included in the "Shop_production.csv" file because they were eliminated from the model, as mentioned in Section 4.2. Those steps, however, must still be performed.

(3) Machine_utilization.csv

This file gives the machine usage during production. An example is shown in Table A.5.

Table A.5 Example of Machine_utilization.csv

Machine index	Machine usage at $t = 1$ (%)	Machine usage at $t = 2$ (%)	Machine usage at $t = 3$ (%)	Machine usage at $t = 4$ (%)	Machine usage at $t = 5$ (%)	Machine usage at $t = 6$ (%)	Machine usage at $t = 7$ (%)
19	0.14	0.00	0.07	0.00	0.05	0.00	0.04
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
21	0.00	0.00	0.00	0.00	0.00	0.00	0.00
22	0.00	0.00	0.00	0.00	0.00	0.00	0.00

23	1.00	0.94	0.52	0.31	0.51	0.77	0.81
24	0.66	0.25	0.31	0.00	0.00	0.00	0.07
25	0.34	0.08	0.20	0.00	0.00	0.00	0.07
26	0.49	0.05	0.15	0.00	0.00	0.00	0.07
27	0.49	0.04	0.15	0.00	0.25	0.00	0.00
28	0.49	0.00	0.24	0.51	0.00	0.00	0.00

The first column identifies the machine index, which is the same index listed in the "Stations.csv" file. The remaining columns, starting with the 2nd, give the machine usage during successive time periods. All of the values in these cells are continuous and should be interpreted as a percentage of the corresponding time period such as an hour. For example, machine 23 (AM3001) is used 94% of the time in the 2nd time period. It can be seen that machine 23 is quite busy, at least during the first seven periods.

(4) WIP_history(original steps).csv

The file maintains a record of the original WIP before any steps are removed. An example is shown in Table A.6.

Table A.6 Example of WIP_history(original steps).csv

Family index	Step Index	WIP at the end of $t = 1$	WIP at the end of $t = 2$	WIP at the end of $t = 3$	WIP at the end of $t = 4$	WIP at the end of $t = 5$
$i = 1$	$j = 1$	0	0	0	0	0
$i = 1$	$j = 2$	300.00	0	0	0	0
$i = 1$	$j = 3$	92.96	384.98	353.33	330.60	302.95
$i = 1$	$j = 4$	107.04	115.02	141.92	134.45	113.51
$i = 1$	$j = 5$	0	0	4.75	7.70	3.60
$i = 1$	$j = 6$	0	0	0	27.25	79.95
$i = 1$	$j = 7$	0	0	0	0	0
$i = 1$	$j = 8$	300.00	300.00	273.70	260.24	174.95
$i = 1$	$j = 9$	0	0	26.31	0.00	85.29
$i = 1$	$j = 10$	113.91	109.60	101.15	137.25	127.24
$i = 1$	$j = 11$	11.09	5.70	0	3.65	0
$i = 1$	$j = 12$	0	9.69	14.16	0.00	13.66

The first two columns indicate the device index and processing step, respectively. Each column in the remaining part of the table stands for a WIP

profile at the end of a time period. For example the third column gives the number of wafers at each step at the end of the first time period. The 4th column gives the WIP at each step at the end of the 2nd time period, and so on.

(5) WIP_history_LogPoint.csv

The file keeps a record of the WIP history at the logpoint level. An example is shown in Table A.7. Note that in this file all of the wafers of the representative devices are aggregated.

Table A.7 Example of WIP_history_LogPoint.csv

Logpoint number	WIP at end of $t = 1$	WIP at end of $t = 2$	WIP at end of $t = 3$	WIP at end of $t = 4$	WIP at end of $t = 5$	WIP at end of $t = 6$	WIP at end of $t = 7$
110	0	0	0	0	0	0	0
112	300.00	0	0	0	0	0	0
282	717.00	949.50	948.00	918.30	858.85	818.75	807.65
290	455.00	471.93	434.11	463.82	523.26	563.36	533.55
300	1470.00	1495.57	1534.89	1534.89	1491.51	1459.55	1496.10
302	0	25.00	0	0	43.38	0	4.36
305	74.00	74.00	49.00	25.00	25.00	100.34	75.34
310	329.00	277.00	275.00	247.00	195.00	143.00	116.00
312	195.00	247.00	299.00	351.00	269.07	301.84	351.98
318	0	0	0	0	133.93	3.16	1.87
…	…	…	…	…	…	…	…

The first column gives the logpoint number at each step while the second column gives the operation number. The remaining columns indicate the WIP profile of the three representative devices at the end of each time period. For example, at logpoint 282 and operation 717, there are a total of 949.5 wafers in queue. This value was obtained by summing the WIP of the three representative devices.

(6) FinalWIP(reduced).csv

This file contains the final WIP at the end of each run. For example, if the code is run for a 1-week problem four times in a row time instead of once for a 4 week problem, the file will contain the WIP profiles at the end of each week. The WIP

profiles give the number of wafer at the reduced steps instead of the original steps. An example is shown in Table A.8.

Table A.8 Example of FinalWIP(reduced).csv

Family	Step index	WIP
C035	1	0
C035	2	0
C035	3	0
C035	4	0
C035	5	0
C035	6	0
C035	7	0
C035	8	0
C035	9	0
C035	10	23.46
C035	11	21.59
C035	12	34.62

The first column indicates the index of the representative device. The second column gives the index for the steps. The last column reports the number of wafers of the device at the corresponding step. For example, there are 23.4577 wafers of device 1 at step 10.

(7) Process_rate.csv

This file contains the effective processing rates that were computed for each device at each step. Table A.9 contains an example.

Table A.9 Example of Process_rate.csv

Device index	Step index	Machine index	Machine MistiID	Processing rate
$i = 22$	$j = 21$	$m = 29$	AP1001	$r_{ijm} = 0.45$
$i = 22$	$j = 21$	$m = 29$	AP1002	$r_{ijm} = 0.45$
$i = 22$	$j = 21$	$m = 29$	AP1003	$r_{ijm} = 0.45$
$i = 22$	$j = 21$	$m = 29$	AP1004	$r_{ijm} = 0.45$
$i = 22$	$j = 21$	$m = 29$	AP1005	$r_{ijm} = 0.45$
$i = 22$	$j = 21$	$m = 29$	AP1006	$r_{ijm} = 0.45$
$i = 22$	$j = 21$	$m = 29$	AP1007	$r_{ijm} = 0.45$
…	…	…	…	…

The first column gives the index of the original devices; that is, the index of the 76 devices in the Routing.csv file before aggregation into representative families. The 22^{th} device, for example, is the representative device for family C035. The second column in the table gives the step index, the third column indicates the machine index, while the forth column lists the machine Misti−ID. The last column gives the processing rate r_{ijm} in wafers/min.

Appendix 2: Field Definitions for Input Files

1. Fields of the “Routing.csv” file.

Facility
Device
Logpoint
Operation
OpnDesc
MachineGrp
EquivOp
OrigRecNo
Par
875
840
824
821
820
819
816
815
810
809
Date
802
949
947
874
837
844
842
814
948
All837
873
950
803
908
883
889
933
934

918
829
905
869
909
910
817
830
812
835
943
862Overflow
845
846
850
851
861
862
882
884
828
All817
All812
All835
All836
882Overflow
836
867
955
956
All814
All875
All873
All874

2. Fields of the “Stations.csv” file are shown as follows.

Facility
Misti-id
Machine Groups
Date
858
811
833
834
838
852
853
854
960
856
939
859
860
861
865
866
868
906
941
942
932
937
855

3. One example of “Family.csv” file is shown as follows.

C035	C027	C021	,
B4JRD15060BPP	B5BJF761924B	EZ/E771676D17	,
B4JRD15060B4P	EZ/E761536Z00	EZ/F771657A02	,
B4JRD15075BPP	EZ/E761536Z02	EZ/F771657A07	,
B4JRD751686J4P	EZ/E761541A00	EZ/F771657A08	,
B4JRF751613D4P	EZ/E761560Z00	EZ/F771657A09	,
B4JRF751625PF	EZ/E761909A16	EZ/F771657A10	,
B4JRF751625PP	EZ/F761503B23	EZ/F771657A11	,
B4JRF7516254F	EZ/F761504A02	EZ/F771657A11L	,
B4JRF7516254P	EZ/F761504B08	EZ/F771657A12	,
B4JRF751672D4P	EZ/F761504B10	EZ/F771657A13	,
B4JRF751989B4P	EZ/F761522A49	EZ/F771657A14	,
B4JSF751613HHT	EZ/F761522A50	EZ/F771657Z11	,
B4JTD15060C	EZ/F761522A51	EZ/F771657Z12	,
…	…	…	,

4. The fields of “wip_data.txt” file are shown as follows.

Date
Logpoint
Operation
Device
Device_type
Prime_Inv
Rewrok_Inv
Hold_Inv
Moves_qty
Num_lots
plan_ct

5. The fields of “Lotstarts.csv” are shown as follows.

Facility
Item
NextLogpoint
EstimatedShipDate
Quantity
StartTime
StartDate
Hot-Flag
Hold-Flag
Logpoint
Lotnumber
day index

6. NonConstrainMachOpnDesc.txt

(List of nonbottleneck operations)

ALIGNMENT
AUTO_VISUAL_INSP
CDSEM_AMATVERITY
CDSEM_HITACHI
CDSEM_KLA8300
CD_SEM_MEASURE
INLINE_PARAMETRIC
KLA_F5_CU
KLA_F5_SCD
KLA_F5_SPECTRA100
LASER_ANNEAL
LASER_MARK
LOT_INSPECT
METAPULSE_MEAS
OPTI_PH_QUAL
OPTIPROBE
PC_CU
PC_NON_MTL
POST_CU_CMP_INSPECT
POST_LOT_MEASURE_CU
RS_MEASURE
SRT_LOT_FORM_M
SRT_LOT_FORM_NM
SRT_METAL_CU
SRT_NON_MTL
STI_CD_MEASURE
WLR
YE_AIT_ADDER
YE_COMPASS_ADDER
YE_DEFECT_REVIEW
YE_ES20_INSPECTION
YE_SEMVISION_INSPECT
YE_SEMVISION_REVIEW
YE_STEALTH_ADDER
YE_EBEAM_INSPECTION
VIPER_INSPECT

Appendix 3: Data Structure

1. Production

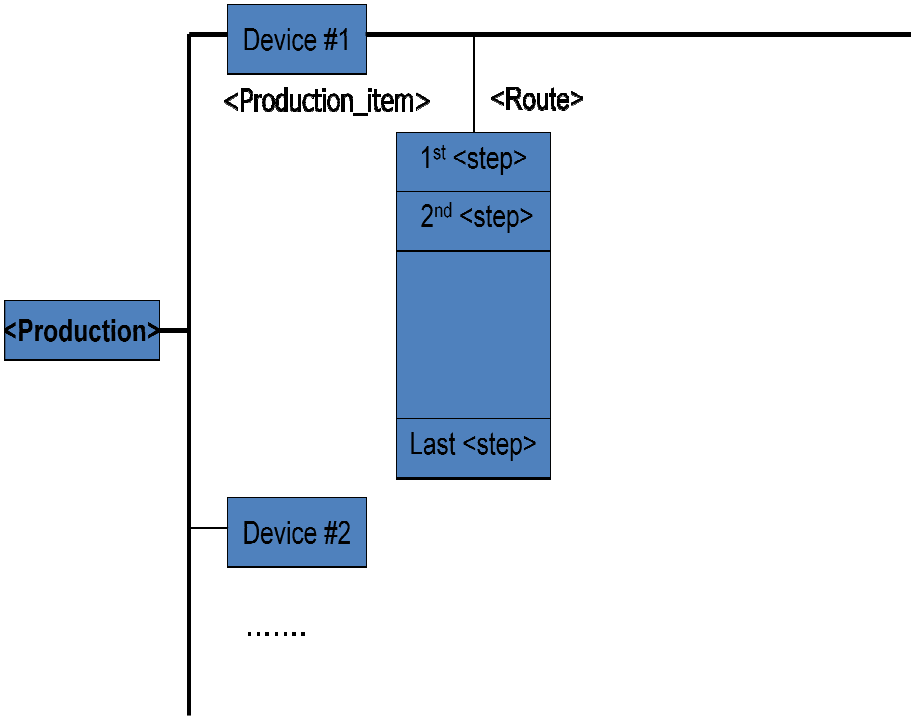

Figure A.1 Data structure of <Production> object

As it can be seen in the figure, the <Production> data object is an array of the <Production_item> data object. Each <Production_item> data object stands for one device. For each device, a <Route> object is included in the <Production_item> object describing the route of the device. A <Route> object is an array of <Step> objects. A <Step> object describes the process within the route, which is a combination of log point number of operation number. More detail information is shown in the <Step> object description.

2. Step

A step corresponds to a line in the "Routing.csv" table which can be indexed by the combination of logpoint number and operation number. The following figure shows the data structure of one <Step> object.

A <Step> object includes the necessary information for one step. First it stores all the *K* parameters which are used to compute the processing rates. The

log point number and the operation number are included as a handle of a step. The "Process Model" describes which mode the step is. The "step_index" is the order number of a step. The "MC_vector" includes the list of machines which can be used to process this step.

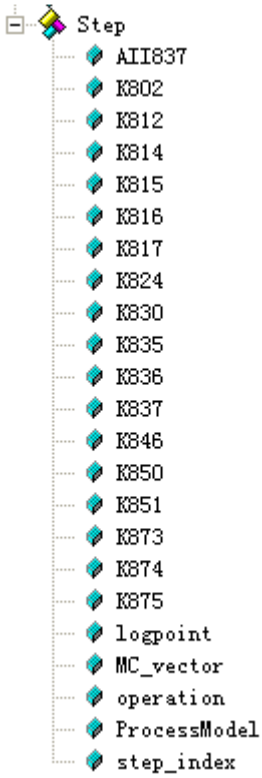

Figure A.2 Data structure of <Step> object

3. Route

   A route is a combination of steps. This can be seen in the following figure.

Figure A.3 Data structure of <Route> object

4. Machine_Set

   The <Machine_Set> object stores the information of the stations. Such information comes from the "Stations.csv" table. The following figure shows the data structure of the <Machine_Set> object.

   The "Misti ID" is the ID associated with the machine. The parameter *K*852 is the capacity multiplier/loading factor which is included in the processing

rate computation. The parameters *K*833 and *K*834 are related to the mean uptime percentage. However, currently these two parameters are not used to compute the processing rate. Instead the uptime percentage is taken into account by using parameter *K*866.

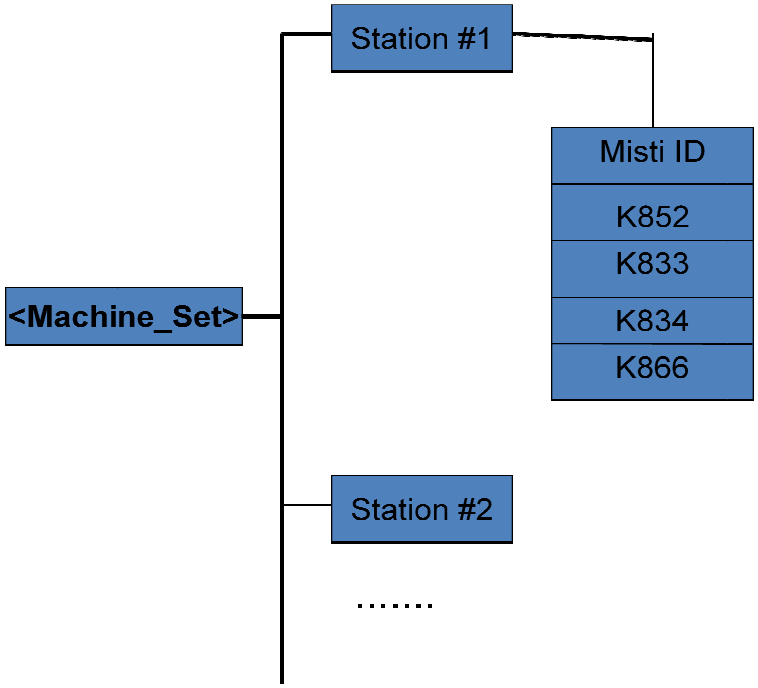

Figure A.4 Data structure of <Machine_Set> object

5. Family

   The <Family> data object stores the family information of the devices. It is an array of family structure. Each family contains the information of family code and the names of the devices belong to this family.

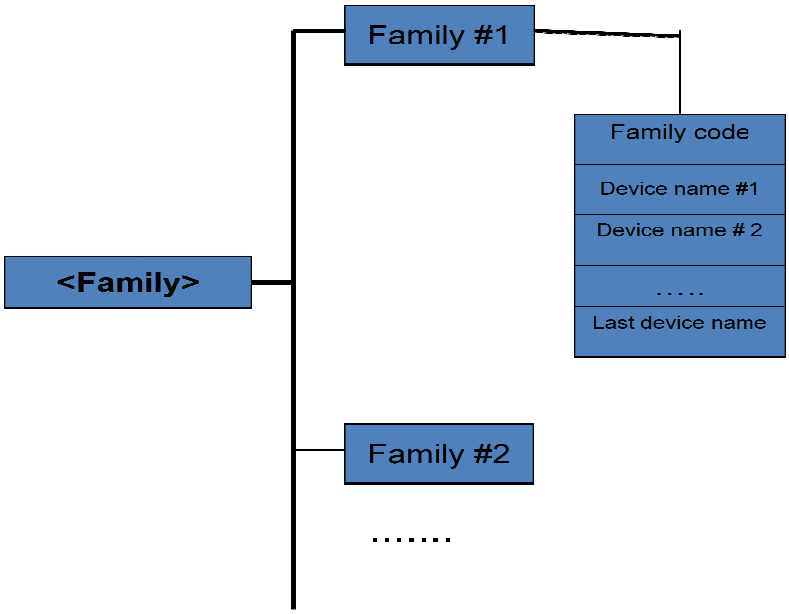

Figure A.5 Data structure of <Family> object

Appendix 4: Solution to the 4-week Problem with Basic Decomposition Scheme

Table A.10 Summary of solution to the 4-week problem by basic decomposition

	T_OUT_{id}			Num. completed			Δ_{id}^{+} or Δ_{id}^{-}		
d	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3
1	328	315	26	328	315	26	0	0	0
2	328	315	26	328	315	26	0	0	0
3	328	315	26	328	315	26	0	0	0
4	328	315	26	328	315	26	0	0	0
5	328	315	26	328	315	26	0	0	0
6	328	315	26	328	315	26	0	0	0
7	328	315	26	328	315	26	0	0	0
8	328	315	26	328	296.85	26	0	18.15	0
9	328	315	26	322.50	267.90	26	5.50	47.10	0
10	328	315	26	330.26	260.14	26	–2.26	54.86	0
11	328	315	26	291.30	299.10	26	36.70	15.90	0
12	328	315	26	210.50	379.90	26	117.50	–64.90	0
13	328	315	26	318.57	271.83	26	9.43	43.17	0
14	328	315	26	328	262.40	26	0.00	52.60	0
15	328	315	26	275.40	315.00	26	52.60	0.00	0
16	328	315	26	290.05	300.35	26	37.95	14.65	0
17	328	315	26	260.75	329.65	26	67.25	–14.65	0
18	328	315	26	307.50	282.90	26	20.50	32.10	0
19	328	315	26	322.20	268.20	26	5.80	46.80	0
20	328	315	26	330.93	259.48	26	–2.92	55.52	0
21	328	315	26	325.08	265.33	26	2.92	49.68	0
22	328	315	26	275.91	314.49	26	52.09	0.51	0
23	328	315	26	281.60	308.80	26	46.40	6.20	0
24	328	315	26	324.76	265.64	26	3.24	49.36	0
25	328	315	26	279.64	310.76	26	48.36	4.24	0
26	328	315	26	293.99	296.42	26	34.02	18.58	0
27	328	315	26	328.53	261.88	26	–0.53	53.13	0
28	328	315	26	327.48	262.93	26	0.53	52.07	0
Total	9184	8820	728	8648.95	8284.95	728	535.08	535.07	0

