Fall 2022

Heuristics For Capacity Allocation And Queue Assignment In Congested Service Systems With Stochastic Customer Demand And Immobile Servers

Adam Colley
Southern Methodist University, acolley@smu.edu

HEURISTICS FOR CAPACITY ALLOCATION AND QUEUE ASSIGNMENT IN CONGESTED SERVICE SYSTEMS WITH STOCHASTIC CUSTOMER DEMAND AND IMMOBILE SERVERS

Approved by:

Dr. Eli Olinick

Dr. Sila Cetinkaya

Dr. Richard Barr

Dr. Gheorghe Spiride

Dr. Michael Hahsler

HEURISTICS FOR CAPACITY ALLOCATION AND QUEUE ASSIGNMENT IN CONGESTED SERVICE SYSTEMS WITH STOCHASTIC CUSTOMER DEMAND AND IMMOBILE SERVERS

A Dissertation Presented to the Graduate Faculty of the

Lyle School of Eningeering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Operations Research

by

Adam Q. Colley

(B.S., DeVry University, 2009)
(M.B.A., Keller Graduate School of Management, 2010)

December 17, 2022

ACKNOWLEDGMENTS

I would like to thank the Department of Engineering Management, Information, and Systems of Southern Methodist University.

Colley , Adam Q. B.S., DeVry University, 2009
M.B.A., Keller Graduate School of Management, 2010

Heuristics for Capacity Allocation and Queue Assignment in Congested Service Systems with Stochastic Customer Demand and Immobile Servers

Advisor: Professor Eli Olinick
Doctor of Philosophy degree conferred December 17, 2022
Dissertation completed October 31, 2022

We propose easy-to-implement heuristics for a problem referred to in the literature as the facility location problem with immobile servers, stochastic demand, and congestion, or the service system design problem. The problem is typically posed as one of allocating capacity to a set of $M/M/1$ queues to which customers with stochastic demand are assigned with the objective of minimizing a cost function composed of a fixed capacity-acquisition cost, a variable customer-assignment cost, and an expected-waiting-time cost. The expected-waiting-time cost results in a non-linear term in the objective function of the standard binary programming formulation of the problem. Thus, the solution approaches proposed in the literature are either sophisticated linearization or relaxation schemes, or metaheuristics. In this study we demonstrate that an ensemble of straight-forward, greedy heuristics can find high-quality solutions in less than one second of CPU time. In addition to filling the gap in the literature using the heuristic, a noniterating linear model was also developed and the existing iterating linear model was tested using various stopping criteria while also examining the affects of additional constraints on the problem similar to issues being discussed in current literature. In many cases, our heuristic approach finds solutions of the same or better quality than those found with expensive, state-of-the art mathematical programming software, in particular a commercial non-linear MIP solver given a

five-minute time limit.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I dedicate this dissertation to Heather L. Colley for sticking by me during my schooling and encouraging me throughout.

Chapter 1

INTRODUCTION

There are many circumstances in which servers must be assigned to a queue and cannot be reassigned to a different queue without great cost (if at all). These are called immobile servers. For the immobile server problem, the queues are called facilities and the number of immobile servers assigned to those facilities drives the capacity for that facility. If customers are assigned to a specific facility—and cannot be reassigned, the problem is greatly simplified as only capacities need to determined. However, the basic immobile server problem not only includes capacity assignment, but also includes customer assignment to each facility.

In the literature, immobile server problem (ISP) is typically posed as allocating capacity to a set of $M/M/1$ queues to which customers with stochastic demand are assigned with the objective of minimizing a cost function composed of a fixed capacity-acquisition cost, a variable customer-assignment cost, and an expected-waiting-time cost. The expected-waiting-time cost results in a non-linear term in the objective function of the standard binary programming formulation of the problem. This dissertation proposes a novel linearization of the objective function and fast, easy-to-implement heuristics.

1.1. Scope

This dissertation examines exact solution of the ISP by solving the base nonlinear mixed-integer program (NMIP), the iterating linearized mixed-integer program (MIP) developed by Elhedhli in 2006 [22], and a noniterating linear mixed-integer program

developed as part of this research. It then examines what properties of the data set make for an easier or more difficult problem to solve using those models along with adding additional constraints on the maximum number of open facilities and/or the maximum system capacity. Finally, it explores a heuristic which can be used to significantly shorten the time needed, the situations in which it would be useful, and the setup of the data—if possible—to give it the highest chance of finding the optimal solution.

In order to analyze the efficiency and accuracy of each model, 695 data sets were tested. These data sets can be grouped in six major categories: Holmberg (30 data sets pulled from Holmberg 1999 [28]), small with varying costs (36 data sets manually designed to identify difficulty), various sizes with linear costs (54 data sets pulled from six actual clients using a Taguchi L9 Orthogonal Array approach to the design of experiments), various sizes with nonlinear costs (54 data sets pulled from the same six actual clients with the costs restructured varying the customer grouping and customer wait-time cost), sizing (512 data sets manually designed for testing increasing numbers of customers, facilties, and capacity levels up to 10,000 customers, 100 facilities, and 20 capacity levels), and saw-tooth (nine extra small data sets manaully created for brute-force analysis within Microsoft Excel).

The actual data from six clients were from performance tuning work manually performed on their system. Their data was scrubbed of any identifying information and kept for research purposes. The system used by the clients is IBM Sterling B2B Integrator (B2Bi [1]) and allows for business processes (customers in the immobile server problem) to be placed into priority queues (facilities in the immobile server problem) which are assigned a specific number of threads (capacity level in the immobile server problem). The six clients ranged from small to large and with varying business process needs so that any heuristic capable of being integrated with B2Bi

would satisfy a wide range of client needs.

1.2. Literature Review

Motivated by B2Bi, we propose easy-to-implement heuristics for a problem referred to in the literature as the facility location problem with immobile servers, stochastic demand, and congestion [22], or the service system design problem [7, 8, 9], or immobile server problem (ISP). The problem is typically posed as one of allocating capacity to a set of $M/M/1$ queues to which customers with stochastic demand are assigned with the objective of minimizing a cost function composed of a fixed capacity-acquisition cost, a variable customer-assignment cost, and an expected-waiting-time cost. Although our study of the problem is motivated by an application in telecommunications, the problem may arise in other settings as well. For example, Armiri [7] notes applications to warehouse location [10], and waste collection and disposal (e.g., [5] and [41]). Motivated by manufacturing applications, Rajagopalan and Yu [40] develop a similar model and apply it to a chemical testing facility; in this case, the problem is to assign samples to machines of varying age, condition, and capability for gas chromatography analysis.

By constraining the solution space so that customers are always assigned to the nearest facility (i.e., in a way that minimizes the assignment cost), Wang et al. [46] develop heuristic solution procedures that are tested on bank-location data. Li et al. [30] apply a similar model to determine optimal placement of proxy websites. Castillo et al. [17] list applications such as motor-vehicle-inspection stations and walk-in health clinics as problem instances in which customers typically, but not necessarily, choose the closest service facility by extending the standard model to include a probability distribution for each customer's service-facility choice. Marianov and Serra [32] consider related coverage models in which the goal is to minimize the

number of service facilities such that each customer is assigned to a single facility subject to probabilistic quality-of-service constraints (e.g., an upper bound constraint on the probability of a customer spending more than a given amount of time waiting in the queue). Survey papers by [13] and Boffey et al. [14] catalog numerous other variations on the standard ISP such as models/applications with finite queues.

Since ISP is $\mathcal{NP}$-hard [5, 7], heuristic solution procedures are common in the literature. Citing the computational complexity for the standard problem, Armiri [7] proposes two heuristics solution procedures based on Lagrangean relaxation, and demonstrates through a computational study that they can find high-quality solutions (less than a 1% optimality gap) within reasonable solution times for a design problem (e.g., problem instances with 500 customers, 30 candidate service-facility locations, and five capacity levels are solved within 90 minutes of CPU time).

For the standard ISP, Elhedhli [22] proposes a linearization of the cost function using piecewise-linear approximations and a cutting-plane algorithm for solving the resulting mixed integer linear program. The cutting-plane algorithm is shown to converge in a finite number iterations and produce a solution that is optimal for the original, non-linear problem. In an empirical demonstration, this approach finds exact solutions to problem instances with as many as 100 customers, 20 candidate service-facility locations, and three levels of capacity.

As far as we can tell, research on the ISP after [22] has focused on variations and extensions of the basic model. The following is a representative survey of two streams of this work: exact methods and metaheuristics.

1.2.1. Exact Methods

Vidyarthi and Jayaswal [45] apply a similar approach to the one in [22] to find solutions within a given optimality tolerance to the ISP with $M/G/1$ queues, and apply their methodology to problem instances with as many as 400 customers, 25

candidate service locations, five levels of capacity and various ranges for waiting-time costs and ratios between the mean and standard deviation of the service time.