WIP profiles of the three devices at the end of each week

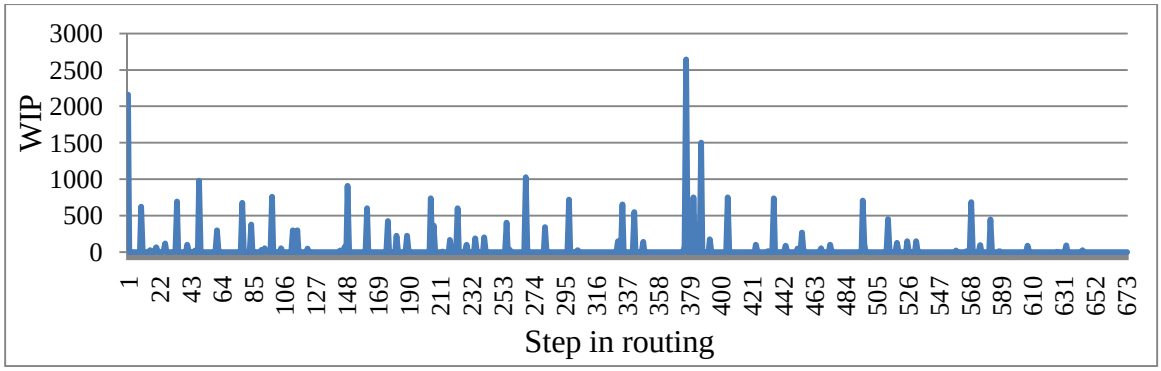

Figure A.6 WIP profile of C_1 at the end of the 1st week

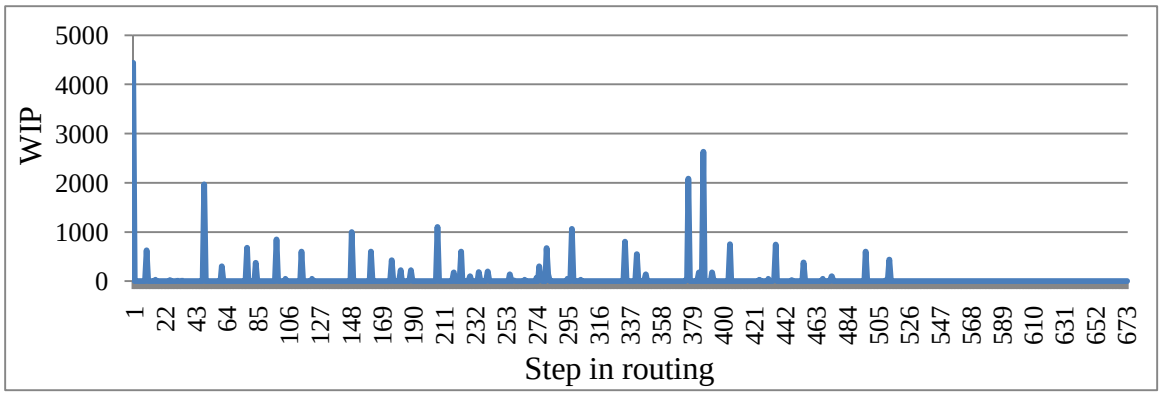

Figure A.7 WIP profile of C_1 at the end of the 2nd week

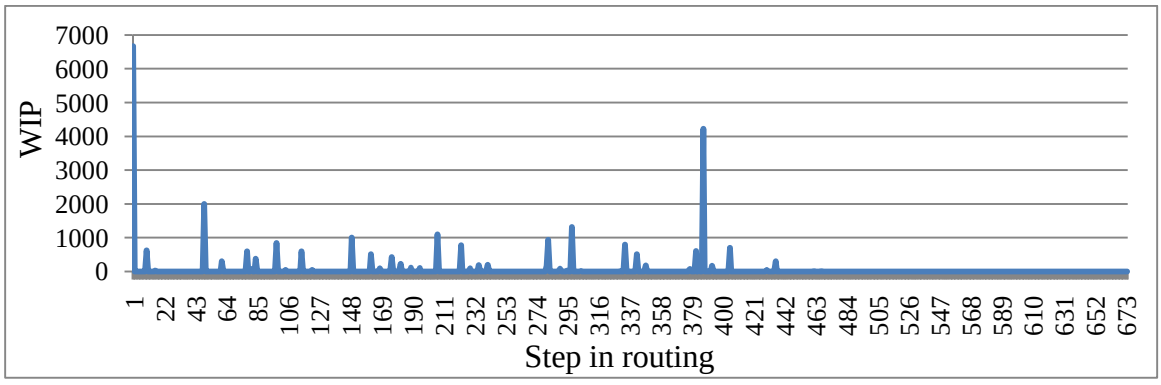

Figure A 8 WIP profile of C_1 at the end of the 3rd week

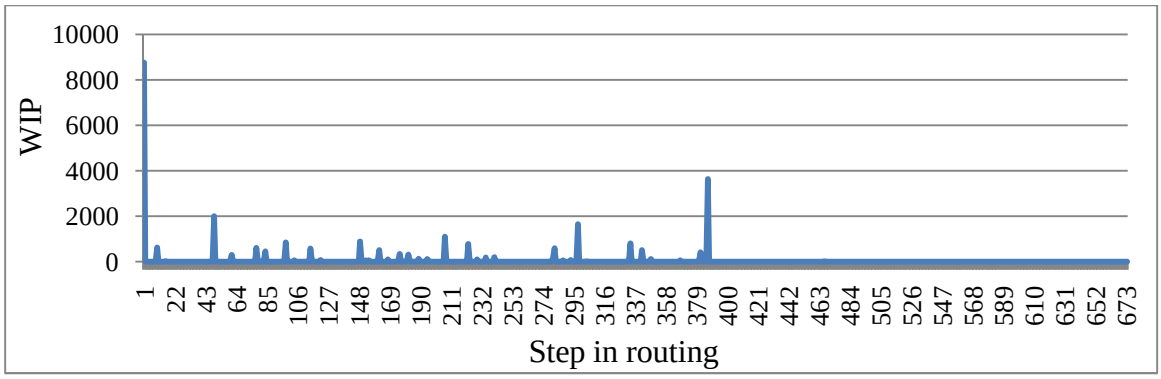

Figure A.9 WIP profile of C_1 at the end of the 4th week

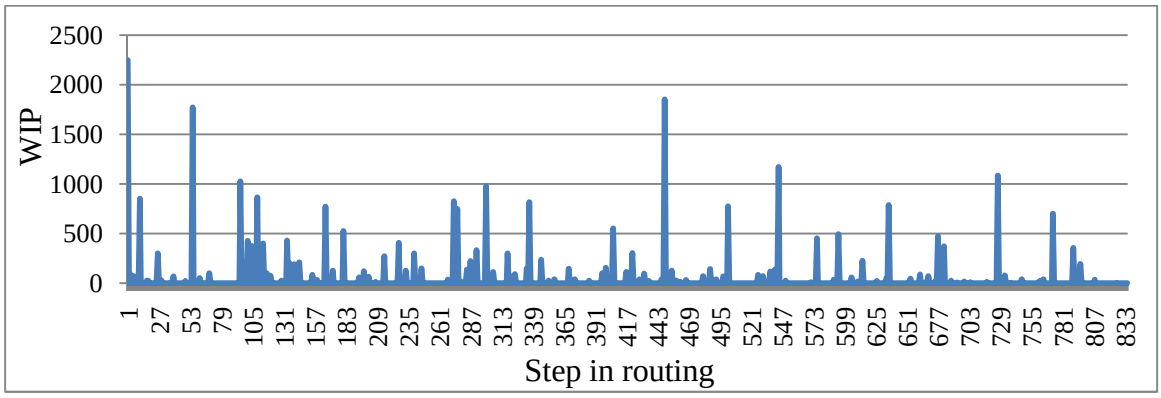

Figure A.10 WIP profile of C_2 at the end of the 1st week

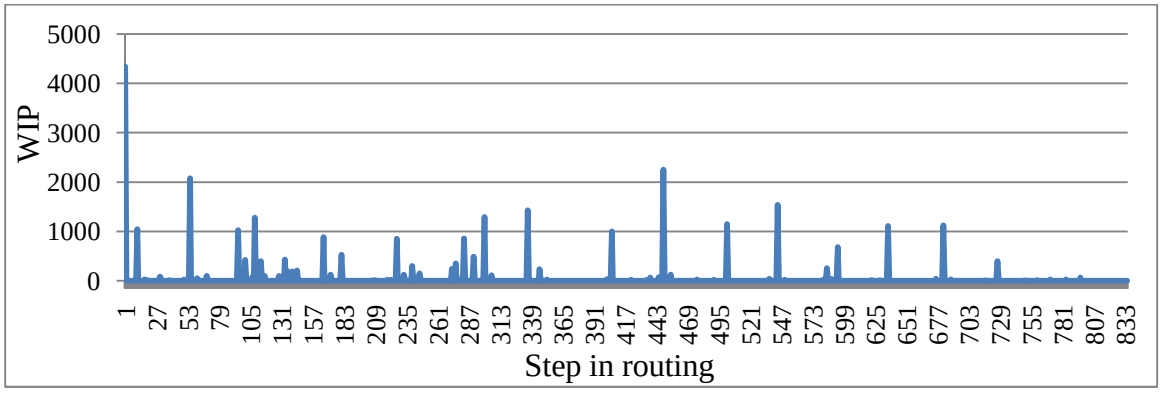

Figure A.11 WIP profile of C_2 at the end of the 2nd week

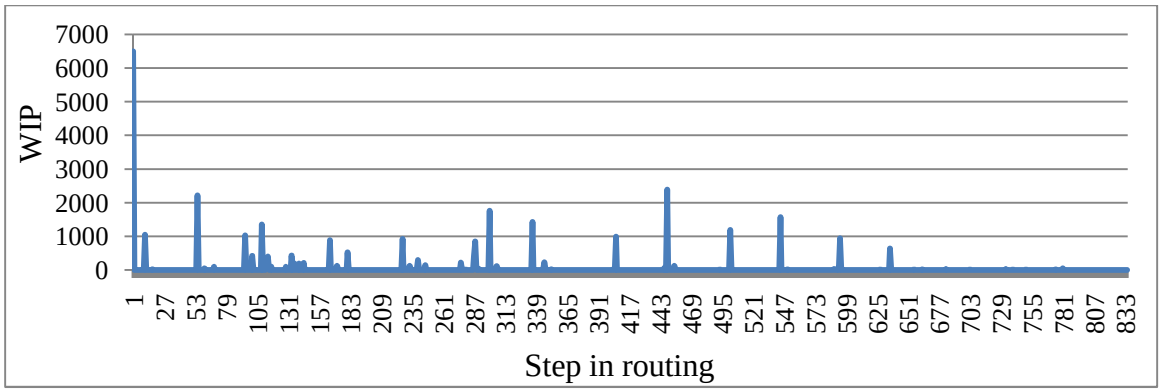

Figure A.12 WIP profile of C_2 at the end of the 3rd week

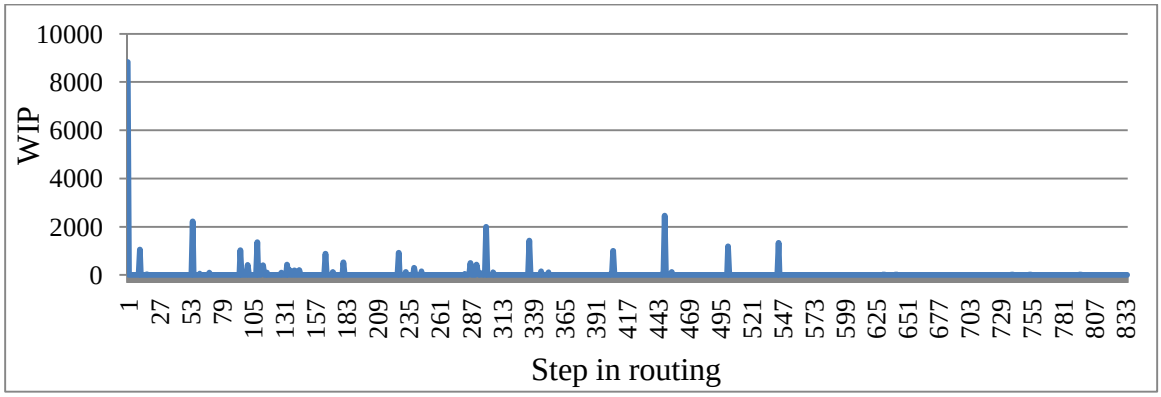

Figure A.13 WIP profile of C_2 at the end of the 4th week

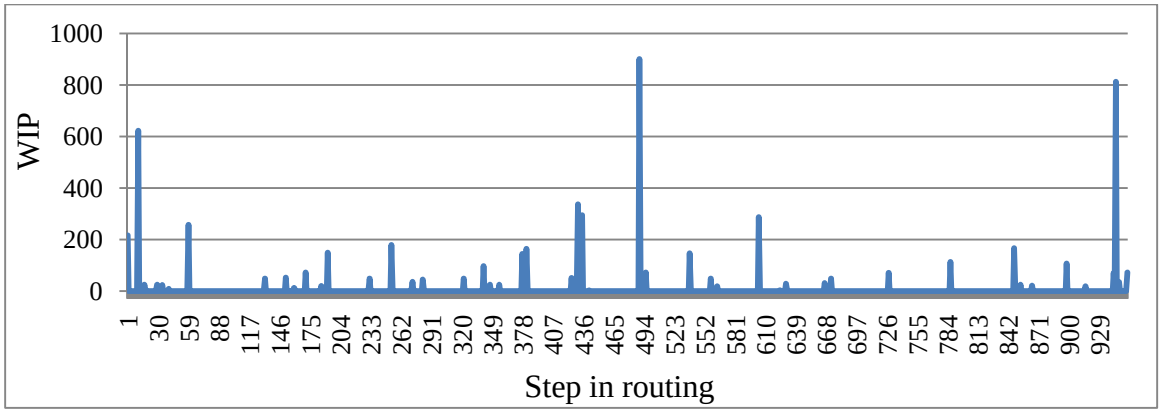

Figure A.14 WIP profile of C_3 at the end of the 1st week

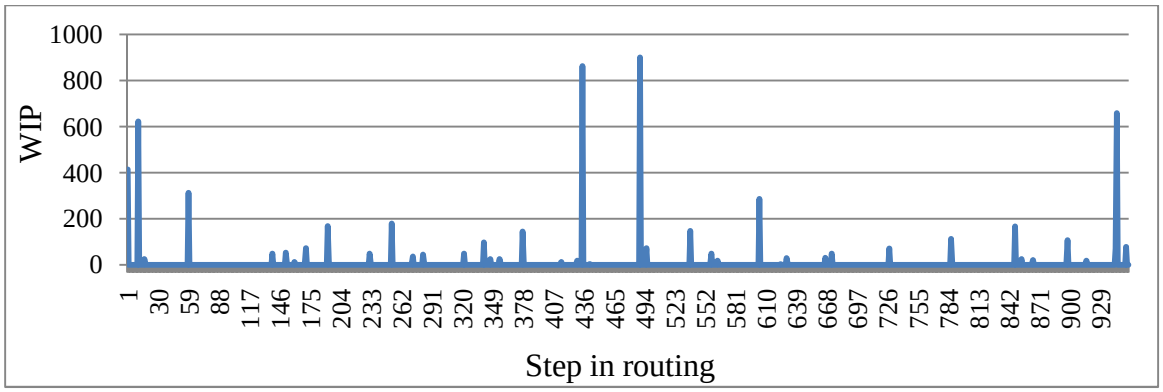

Figure A.15 WIP profile of C_3 at the end of the 2nd week

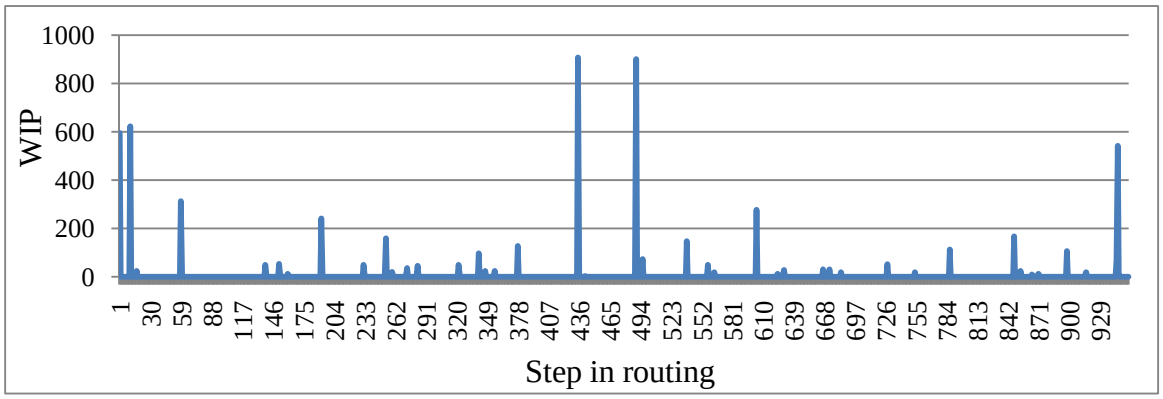

Figure A.16 WIP profile of C_3 at the end of the 3rd week

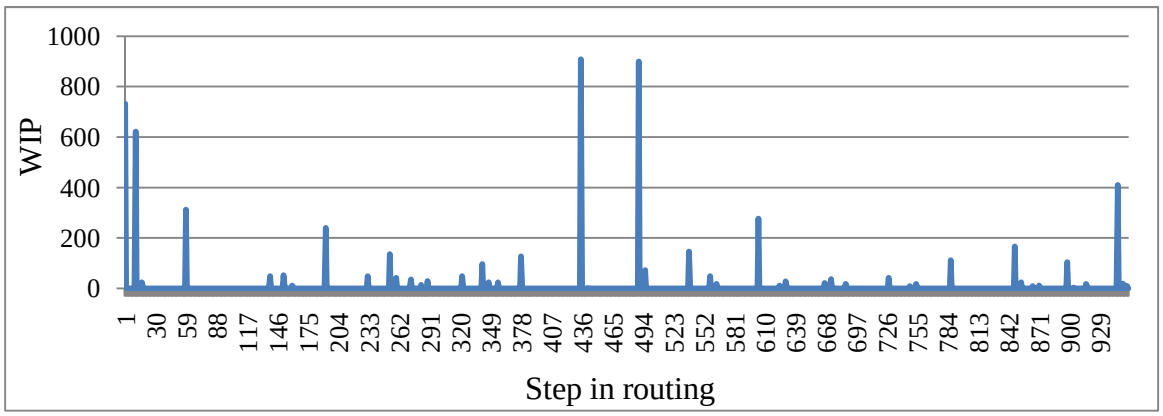

Figure A.17 WIP profile of C_3 at the end of the 4th week

Appendix 5: Solution to the 4-week Problem with Decomposition Scheme

Table A.11 Summary of solution to the 4-week problem with rescheduling and heuristic scheme

	T_OUT_{id}			Num. completed			Δ_{id}^{+} or Δ_{id}^{-}		
d	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3
1	328	315	26	328	315	26	0	0	0
2	328	315	26	328	315	26	0	0	0
3	328	315	26	328	315	26	0	0	0
4	328	315	26	328	315	26	0	0	0
5	328	315	26	328	315	26	0	0	0
6	328	315	26	328	315	26	0	0	0
7	328	315	26	328	315	26	0	0	0
8	328	315	26	328	315	26	0	0	0
9	328	315	26	328	315	26	0	0	0
10	328	315	26	328	315	26	0	0	0
11	328	315	26	328	315	26	0	0	0
12	328	315	26	328	315	26	0	0	0
13	328	315	26	328	315	26	0	0	0
14	328	315	26	328	315	26	0	0	0
15	328	315	26	328	315	26	0	0	0
16	328	315	26	328	315	26	0	0	0
17	328	315	26	328	315	26	0	0	0
18	328	315	26	328	315	26	0	0	0
19	328	315	26	328	315	26	0	0	0
20	328	315	26	328	315	26	0	0	0
21	328	315	26	328	315	26	0	0	0
22	328	315	26	328	315	26	0	0	0
23	328	315	26	328	315	26	0	0	0
24	328	315	26	328	315	26	0	0	0
25	328	315	26	328	315	26	0	0	0
26	328	315	26	328	315	26	0	0	0
27	328	315	26	328	315	26	0	0	0
28	328	315	26	328	315	26	0	0	0
Total	9184	8820	728	9184	8820	728	0	0	0

WIP profiles of the three devices at the end of each week

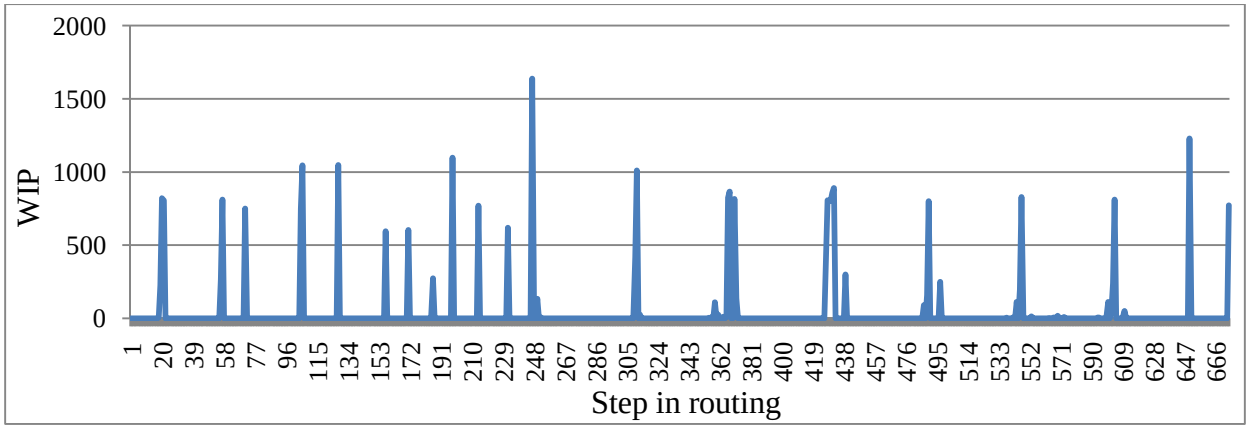

Figure A.18 WIP profile of C_1 at the end of the 1st week

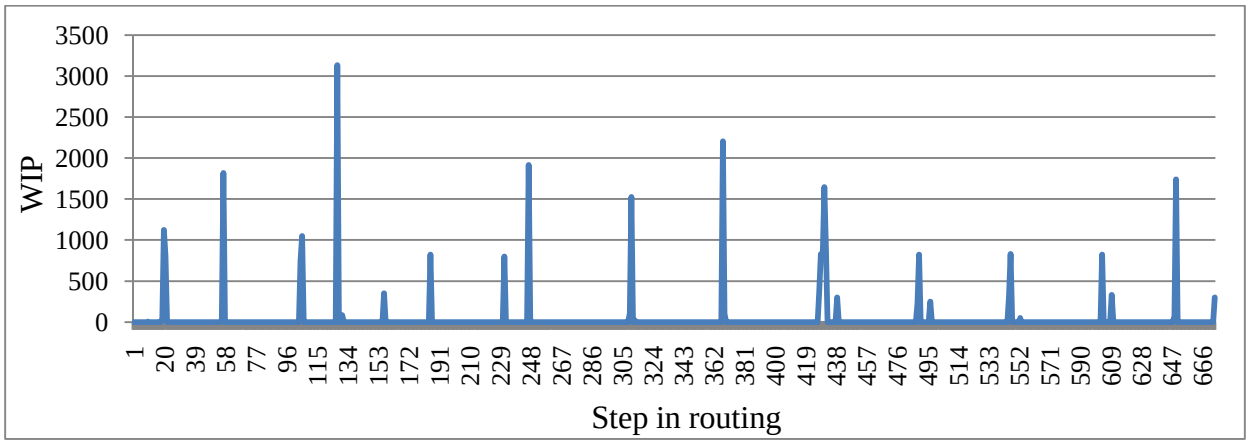

Figure A.19 WIP profile of C_1 at the end of the 2nd week

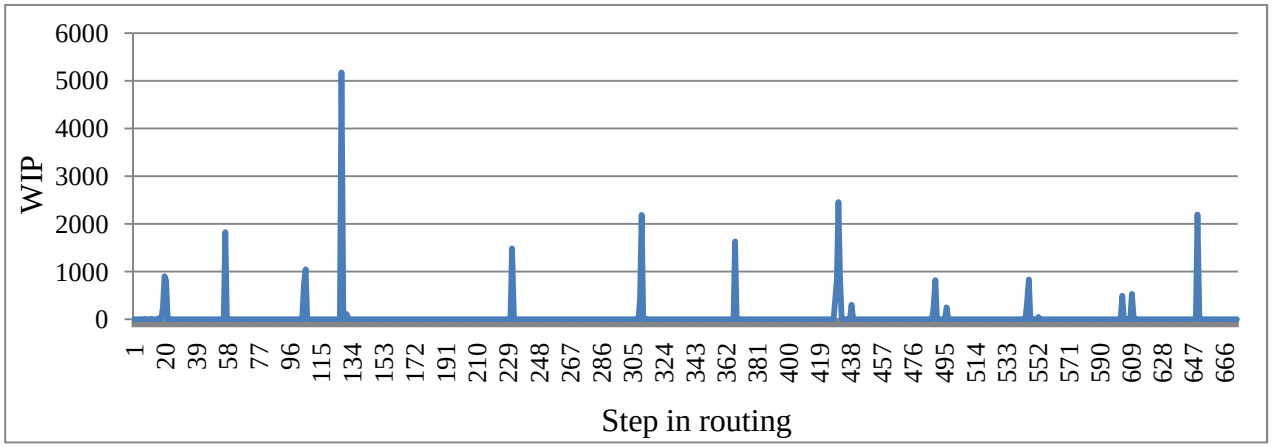

Figure A.20 WIP profile of C_1 at the end of the 3rd week

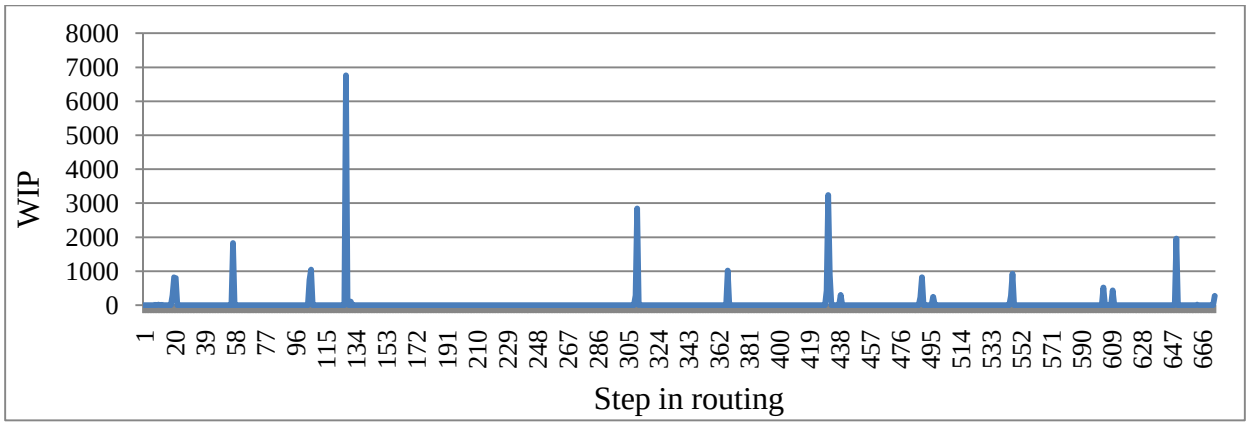

Figure A.21 WIP profile of C_1 at the end of the 4th week

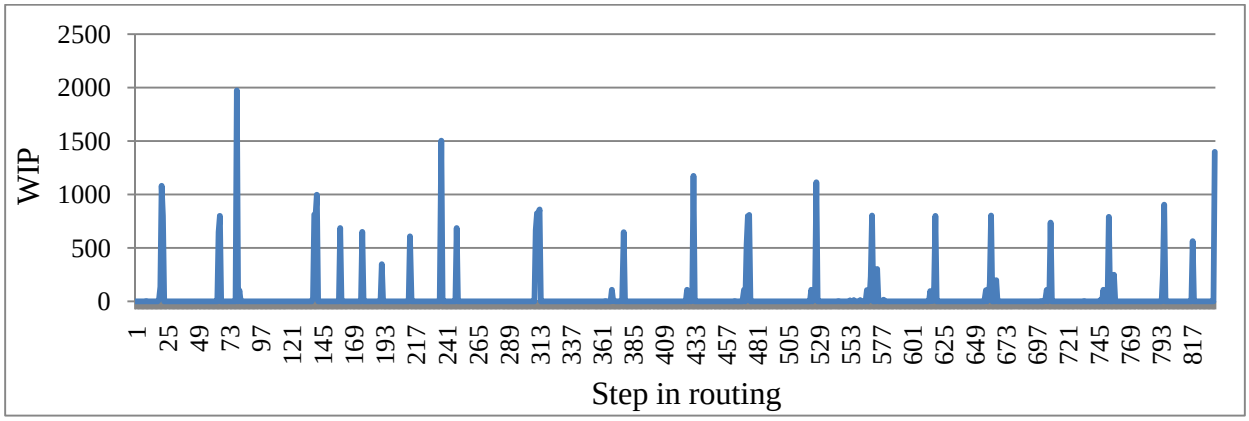

Figure A.22 WIP profile of C_2 at the end of the 1st week

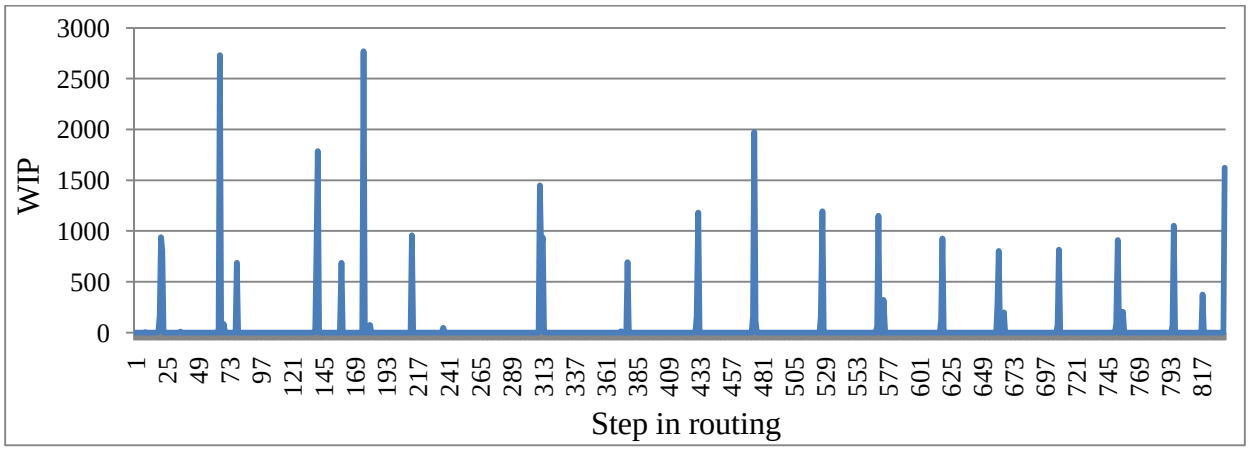

Figure A.23 WIP profile of C_2 at the end of the 2nd week

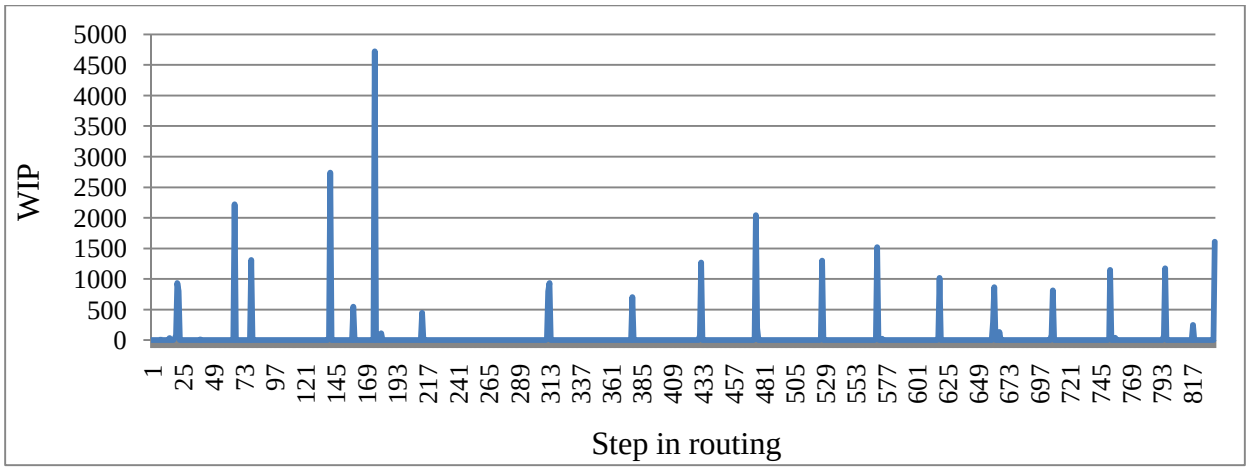

Figure A.24 WIP profile of C_2 at the end of the 3rd week

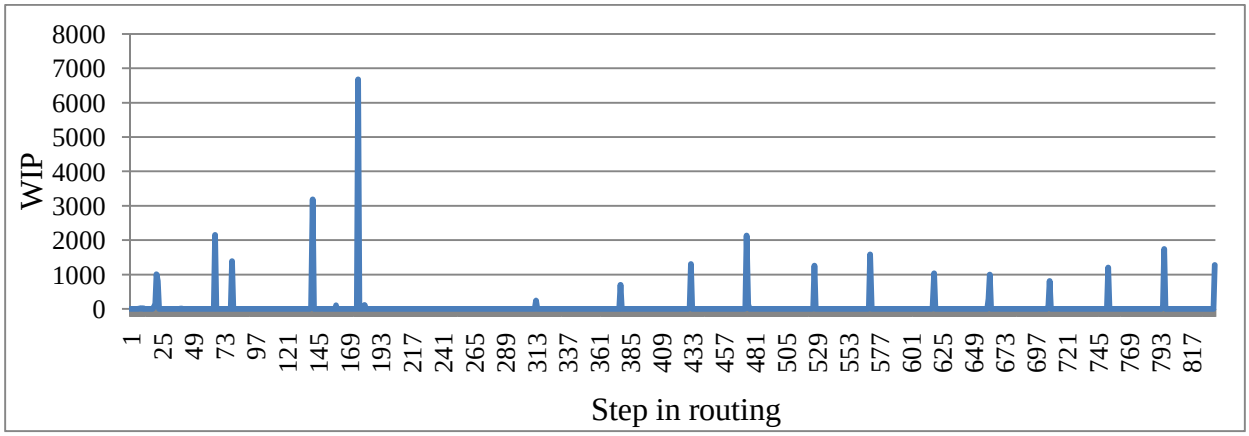

Figure A.25 WIP profile of C_2 at the end of the 4th week

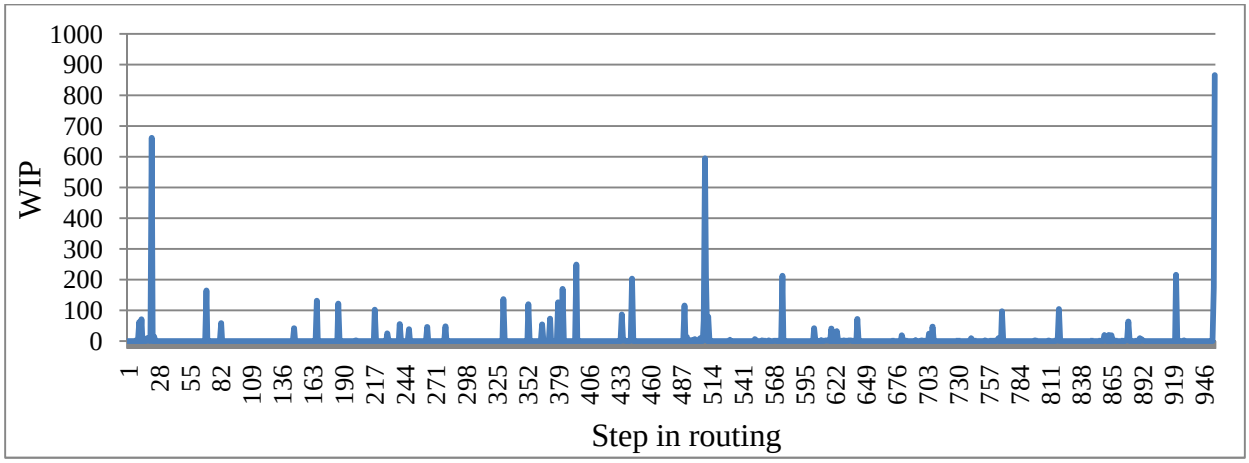

Figure A.26 WIP profile of C_3 at the end of the 1st week

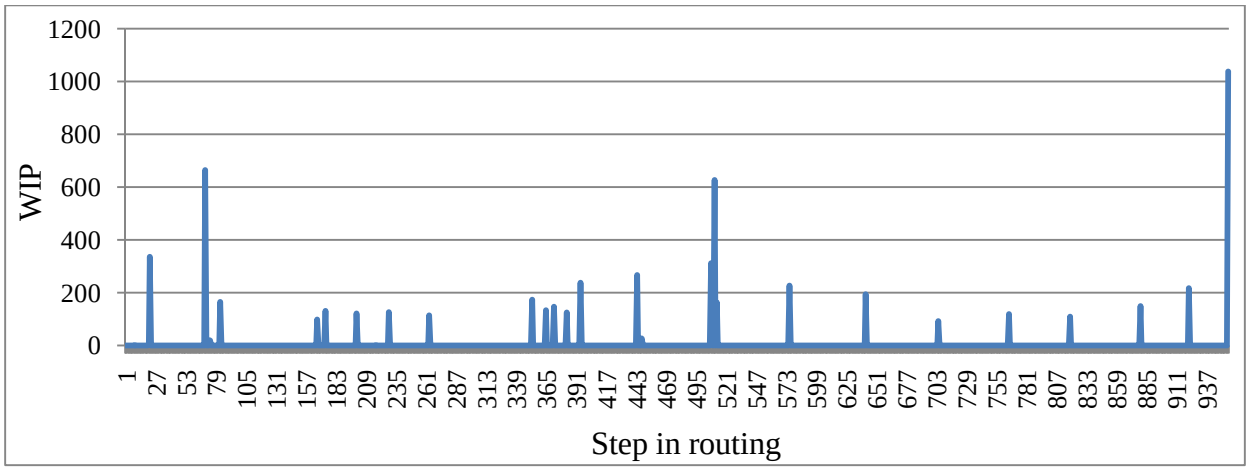

Figure A.27 WIP profile of C_3 at the end of the 2nd week

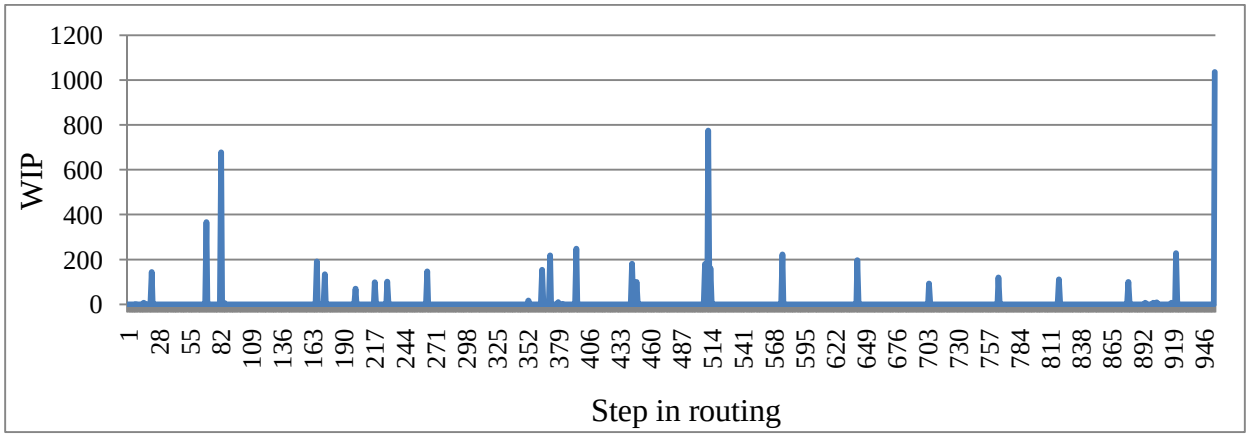

Figure A.28 WIP profile of C_3 at the end of the 3rd week

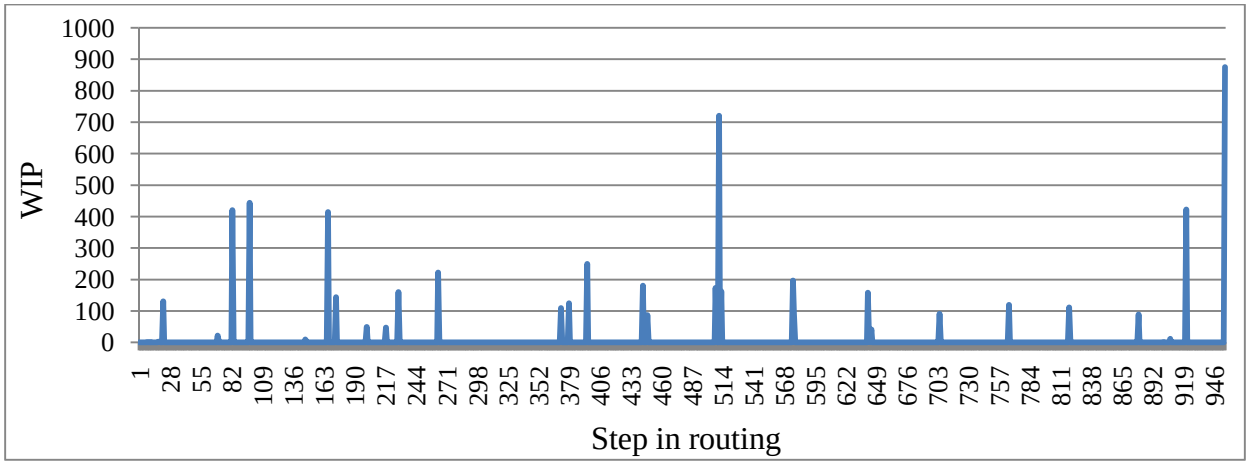

Figure A.29 WIP profile of C_3 at the end of the 4th week

Appendix 6: Daily Input for the 13-week Problem

d	$R_{1d}(1)$	$R_{2d}(1)$	$R_{3d}(1)$	d	$R_{1d}(1)$	$R_{2d}(1)$	$R_{3d}(1)$	d	$R_{1d}(1)$	$R_{2d}(1)$	$R_{3d}(1)$
1	300	300	24	32	300	325	120	63	375	250	72
2	300	300	24	33	250	325	0	64	300	275	144
3	312	300	24	34	250	300	24	65	400	200	72
4	300	299	24	35	250	348	24	66	400	448	0
5	275	450	0	36	275	325	0	67	399	300	0
6	299	349	72	37	275	312	12	68	400	300	0
7	375	249	48	38	275	324	48	69	400	250	0
8	325	287	48	39	300	300	96	70	312	246	24
9	325	300	6	40	300	300	48	71	375	300	0
10	326	300	24	41	325	300	0	72	400	300	0
11	325	425	24	42	300	311	72	73	400	250	48
12	325	300	0	43	300	300	6	74	424	325	24
13	325	300	24	44	300	300	12	75	400	362	0
14	325	175	72	45	300	300	0	76	400	375	24
15	325	300	24	46	350	300	36	77	374	325	48
16	325	275	6	47	325	312	96	78	308	350	51
17	325	300	0	48	350	300	60	79	300	412	0
18	325	350	24	49	337	324	24	80	300	300	0
19	325	300	24	50	300	300	0	81	300	362	24
20	300	312	54	51	300	300	0	82	400	300	24
21	300	324	48	52	325	312	0	83	400	300	0
22	325	325	0	53	325	300	24	84	400	350	48
23	325	325	0	54	337	262	24	85	300	348	24
24	300	325	0	55	300	356	24	86	400	300	0
25	300	300	24	56	300	349	54	87	400	300	24
26	300	300	72	57	300	300	0	88	300	400	12
27	275	387	0	58	400	300	0	89	300	425	0
28	275	375	42	59	300	348	0	90	400	274	24
29	275	325	0	60	300	300	0	91	429	350	48
30	300	325	0	61	400	275	24				
31	300	312	24	62	350	312	72				

Appendix 7: Solution to the 13-week Problem with Decomposition Scheme

Table A.12 Summary of the solution to the 13-week problem with rescheduling and dispatching heuristic

	T_OUT_{id}			Num. completed			Δ_{id}^{+} or Δ_{id}^{-}		
d	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3
1	328	315	26	328	315	26	0	0	0
2	328	315	26	328	315	26	0	0	0
3	328	315	26	328	315	26	0	0	0
4	328	315	26	328	315	26	0	0	0
5	328	315	26	328	315	26	0	0	0
6	328	315	26	328	315	26	0	0	0
7	328	315	26	328	315	26	0	0	0
8	328	315	26	328	315	26	0	0	0
9	328	315	26	328	315	26	0	0	0
10	328	315	26	328	315	26	0	0	0
11	328	315	26	328	315	26	0	0	0
12	328	315	26	328	315	26	0	0	0
13	328	315	26	328	315	26	0	0	0
14	328	315	26	328	315	26	0	0	0
15	328	315	26	328	315	26	0	0	0
16	328	315	26	328	315	26	0	0	0
17	328	315	26	328	315	26	0	0	0
18	328	315	26	328	315	26	0	0	0
19	328	315	26	328	315	26	0	0	0
20	328	315	26	328	315	26	0	0	0
21	328	315	26	328	315	26	0	0	0
22	328	315	26	328	315	26	0	0	0
23	328	315	26	328	315	26	0	0	0
24	328	315	26	328	315	26	0	0	0
25	328	315	26	328	315	26	0	0	0
26	328	315	26	328	315	26	0	0	0
27	328	315	26	328	315	26	0	0	0
28	328	315	26	328	315	26	0	0	0
29	328	315	26	328	315	26	0	0	0
30	328	315	26	328	315	26	0	0	0
31	328	315	26	328	315	26	0	0	0
32	328	315	26	328	315	26	0	0	0
33	328	315	26	328	315	26	0	0	0
34	328	315	26	328	315	26	0	0	0
35	328	315	26	328	315	26	0	0	0
36	328	315	26	328	315	26	0	0	0

37	328	315	26	328	315	26	0	0	0
38	328	315	26	328	315	26	0	0	0
39	328	315	26	328	315	26	0	0	0
40	328	315	26	328	315	26	0	0	0
41	328	315	26	328	315	26	0	0	0
42	328	315	26	328	315	26	0	0	0
43	328	315	26	328	315	26	0	0	0
44	328	315	26	328	315	26	0	0	0
45	328	315	26	328	315	26	0	0	0
46	328	315	26	328	315	26	0	0	0
47	328	315	26	328	315	26	0	0	0
48	328	315	26	328	315	26	0	0	0
49	328	315	26	328	315	26	0	0	0
50	328	315	26	328	315	26	0	0	0
51	328	315	26	328	315	26	0	0	0
52	328	315	26	328	264.81	26	0	50.20	0
53	328	315	26	310.4	280	26	17.6	35	0
54	328	315	26	295.2	295.20	26	32.8	19.80	0
55	328	315	26	300.70	289.70	26	27.30	25.30	0
56	328	315	26	275.4	315	26	52.6	0	0
57	328	315	26	328	262.4	26	0	52.6	0
58	328	315	26	304.48	285.92	26	23.52	29.08	0
59	328	315	26	323.02	267.38	15.18	4.98	47.62	10.82
60	328	315	26	295.2	295.2	0	32.8	19.8	26
61	328	315	26	310.4	280	0	17.6	35	26
62	328	315	26	267.9	322.5	0	60.1	−7.5	26
63	328	315	26	282.9	307.5	0	45.1	7.5	26
64	328	315	26	328	262.4	0	0	52.6	26
65	328	315	26	301.5	288.9	0	26.5	26.1	26
66	328	315	26	315.35	275.06	0	12.65	39.95	26
67	328	315	26	295.82	294.58	0	32.18	20.42	26
68	328	315	26	308.86	281.54	0	19.14	33.46	26
69	328	315	26	286.98	303.42	0	41.02	11.58	26
70	328	315	26	275.4	315	0	52.6	0	26
71	328	315	26	328	262.4	0	0	52.6	26
72	328	315	26	302.8	287.6	0	25.2	27.4	26
73	328	315	26	322.95	267.45	0	5.05	47.55	26
74	328	315	26	291.97	298.43	0	36.03	16.57	26
75	328	315	26	308.7	281.7	0	19.3	33.3	26
76	328	315	26	282.08	308.32	0	45.92	6.68	26
77	328	315	26	275.4	315	0	52.6	0	26
78	328	315	26	328	262.4	0	0	52.6	26
79	328	315	26	300.62	289.78	0	27.38	25.22	26

80	328	315	26	316.8	273.6	0	11.2	41.4	26
81	328	315	26	291.97	298.43	0	36.03	16.57	26
82	328	315	26	308.7	281.7	0	19.3	33.3	26
83	328	315	26	284.21	306.19	0	43.79	8.81	26
84	328	315	26	281.6	308.8	0	46.4	6.2	26
85	328	315	26	328	262.4	0	0	52.6	26
86	328	315	26	300.7	289.7	0	27.3	25.3	26
87	328	315	26	319.43	270.97	0	8.57	44.03	26
88	328	315	26	285.38	305.02	0	42.62	9.98	26
89	328	315	26	307.29	283.11	0	20.71	31.89	26
90	328	315	26	288.2	302.2	0	39.8	12.8	26
91	328	315	26	282.9	307.5	0	45.1	7.5	26
Total	29848	28665	2366	28797.2	27614.2	1523.18	1050.8	1050.8	843