Elhedli et al. [23] study an extension of the ISP in which server capacity is a continuous variable and the installation cost is proportional to the square root of the capacity. They propose solution approaches based on a piecewise-linear approximation of the non-linear terms in the objective function, and a cutting plane method based on Lagrangian relaxation and second-order cone programming. These approaches are demonstrated to find high quality solutions for instances with up to 25 facilities and 100 customers. Hoseinpour [29] generalizes the model in [23] to any non-decreasing concave function. Stiermaier [42] considers a budget for the number servers assigned, but the objective function does not have the capacity cost term.

The B2Bi use case fits the standard model in which all customers must be served. Other authors have investigated use cases that make a trade-off between cost and potential loss of customers who have limited patience for waiting in line. Aboolian et al. [4] consider a profit-maximizing variation of the ISP in which customer demand varies inversely with waiting time. In addition to the standard model with $M/M/1$ queues and discrete capacity levels, they also propose a model with a fixed service rate and $M/M/k$ queues (i.e., the number of servers at each facility is a decision variable). Boffey et al. [15] also consider lost demand and study an $M/E_r/n/N$ model with service-level constraints limiting the amount of lost traffic. In a related study [31], the authors consider an $M/E_r/n/N$ model in which given service-level requirements are enforced by constraints on server utilization.

1.2.2. Metaheuristics

Pasandideh and Chambari [36] and Chambari et al. [18] adapt standard genetic algorithms (GAs) from the literature, non-dominated ranked genetic algorithm for solving multi-objective optimization problems (NRGA) [6] and the fast elitist non-

dominated sorting genetic algorithm for multi-objective optimization (NSGA-II) [20], to a bi-objective variation of the ISP with finite queues ($M/M/1/K$) and a budget constraint on the number of service facilities used. The two objectives in the model are to minimize the time customers spend traveling to the facilities and waiting for service, and to minimize the servers' idle time. To specialize the GAs for ISP, the authors introduce a scheme for encoding an ISP solution as a vector (chromosome) in a way that implements the combination of two parent solutions to produce an offspring solution (crossover) as a linear combination of the parent vectors, and a simple mechanism for random mutation of chromosome's for individuals in the population. The initial population of solutions are essentially random assignments of customers to facilities and random allocations capacity to servers. Thus, there is no guarantee that any individual in the initial population represents a feasible solution to the ISP. Likewise, the offspring produced by pairing two members of the population might be infeasible even if both of its parents are feasible. Therefore, the fitness functions in the GAs have penalty terms for infeasibility so that the natural selection process modeled by the GA favors feasibility. The authors report results from test instances with 6 to 22 customers and 5 to 20 potential service facilities. The GAs were implemented in Matlab on a Pentium 1860 processor with one GB RAM. The reported running times range from 223.23 to 479.49 CPU seconds with an average CPU time of 378.94 seconds.

Pasandideh and Niaki [37] present a GA for a similar bi-objective variant of ISP with a budget for the number of facilities selected. In this case, all servers have the same fixed capacity; this allows the GA to encode solutions as binary matrices in which an entry of 1 in row i and column j indicates the assignment of customer i to facility j as well as the selection of facility j. Using binary matrices for chromosomes also allows for a more straightforward crossover mechanism compared to the one in

[18, 36]. The GA uses a desirability function [21] to address the trade-off between the two objectives and penalties for violating constraints; thus, the optimization problem becomes an unconstrained problem with a single objective function. As in [36, 18], an iteration count is used as the stopping criterion for the GA. The GA is evaluated on a set of 12 problem instances in which the number of customers and potential facilities used range from 3 to 150 and 4 to 65, respectively. The authors report running times of less than one minute to 40 minutes of CPU time with an average of 5.17 minutes for their GA. They also implemented a mathematical programming formulation of the problem in LINGO. Using LINGO, the solution times ranged from less than one minute to over eight hours of CPU time with an average of 80 minutes. The LINGO model was unable to solve the largest problem in the data set, which has 150 customers and 65 potential facilities of which at most 15 may be selected. As measured by the desirability function, which ranges from zero to one, the difference between the objective function values of the solution returned by LINGO and the best solution found by the GA ranged from -0.0446 to 0.0002.

Rahmati et al. [39] further extend the model by including a third objective, cost, and treating each service facility as an $M/M/s$ queue. They propose a multi-objective version of the harmony search metaheuristic (MOHS) framework [24] and test it against GA's based on NRGA and NSGA-II and on a set of 20 problem instances with up to 3,500 customers, and 700 servers distributed among 1,100 potential facility locations. Like a genetic algorithm, MOHS uses randomization to generate a population of solutions and to combine and/or modify those solutions to generate new members of the population, and a fitness function to guide the inclusion and exclusion of solutions from generation of the population to the next. The algorithms in [39] were implemented in MATLAB on a laptop with a 2GHz CPU and 8 GB of RAM. The solution times using MOHS, NRGA, and NSGA-II ranged from 19.34 to

92.94 seconds of CPU time with a mean of 43.86 seconds, 37.92 to 183.93 seconds with a mean of 87.61 seconds, 22.34 to 117.74 seconds of with a mean of 62.82 seconds, respectively. The differences in solution times were found to be statistically significant at the 95% confidence level. The MOHS algorithm also outperformed the two genetic algorithms in measures of the quality of the Pareto frontier (e.g., diversity and number of pareto-optimal solutions).

Hajipour et al. [27] extend the model even further by introducing an additional budget on capacity. They compare genetic and harmony search algorithms simliar to those proposed by Rahmati et al. [39], and a multi-objective simulated annealing algorithm. As in [39], the algorithms in were implemented in MATLAB on a laptop with a 2GHz CPU and 8 GB of RAM, and test on a set of 25 problem instances with up to 5,500 customers and 1,800 facilities. The harmony search algorithm solved all problem instances within a minute of CPU time

Akrat and Jafari [12] address a ISP variant in which the selected facilities have the same service rate and customers are assigned to the nearest facility. The objective is to minimize the total travel and waiting time subject to a constraint on the average waiting time at any server. They develop an integer programming model, a simulated annealing heuristic, and a genetic algorithm for the problem. The solution approaches are tested on problems with 10 to 25 customers and 3 to 15 facilities.

Zamani et al. [47] consider the possibility of service interruptions (e.g., ATM malfunctions) to the ISP. The timing and duration of service interruptions are an additional source of uncertainty in the system. The authors propose a genetic algorithm and an ant lion heuristic [33] to solve this complex variation of the fundamental problem, as well as a non-linear mixed integer program that they attempt to solve with Baron. In their numerical testing, the allow Baron up to three hours of CPU time to solve a given problem instance. Testing instances with 50 to 200 customers

and 10 to 30 facilities, they find that Baron runs for the full three hours in most cases (61 out of 71) without finding a provably optimal solution. The metaheuristics require at most several minutes to solve each these problem instances and produce solutions with comparable quality to the exact approach. Interestingly, this is one of only two works cited herein in which the authors report attempts to validate their proposed solution approaches with a non-linear mixed integer programming solver.

The heuristics for the ISP surveyed in this section are fundamentally different than the one developed in this dissertation. Once the representation of the ISP solution as a chromosome is determined, the focus in this stream of research is on tuning the parameters of the GA or HS framework. This is an important task in the making algorithm effective, but it doesn't involve or exploit the structure of the ISP. Another important difference is that most of the algorithms discussed in this section are designed for finding pareto-optimal solutions to multi-objective variations of the ISP. So, they propose schemes for combining the objectives (with a penalty for infeasibility) into a single fitness or desirability measure that ranges from zero to one. The ISP considered in this dissertation has a single cost function in which the user-defined trade-offs between competing objectives are specified in the data.

1.2.3. Math-Programming-Based Heuristics

Some of the papers that develop exact solution procedures for the ISP also present heuristics that are guided by the optimization process. For example, Agnihothri et al. [5] describe a heuristic that attempts to derive a feasible solution from each iteration of their Lagrangian relaxation procedure. In the procedure, the constraints ensuring that every customer is assigned to exactly one facility are relaxed in the lower-bound problem. The heuristic attempts to find a feasible solution to the ISP from the solution to the lower-bound problem by assigning each customer that has either not been assigned to a facility or has been assigned to multiple facilities to

the open facility that minimizes a particular cost function. If this yields a feasible assignment, then the heuristic attempts to improve the solution by checking to see if there are any customers whose assignment could be changed to another facility that has sufficient capacity in a way that reduces the total cost. Amiri [7, 8, 9] describes a similar heuristic for attempting to construct a primal solution at each step of a Lagrangian relaxation procedure.

Stiermaier [42] proposes metaheuristics for a special case of the ISP proposed by Aboolian et al. [3] in which customers are always assigned to the nearest facility. The initial solution for each of the metaheuristics is produced by the "descent" heuristic developed in [3]. The descent heuristics applies an algorithm that optimally assigns customers to a given subset of facilities, $S \subset M$. The algorithm then applies local search to a neighborhood of S defined by operations on the set (e.g., adding an element from $M \setminus S$ to S).

1.2.4. Greedy Heuristics

Some of the math-programming-based heuristics use greedy constructive procedures within a more complex algorithm (e.g., Amiri [7, 8, 9]), but we found only two examples of stand-alone greedy heuristics for ISP in the literature. Wang et al. [46] propose a greedy dropping-heuristic (GD) that starts with all service facilities open and sequentially closes a given number of facilities. The GD heuristic starts will all facilities open and assigns each customer to its preferred (facility); the data are such that this solution is always feasible. Next, the GD heuristic closes one facility at a time, reassigns the affected customers, and computes the solution cost (∞ if the solution is not feasible). Whichever of the new set of feasible solutions is feasible becomes the incumbent and the process continues until a local minimum is found. The authors note that the GD heuristic is adapted from an algorithm typically used in for the uncapacitated facility location (UFL) problem. To improve solution quality,

the GD heuristic is embedded in a Tabu Search [25] (TS) framework. Both heuristics are compared to a Lagrangian relaxation scheme, which is shown to converge to solutions with a given optimality tolerance, in a computation study on problem instances with up to 459 costumers, 84 candidate service-facility locations, and three capacity levels. The GD takes fractions of a second to run, but does not always find feasible solutions, whereas the TS finds solutions in all cases in an average of 2.3 seconds. The Lagrangian approach also finds solutions solutions in all cases and is particularly efficient when the server utilization rate is relatively low; however, its running time increases with the given optimality tolerance and can be relatively long (16 to 40 minutes) in some cases.

In [5] the authors also describe a heuristic that can be applied independently from the Lagrangian relaxation procedure. This heuristic starts with all facilities open and assigns customers to facilities one at a time in a greedy fashion whereby the current customer is assigned to the facility that minimizes the resulting change in the cost function. As with the first heuristic, a post-processing/improvement check is applied if the greedy assignment produces a feasible solution. This is similar to the UFL-based heuristic proposed by Wang et al. [46]. This type of heuristic is the most similar we have found in the literature to the heuristic developed in this dissertation. One important distinction is that unlike the problem studied in this dissertation, the ISPs considered by [5] and [46] do not involve deciding how much capacity to allocate to the servers. The heuristic developed in this dissertation is novel in that it employs multiple capacity allocation mechanisms and multiple greedy customer-assignment strategies.

1.3. Goals

The primary goal is to find a heuristic which works quickly enough for real-time calculations (not slower than one solution per second) that can be written into a Java-based application (although the language is not important) and finds an acceptably good solution within the time constraints. This would allow for the heuristic to be coded into the B2Bi application and deployed to client sites so that performance tuning is done real-time. The end goal is to provide an adaptive solution to meet customer needs as they change over time and be able to dynamically and quickly update tuning in response to a sudden spike in traffic.

The secondary goal is to determine if the difficulty of finding a solution can be determined. Identifying what factors are responsible for the problem's difficulty is part of this goal. Understanding how the problem data could be adjusted to reduce difficulty is the ultimate end goal.

The final goal is to determine what effect additional constraints have on the efficiency of the models and heuristic. The additional constraints tested are a constraint on the maximum number of open facilities in the solution and a constraint on the maximum capacity of the overall system. This provides four scenarios consisting of the base model with neither additional constraint, only the additional constraint for the maximum system capacity, only the additional constraint for the maximum number of open facilities, and both additional constraints.

Each model was tested with each data set, with and without each of the two additional constraints, with and without giving the models a starting point, and a five-minute maximum run time. The heuristic was tested with each data set, with and without each of the two additional constraints, and a five-minute maximum run time (the results were used as the starting point for the similarly constructed model when testing the models with a starting point). The heuristic also stored results for

stopping after the initial greedy algorithm so the results could be compared with the longer-running version of the heuristic which included customer reassignment tests.

1.4. Contributions

This dissertation develops a fast, effective heuristic for the immobile server problem that is easy to implement. This fills a gap between computationally intensive solution approaches in the literature: exact methods, that require specialized optimization software to implement, and meta-heuristics. The exact methods in the literature involve iteratively solving sequences of mixed integer programs. Another contribution of this dissertation is a stand-alone mathematical programming model that can be solved with a single, straightforward application of a commercial solver. A relatively large testbed of new problem instances, many of which are orders of magnitude larger than those in the literature, was generated to evaluate the new solution approaches. These data sets have been made available to share with other researchers on the SMU Scholar website. In the computational study with the testbed data and data from the literature, the heuristic was shown to be very effective for time-limited use cases such as reconfiguring resources for B2Bi clients and content delivery networks. The heuristic was also shown to improve the performance of the exact methods by quickly finding high-quality incumbent solutions. Through statistical and sensitivity analysis, the dissertation also provides a better understanding of the effect of customer waiting time cost and capacity budgets on solution with exact methods.

Chapter 2

Immobile Server Problem and its Cost Components

The input for the problem consists of a set of traffic streams (customers) denoted by $i \in \{1, 2, ..., m\}$, single-server queues denoted by $j \in \{1, 2, ..., n\}$, and server-capacity levels denoted by $k \in \{1, 2, ..., K\}$, which are described using the following parameters: λ_i is the average rate that customer i sends jobs to the queuing system, μ_{jk} is the average service rate for queue j if its server is allocated capacity level k, c_{ij} is the cost of assigning the traffic for customer i to queue j, t is the waiting cost per job per time unit, and f_{jk} is the cost of allocating capacity level k to queue j. With those inputs—which are used by all mathematical models and the heuristic—along with the solution given by the two binary decision variables of x_{ij} to denote customer i is assigned to queue j and y_{jk} to denote queue j is operating at capacity level k, the base nonlinear Immobile Server Problem is:

Minimize:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\lambda_i x_{ij} + t\sum_{j=1}^{n}\frac{\sum_{i=1}^{m}\lambda_i x_{ij}}{\sum_{k=1}^{K}\mu_{jk}y_{jk} - \sum_{i=1}^{m}\lambda_i x_{ij}} + \sum_{j=1}^{n}\sum_{k=1}^{K} f_{jk}y_{jk} \qquad (2.1)$$

Subject To:

$$\sum_{i=1}^{m} \lambda_i x_{ij} - \sum_{k=1}^{K} \mu_{jk} y_{jk} \leq 0 \qquad j \in \{1, ..., n\} \tag{2.2}$$

$$\sum_{k=1}^{K} y_{jk} \leq 1 \qquad j \in \{1, ..., n\} \tag{2.3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1, ..., m\} \tag{2.4}$$

$$x_{ij} \in \{0, 1\} \qquad i \in \{1, ..., m\}, j \in \{1, ..., n\} \tag{2.5}$$

$$y_{jk} \in \{0, 1\} \qquad j \in \{1, ..., n\}, k \in \{1, ..., K\} \tag{2.6}$$

The objective function (2.1) is to minimize the three costs described in further detail in this chapter: the facility assignment cost, the customer wait-time cost, and the facility capacity cost. Constraint (2.2) forces there to be sufficient capacity assigned to the facility ($\mu_{jk} y_{jk}$) to meet the demand assigned to the facility ($\lambda_{ij} x_{ij}$). Constraint (2.3) specifies that a facility may only have one capacity level assigned (as y_{jk} is binary from constraint (2.6)), but may also have no capacity level assigned—signifying the facility is unused. Constraint (2.4) specifies that a customer must be assigned to one and only one facility (as x_{jk} is binary from constraint (2.5)).

A few of the many applications this problem can solve are selecting the best physical location for new facilities, the number of devices needed—such as PBX's—at existing facilities, airport security to determine number of agents for each lane, Content Delivery Networks [38]—which need the capability to quickly adapt to changing demands, and Sterling B2B Integrator (B2Bi) for application performance tuning. However, the base model does have a few issues that might need to be addressed such as allowing for division by zero—especially in allowing for a facility to be unused, whether customers should be individuals or groups of individuals, and how to structure/calculate the costs—which could be highly subjective to the person making that determination.

2.1. Facility Assignment Cost

The $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \lambda_i x_{ij}$ portion of equation (2.1) is the facility assignment cost. It drives customer assignment to a preferred facility—provided it does not violate feasibility. Since a customer's facility cost (c_{ij}) can vary by facility, the facility with the lowest cost for that customer is considered the customer's preferred facility.