WIP profiles of the three devices at the end of each week

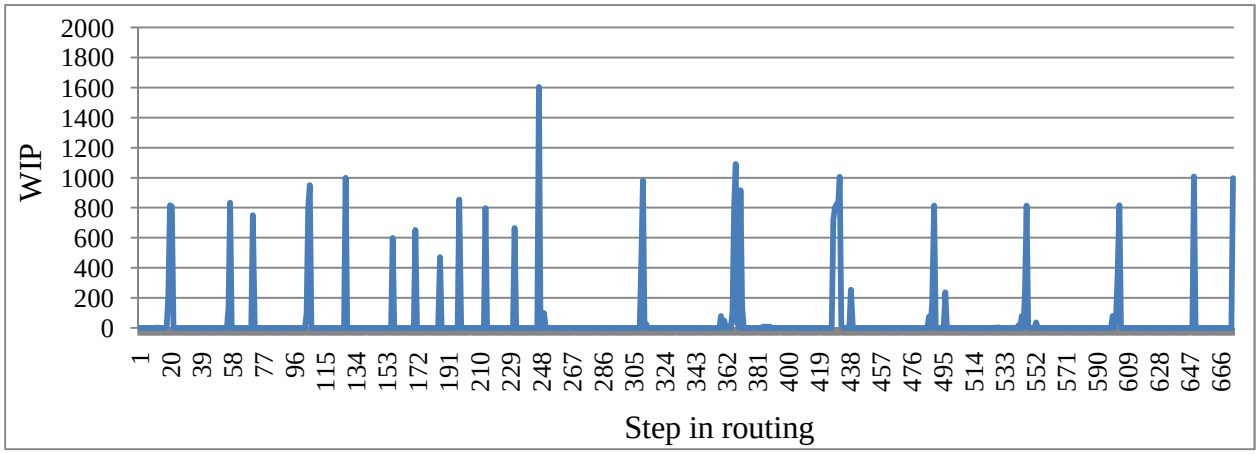

Figure A.30 WIP profile of C_1 at the end of the 1st week

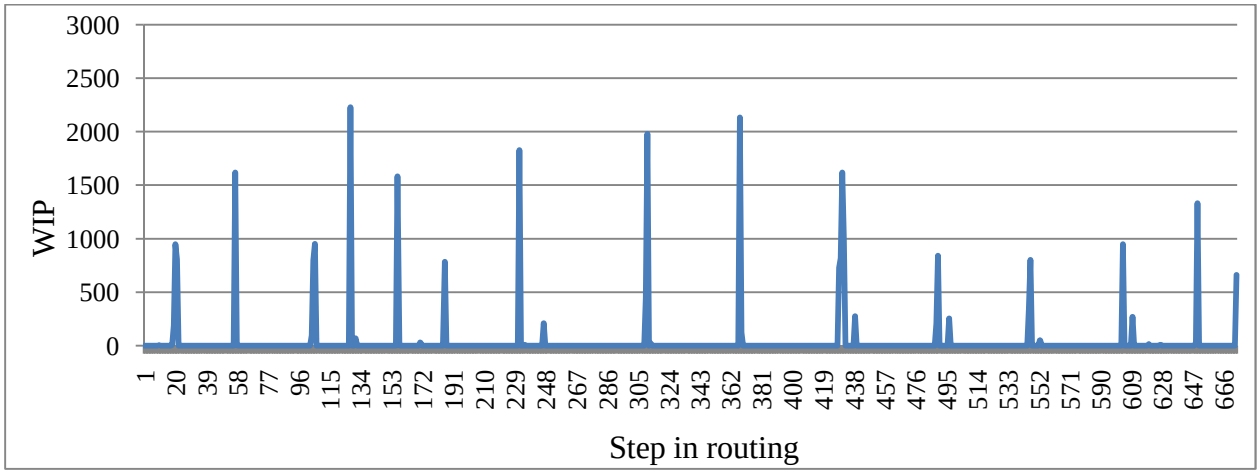

Figure A.31 WIP profile of C_1 at the end of the 2nd week

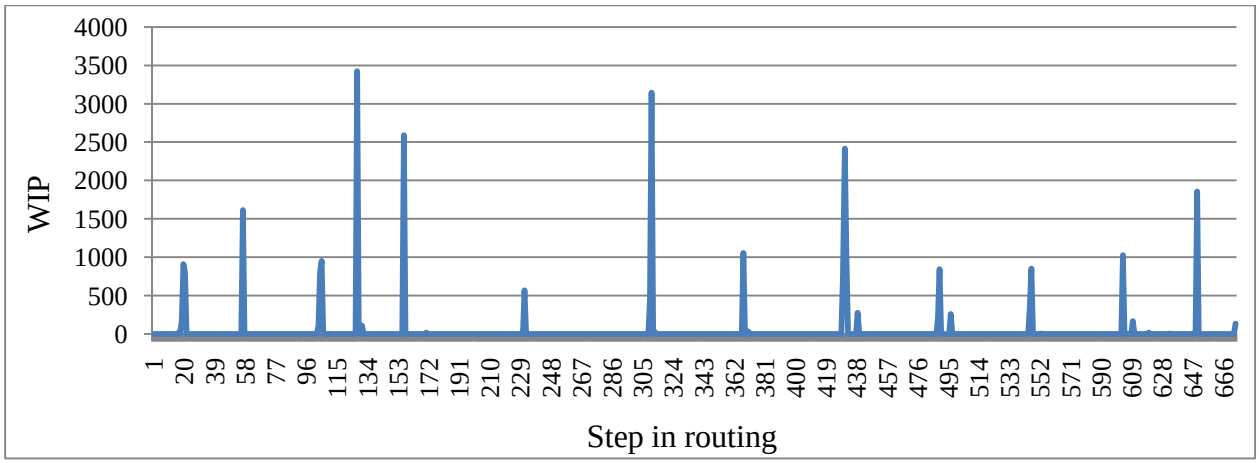

Figure A.32 WIP profile of C_1 at the end of the 3rd week

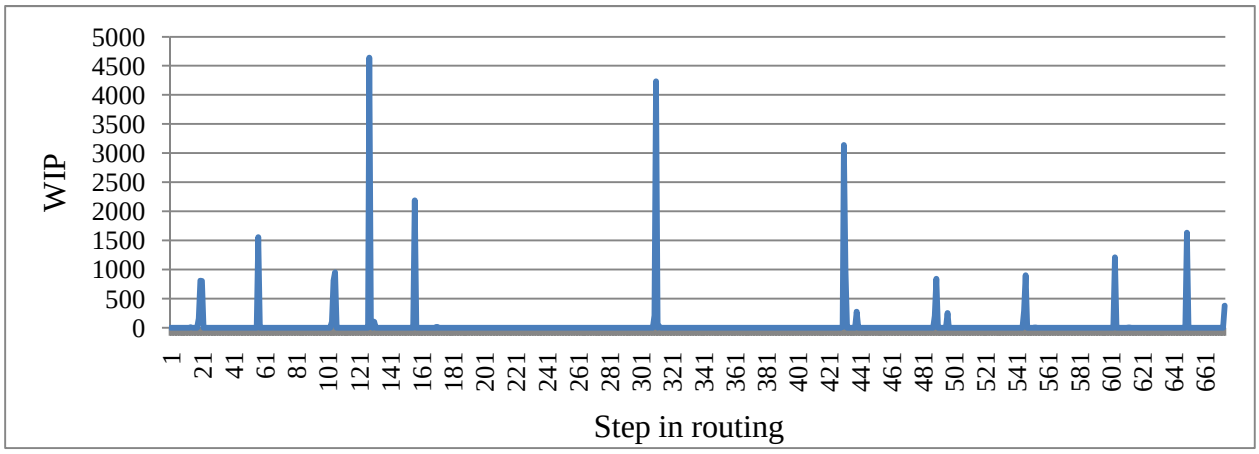

Figure A.33 WIP profile of C_1 at the end of the 4th week

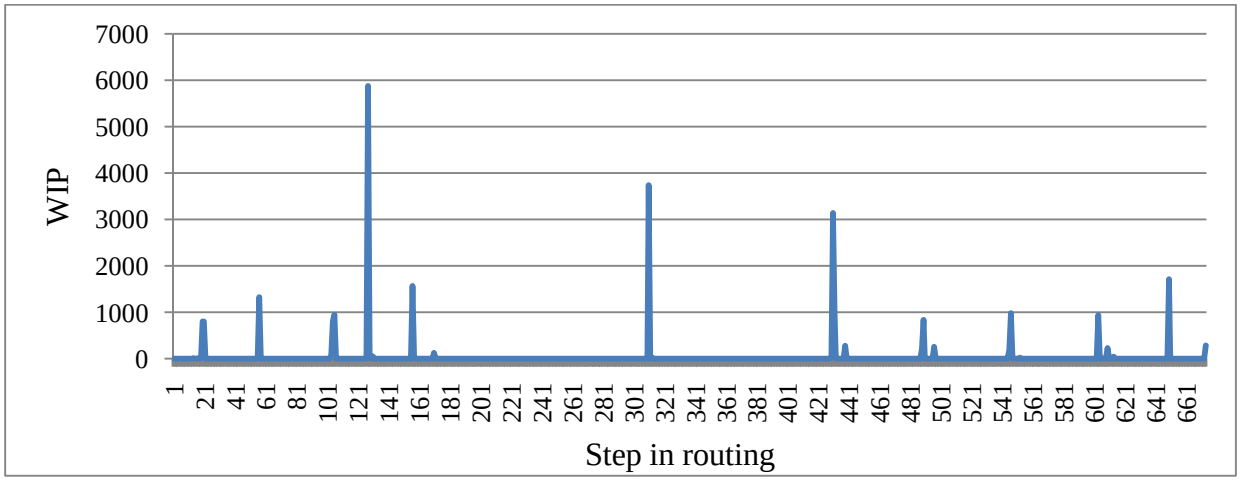

Figure A.34 WIP profile of C_1 at the end of the 5th week

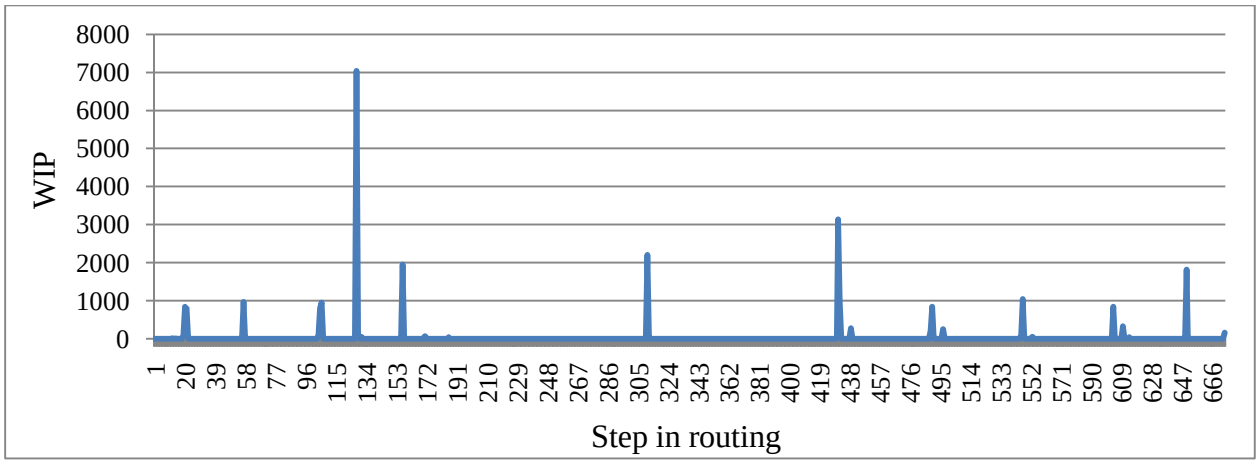

Figure A.35 WIP profile of C_1 at the end of the 6th week

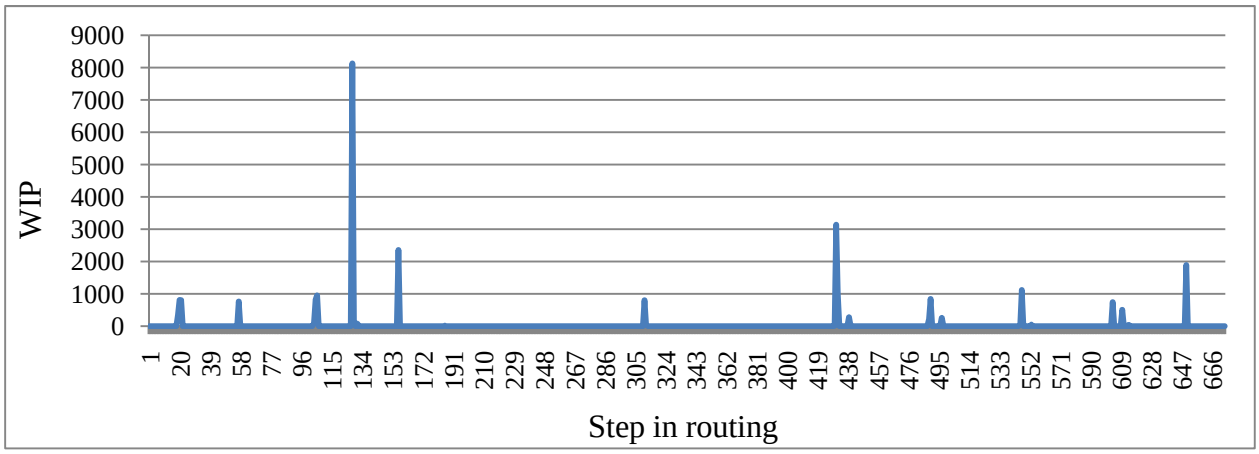

Figure A. 36 WIP profile of C_1 at the end of the 7th week

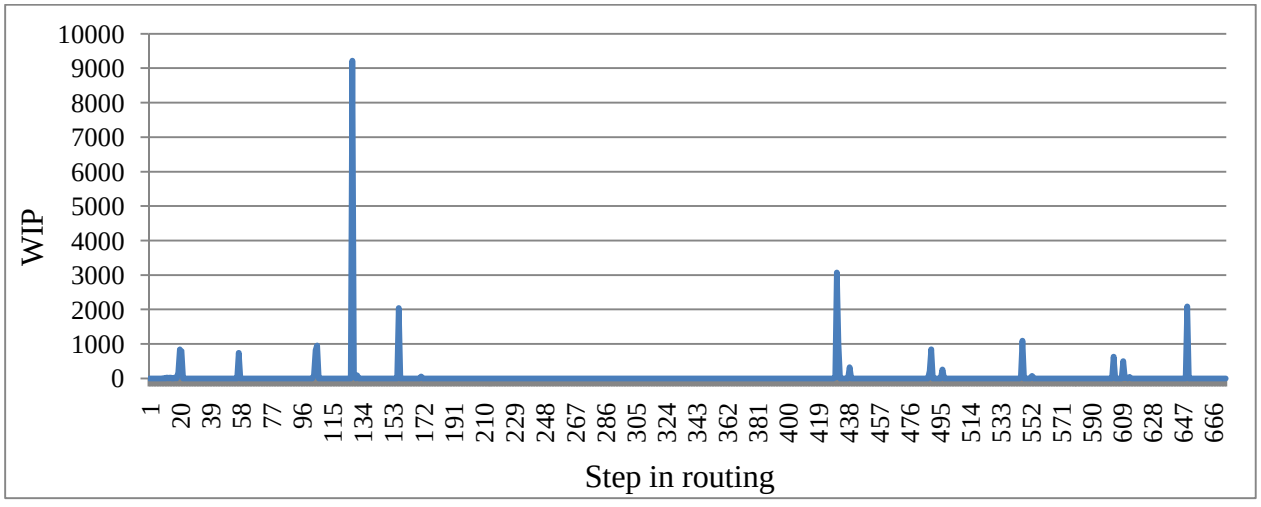

Figure A. 37 WIP profile of C_1 at the end of the 8th week

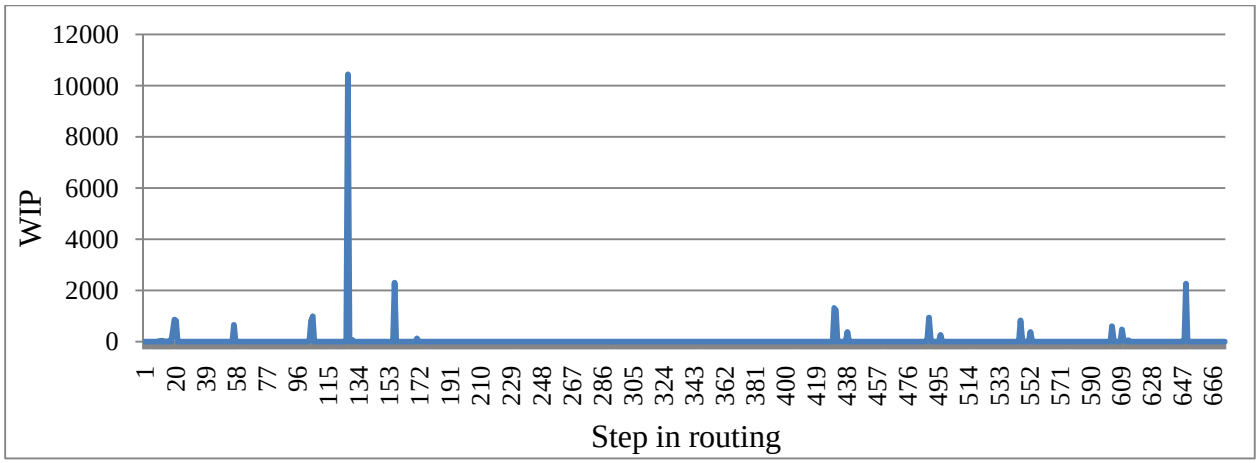

Figure A.38 WIP profile of C_1 at the end of the 9th week

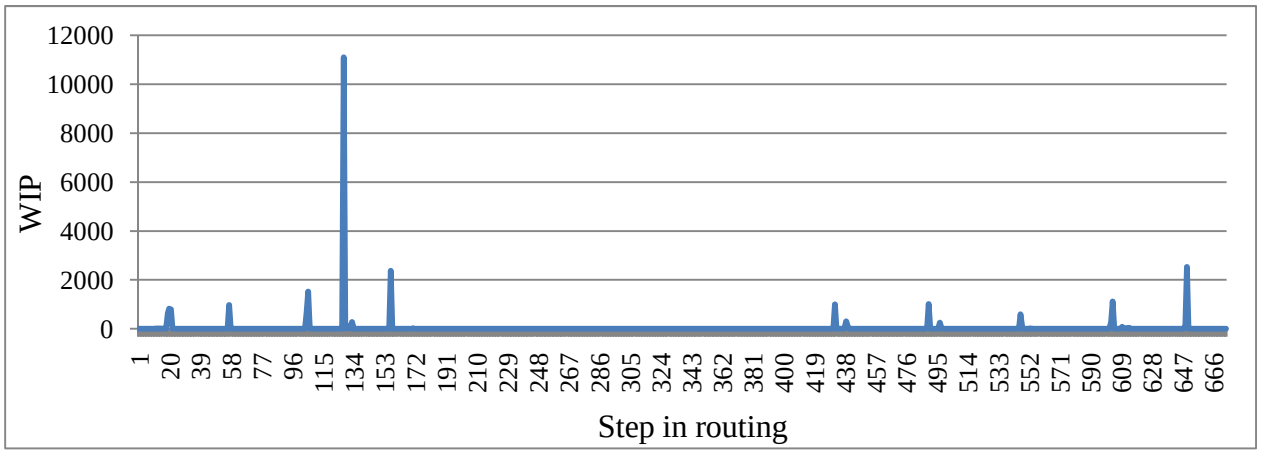

Figure A.39 WIP profile of C_1 at the end of the 10th week

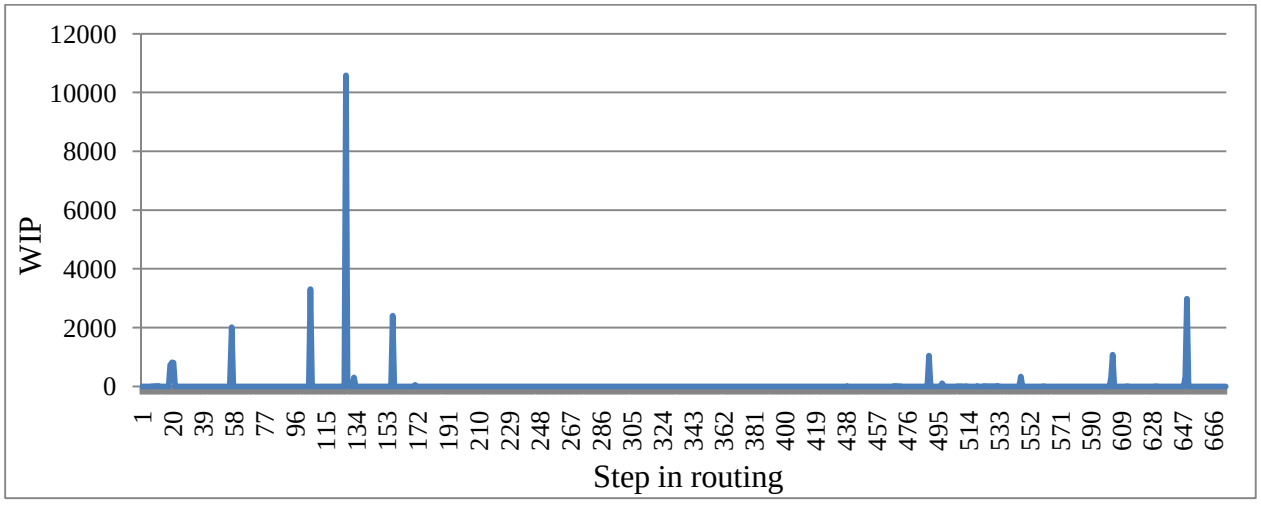

Figure A.40 WIP profile of C_1 at the end of the 11th week

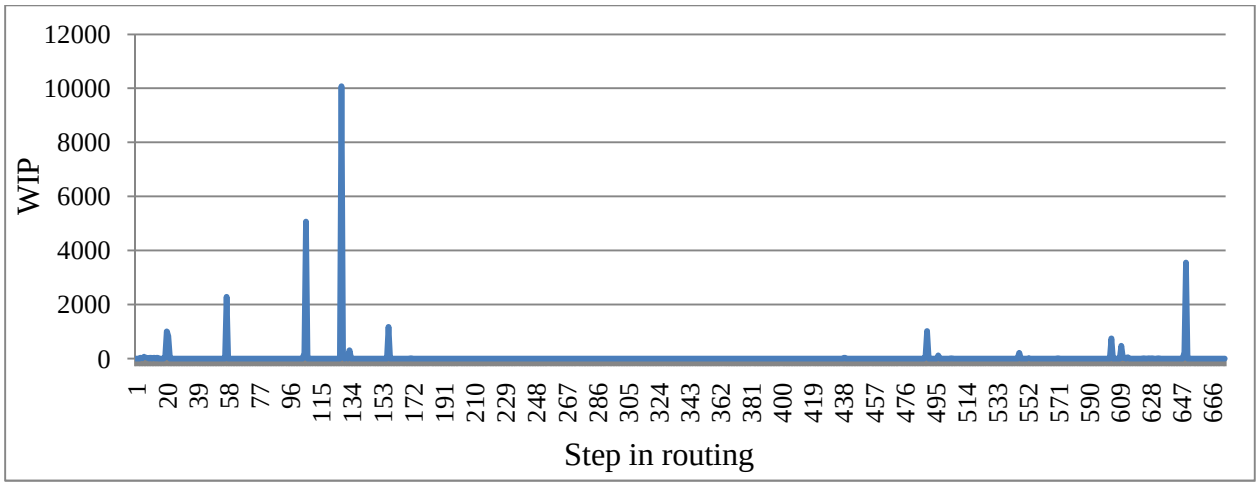

Figure A.41 WIP profile of C_1 at the end of the 12th week

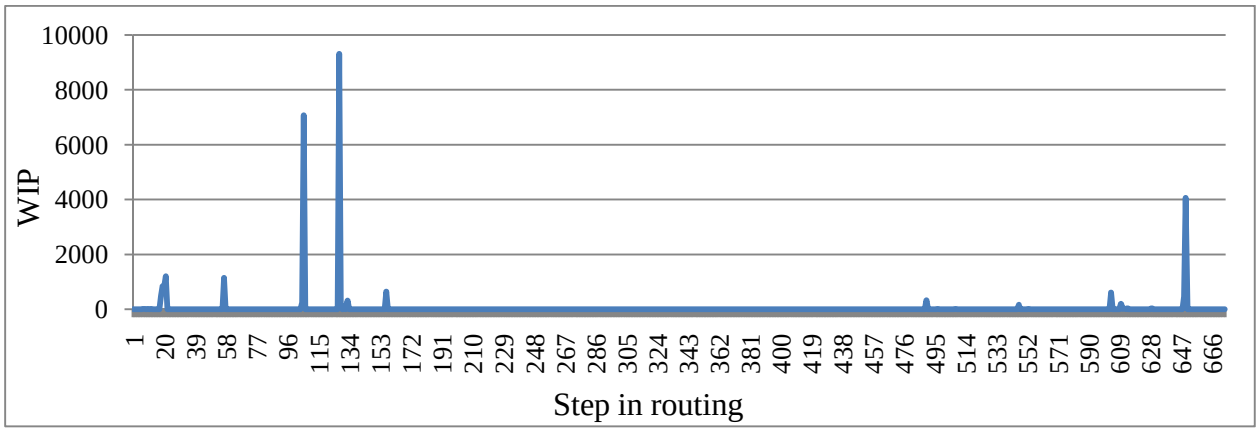

Figure A.42 WIP profile of C_1 at the end of the 13th week

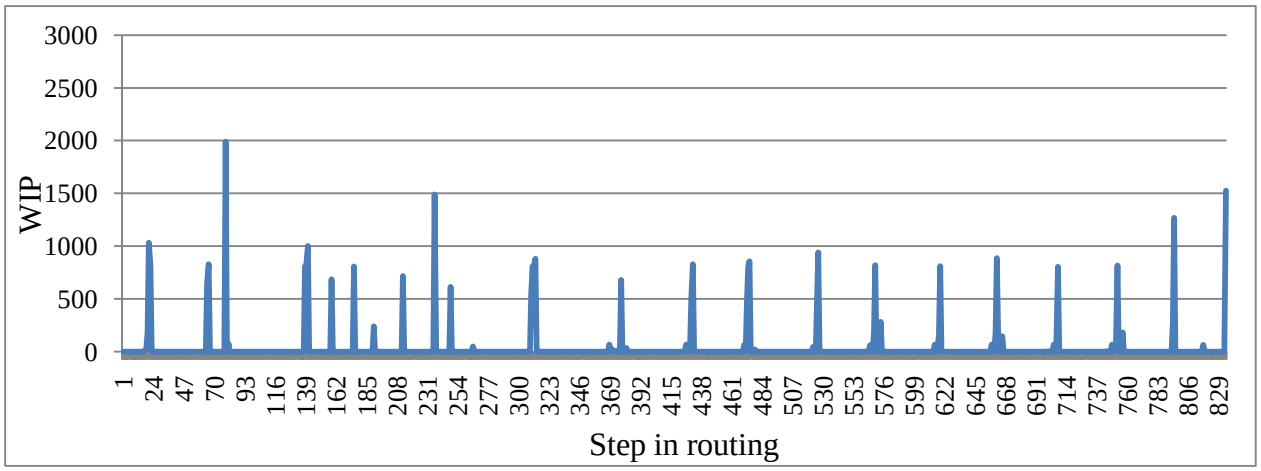

Figure A.43 WIP profile of C_2 at the end of the 1st week

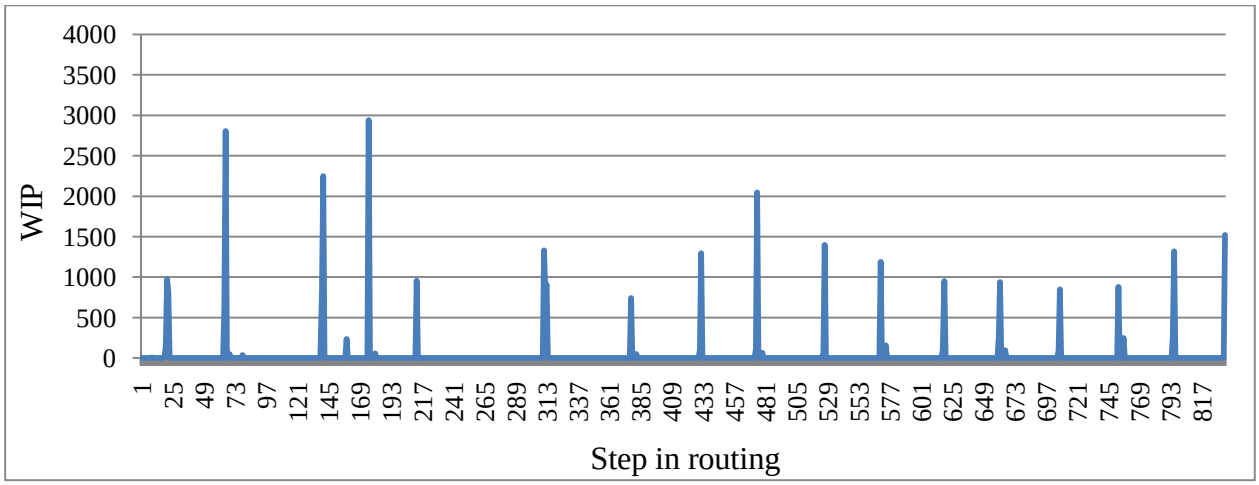

Figure A.44 WIP profile of C_2 at the end of the 2nd week

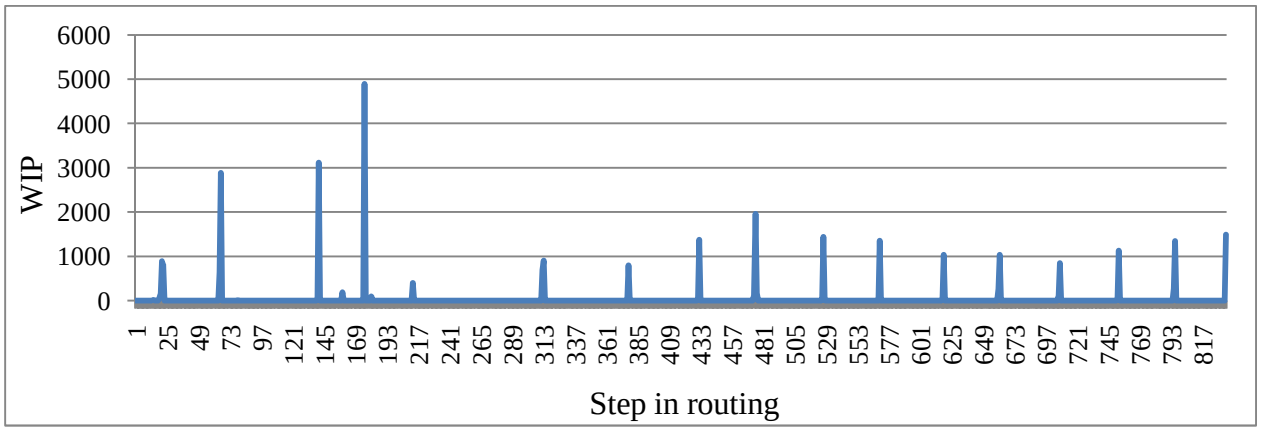

Figure A.45 WIP profile of C_2 at the end of the 3rd week

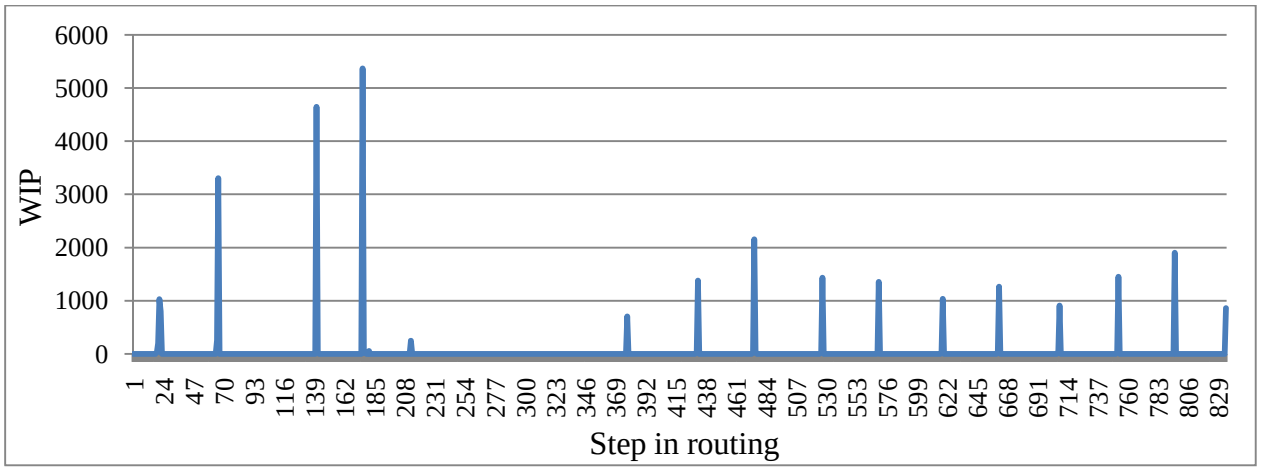

Figure A.46 WIP profile of C_2 at the end of the 4th week

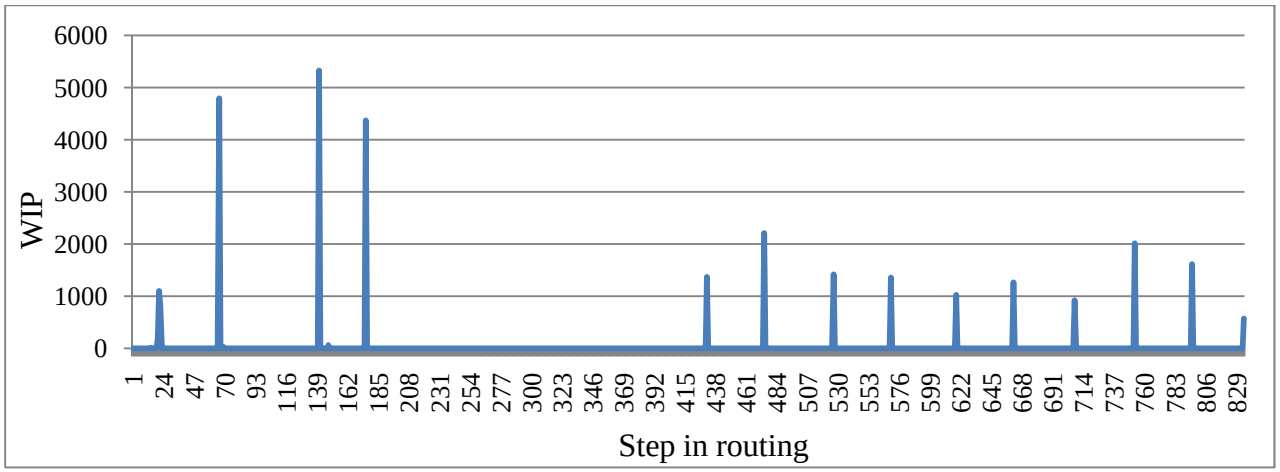

Figure A.47 WIP profile of C_2 at the end of the 5th week

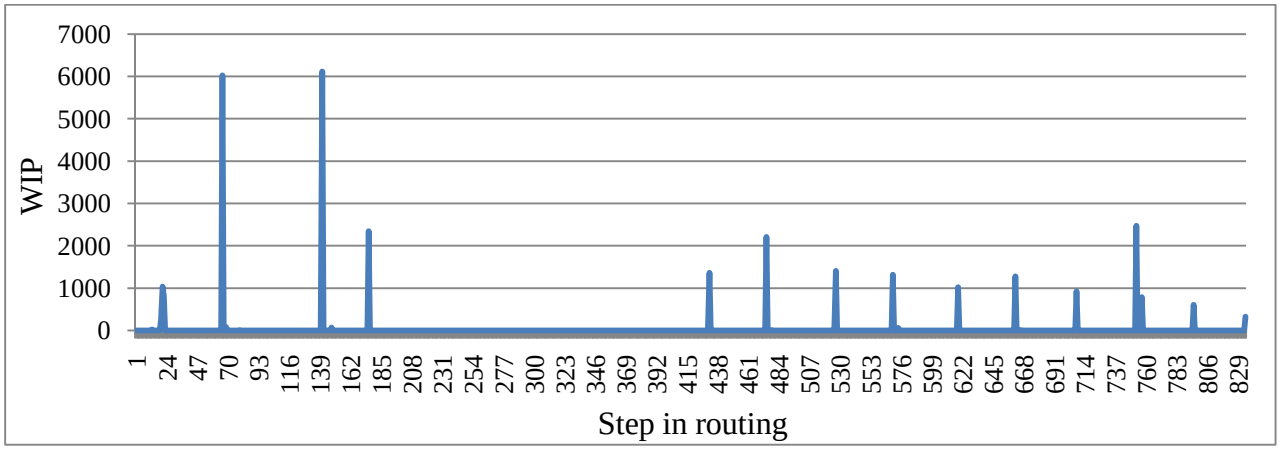