The total cost being applied is $c_{ij}\lambda_i$ if the customer is assigned to that facility ($x_{ij} = 1$). Therefore, customers with a larger λ_i tend to get assigned to their preferred facilities as a smaller change in c_{ij} between facilities for those customers has a larger change in the overall solution value.

The value for the customer's demand (λ_i) does not vary by facility in this model as the assumption is that the customer has certain requirements to be met regardless of which facility the customer gets assigned. Depending on the type of service being provided, this might not be accurate. For example, an elite grocery store chain that assigns a customer to a location that requires more travel might get less overall business from that customer as the customer might only travel to that location for products that cannot be purchased at a competitor with a facility that is closer to the customer.

The complexity created by the facility assignment cost is that customers could form groups with similar costs given different facility assignments. It could be as simple as swapping two customers between facilities, but as complex as moving every customer to a different facility while maintaining a similar cost. This can cause a large difference in the actual solution with a small change in the solution cost.

2.2. Facility Capacity Cost

The $\sum_{j=1}^{n}\sum_{k=1}^{K} f_{jk}y_{jk}$ portion of equation (2.1) is the facility capacity cost. It drives capacity of the facility to be the lowest feasible value—meaning it tries to keep customers from being assigned to the facility if that customer assignment causes the required capacity to increase to the next higher incremental value by trying to assign customers to the facility with the lowest f_{jk} values. The y_{jk} value specifies if the facility is being operated at the specific capacity level and f_{jk} is the total cost of operating the facility at that capacity level.

The facility capacity cost (f_{jk}) generally represents the cost of opening the facility—assuming it's a new facility—plus the cost of being able to service a specific amount of demand—sometimes referred to as a customer access cost. Even in the models which break apart these components, the customer access cost does not vary by customer and facility—only by facility. In order to vary by customer assignment and capacity level, both the customer assignment (x_{ij}) and facility capacity level (y_{jk}) would be needed in this cost—creating a nonlinear cost using $x_{ij}y_{jk}$ to denote the customer is assigned to the facility and the facility is operating at the specific capacity level. An additional binary decision variable (w_{ijk}) could be used so this cost could stay linear ($x_{ij} + y_{jk} \leq w_{ijk} + 1$) at the expense of adding mnK additional constraints.

This base model does not require the increments in facility capacity cost to be linear—or even convex. They are simply break points used to determine a specific capacity level (μ_{jk}) and cost (f_{jk}). One of the reasons for splitting the cost into its components is to apply a concave cost function (to represent economies of scale) in determining the customer access cost. However, these calculations are done prior to the solver attempting to find the optimal solution and result in a set of μ_{jk} and f_{jk} values that continue to increase at each higher capacity level.

2.3. Customer Wait-Time Cost

The $t \sum_{j=1}^{n} \frac{\sum_{i=1}^{m} \lambda_i x_{ij}}{\sum_{k=1}^{K} \mu_{jk} y_{jk} - \sum_{i=1}^{m} \lambda_i x_{ij}}$ portion of equation (2.1) is the customer wait-time cost. It drives demand and capacity of the facility to be level across all facilities by penalizing overloading some of the facilities to the point where the customers assigned to the facility experience long wait times to be serviced on average. This cost component causes the problem to be nonlinear and greatly increases the difficulty in solving the problem. The t values in our data sets are either a given number or calculated to be the maximum $c_{ij}\lambda_i$. In our testing, a β value is used to vary the value of t to determine the effect of t on the solution or run time.

Some models have been developed to linearize this component by either iteratively calculating a lower bound until the same solution is achieved twice, or by using additional constraints to calculate the appropriate ratio without the need to iterate. Both provide increased speed in finding a solution compared to using a nonlinear solver—however the solution cannot be guaranteed to be globally optimal. As discussed later, the Elhedhli model also provides for some interesting results when given a good starting point as the model is not using the actual objective value to minimize the cost, but an approximate linearized objective value. Additionally, the Colley model can be highly sensitive to the tolerance levels used by the MIP solver. Even linearizing the customer wait-time cost does not affect the increase in difficulty in finding the optimal solution.

2.4. Cost Interactions

The three cost components interact with each other to affect optimal customer assignment and facility capacity level. The facility assignment and capacity costs are competing objectives. The facility assignment cost is minimized when all customers are assigned to their preferred (e.g., nearest) facility. From this perspective, a given

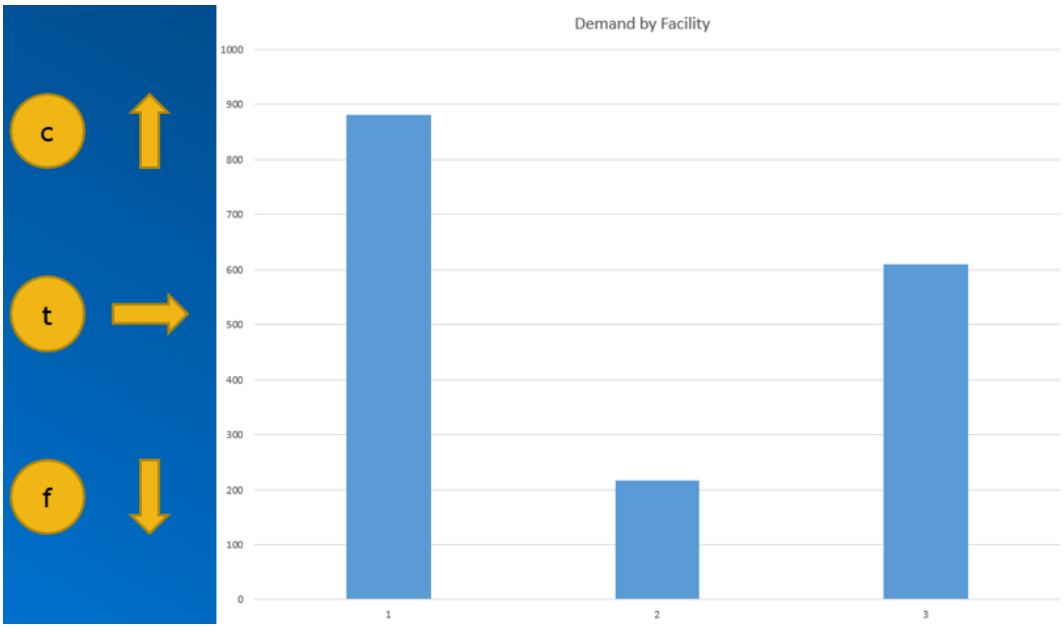


Figure 2.1. Cost Interaction

facility can be seen as attracting as much demand as possible from a particular set of customers. At the same time, however, the facility capacity cost can be minimized for a given facility when it is assigned as little demand as possible. The customer wait-time cost can be seen as an incentive to assign customers evenly across all facilities.

The customer wait-time cost is required in order to keep a facility from being overloaded if it is the preferred facility for a large portion of the customers. The t value helps scale the customer wait-time cost so that it can compete with the facility assignment cost. This is purpose of running the data multiple times with a different β so that the correct scaling can be found to offset the competing costs.

The lowest possible customer wait-time cost is with each facility at maximum capacity and the demand perfectly spread across all facilities. This is because the ratio $(\frac{\lambda}{\mu-\lambda})$ increases faster as λ increases than the corresponding decrease in another facility. If t varied by facility, then this would not necessarily be the case. However,

t treats all facilities with equal importance in the base model.

2.5. Costs in IBM Sterling B2B Integrator

Since IBM Sterling B2B Integrator (B2Bi) is a software application, the costs cannot be calculated like they would for physical facilities and customers (e.g. using the distance between the customer and the facility to calculate a cost). Therefore, there is a great deal of leeway in determining the three costs in B2Bi.

For example, certain types of business processes perform similar types of functionality and could be grouped together. The priority queues the business processes could be assigned to could be predetermined for specific types of functionality and therefore business processes with that functionality would have that priority queue as its preferred "facility" with the lowest cost. Costs could be linearly or nonlinearly increased from there based on the desirability of that business process running in another priority queue.

Additionally, certain priority queues might need more idle threads available in case new busines processes are executed. These priority queues could be given lower facilitiy capacity costs to help drive additional threads to that priority queue.

Customer wait-time cost would need to be set to appropriately balance out the other costs so that any single priority queue is not overloaded with assigned business processes.

Capacity and arrival rates would also be calculated based on the amount of work being performed during a specific time period. The capacity would be the time period multiplied by the number of threads at that capacity level (e.g. 3,600, 7,200, and 10,800 could be three capacity levels for 1, 2, or 3 threads being available over a one hour time period using seconds as the unit of measure). The arrival rate would be the average number of seconds a business process was executing in any given hour.

Because business processes can have multiple instances executing at the same time in B2Bi, the arrival rate could be higher than the number of seconds in an hour.

Chapter 3

Mathematical Models

Three different mathematical approaches were compared: the nonlinear base model which is assumed to be the most accurate provided it completes within the allowed time—but also the most difficult to solve, the approach developed by Elhedhli in 2006 [22] which trades accuracy for speed, and the Colley linear model developed in this dissertation.

3.1. Nonlinear

The nonlinear model was solved using Baron (19.12.7 [44]). It was given a maximum run time of five minutes. When giving a starting point, the x_{ij} and y_{jk} decision variables are preset to the values calculated by the heuristic presented in Chapter 4 of this dissertation before the model is solved.

3.2. Elhedhli Linear

The Elhedhli linear model was solved using Gurobi (9.1.2 [26]). The model linearizes customer wait-time cost using a new r_{jh} parameter where h is the iteration and H represents the number of iterations. The r_{jh} value is calculated at the end of each iteration for the next iteration to use.

New R_j and z_{jk} decision variables are then used to linearize the objective function. The value for z_{jk} is the ratio of the demand to the capacity for the facility. The value for R_j is the lower bound of the ratios calculated from the current z_{jk} and the ratios (r_{jh}) from previous iterations. The new objective function component for customer

wait-time cost is $t \sum_{j=1}^{n} R_j$.

The Elhedhli linear formulation from his paper in 2006 [22] is therefore:

Minimize:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\lambda_i x_{ij} + t\sum_{j=1}^{n} R_j + \sum_{j=1}^{n}\sum_{k=1}^{K} f_{jk}y_{jk} \tag{3.1}$$

Subject To:

$$\sum_{i=1}^{m} \lambda_i x_{ij} - \sum_{k=1}^{K} \mu_{jk} z_{jk} = 0 \qquad j \in \{1,...,n\} \tag{3.2}$$

$$\sum_{k=1}^{K} y_{jk} \leq 1 \qquad j \in \{1,...,n\} \tag{3.3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1,...,m\} \tag{3.4}$$

$$z_{jk} - y_{jk} \leq 0 \qquad j \in \{1,...,n\}, k \in \{1,...,K\} \tag{3.5}$$

$$z_{jk} - \frac{R_j}{(1+r_{jh})^2} \leq \frac{r_{jh}^2}{(1+r_{jh})^2} \qquad j \in \{1,...,n\}, k \in \{1,...,K\}, h \in \{1,...,H\} \tag{3.6}$$

$$x_{ij} \in \{0,1\} \qquad i \in \{1,...,m\}, j \in \{1,...,n\} \tag{3.7}$$

$$y_{jk} \in \{0,1\} \qquad j \in \{1,...,n\}, k \in \{1,...,K\} \tag{3.8}$$

$$0 \leq z_{jk} \leq 1 \qquad j \in \{1,...,n\}, k \in \{1,...,K\} \tag{3.9}$$

$$0 \leq R_j \qquad j \in \{1,...,n\} \tag{3.10}$$

The formula in the constraint to limit R_j implies that the exact value for R_j matches the true ratio; however, since it is an inequality in the constraint, the value for R_j could be lower than the true value of the ratio, thus making the solution cost the model returns to be lower than the actual solution cost. The model is solved iteratively with H increasing at each iteration until the same solution is found twice in a row.

	Facility 1			Facility 2			Facility 3			
h	z	R	r	z	R	r	z	R	r	Solution Cost
1	86.35%	-	6.3277	98.76%	-	79.6452	96.33%	-	26.2838	1,174,126.0000
2	66.10%	-	1.9502	97.48%	-	38.6825	83.32%	-	4.9963	1,196,920.0000
3	80.11%	3.1694	4.0283	97.48%	38.6825	38.6825	69.31%	-	2.2588	1,222,031.1895
4	78.85%	3.7103	3.7292	97.48%	38.6825	38.6825	70.57%	2.3924	2.3981	1,223,791.1382
5	78.85%	3.7292	3.7292	97.48%	38.6825	38.6825	70.57%	2.3981	2.3981	1,223,805.9135
6	78.85%	3.7292	3.7292	97.48%	38.6825	38.6825	70.57%	2.3981	2.3981	1,223,805.9135

Figure 3.1. Elhedhli Linear Model Example

When giving a starting point, the x_{ij} and y_{jk} decision variables are preset to the values calculated by the heuristic. The r_{jh} values are calculated as if the model itself was executed with one iteration already. Then the model begins its normal iteration.

Figure 3.1 is an example of the z_{jk} and R_j values calculated for each iteration with the new r_{jh} value calculated based on the incumbent solution. The solution cost can be seen to increase as R_j and r_{jh} converge. The z_{jk} value shown is the percentage of demand versus the selected capacity level for that facility during that iteration—as the capacity level can change from one iteration to another.

This also demonstrates that stopping based on the same solution twice in a row, stopping based on the solution cost converging, and stopping based on R_j converging to r_{jH} are three possible stopping conditions for this mathematical model. While stopping based on the same solution would be an exact match, stopping based on converging values would be tested against some tolerance. In order to determine which stopping point is best, all three stopping conditions were tested for each scenario, starting point, and data set. Stopping based on solution cost checked for the absolute difference in solution cost to be within 1e-08 while stopping based on R_j convering to r_{jH} checked for a tolerence of 1e-03 for the percentage difference of each facility.

3.3. Colley Linear

The Colley linear model was solved using Gurobi (9.1.2). In order to linearize the objective function (2.1), a new continuous decision variable q_{jk} will represent the (average) waiting time at facility j if facility j is assigned capacity level k, and zero otherwise. Using this notation, the waiting time at facility j is $\sum_{k=1}^{K} q_{jk}$, and the waiting time term in the objective function can be replaced with the linear term $t \sum_{j=1}^{n} \sum_{k=1}^{K} q_{jk}$.

Let continuous decision variable $v_{ij} = 1 + \sum_{k=1}^{K} q_{jk}$ if customer i is assigned to facility j (i.e. if $x_{ij} = 1$), and zero otherwise. Additionally, a continuous parameter M is introduced which must be a least one plus the largest possible average wait time. This can be found by using another mathematical model or estimated by taking the largest μ_{jk} value if all parameters are integers. Care should be taken to not have the value be too low as it could over constrain the problem and/or cause the problem to become infeasible. This produces the following mixed-integer linear program:

Minimize:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \lambda_i x_{ij} + t \sum_{j=1}^{n} \sum_{k=1}^{K} q_{jk} + \sum_{j=1}^{n} \sum_{k=1}^{K} f_{jk} y_{jk} \tag{3.11}$$

Subject To:

$$\sum_{i=1}^{m} \lambda_i v_{ij} - \sum_{k=1}^{K} \mu_{jk} q_{jk} = 0 \qquad j \in \{1, ..., n\} \tag{3.12}$$

$$\sum_{k=1}^{K} y_{jk} \leq 1 \qquad j \in \{1, ..., n\} \tag{3.13}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1, ..., m\} \tag{3.14}$$

$$q_{jk} \leq M y_{jk} \qquad j \in \{1, ..., n\}, k \in \{1, ..., K\} \tag{3.15}$$

$$v_{ij} \geq \left(1 + \sum_{k=1}^{K} q_{jk}\right) - M\left(1 - x_{ij}\right) \qquad i \in \{1, ..., m\}, j \in \{1, ..., n\} \tag{3.16}$$

$$x_{ij} \in \{0,1\} \qquad i \in \{1,...,m\}, j \in \{1,...,n\} \tag{3.17}$$

$$y_{jk} \in \{0,1\} \qquad j \in \{1,...,n\}, k \in \{1,...,K\} \tag{3.18}$$

$$0 \leq q_{jk} \qquad j \in \{1,...,n\}, k \in \{1,...,K\} \tag{3.19}$$

$$0 \leq v_{ij} \qquad i \in \{1,...,m\}, j \in \{1,...,n\} \tag{3.20}$$

In order to justify the use of q_{jk} and v_{ij}, let the following variables represent specific summations and sets:

1. Let $I_j \subseteq \{1,2,...,m\}$ denote the customers assigned to facility j: $I_j \subseteq \{i \in 1,...,m : x_{ij} = 1\}$
2. Let $\hat{\lambda}_j$ be the arrival rate at facility j: $\hat{\lambda}_j = \sum_{i=1}^{m} \lambda_i x_{ij} = \sum_{i \in I_j} \lambda_i$.
3. Let $\hat{\mu}_j$ be the capacity at facility j: $\hat{\mu}_j = \sum_{k=1}^{K} \mu_{jk} y_{jk}$ (from (3.13) and (3.18) no more than one y_{jk} may have a value of one).
4. Let $\hat{q}_j$ be the average waiting time at facility j: $\hat{q}_j = \sum_{k=1}^{K} q_{jk} y_{jk}$ (from (3.13) and (3.18) no more than one y_{jk} may have a value of one).