Figure A.48 WIP profile of C_2 at the end of the 6th week

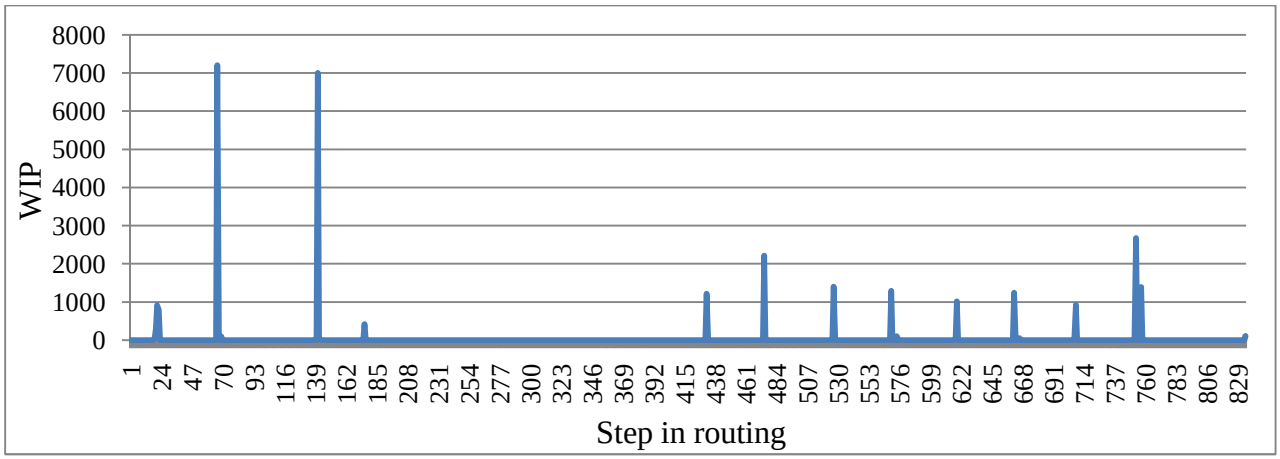

Figure A.49 WIP profile of C_2 at the end of the 7th week

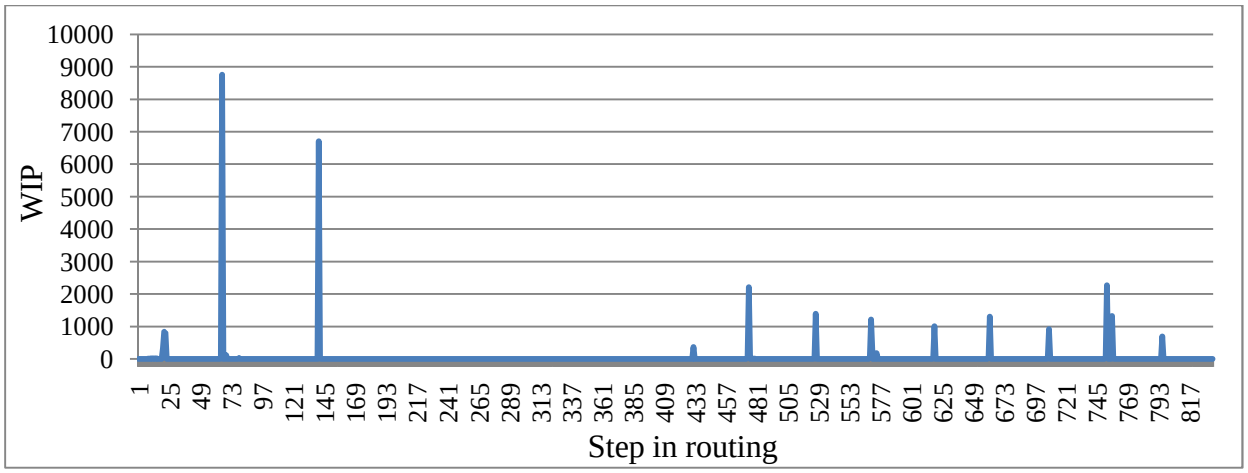

Figure A.50 WIP profile of C_2 at the end of the 8th week

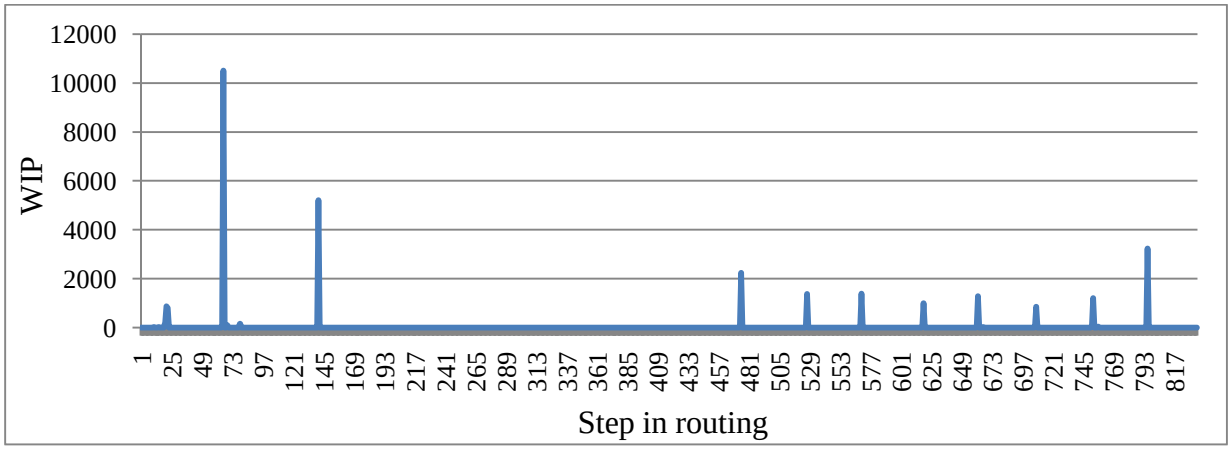

Figure A.51 WIP profile of C_2 at the end of the 9th week

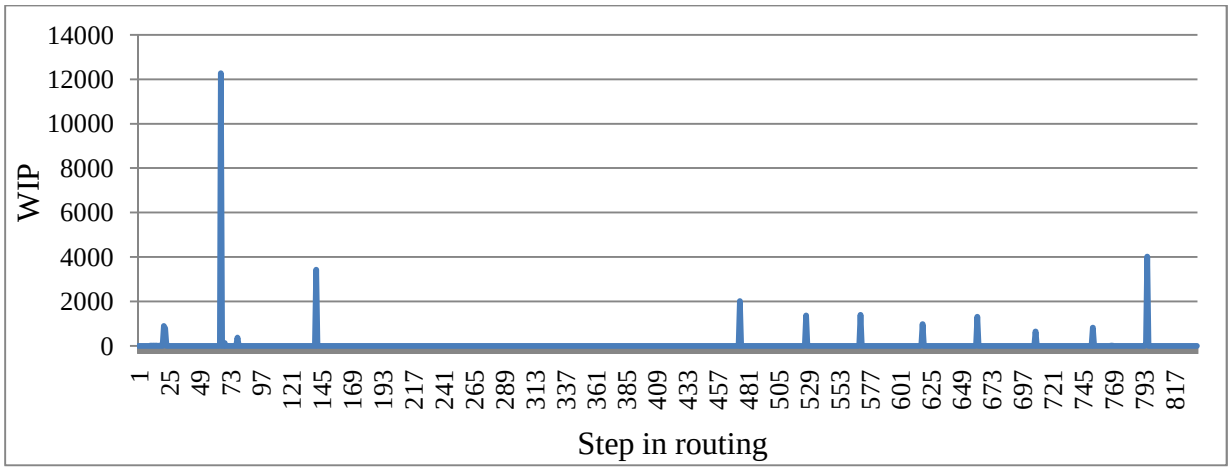

Figure A.52 WIP profile of C_2 at the end of the 10th week

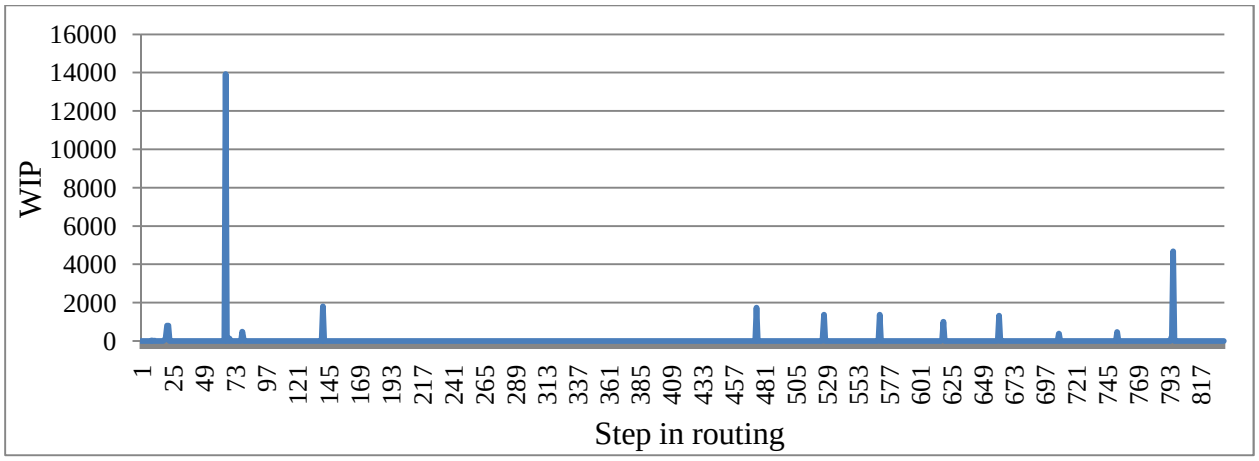

Figure A.53 WIP profile of C_2 at the end of the 11th week

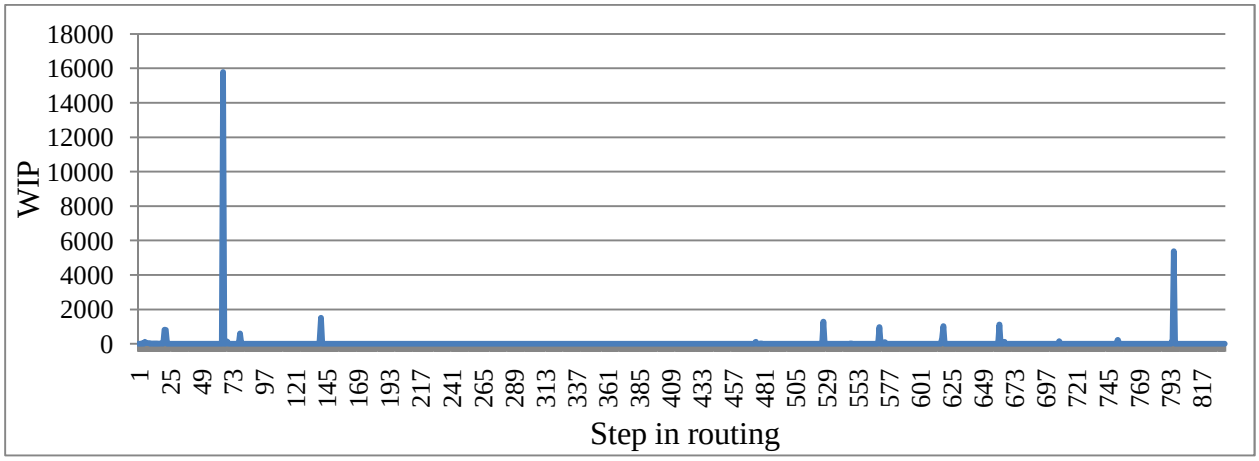

Figure A.54 WIP profile of C_2 at the end of the 12th week

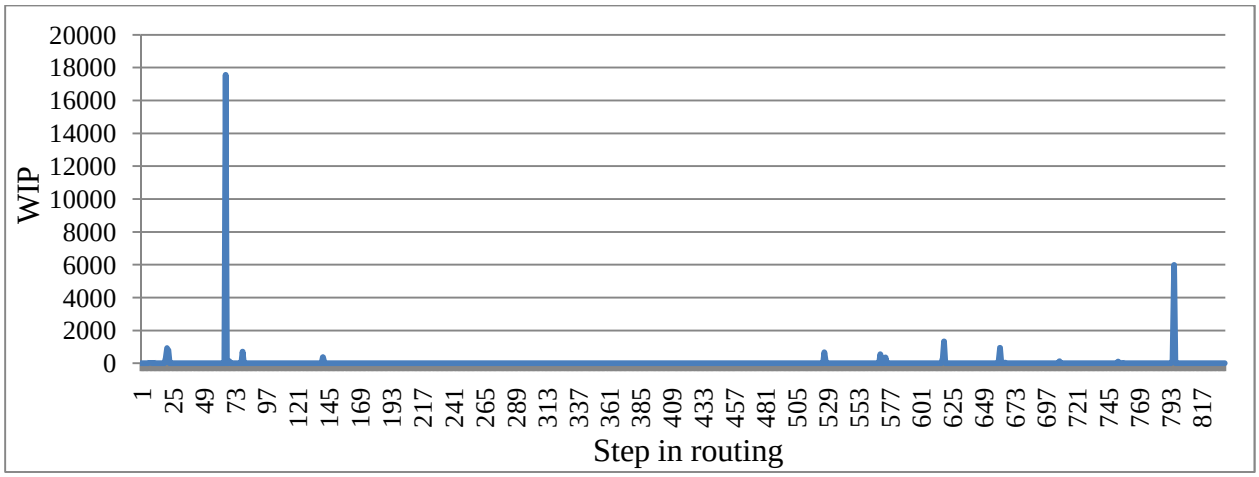

Figure A.55 WIP profile of C_2 at the end of the 13^{th} week

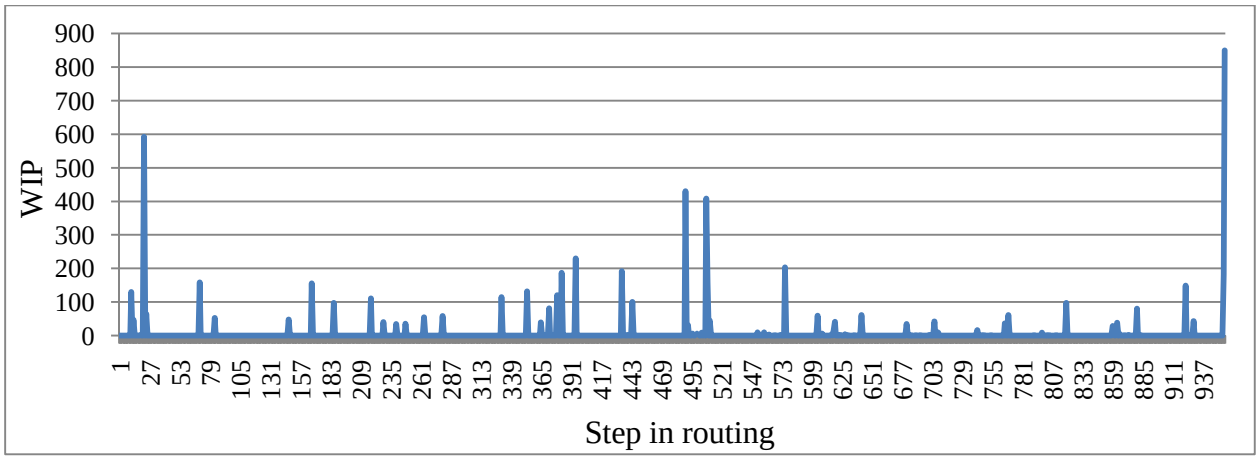

Figure A.56 WIP profile of C_3 at the end of the 1st week

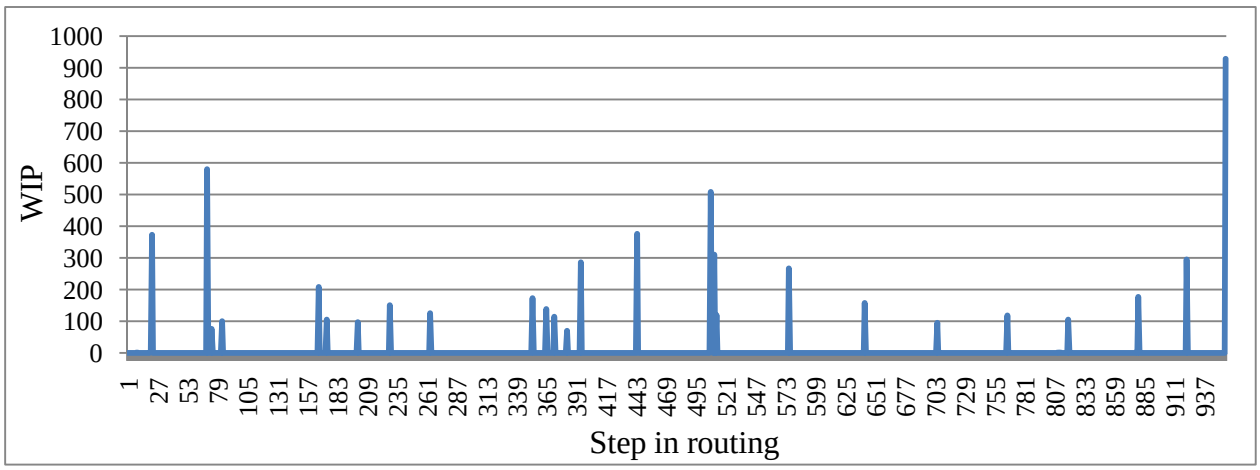

Figure A.57 WIP profile of C_3 at the end of the 2nd week

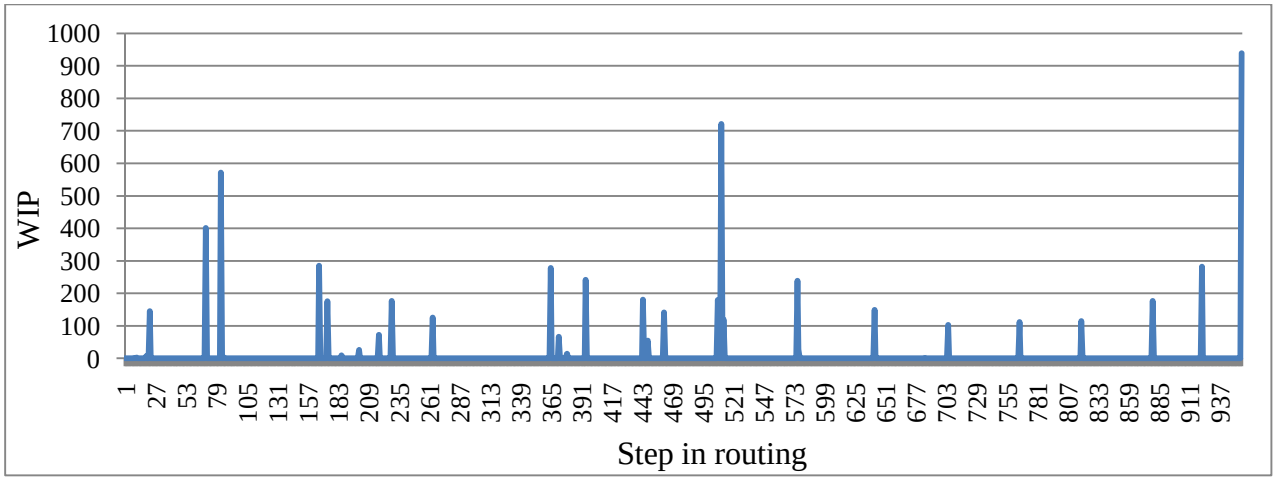

Figure A.58 WIP profile of C_3 at the end of the 3rd week

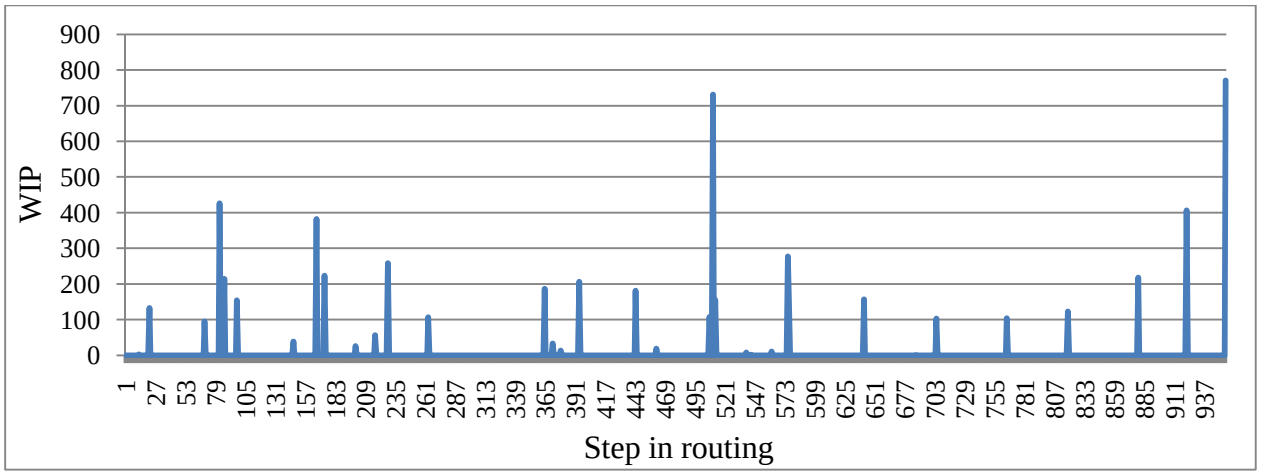

Figure A.59 WIP profile of C_3 at the end of the 4th week

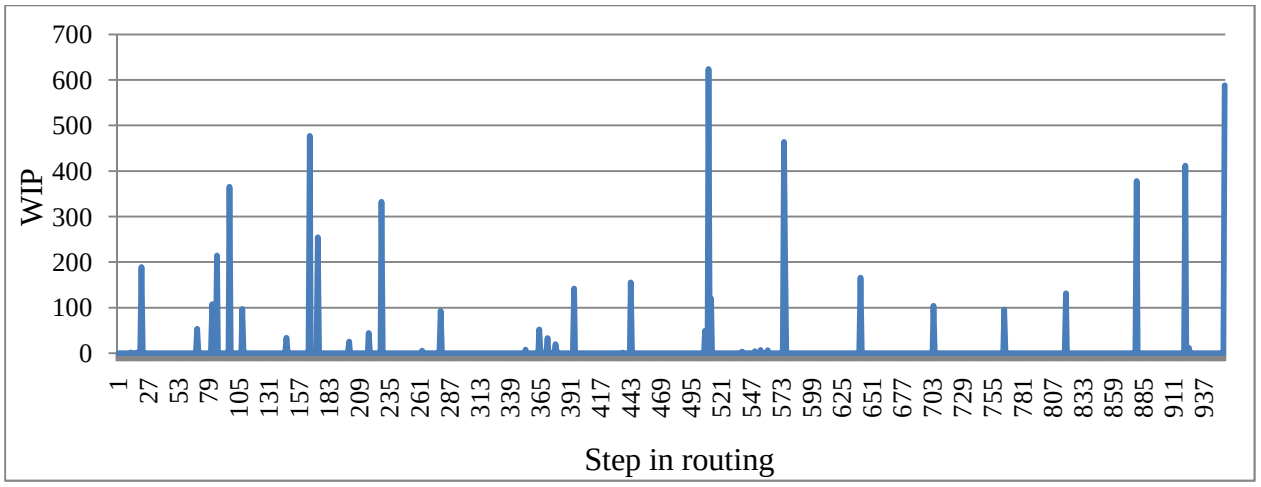

Figure A.60 WIP profile of C_3 at the end of the 5th week

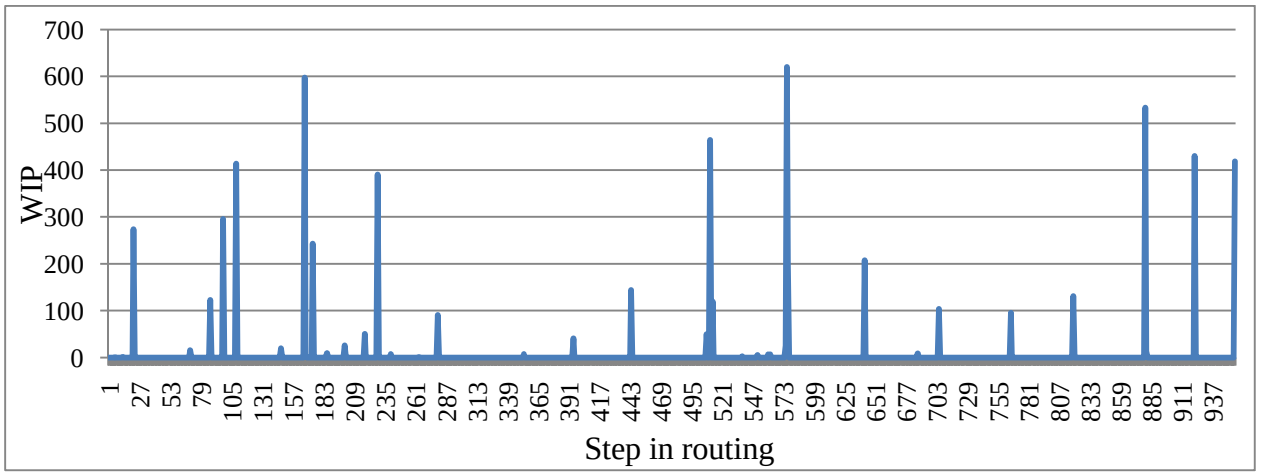

Figure A.61 WIP profile of C_3 at the end of the 6th week

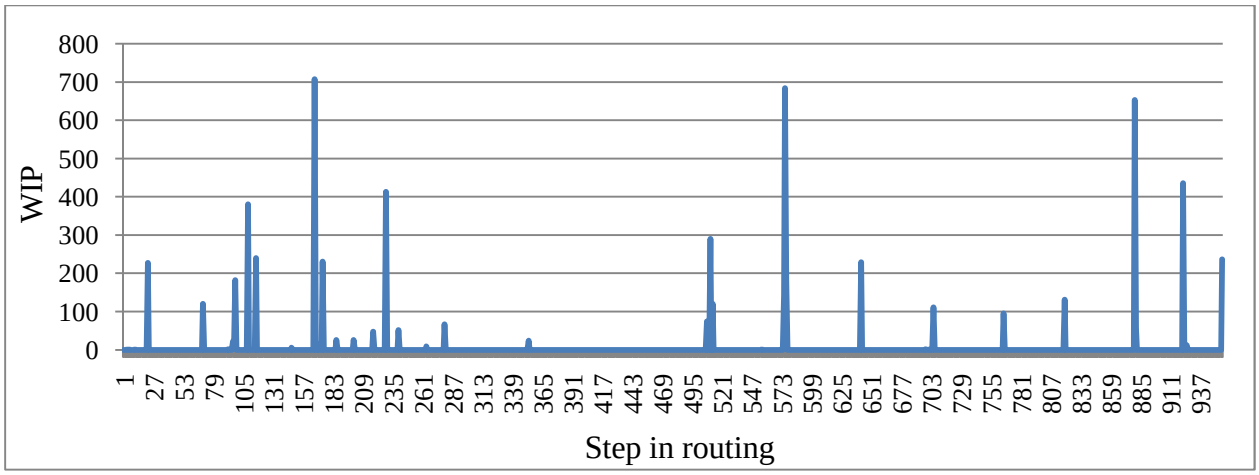

Figure A.62 WIP profile of C_3 at the end of the 7th week

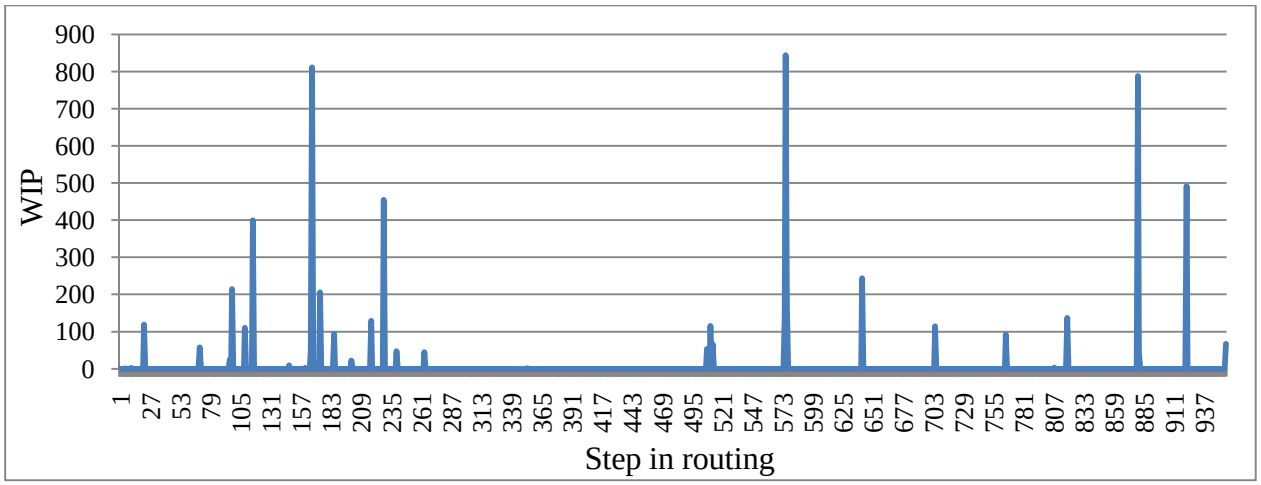

Figure A.63 WIP profile of C_3 at the end of the 8th week

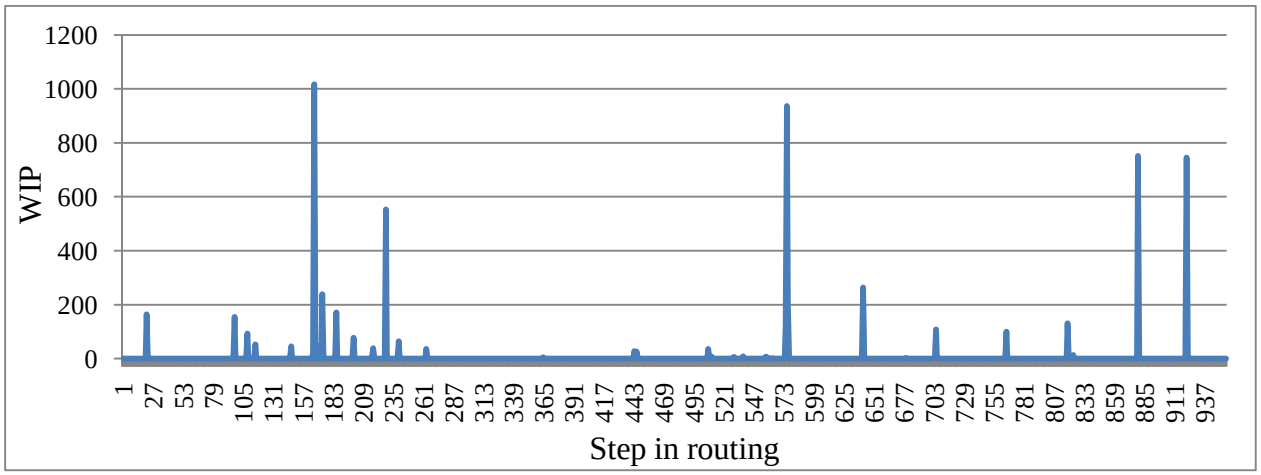

Figure A.64 WIP profile of C_3 at the end of the 9th week

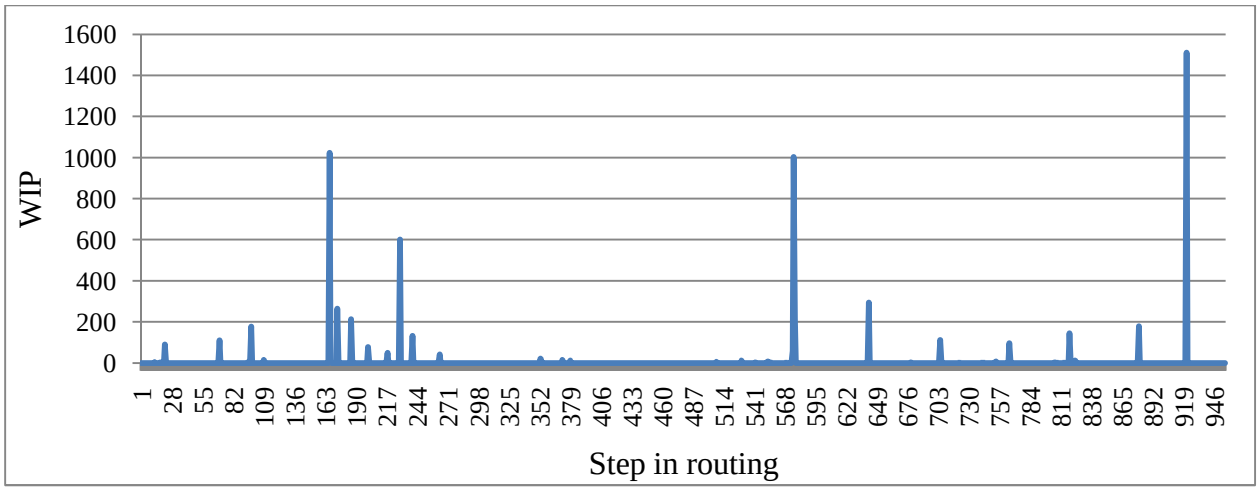

Figure A.65 WIP profile of C_3 at the end of the 10th week

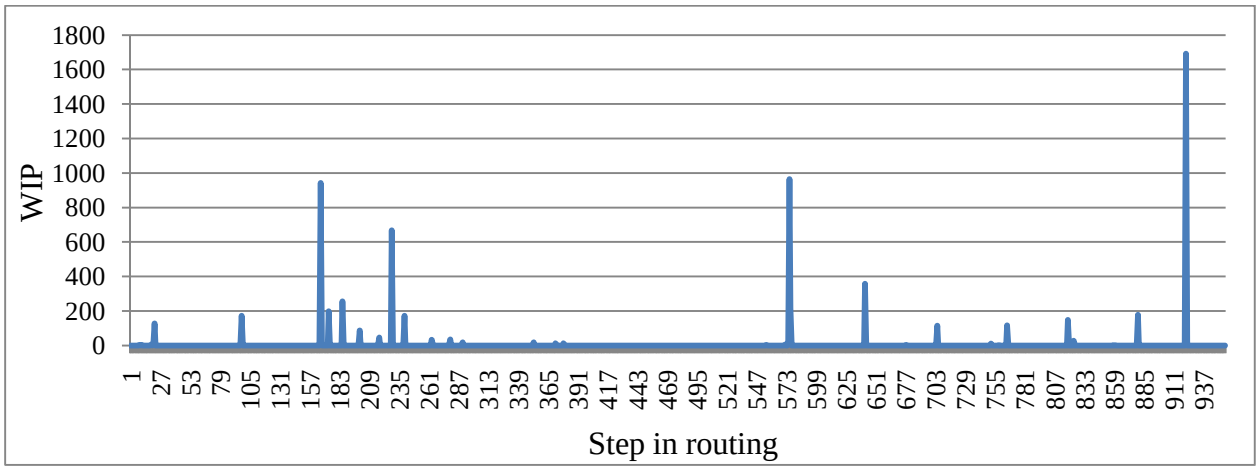

Figure A.66 WIP profile of C_3 at the end of the 11th week

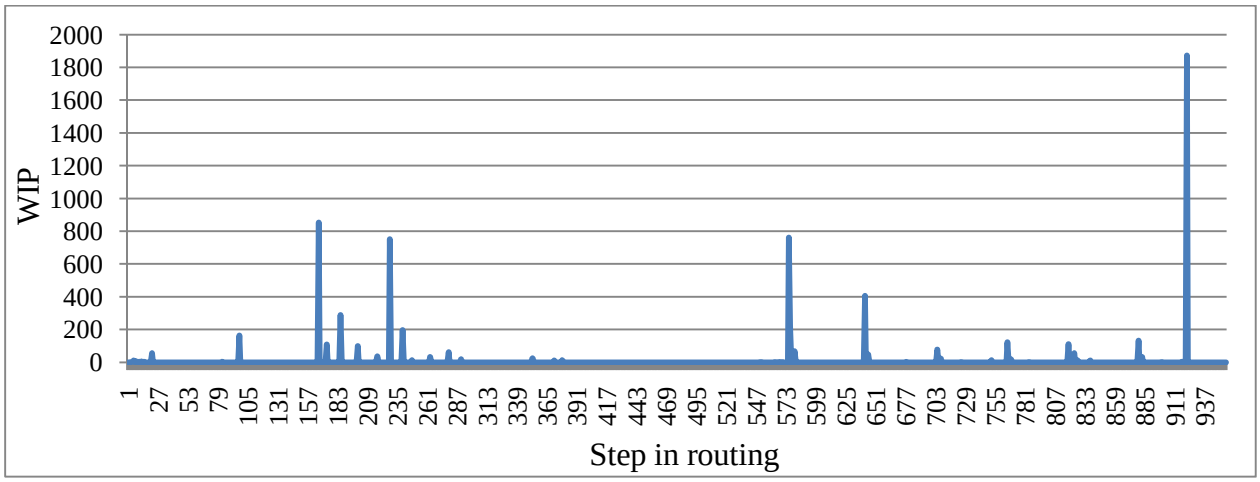

Figure A.67 WIP profile of C_3 at the end of the 12th week