Lemma 3.1 *Since both $\hat{\mu}_j$ and $\hat{q}_j$ are based on y_{jk} and from (3.13) and (3.18), no more than one y_{jk} may have a value of one, then $\sum_{k=1}^{K} \mu_{jk} q_{jk}$ in (3.12) equals $\hat{\mu}_j \hat{q}_j$.*

From (3.16) v_{ij} must be no smaller than one plus $\hat{q}_j$ if x_{ij} is equal to one, otherwise v_{ij} may be any positive value. Because the objective function wants to minimize q_{jk}, it will attempt to set those values to zero. However, if a customer is assigned to facility j, then v_{ij} must be at least one and from (3.12) q_{jk} must be greater than zero. Therefore, v_{ij} must also increase, causing q_{jk} to continue to increase until (3.12) and (3.16) are both satified, which would be when v_{ij} is exactly equal to one plus $\hat{q}_j$ as any value over that would cause $\hat{q}_j$ to be larger than it needs to be.

Therefore, in an optimal solution, $\hat{q}_j$ is equal to the average waiting time at facility j as shown below:

Case 1: Facility j is used (i.e., $\sum_{i=1}^{m} x_{ij} \geq 1$):

$$\sum_{i=1}^{m} \lambda_i v_{ij} = \sum_{k=1}^{K} \mu_{jk} q_{jk} \quad \Rightarrow \quad (3.21)$$

$$\sum_{i \in I_j} \lambda_i v_{ij} = \sum_{k=1}^{K} \mu_{jk} q_{jk} \quad \Rightarrow \quad (3.22)$$

$$\sum_{i \in I_j} \lambda_i v_{ij} = \hat{\mu}_j \hat{q}_j \quad \Rightarrow \quad (3.23)$$

$$\sum_{i \in I_j} \lambda_i (1 + \hat{q}_j) = \hat{\mu}_j \hat{q}_j \quad \Rightarrow \quad (3.24)$$

$$\sum_{i \in I_j} \lambda_i + \sum_{i \in I_j} \lambda_i \hat{q}_j = \hat{\mu}_j \hat{q}_j \quad \Rightarrow \quad (3.25)$$

$$\sum_{i \in I_j} \lambda_i = \hat{\mu}_j \hat{q}_j - \sum_{i \in I_j} \lambda_i \hat{q}_j \quad \Rightarrow \quad (3.26)$$

$$\hat{\lambda}_j = \hat{\mu}_j \hat{q}_j - \hat{\lambda}_j \hat{q}_j \quad \Rightarrow \quad (3.27)$$

$$\hat{\lambda}_j = \hat{q}_j (\hat{\mu}_j - \hat{\lambda}_j) \quad \Rightarrow \quad (3.28)$$

$$\hat{q}_j = \frac{\hat{\lambda}_j}{\hat{\mu}_j - \hat{\lambda}_j} \quad (3.29)$$

Equation (3.23) follows from (3.22) by Lemma 3.1. Equation (3.24) follows from (3.23) by definition of $\hat{q}$, and (3.27) follows from (3.26) by definition of $\hat{\lambda}$.

Case 2: Facility is not used (i.e., $\sum_{i=1}^{m} x_{ij} = 0$): From (3.16), all v_{ij} for facility j can be equal to zero, allowing $\hat{q}_j$ to also equal 0 from (3.12).

3.4. Mathematical Model Size Comparison

Using m for the number of customers, n for the number of facilities, K for the capacity levels, and H for the number of prior iterations when solving the Elhedhli model, the size of each model is given in Table 3.1:

Table 3.1. Mathematical Model Size Comparison

Model	**Constraints**	**Decision Variables**
Nonlinear	$2n + m + mn + nK$	$mn + nK$
Elhedhli Linear	$2n + m + mn + 2nK + nKH$	$mn + 2nK + n$
Colley Linear	$2n + m + 2mn + 2nK$	$2mn + 2nK$

While the Elhedhli linear model above grows with each iteration, the formula listed would be applicable to the final iteration as it is the largest number of constraints. The size of the model depends on the data set, if a starting point was given, and the number of iterations required. The more iterations required, the larger the model becomes and the more time required for solving each additional iteration. Even with this, the Elhedhli linear model usually performs in the least amount of elapsed time compared to the other mathematical models.

Chapter 4

Heuristic

The heuristic developed in this dissertation for handling even more stringent time constraints in exchange for accuracy was written in Java. However, any programming language could be used. Java was chosen as one of the potential uses is with IBM Sterling B2B Integrator (B2Bi), which is Java-based. The heuristic's base greedy algorithm is a Constructive Heuristic [16] designed for maximum speed so that a solution can be found in no longer than one second. Additional customer reassignment alogrithms using repetitive and distinct Hill Climbing [16] local search heuristics can also be applied by the heuristic to potentially get a better result while incurring some additional elapsed time.

The elapsed time is important for B2Bi as unexpected or abnormally large spikes in traffic can occur. For example, many retail companies experience abnormally large spikes in traffic on Black Friday and the following month. Also, the marketing department could potentially create an unexpected increase in traffic with a vastly successful promotion. Both cases currently require the information technology department to adjust tuning prior to the spike—provided the spike is expected—and to be adjusted back to normal parameters after the spike in traffic. A missed change in the tuning could bottleneck the system, potentially causing a loss of revenue. Integrating this heuristic into B2Bi for automatically tuning the system could potentially save the company from experiencing lost sales due to a spike in traffic. Therefore, finding a method of constructing the data set and costs that minimizes the solution error using just the heuristic's greedy alrogithm would be a best case scenario.

The rest of this chapter is structured as follows: Section 4.1 gives an abstract description of the heuristic and pseudocode for its main routine. Section 4.2 illustrates the heuristic with a small problem instance (data set ST1b). Pseudocode for the subroutines of the heuristic are given in sections 4.3, 4.4, and 4.5, and a worst-case complexity analysis is presented in section 4.6.

4.1. Basic Structure

Variables which represent an indexed set of values—such as λ_i respresenting the demand for customer i—use subscripts for the single value being accessed or use a bar ($\bar{\lambda}$) when refering to the set as a whole. This nomenclature is used throughout the pseudocode in this dissertation. As mentioned in the introduction for Chapter 2, all pseudocode has access to the input data (m, n, K, t, $\bar{\lambda}$, $\bar{c}$, $\bar{\mu}$, and $\bar{f}$). Which are therefore not listed as inputs to the pseudocode routines. Comments are also included in the pseudocode using /* ... */ notation.

In terms of the heuristic, the greedy constructive algorithm initially assigning customers to facilities is called "Phase 1" and the customer reassignment logic is called "Phase 2" so that the results of running only the greedy constructive algorithm can be compared with the complete heuristic. All Phase 1 routines require a sorted list of customers ($\bar{L}$) and output a solution of $\bar{x}$ and $\bar{y}$. All Phase 2 routines take the solution of $\bar{x}$ and $\bar{y}$ as both the input and the output.

For this dissertation, seven customer lists were calculated from the data set to provide a Deterministic Search [16]. The seven customer lists derived from the data set are sorted using λ_i, c_{min}, c_{avg}, c_{max}, $\lambda_i c_{min}$, $\lambda_i c_{avg}$, and $\lambda_i c_{max}$—where c_{min} is the minimum c for the customer, c_{avg} is the average c for the customer, and c_{max} is the maximum c for the customer. Additionally, three randomly sorted customer lists were included; this allows for a potentially better solution to be found, but

potentially results in different solutions across multiple runs of the heuristic against the same input.

Phase 1 has three possible capacity level options that can be utilitized: forcing a minimum capacity (4.4), unforced capacity (4.5), and forcing a maximum capacity (4.6). These are represented below with three different sets of pseudocode. Each of the ten lists were sorted both ascending (represented with $\triangle$) and descending (represented with $\bigtriangledown$) when calling the three capacity level options, creating sixty executions for Phase 1 within this heuristic. All sixty instances were executed simultaneously in separate threads to minimize the elapsed time.

All sixty variants of the greedy constructive algorithm of Phase 1 must complete before any iterations of Phase 2 are executed in the heuristic for this dissertation. An additional time contraint of five minutes was also applied to be consistent with the time limit applied to the mathematical models.

In this heuristic, Phase 2 continues to loop until the solution no longer changes. Within this outer loop, three subroutines are executed in the sequence of One Customer Move (4.7), Two Customer Move (4.8), and Swap Customers (4.9).