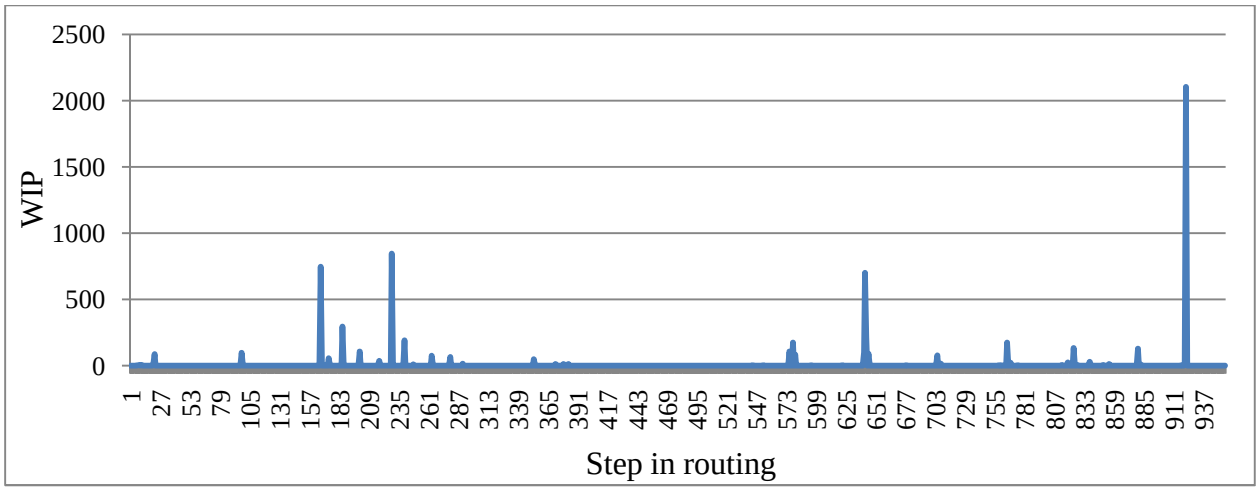

Figure A.68 WIP profile of C_3 at the end of the 13th week

Appendix 8: Benders Decomposition

Model (3) contains a series of block constraints with overlapping variables, where the blocks correspond to different time periods and a subset of the variables appears in the adjacent blocks for periods $t-1$ and t. In visual terms, the overlapping constraints form a staircase. The general problem is called a *multistage linear program* and has the following form:

$$w = \text{Minimize} \quad \mathbf{c}^0\mathbf{x}^0 + \mathbf{c}^1\mathbf{x}^1 + \mathbf{c}^2\mathbf{x}^2 + \cdots + \mathbf{c}^N\mathbf{x}^N \tag{17a}$$

$$\text{subject to} \quad -\mathbf{B}^0\mathbf{x}^0 + \mathbf{A}^1\mathbf{x}^1 = \mathbf{b}^1 \tag{17b}$$

$$-\mathbf{B}^1\mathbf{x}^1 + \mathbf{A}^2\mathbf{x}^2 = \mathbf{b}^2 \tag{17c}$$

$$-\mathbf{B}^2\mathbf{x}^2 + \mathbf{A}^3\mathbf{x}^3 = \mathbf{b}^3 \tag{17d}$$

$$\text{O}$$

$$-\mathbf{B}^{N-1}\mathbf{x}^{N-1} + \mathbf{A}^N\mathbf{x}^N = \mathbf{b}^N \tag{17e}$$

$$\mathbf{x}^0 \geq \mathbf{0},\ \mathbf{x}^1 \geq \mathbf{0}, \ldots, \mathbf{x}^N \geq \mathbf{0} \tag{17f}$$

where all vectors and matrices are of appropriate dimension and $\mathbf{B}^k$ ($k = 1,\ldots,N-1$) reflects the overlapping or staircase nature of the constraints. To decompose model (17), we introduce a sequence of vectors $\mathbf{z}^1, \mathbf{z}^2,\ldots,\mathbf{z}^N$ with the same dimensions as $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N$ and write each constraint as

$$-\mathbf{B}^{k-1}\mathbf{z}^{k-1} + \mathbf{A}^k\mathbf{x}^k = \mathbf{b}^k$$

with the restriction that $\mathbf{B}^k\mathbf{x}^k = \mathbf{B}^k\mathbf{z}^k$, $k = 1,\ldots,N$. Now, if the $\mathbf{z}$ variables are treated the same way that the integer variables are treated in Benders algorithm for solving mixed-integer linear programs, we get the following N primal subproblems.