The main heuristic process for this dissertation can be represented as:

Algorithm 4.1 *Main*

1: $x_{ij} \leftarrow 0 \ \forall \ i \in \{1..m\}, j \in \{1..n\}$

2: $y_{jk} \leftarrow 0 \ \forall \ j \in \{1..n\}, k \in \{1..K\}$

3: $z \leftarrow \infty$

/ Initialize and populate a set of 10 customer lists */*

4: $\bar{M} \leftarrow \{1..10\}$

5: $M_1 \leftarrow \forall \ i \in \{1..m\} \triangle \lambda_i$

6: $M_2 \leftarrow \forall \ i \in \{1..m\} \triangle min(c_{ij} \forall j \in \{1..n\})$

7: $M_3 \leftarrow \forall \ i \in \{1..m\} \triangle avg(c_{ij} \forall j \in \{1..n\})$

8: $M_4 \leftarrow \forall \ i \in \{1..m\} \triangle max(c_{ij} \forall j \in \{1..n\})$

9: $M_5 \leftarrow \forall \ i \in \{1..m\} \triangle \lambda_i min(c_{ij} \forall j \in \{1..n\})$

10: $M_6 \leftarrow \forall \ i \in \{1..m\} \triangle \lambda_i avg(c_{ij} \forall j \in \{1..n\})$

11: $M_7 \leftarrow \forall \ i \in \{1..m\} \triangle \lambda_i max(c_{ij} \forall j \in \{1..n\})$

12: $M_8 \leftarrow \forall \ i \in \{1..m\} \triangle random()$

13: $M_9 \leftarrow \forall \ i \in \{1..m\} \triangle random()$

14: $M_{10} \leftarrow \forall \ i \in \{1..m\} \triangle random()$

/ Execute Phase 1 with customer lists sorted ascending and then descending */*

15: ***for*** $s = 1$ ***to*** 2 ***do***

16: ***for*** $a = 1$ ***to*** 10 ***do***

17: $(\bar{\chi}, \bar{\gamma}) \leftarrow ForcedMinimumCapacity\ 4.4\ (M_a)$

18: $Z \leftarrow CalculateCost\ 4.2\ (\bar{\chi}, \bar{\gamma})$

/ Best Phase 1 Solution? */*

19: ***if*** $Z < z$ ***then***

20: $z \leftarrow Z$

21: $\bar{x} \leftarrow \bar{\chi}$

22: $\bar{y} \leftarrow \bar{\gamma}$

23: ***end if***

24: $(\bar{\chi}, \bar{\gamma}) \leftarrow UnforcedCapacity\ 4.5\ (M_a)$

25: $Z \leftarrow CalculateCost\ 4.2\ (\bar{\chi}, \bar{\gamma})$

/* *Best Phase 1 Solution?* */

26: ***if*** $Z < z$ ***then***

27: $z \leftarrow Z$

28: $\bar{x} \leftarrow \bar{\chi}$

29: $\bar{y} \leftarrow \bar{\gamma}$

30: ***end if***

31: $(\bar{\chi}, \bar{\gamma}) \leftarrow ForcedMaximumCapacity\ 4.6\ (M_a)$

32: $Z \leftarrow CalculateCost\ 4.2\ (\bar{\chi}, \bar{\gamma})$

/* *Best Phase 1 Solution?* */

33: ***if*** $Z < z$ ***then***

34: $z \leftarrow Z$

35: $\bar{x} \leftarrow \bar{\chi}$

36: $\bar{y} \leftarrow \bar{\gamma}$

37: ***end if***

38: $M_a \leftarrow \forall\ i \in \{1..m\} \bigtriangledown M_a$ /* *Sort* M_a *descending for the next round* */

39: ***end for***

40: ***end for***

/* *At least one feasible solution?* */

41: ***if*** $z \neq \infty$ ***then***

42: *Save* $\bar{x}$ *and* $\bar{y}$ *as the best Phase 1 Solution*

/* *Reinitialize "best" solution to find best Phase 2 solution* */

43: $x_{ij} \leftarrow 0\ \forall\ i \in \{1..m\}, j \in \{1..n\}$

44: $y_{jk} \leftarrow 0\ \forall\ j \in \{1..n\}, k \in \{1..K\}$

45: $z \leftarrow \infty$

/* *Execute Phase 2* */

46: ***for all*** $(\bar{X}, \bar{Y}) \in$ *feasible and unique Phase 1 solutions* ***do***

47: $Z \leftarrow CalculateCost\ 4.2\ (\bar{X}, \bar{Y})$

48: $l \leftarrow 1$

49: ***while*** $l = 1$ ***do***

50: $l \leftarrow 0$

51: $(\bar{\chi}, \bar{\gamma}) \leftarrow$ *One Customer Move 4.7* $(\bar{X}, \bar{Y})$

52: $\Delta \leftarrow CalculateCost\ 4.2\ (\bar{\chi}, \bar{\gamma})$

/* *Best Solution?* */

53: ***if*** $\Delta < Z$ ***then***

54: $Z \leftarrow \Delta$

55: $\bar{X} \leftarrow \bar{\chi}$

56: $\bar{Y} \leftarrow \bar{\gamma}$

57: $l \leftarrow 1$

58: ***end if***

59: $(\bar{\chi}, \bar{\gamma}) \leftarrow$ *Two Customer Move 4.8* $(\bar{X}, \bar{Y})$

60: $\Delta \leftarrow CalculateCost\ 4.2\ (\bar{\chi}, \bar{\gamma})$

/* *Best Solution?* */

61: ***if*** $\Delta < Z$ ***then***

62: $Z \leftarrow \Delta$

63: $\bar{X} \leftarrow \bar{\chi}$

64: $\bar{Y} \leftarrow \bar{\gamma}$

65: $l \leftarrow 1$

66: ***end if***

67: $(\bar{\chi}, \bar{\gamma}) \leftarrow$ *Swap Customers 4.9* $(\bar{X}, \bar{Y})$

68: $\Delta \leftarrow CalculateCost\ 4.2\ (\bar{\chi}, \bar{\gamma})$

/* *Best Solution?* */

69: ***if*** $\Delta < Z$ ***then***

70: $Z \leftarrow \Delta$

71: $\bar{X} \leftarrow \bar{\chi}$

72: $\bar{Y} \leftarrow \bar{\gamma}$

73: $l \leftarrow 1$

74: ***end if***

75: ***end while***

/ Best Phase 2 Solution? */*

76: ***if*** $Z < z$ ***then***

77: $z \leftarrow Z$

78: $\bar{x} \leftarrow \bar{\chi}$

79: $\bar{y} \leftarrow \bar{\gamma}$

80: ***end if***

81: ***end for***

82: *Save* $\bar{x}$ *and* $\bar{y}$ *as the best Phase 2 Solution*

83: ***end if***

4.2. Illustration

Using a data set with five customers ($m = 5$), three facilities ($n = 3$), and three capacity levels ($K = 3$), the heuristic can be illustrated graphically in a step-by-step fashion. The data set has $t = 1000$ and the following indexed values:

Table 4.1. Customer-related Parameters

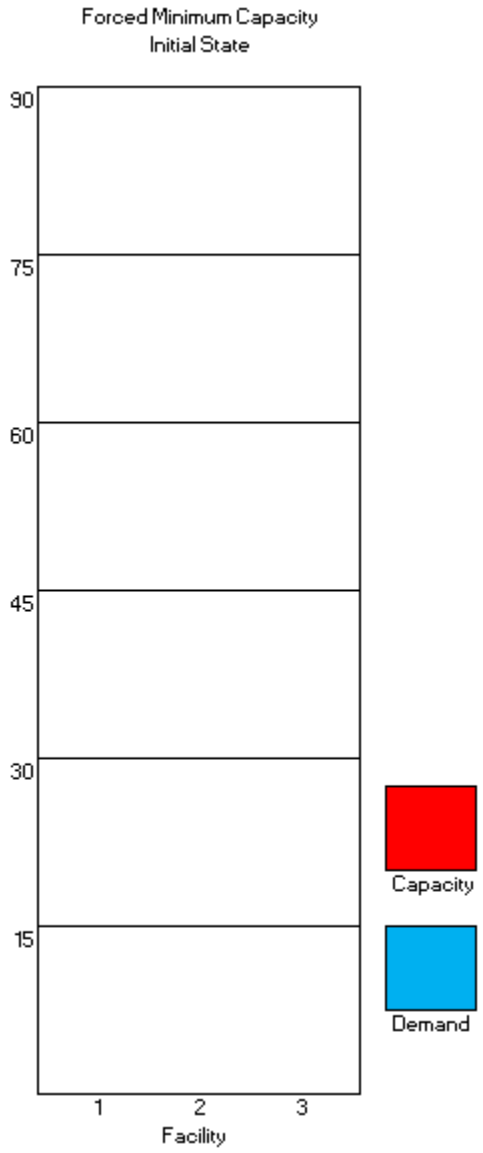
Customer (i)	**Demand** (λ_i)	**Assignment Cost** (c_{ij})			**max**($\lambda_i c_{ij}$)
		1	**2**	**3**	
1	15	20	10	15	300
2	25	30	10	20	750
3	15	45	30	15	675
4	10	25	75	50	750
5	20	40	20	60	1200