$$w^P(\mathbf{z}^k) = \text{Minimize} \quad \mathbf{c}^k\mathbf{x}^k \tag{18a}$$

$$\text{subject to} \quad \mathbf{A}^k\mathbf{x}^k = \mathbf{b}^k + \mathbf{B}^{k-1}\mathbf{z}^{k-1} \tag{18b}$$

$$\mathbf{B}^k\mathbf{x}^k = \mathbf{B}^k\mathbf{z}^k \tag{18c}$$

$$\mathbf{x}^k \geq \mathbf{0} \tag{18d}$$

For the moment, assume that all the variables that appear in model (18) are continuous. Letting $\mathbf{u}^k$ and $\mathbf{y}^k$ be the dual variables associated with constraints (18b) and (18c), respectively, the dual of (18) is

$$w^D(\mathbf{z}^k) = \text{Maximize} \quad \mathbf{u}^k(\mathbf{b}^k + \mathbf{B}^{k-1}\mathbf{z}^{k-1}) + \mathbf{y}^k(\mathbf{B}^k\mathbf{z}^k)$$

$$\text{subject to} \quad \mathbf{u}^k\mathbf{A}^k + \mathbf{y}^k\mathbf{B}^k \leq \mathbf{c}^k \tag{19}$$

where $w^D(\mathbf{z}^k) = w^P(\mathbf{z}^k)$. Thus, model (3) is equivalent to

$$w = \text{Minimize} \sum_{k=1}^{N} w^D\left(\mathbf{z}^k\right) \tag{20}$$

To solve (20), we need to create a master problem. Let E^k be the set of extreme points associated with the feasible region in (19) and let R^k be the set of extreme rays, $k = 1,\ldots,N$. Benders' master problem can be written as

$$w = \text{Minimize} \quad \sum_{k=1}^{N} \eta_k \tag{21a}$$

$$\text{subject to} \quad \eta_k \geq \mathbf{u}^{ke}(\mathbf{b}^k + \mathbf{B}^{k-1}\mathbf{z}^{k-1}) + \mathbf{y}^{ke}(\mathbf{B}^k\mathbf{z}^k),\ \forall\ e \in E^k\ ,\ k = 1,\ldots,N \tag{21b}$$

$$0 \geq \mathbf{v}^{kr}(\mathbf{b}^k + \mathbf{B}^{k-1}\mathbf{z}^{k-1}) + \mathbf{q}^{kr}(\mathbf{B}^k\mathbf{z}^k),\ \ \forall\ r \in R^k\ ,\ k = 1,\ldots,N \tag{21c}$$

where the constraints in (21b) are called optimality cuts and those in (21c) are called feasibility cuts. The standard Benders algorithm can now be used to solve (21), where at each iteration, cuts are added by solving either the N dual subproblems in (19) or the N primal subproblems (18) and then computing the corresponding dual variables $\mathbf{u}^k$ and $\mathbf{y}^k$ or dual extreme rays $\mathbf{v}^k$ and $\mathbf{q}^k$.

Appendix 9: Pseudocodes for Neighborhood Search in Capacitated Clustering

The following definitions, together with the aforementioned notation in Chapter 3, are used to describe the phase II procedures.

Parameters

β	a percentage serves as tolerance
$TEW(x)$	sum of edge weights in every cluster of solution x
$c(i, x)$	cluster to which node i is assigned in solution x
ben_i_s	benefit obtained by shifting node i to cluster s
best_ben_i_s	best benefit from *ben_i_s*
benefit	benefit obtained by moving node i
best_ben	best benefit found during the local search for all of the nodes
i^{best}	node to be moved with *best_ben*
s^{best}	cluster to which i^{best} is assigned in the best move
j^{best}	node to be moved from s^{best} if $W_{s^{best}}(x) + w_{i^{best}} \geq C^{max}$
s_1^{best}	cluster to which node j^{best} is moved to
j', j^*	temporary variables for the node to be shifted
s_1, s_1', s_1^*	temporary variables for the cluster to which j' and j^* are shifted
Δ	percentage decrement for tolerance β

Procedure: $N_1(x, w, c, \beta, C^{min}, C^{max}, x^*)$

Input: current solution x, node weights vector w, edge weights matrix c, capacity bounds C^{min} and C^{max} and tolerance β.

Output: local solution x^* with respect to neighborhood $N_1(x)$.

Step 1: Initialization

$$TEW(x) = \sum_{k=1}^{p} \sum_{i,\, j \in V_k(x)} c_{ij} \; ; \; best\_ben = -\infty; \; x^* = x;$$

Step 2: for every node perform an N_1 local search

for ($i \in V$){
$k = c(i, x)$;
if ($W_k(x) - w_i \geq C^{min}$){
best_ben_i_s $= -\infty$;
for ($s = 1,\ldots,p$ and $s \neq k$){
if ($W_s(x) + w_i \leq C^{max}$){

$$ben\_i\_s = \sum_{j \in V_s^*(x)} c_{ij} - \sum_{j \in V_k(x) \setminus \{i\}} c_{ij} \; ;$$

}else{//cluster s will exceed the upper bound

$$(j', s_1') = \underset{j, s_1}{\text{argmax}} \{ \sum_{l \in V_s(x) \setminus \{j\}} c_{il} - \sum_{l \in V_k(x) \setminus \{i\}} c_{il} + \sum_{l \in V_{s_1}(x)} c_{jl} - \sum_{l \in V_s(x)} c_{jl} \, ,$$

$\forall\, j \in V_s, j \neq i, s_1 = 1,\ldots,p, s_1 \neq s, C^{min} \leq W_s(x) + w_i - w_j \leq C^{max}$ and $W_{s_1}(x) + w_j \leq C^{max}$ };

$ben\_i\_s = \sum_{l \in V_s(x) \setminus \{j'\}} c_{il} - \sum_{l \in V_k(x) \setminus \{i\}} c_{il} + \sum_{l \in V_{s_1'}(x)} c_{j'l} - \sum_{l \in V_s(x)} c_{j'l}$;

}

if ($ben\_i\_s > best\_ben\_i\_s$){

$bes\_bent\_i\_s = ben\_i\_s$; $s^* = s$; $benefit = ben\_i\_s$;

if ($W_{s^*}(x) + w_i \geq C^{\max}$){

$j^* = j'$; $s_1^* = s_1'$;

}

}

}//end_for_s

if ($benefit > best\_ben$) {

$best\_ben = benefit$; $i^{\text{best}} = i$; $s^{\text{best}} = s^*$;

if ($W_{s^{\text{best}}}(x) + w_{i^{\text{best}}} \geq C^{\max}$){

$j^{\text{best}} = j^*$; $s_1^{\text{best}} = s_1^*$;

}

}

}

}

Step 3: if ($best\_ben > -\beta \times t_w(x)$){

$k = c(i^{\text{best}}, x)$;

$V_k(x^*) \leftarrow V_k(x) \setminus \{i^{\text{best}}\}$;

$V_{s^{\text{best}}}(x^*) \leftarrow V_{s^{\text{best}}}(x) \cup \{i^{\text{best}}\}$;

if ($|V_{s^{\text{best}}}(x)| = C^{\max}$){

$V_{s^{\text{best}}}(x^*) \leftarrow V_{s^{\text{best}}}(x^*) \setminus \{j^{\text{best}}\}$; $V_{s_1^{\text{best}}}(x^*) \leftarrow V_{s_1^{\text{best}}}(x) \cup \{j^{\text{best}}\}$;

}

}

Figure A.69 Pseudocode for local search in neighborhood N_1

Parameters

$k_1, k_2, s_1, s_2, s_3, s^*$	temporary variables for cluster identification
$ben\_e\_s_l$	temporary variable for benefit obtained, l = 1, 2, 3
e^{best}	edge e of the move with *best_ben*
$i^{\text{best}}, j^{\text{best}}$	the two endpoints of e^{best}
s^{best}	cluster to which e^{best} is assigned in the best move

Procedure: $N_2(x, w, c, \beta, C^{\min}, C^{\max}, x^*)$

Input: current solution x, node weights vector w, edge weights matrix c, capacity bounds $C^{\min}$ and $C^{\max}$ and tolerance β.

Output: optimal solution in neighborhood $N_2(x)$, x^*

Step 1: initialization

$TEW(x) = \sum_{k=1}^{p} \sum_{i,j \in V_k(x)} c_{ij}$; $best\_ben = -\infty$; $x^* = x$;

Step 2: for every edge perform an N_2 local search

for ($e = (i,j) \in E$) {

$k_1 = c(i, x)$; $k_2 = c(j, x)$;

if ($k_1 = k_2 = k$ and $W_k(x) - w_i - w_j \geq C^{\min}$){//$e$ is within cluster k

$$s^* = \underset{s}{\text{argmax}} \{ \sum_{l \in V_s(x)} (c_{il} + c_{jl}) - \sum_{l \in V_k(x) \setminus \{i,j\}} (c_{il} + c_{jl}),$$

$$\forall\, s \in \{1,..,p\} \setminus \{k\} \text{ and } W_s(x) + w_i + w_j \leq C^{\max} \};$$

$$benefit = \sum_{l \in V_{s^*}(x)} (c_{il} + c_{jl}) - \sum_{l \in V_k(x) \setminus \{i,j\}} (c_{il} + c_{jl});$$

} else {

if ($W_{k_1}(x) - w_i \geq C^{\min}$ and $W_{k_2}(x) - w_j \geq C^{\min}$) {

if ($W_s(x) + w_i + w_j \leq C^{\max}$) {

$$s_1 = \underset{s}{\text{argmax}} \{ \sum_{l \in V_s(x)} (c_{il} + c_{jl}) + c_{ij} - \sum_{l \in V_{k_1}(x)} c_{il} - \sum_{l \in V_{k_2}(x)} c_{jl},$$

$$\forall\, s \neq k_1,\, s \neq k_2\};$$

$$ben\_e\_s_1 = \sum_{l \in V_{s_1}(x)} (c_{il} + c_{jl}) + c_{ij} - \sum_{l \in V_{k_1}(x)} c_{il} - \sum_{l \in V_{k_2}(x)} c_{jl};$$

}

if ($W_{k_1}(x) + w_{b_e} \leq C^{\max}$) {

$$ben\_e\_s_2 = \sum_{l \in V_{k_1}(x)} c_{jl} - \sum_{l \in V_{k_2}(x)} c_{jl};\ s_2 = k_1;$$

}

if ($W_{k_2}(x) + w_{a_e} \leq C^{\max}$) {

$$ben\_e\_s_3 = \sum_{l \in V_{k_2}(x)} c_{il} - \sum_{l \in V_{k_1}(x)} c_{il};\ s_3 = k_2;$$

}

$benefit = \max \{ben\_e\_s_1, ben\_e\_s_2, ben\_e\_s_3\}$;

$s^* = s_l$, where $ben\_e\_s_l = benefit$ with $l = 1, 2, 3$;

}

}

if ($benefit > best\_ben$) {

$best\_ben = benefit$;

$e^{\text{best}} = e$; $s^{\text{best}} = s^*$;

}

}

Step 3: if ($best\_ben > -\beta \times t_w(x)$){

$e^{\text{best}} = (i^{\text{best}}, j^{\text{best}})$;

$k_1 = c(i^{\text{best}}, x)$; $k_2 = c(j^{\text{best}}, x)$;

if ($k_1 = k_2$) {

$k = k_1$; $V_k(x^*) = V_k(x) \setminus \{i^{\text{best}}, j^{\text{best}}\}$; $V_{s^{\text{best}}}(x^*) = V_{s^{\text{best}}}(x) \cup \{i^{\text{best}}, j^{\text{best}}\}$;

} else {

if ($s^{\text{best}} = k_1$) {

$V_{k_1}(x^*) = V_{k_1}(x) \cup \{j^{\text{best}}\}$; $V_{k_2}(x^*) = V_{k_2}(x) \setminus \{j^{\text{best}}\}$;

}

else if ($s^{best} = k_2$) {

$V_{k_1}(x^*) = V_{k_1}(x) \setminus \{i^{best}\}$; $V_{k_2}(x^*) = V_{k_2}(x) \cup \{i^{best}\}$;

} else {

$V_{k_1}(x^*) = V_{k_1}(x) \setminus \{i^{best}\}$; $V_{k_2}(x^*) = V_{k_2}(x) \setminus \{j^{best}\}$;

$V_{s^{best}}(x^*) = V_{s^{best}}(x) \cup \{i^{best}, j^{best}\}$;

}

}

}

Figure A.70 Pseudocode for local search in neighborhood N_2

Parameters

(i^{best}, j^{best}) nodes to be swapped corresponding to *best_ben*

Procedure: $N_3(x, w, c, \beta, C^{min}, C^{max}, x^*)$

Input: current solution x, node weights vector w, edge weights matrix c, capacity bounds C^{min} and C^{max} and tolerance β.

Output: local solution x^* with respect to neighborhood $N_3(x)$.

Step 1: initialization

$TEW(x) = \sum_{k=1}^{p} \sum_{i, j \in V_k(x)} c_{ij}$; *best_ben* = $-\infty$; $x^* = x$;

Step 2: for every pair of nodes perform a local search in N_3

for ($k = 1,\ldots,p$){

if ($C^{min} \le W_k(x) - w_i + w_j \le C^{max}$) {

for ($s = k + 1,\ldots,p$){

if ($C^{min} \le W_s(x) - w_j + w_i \le C^{max}$) {

$(i^*, j^*) =$

$\text{argmax}\{ (\sum_{l \in V_s(x)} c_{il} - \sum_{l \in V_k(x) \setminus \{i\}} c_{il}) + (\sum_{l \in V_k(x)} c_{jl} - \sum_{l \in V_s(x) \setminus \{j\}} c_{jl}) :$

$i \in V_k(x)$ and $j \in V_s(x)\}$;

benefit = $(\sum_{l \in V_s(x)} c_{i^* l} - \sum_{l \in V_k(x) \setminus \{i^*\}} c_{i^* l}) + (\sum_{l \in V_k(x)} c_{j^* l} - \sum_{l \in V_s(x) \setminus \{j^*\}} c_{j^* l})$;

if (*benefit* > *best_ben*) {

best_ben = *benefit*;

$i^{best} = i^*$; $j^{best} = j^*$;

}

}

}

}

}

Step 3: if (*best_ben* > $-\beta \times t_w(x)$){

$k = c(i^{best}, x)$; $s = c(j^{best}, x)$;

$V_k(x^*) = (V_k(x) \setminus \{i^{best}\}) \cup \{j^{best}\}$;

$V_s(x^*) = (V_s(x) \setminus \{j^{best}\}) \cup \{i^{best}\}$;

}

Figure A.71 Pseudocode for local search in neighborhood N_3

Appendix 10: Pseudocodes for Back-end Operations

Indices and sets

i	index for machines; $i \in M$
j	index for devices; $j \in D$
k	index for key devices; $k \in K$
l	index for lots; $l \in L$
p	index for package devices; $p \in P$
s	index for routes; $s \in S$
t	index for tooling families; $t \in T$
τ	index for temperature; $\tau \in \overline{T}$
λ	index for tooling setup; $\lambda \in \Lambda$
n, m	index for temperature combinations; $n, m \in N$
D	set of all devices, including regular devices, key devices and package devices
K	set of key devices; $K \subseteq D$
L	set of lots in WIP at current operation
$L(i)$	set of lots that can be processed on machine i
$L(i,j)$	set of lots consisting of device j that can be processed on machine i
$L(i,\lambda)$	set of lots that can be processed on machine i with tooling setup λ
M	set of machines (each machine falls into a machine group)
$M(l)$	set of machines that can process lot l
N	set of feasible temperature combinations for machines and tooling
$\overline{N}(n)$	set of temperature combinations that intersect combination n.
P	set of package devices; $P \subseteq D$
S	set of routes
$S(i)$	set of routes that use machine i
$S(i,l)$	set of routes that use machine i to process lot l
$S(i,l,\lambda)$	set of routes that use machine i to process lot l with tooling setup λ
T	set of tooling families
$\overline{T}$	set of operating temperatures
$\overline{T}(n)$	set of operating temperatures that are elements of temperature combination n
Λ	set of tooling setups that used in the routs S
$\Lambda(i)$	set of tooling setups that can be installed in machine i
$\Lambda(i,t,\tau)$	set of tooling setups that can be installed in machine i using tooling family t under temperature τ

Parameters and data

$b_{\lambda t}$	number of tooling pieces from family t required by tooling setup λ
n_{tm}^{tooling}	number of tooling pieces from family t available under temperature combination m
n_l^{chips}	number of chips in lot l

$n_k^{\text{min\_chips}}$	minimum number of chips associated with key device k required to be processed over the planning period
$n_p^{\text{min\_chips}}$	minimum number of chips associated with package device p required to be processed over the planning period
r_{ils}	processing rate of lot l on machine i using route s (chips per hour)
w_l	weight (benefit) associated with processing lot l (function of lot age and the remaining planned cycle time)
w_1^k	weight (penalty) associated with shortage of package device k
w_2^p	weight (penalty) associated with shortage of key device p
ε_s	penalty for choosing route s
C	normalizing constant associated with key and package device shortages
H_i	(capacity) number of hours available on machine i over the planning period

Decision variables

x_{ils}	1 if lot l is processed by machine i with route s, 0 otherwise
$y_{i\lambda}$	1 if machine i is using tooling setup λ, 0 otherwise
Δ_1^k	shortage of key device k
Δ_2^p	shortage of package device p

Figure A.72 Notation for setup and assignment model

Procedure: $N_1(x^{\text{IP}}, \overline{\lambda}_i, x^*)$

Input: Feasible IP solution x^{IP}, current machine setup $\overline{\lambda}_i$, $\forall\ i \in M$

Output: Local optimum x^* in neighborhood $N_1(x^{\text{IP}})$

Step 1: //Initialize L_0, L_i, t_i^{IP} and x^* according to x^{IP}

$L_0 \leftarrow \emptyset$; $L_i \leftarrow \emptyset$; $t_i^{\text{IP}} = 0$, $\forall\ i \in M$;

$x_{ils}^* = x_{ils}^{\text{IP}}$, $\forall\ i \in M$, $l \in L(i, \overline{\lambda}_i)$, $s \in S(i,l, \overline{\lambda}_i)$;

for ($i \in M$, $l \in L(i, \overline{\lambda}_i)$, $s \in S(i,l, \overline{\lambda}_i)$) {//for each index combination

if ($x_{ils}^{\text{IP}} = 1$){ // lot l is processed by machine i by route s

//update L_i and t_i^{IP}

$L_i \leftarrow L_i \cup \{l\}$; $t_i^{\text{IP}} = t_i^{\text{IP}} + n_l^{\text{chips}} / r_{ils}$;

}

}

for ($l \in L$) {//for each lot

if ($l \notin L_i$, $\forall\ i \in M$) {//assign lot l to the set of unassigned lots L_0

$L_0 \leftarrow L_0 \cup \{l\}$;

}

}

Step 2: Sort the lots $l \in L_0$ in nonincreasing order of $ben(l)$

Step 3: For ($l \in L_0$) {//for each unassigned lot

for ($i \in M$) {//for each machine

if ($l \in L(i, \bar{\lambda}_i)$){//test if lot l can be assigned to machine i using setup $\bar{\lambda}_i$

//get the route associated to the highest processing rate using tooling setup $\bar{\lambda}_i$

$s = \text{argmax}\{r_{ils}, s \in S(i,l,\bar{\lambda}_i)\}$;

//test if there is enough time to assign lot l to machine i

if ($t_i^{\text{IP}} + n_l^{\text{chips}} / r_{ils} \le H_i$){

//assign lot l to machine i, update L_i, L_0 and t_i^{IP}

$L_i \leftarrow L_i \cup \{l\}$; $t_i^{\text{IP}} = t_i^{\text{IP}} + n_l^{\text{chips}} / r_{ils}$; $L_0 \leftarrow L_0 \setminus \{l\}$; $x_{ils}^* = 1$;

//update output and shortage given new L_i and L_0

$out(d_l) = out(d_l) + n_l^{\text{chips}}$; $sh(d_l) = n(d_l) - th(d_l)$;

//update the lot benefit measure $ben(l)$, $l \in L_0$

for ($l' \in L_0$ and $d_{l'} = d_l$) {

$$ben(l) = w_l + \left(w_{d_l} / C\right) \cdot \min\left\{n_l^{\text{chips}}, sh(d_l)\right\} \cdot I_{\{sh(d_l)>0\}} \cdot I_{\{d_l \in \{K \cup P\}\}};$$

}

Re-sort the lots in L_0 given updated $ben(l)$, $l \in L_0$;

}

}

}

}

Figure A.73 Pseudocode for $N_1(x^{\text{IP}})$ local search

Procedure: $N_2(x^{\text{IP}}, \bar{\lambda}_i, x^*)$

Input: Feasible IP solution x^{IP}, current machine setup $\bar{\lambda}_i$, $\forall\, i \in M$

Output: Local optimum x^* in neighborhood $N_2(x^{\text{IP}})$

Step 1: //Initialize L_0, L_i , t_i^{IP} and x^* according to x^{IP}

$L_0 \leftarrow \emptyset$; $L_i \leftarrow \emptyset$; $t_i^{\text{IP}} = 0$, $\forall\, i \in M$;

$x_{ils}^* = x_{ils}^{\text{IP}}$, $\forall\, i \in M$, $l \in L(i, \bar{\lambda}_i)$, $s \in S(i,l,\bar{\lambda}_i)$;

for ($i \in M$, $l \in L(i, \bar{\lambda}_i)$, $s \in S(i,l,\bar{\lambda}_i)$) {//for each index combination

if ($x_{ils}^{\text{IP}} = 1$){ // lot l is processed by machine i by route s

//update L_i and t_i^{IP}

$L_i \leftarrow L_i \cup \{l\}$; $t_i^{\text{IP}} = t_i^{\text{IP}} + n_l^{\text{chips}} / r_{ils}$;

}

}

for ($l \in L$) {//for each lot

if ($l \notin L_i$, $\forall\, i \in M$) {//assign lot l to the set of unassigned lots L_0

$L_0 \leftarrow L_0 \cup \{l\}$;

}
}
Step 2: Sort the lots $l \in L_0$ in nonincreasing order of $ben(l)$
Step 3: for ($l \in L_0$) {//for each unassigned lot
for ($i \in M$) {//for each machine
if ($l \in L(i, \bar{\lambda}_i)$){ //test if lot l can be assigned to machine i using setup $\bar{\lambda}_i$
for ($l' \in L_i$){//for each lot assigned to machine i
//get the route associated to the highest processing rate using tooling setup $\bar{\lambda}_i$
$s = \text{argmax}\{r_{ils}, s \in S(i,l,\bar{\lambda}_i)\}$;
$s' = \text{argmax}\{r_{il's}, s \in S(i,l',\bar{\lambda}_i)\}$;
//test if there is enough time to process l if the swap is made
if ($t_i^{\text{IP}} + n_l^{\text{chips}} / r_{ils} - n_{l'}^{\text{chips}} / r_{il's} \le H_i$ and $ben(l) > ben(l')$){
//swap lots l and l', update sets L_0, L_i $\forall$ $i \in M$, t_i^{IP} and x^*
$L_i \leftarrow L_i \setminus \{l'\}$; $L_i \leftarrow L_i \cup \{l\}$;
$L_0 \leftarrow L_0 \setminus \{l\}$; $L_0 \leftarrow L_0 \cup \{l'\}$;
$t_i^{\text{IP}} = t_i^{\text{IP}} + n_l^{\text{chips}} / r_{ils} - n_{l'}^{\text{chips}} / r_{il's}$;
$x^*_{ils} = 1$; $x^*_{il's} = 0$;
//update output and shortage given updated L_i and L_0
$out(d_l) = out(d_l) + n_l^{\text{chips}}$; $sh(d_l) = n(d_l) - th(d_l)$;
//update the lot benefits $ben(l)$, $l \in L_0$
for ($l \in L$) {
$$ben(l) = w_l + \left(w_{d_l} / C\right) \cdot \min\left\{n_l^{\text{chips}}, sh(d_l)\right\} \cdot I_{\{sh(d_l)>0\}} \cdot I_{\{d_l \in \{K \cup P\}\}};$$
}
Re-sort the lots in L given new $ben(l)$, $l \in L$;
}
}
}
}
}
}

Figure A.74 Pseudocode for $N_2(x^{\text{IP}})$ local search

Procedure: KP_heur(j, λ, L_0, ben, L^*)
Input: SIM_j, tooling setup λ, set of unassigned lots L_0
Output: Benefit ben from setting up the machines in SIM_j with tooling λ; set of lots L^* assigned to (j, λ) combination

Step 1: Sort the lots in L_0 according to $ben(l)\Big/\left(n_l^{\text{chips}}/r_{ils}\right)$ in nonincreasing order;

Step 2: Pick one machine $i \in SIM_j$;
$t = 0$; $ben = 0$;
for ($l \in L_0$) {
$s = \text{argmax}\{r_{ils} : s \in S(i, l, \lambda)\}$; //choose the route with the highest processing rate
if ($t + n_l^{\text{chips}}/r_{ils} \le H_i$) {//test if there is enough time to process lot l
put $t \leftarrow t + n_l^{\text{chips}}/r_{ils}$; $ben \leftarrow ben + ben(l)$; $L^* \leftarrow L^* \cup \{l\}$;
}
}

Figure A.75 Pseudocode to compute the benefit associated with machine-tooling combination (j, λ)

Procedure: LPLB($\bar{x}$, $\bar{y}$, K, x^*, y^*)
Input: Current IP solution ($\bar{x}$, $\bar{y}$), neighborhood radius K
Output: Improved solution (y^*, x^*)
Step 1: Construct the local branch cut $\sum_{(i,\lambda)\in\bar{Y}}(1-y_{i\lambda}) \le K$;
Include the cut into model (1) and solve as LP and obtain solution y^{LP};
Step 2: for ($iter = 1, 2, \ldots, n^{\text{LPLB}}$) {
Simulate machine setups from y^{LP} and denote as y^s;
Solve LLP to get LLP_heur(y^s) and x^s;
if ($sim\_obj(x^s, y^s) > sim\_obj(\bar{x}, \bar{y})$) {
$x^* = x^s$; $y^* = y^s$;
}
}

Figure A.76 Pseudocode of LPLB

Procedure: Phase_I(L, M, T, Λ, S^{PhaseI})
Input: Set of lots L, set of machine M, set of tooling families T, and set of tooling setups Λ
Output: Set of initial solutions S^{PhaseI}
Step 1: //initialization
Construct *SIM* from M;
Initialize SL;
$S^{\text{PhaseI}} \leftarrow \emptyset$;
Step 2: for ($k = 1, 2, \ldots, n^{\text{PhaseI}}$) {
Put $L_0 \leftarrow L$; $L_i \leftarrow \emptyset$, $\forall\, i \in M$;
Compute $ben(l)$, $\forall\, l \in L$;
Sort the lots in L_0 according to $ben(l)$ in nonincreasing order;
while (some machine $i \in M$ is idle and sufficient tooling $t \in T$ is available){
//construct CL

```
                for (all feasible (j, λ) combinations) {
                    KP_heur(j, λ, L0, b, L*);
                    Append the triplet (j, λ, b) to CL
                }
                Sort CL according to benefit b;
                Construct RCL;
                Assign probability to the elements in RCL according to S_iλ in SL;
                Select one RCL element randomly: (j*, λ*, b*);
                //perform the machine tooling setup
                Find an available machine i ∈ SIM_j*
                Set y_iλ* = 1; y_iλ = 0, ∀ λ ∈ Λ(i) \ {λ*};
                KP_heur(j*, λ*, L0, b, L*);
                Put L_i ← L*; L0 ← L0 \ L*;
                Update SL;
                Update ben(l), l ∈ L0;
                Update machine and tooling usage;
            }
            //given the machine setups y*,
            Solve LLP to get LLP_heur(y*) and x*;
            Put S^PhaseI ← S^PhaseI ∪ {(x*, y*)};
        }
```

Figure A.77 Pseudocode of GRASP phase I

```
Procedure: GRASP(L, M, T, Λ, x*, y*)
Input:  Set of lots L, set of machines M, set of tooling families T, and set of tooling
        setups Λ
Output: Best solution found to model (1), (x*, y*)
Step 1: Phase_I(L, M, T, Λ, S^PhaseI);
        Select a subset of top elements in S^PhaseI; denote as S^sub;
Step 2: Denote the jth element of S^sub as (x^j, y^j);
        //initialize (x*, y*)
        x* = x^1; y* = y^1;
        for (j = 1, 2,…,|S^sub|) {
            LPLB(x^j, y^j, K, x^j*, y^j*);
            if (sim_obj(x^j*, y^j*) > sim_obj(x*, y*)) {
                x* = x^j*; y* = y^j*;
            }
        }
```

Figure A.78 Pseudocode of GRASP

```
Procedure: Random_Case_Generator()
Step 1: (Parameters) Let L^test = set of lots, M^test = set of machines, m_g = number of
        machine groups, t^test = number of tooling pieces, T^test = set of tooling families, n_D
```

= number of devices, n_K = number of key devices, n_P = number of package devices, n_{temp} = number of operating temperatures, S^{test} = set of routes, and Λ^{test} = set of tooling setups.

Step 2: (Lot generation) Generate the lot sizes uniformly in the range $[n_1, n_2]$, where n_1 is the minimum lot size and n_2 is the maximum lot size according to the original dataset. Generate the lot weights uniformly in the range $[w_{min}, w_{max}]$. For each lot $l \in L^{test}$, randomly select one of the n_D devices to be the device contained in the lot.

Step 3: (Machine generation) For each machine $i \in M^{test}$, randomly select one of the m_g machine groups, select one or more operating temperatures from the n_{temp} operating temperatures.

Step 4: (Tooling generation) For each tooling piece $t \in \{1, 2,\ldots, t^{test}\}$, randomly select one of the $|T^{test}|$ tooling families, select one or more operating temperatures from the n_{temp} operating temperatures.

Step 5: (Route generation) Let $S^{test} \leftarrow \emptyset$. For a device *d*, randomly select one or more machine groups. For each device-machine group combination, a new route *s* is generated and $S^{test} \leftarrow S^{test} \cup \{s\}$. The corresponding processing rates are uniformly generated from [15000, 150000] in the units of parts per hour (PPH). Go to the next device.

Step 6: (Tooling setup generation) Let $\eta = |S^{test}|/|\Lambda^{test}|$ which is approximately 0.5 in the original dataset. Set $|\Lambda^{test}| = \eta \cdot |S^{test}|$. For each setup $\lambda \in \{1, 2,\ldots,|\Lambda^{test}|\}$, randomly select one of the n_{temp} temperatures and one or more tooling families, each with probability $(1/|T^{test}|)$. Then draw at random the number of tooling pieces from {1, 2, 3, 4} for each family realized.

Step 7: (Link S^{test} to Λ^{test}) For each route $s \in S^{test}$, pick one of the $|\Lambda^{test}|$ setups at random.

Step 8: (Device generation) Randomly select n_K devices to be key devices and another n_P devices to be package devices. For each key or package device, the minimum output is randomly chosen from $[n_{min}, n_{max}]$,where n_{min} is the minimum target output and n_{max} is the maximum target output in the original dataset.

Figure A.79 Pseudocode of random cases generator

Bibliography

Ahmadi, S., Osman, I.H. (2005). Greedy random adaptive memory programming search for the capacitated clustering problem. European Journal of Operational Research, 162(1), 30-44.

Asmundsson, J., Rardin, R.L., Uzsoy, R. (2006). Tractable nonlinear production planning models for semiconductor wafer Fabrication Facilities. IEEE Transactions on Semiconductor Manufacturing, 19(1), 95-111.

Bai, X., Gershwin, S.B. (1994). Scheduling manufacturing systems with work-in-process inventory control: multiple-part-type systems. International Journal of Production Research, 32(2), 365-385.

Bard, J.F., Jarrah, A.I. (2009). Large-scale constrained clustering for rationalizing pickup and delivery operations. Transportation Research Part B: Methodological, 43(5), 542-561.

Bard, J.F., Purnomo, H.W. (2005). Preference scheduling for nurses using column generation. European Journal of Operational Research, 164(2), 510-534.

Bard, J.F., Purnomo, H.W. (2007). Cyclic preference scheduling of nurses using a Lagrangian-based heuristic. Journal of Scheduling, 10(1), 5-23.

Bard, J.F., Rojanasoonthon, S. (2006). A branch & price algorithm for parallel machine scheduling with time windows and job priorities. Naval Research Logistics, 53(1) 24-44.

Barreto, S., Ferreira, C., Paixao, J., Santos, B.S. (2006). Using clustering analysis in a capacitated location-routing problem. European Journal of Operational Research, 179, 968-977.

Bertsimas, D., Sethuraman, J. (2002). From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective. Mathematical Programming, Series A, 92, 61-102.

Bertsimas, D., Gamarnik, D., Sethuraman, J. (2003). From fluid relaxations to practical algorithms for high-multiplicity job-shop scheduling: the holding cost objective. Operations Research, 51(5), 798-813.

Boudia, M., Louly, M.A.O., Prins, C. (2006). A reactive GRASP and path relinking for a combined production-distribution problem. Computers & Operations Research, 34(11), 3402-3419.

Brucker, J. (1978). On the complexity of clustering problem. Lecture Notes in Economics and Mathematical Systems, 157, 45-54, Springer, Berlin / Heidelberg.

Boschetti, M., Maniezzo, V. (2009). Benders decomposition, Lagrangian relaxation and metaheuristic design. Journal of Heuristics, 15(3), 283-312.

Cano, J.R., Cardon, O., Herrera, F., Sanchez, L. (2002). A greedy randomized adaptive search procedure applied to the clustering problem as an initialization process using *k*-means as a local search procedure. Journal of Intelligent & Fuzzy Systems, 12, 235-242.

Carey, M.R., Johnson, D.S. (1979). Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, New York.

Chacon, G.R., Ballego, F., James, S. (2005). Manufacturing operations improvement with loop management. Technical report no. 231, Texas Instruments, Dallas, Texas.

Chiang, T.C., Shen, Y.S., Fu, L.C. (2008a). A new paradigm for rule-based scheduling in the wafer probe centre. International Journal of Production Research, 46(15), 4111-4133.

Chiang, D.M., Guo, R.-S., Pai, F.-Y. (2008b). Improved customer satisfaction with a hybrid dispatching rule in semiconductor back-end factories. International Journal of Production Research, 46(17), 4903-4923.

Chiou,Y.-C., Lan. L.W. (2001). Genetic clustering algorithms. European Journal of Operational Research, 135(2), 413-427.

Daganzo, C.F. (2005). Logistics System Analysis, 4th Edition, Springer, Berlin.

Dai, J.G. (1995). On positive Harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. Annals of Applied Probability, 5(1), 49-77.

Dai, J.G., Weiss, G. (2002). A fluid heuristic for minimizing makespan in job-shops. Operations Research, 50(4), 692-707.

Feo, T.A., Resende, M.G.C. (1995). Greedy randomized adaptive search procedures. Journal of Global Optimization, 6, 109-134.

Ferreira, C.E., Martin, A., de Souza, C.C., Weismantel, R., Wolsey, L.A. (1998). The node capacitated graph partitioning problem: A computational study. Mathematical Programming, 81(2), 229-256.

Fischetti, M., Lodi, A. (2003). Local branching. Mathematical Programming, 98(1) 23-47.

Fischetti, M., Luzzi, I. (2009). Mixed-integer programming models for nesting problems. Journal of Heuristics, 15(3), 201-226.

Fordyce, K., Dalton, D., Gerard, B., Jesse, R., Sullivan, G. (1992). Daily output planning: integrating operations research, artificial intelligence, and real-time decision support with APL2. Expert Systems With Applications, 5, 245-256.

Freed, T., Doerr, K.H., Chang, T. (2006). In-house development of scheduling decision support systems: case study for scheduling semiconductor device test operations. International Journal of Production Research, 45(21), 5075-5093.

Frank, A. (1994). On the edge-connectivity algorithm of Nagamochi and Ibaraki. Working paper, ARTEMIS-IMAG, Université de Grenoble, Grenoble, France.

Glassey, C.R., Resende, M.G.C. (1988). Closed-loop job release for VLSI circuit manufacturing. IEEE Transactions on Semiconductor Manufacturing, 1(1), 36-46.

Glover, F., Laguna, M., Marti, R. (2000). Fundamentals of scatter search and path relinking. Control and Cybernetics, 29(3), 653-684.

Johnson, E.L., Mehrotra, A., Nemhauser, G.L. (1993). Min-cut clustering. Mathematical Programming, 62(1), 133-151.

Hansen, P., Mladenovic, N. (1997). Variable neighborhood search. Computers & Operations Research, 24(11), 1097-1100.

Hansen, P., Mladenovic, N. (2001). J-means: a new local search heuristic for minimum sum of squares clustering. Pattern Recognition, 34(2), 405-413.

Hu, B., Leitner, M., Raidl, G.R. (2008). Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. Journal of Heuristics, 14(5), 473-499.

Hughes, R.A., Shott, J.D. (1986). The future of automation for high-volume wafer fabrication and ASIC manufacturing. Proceedings of the IEEE, 74(12), 1775-1793.

Kaufman, L., Roussweuw, P. (1990). Finding Groups in Data: An Introductory to Cluster Analysis, Wiley, New York.

Karger, D.R., Stein, C. (1996). A new approach to the minimum cut problem. Journal of ACM, 43(4), 601-640.

Knutson, K., Kempf, K., Fowler, J.W., Carlyle, M. (1999). Lot-to-order matching for a semiconductor assembly and test facility. IIE Transactions on Scheduling & Logistics, 31(11), 1103-1111.

Kontoravdis, G., Bard, J.F. (1995). A GRASP for the vehicle routing problem with time windows. ORSA Journal on Computing, 7(1), 10-23.

Laporte, S., Chapleau, S., Landry, P.-E., Mercure, H. (1989). An algorithm for the design of mailbox collection routes in urban areas. Transportation Research Part B: Methodological, 23(4), 271-280.

Leachman, R.C. (2002). Application of mathematical optimization to semiconductor production planning. In Resende, M., Pardolos, P. (eds.), Handbook of Applied Optimization. Oxford University Press, New York, 746-762.

Leachman, R.C., Carmon (Freed), T. (1992). On capacity modeling for production planning with alternative machine types. IIE Transactions on Scheduling & Logistics, 24(4), 62-72.

Leachman, R.C., Jeenyoung, K., Lin, V. (2002). SLIM: short cycle time and low inventory in manufacturing at Samsung electronics. Interfaces, 32(1), 61-77.

Lee, Y.H., Kim, T. (2002). Manufacturing cycle time reduction using balance control in the semi-conductor fabrication line. Production Planning & Control, 13:529-540.

Lin, V. (1999). Advanced semiconductor production planning. Doctoral dissertation, Department of Industrial Engineering and Operations Research, University of California, Berkeley.

Lorena, L.A.N., Senne, E.L.F. (2004). A column generation approach to capacitated *p*-median problems. Computers & Operations Research, 31(6), 863-876.

Lozinski, C., Glassey, C.R. (1988). Bottleneck starvation indicators for shop floor control. IEEE Transactions on Semiconductor Manufacturing, 1(4), 147-153.

Mehrotra, A., Trick, M.A. (1998). Cliques and clustering: a combinatorial approach. Operations Research Letters, 22(1), 1-12.

Mulvey, J.M., Beck, M.P. (1984). Solving capacitated clustering problems. European Journal of Operational Research, 18(3), 339-48.

Nagmochi, H., Ibaraki, T. (1992). Computing edge-connectivity in multigraphs and capacitated graphs, SIAM Journal of Discrete Mathematics, 5(1), 54-66.

Narahari, Y., Khan, K. (1996). Performance analysis of scheduling policies in re-entrant manufacturing systems. Computers & Operations Research, 23(1), 37-57.

Negreiros, M., Palhano, A. (2006). The capacitated centered clustering problem. Computers & Operations Research, 33(6), 1639-1663.

Newell, G.F., Daganzo, C.F. (1986). Design of multiple-vehicle delivery tours – I: A ring-radial network. Transportation Research Part B: Methodological, 20B(5), 345-363.

Osman, I.H., Ahmadi, S. (2007). Guided construction search metaheuristics for the capacitated *p*-median problem with single source constraint. Journal of the Operational Research Society, 58(1), 100-114.

Ouyang, Y. (2007). Design of vehicle routing zones for large-scale distribution systems. Transportation Research Part B: Methodological, 41(10), 1079-1093.

Ovacik, I.M., Uzsoy, R. (1996). Decomposition methods for scheduling semiconductor testing facilities. International Journal of Flexible Manufacturing Systems, 8, 357-388.

Prais, M., Ribeiro, C.C. (1999). Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing, 12(3), 164-176.

Pfund, M.E., Mason, S.J., Fowler, J.W. (2006). Semiconductor manufacturing scheduling and dispatching. In Herrmann, J.W. (ed.), Handbook of Scheduling, Chapter 9, pp. 213-241, Springer, Berlin.

Pinedo, M.L. (2008). Scheduling Theory, Algorithms, and Systems, Third Edition, Springer, New York.

Purnomo, H.W., Bard, J.F. (2007). Cyclic preference scheduling for nurses using branch and price. Naval Research Logistics, 54(2), 200-220.

Rojanasoonthon, S., Bard, J.F. (2005). A GRASP for parallel machine scheduling with time windows. INFORMS Journal on Computing, 17(1), 32-51.

Rivera, D.E., Vargas-Villami, F.D., Kampf, K.G. (2003). A hierarchical approach to production control of reentrant semiconductor manufacturing lines. IEEE Transaction on Control Systems Technology, 11(4), 578-587.

Robinson, J.K., Fowler, J.W., Bard, J.F. (1995). The use of upstream and downstream information in scheduling semiconductor batch operations. International Journal of Production Research, 33(7), 1849-1869.

Rybko, A.N., Stolyar, A.L. (1992). Ergodicity of stochastic processes describing the operations of open queueing networks. Problems of Information Transmission, 28, 199-220.

Saito, K. (2007). A robust dispatching algorithm for autonomous distributed manufacturing of mixed VLSI products. IEEE Transactions on Semiconductor Manufacturing, 20(3), 299-305.

Song, Y., Zhang, M.T., Yi, J., Zhang, L., Zheng, L. (2007). Bottleneck station scheduling in semiconductor assembly and test manufacturing using ant colony optimization. IEEE Transactions on Automation Science and Engineering, 4(4), 569-578.

Stray, J., Fowler, J.W., Carlyle, W.M., Rastogi, A.P. (2006). Enterprise-wide semiconductor manufacturing resource planning. IEEE Transactions on Semiconductor Manufacturing, 19(2), 259-268.

Tu, Y.M., Chao, Y.H., Chang, S.H., You, H.C. (2005). Model to determine the backup capacity of a wafer foundry. International Journal of Production Research, 43, 339-359.

Uzsoy, R., Lee, C.-Y., Martin-Vega, L.A. (1992). A review of production planning and scheduling models in the semiconductor industry; part I: system characteristics, performance evaluation and production planning. IIE Transaction on Scheduling & Logistics, 24(4), 47-60.

Van Zant, P. (2000). Microchip Fabrication: A Practical Guide to Semiconductor Processing, 4th Edition, McGraw-Hill, NY.

Wein, L.M. (1988). Scheduling semiconductor wafer fabrication. IEEE Transactions on Semiconductor Manufacturing, 1(3), 115-130.

Weiss, G. (1995). On optimal draining of fluid reentrant lines. In Kelly, F.P. and Williams, R.J., editors, Stochastic Networks, volume 71, The IMA volumes in mathematics and its applications, Springer-Verlag, New York, 93-105.

Wolsey, L.A. (1998). Integer Programming, John Wiley & Sons, New York.

Yavuz, M., Akcali, E. (2007). Production smoothing in just-in-time manufacturing systems: a review of the models and solution approaches. International Journal of Production Research, 45(16), 3579-3597.

Zäpfel, G., Missbauer, H. (1993). Production planning and control (PPC) systems including load-oriented order release – problems and research perspectives. International Journal of Production Economics, 30, 107-122.

Zhang, M.T., Niu, S., Deng, S., Zhang, Z., Li, Q., Zheng, L. (2007). Hierarchical capacity planning with reconfigurable kits in global semiconductor assembly and test manufacturing. IEEE Transactions on Automation Science and Engineering, 4(4), 543-552.

VITA

Yumin Deng was born in Huizhou, Guangdong province of P. R. China on August 19, 1980, the son of Liquan Deng and Lizhen Lin. After finishing his work in No.1 High School of Huizhou, he entered Shanghai Jiaotong University in September 1999 for his undergraduate study with a focus on Naval Architecture and Ocean Engineering. He received his Bachelor of Engineering from Shanghai Jiaotong University in July 2003. Right after his undergraduate graduation he was admitted to The University of Texas at Austin as a master student in Civil Engineering. His work was focused on the optimal propeller design during his master research. He received his Master of Science degree in December 2005.

In January 2006, he started his doctoral study in Operations Research & Industrial Engineering, The University of Texas at Austin.

Permanent Address: APT 403, No. 15, Lane 1, Jiangbian Rd.
Xiajiao District, Huizhou, Guangdong, P. R. China, 516001

This manuscript was typed by the author.