Table 4.2. Facility-related Parameters

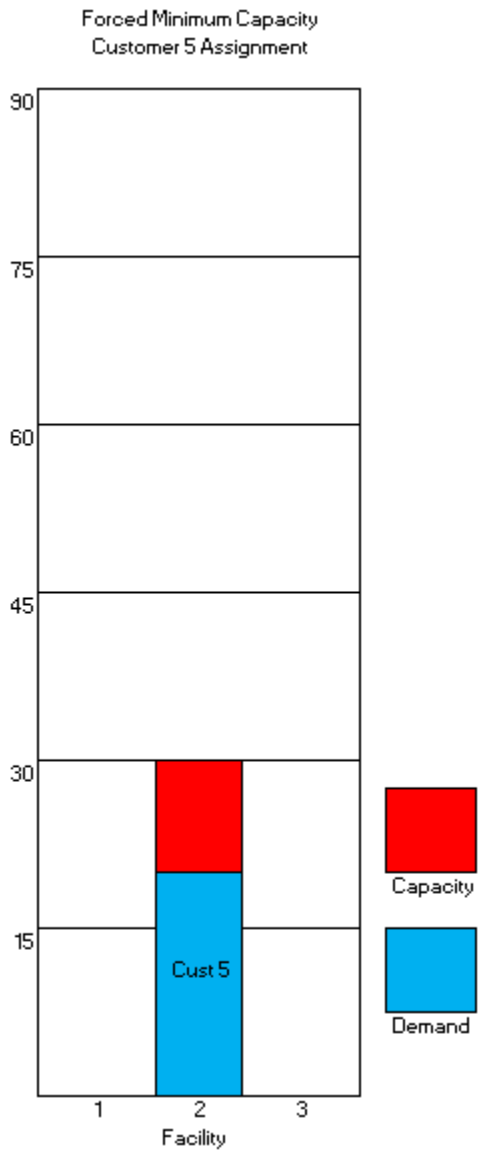
Facility (j)	**Capacity** (μ_{jk})			**Capacity Cost** (f_{jk})		
	1	**2**	**3**	**1**	**2**	**3**
1	30	60	90	90	180	270
2	30	60	90	90	180	270
3	30	60	90	90	180	270

Given a sort order of the maximum facility assignment cost times demand ($\max(\lambda_i c_{ij})$) descending, the customers will be evaluated by Phase 1 in the ordered set $\{5, 2, 4, 3, 1\}$. All three Phase 1 capacity options will be illustrated. Each solution will then be evaluated by Phase 2 as part of the illustration for that Phase 1 capacity option.

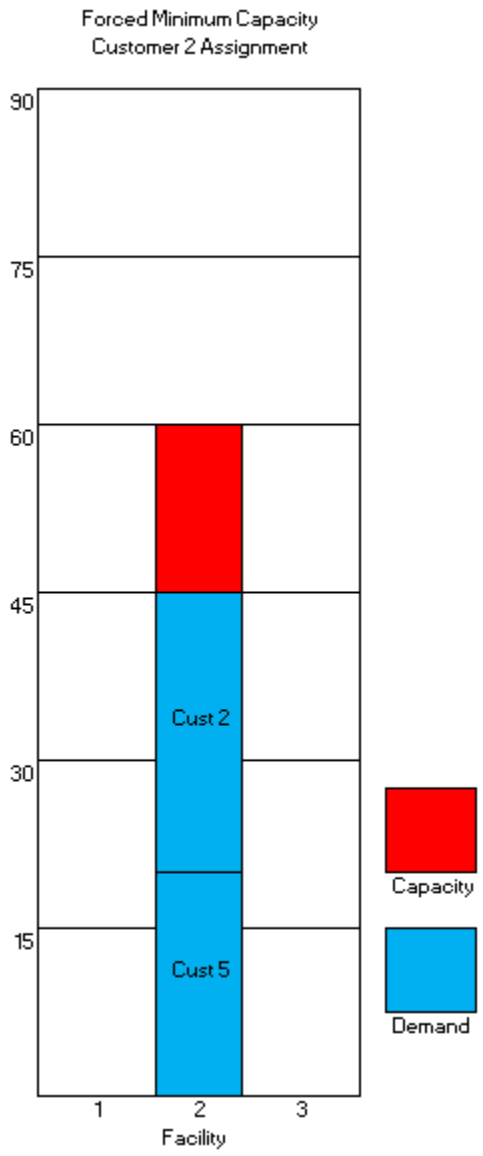
4.2.1. Forced Minimum Capacity Illustration



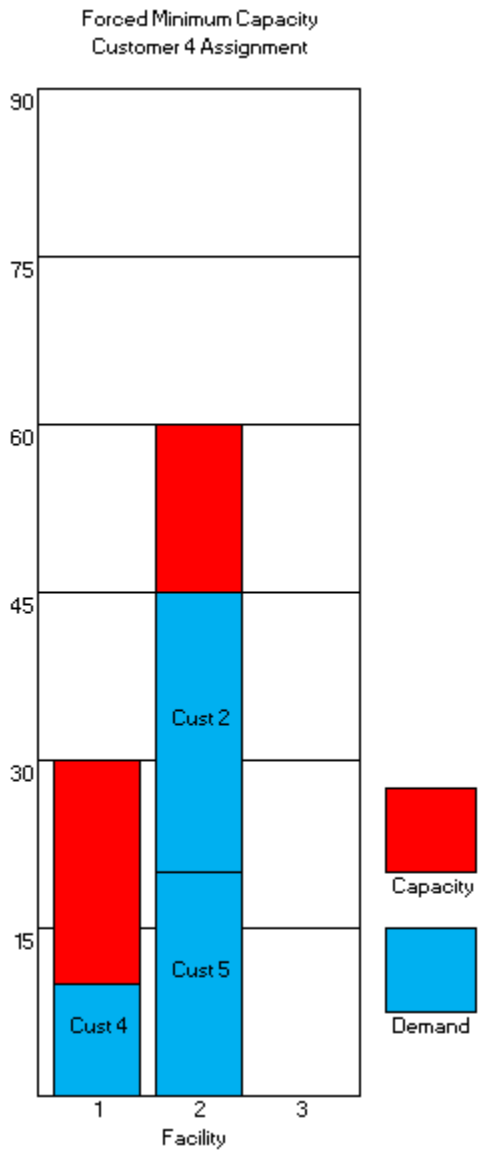
Forced Minimum Capacity starts with no capacity assigned to any facility. The solution cost starts at $0.00.



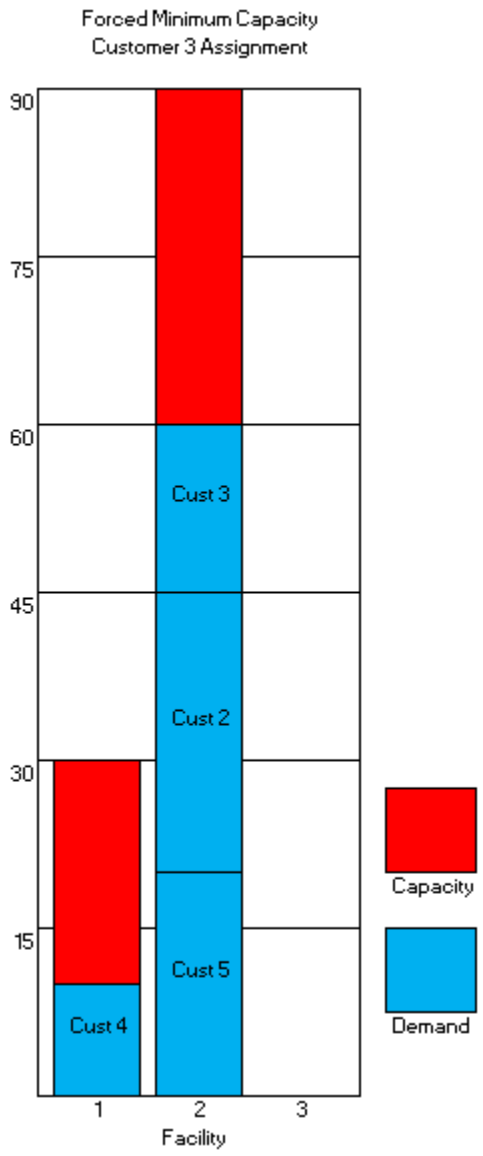
Customer 5 is assigned to Faciltity 2. Because Faciltity 2 had no capacity and Customer 5 has 20 units of demand, Faciltity 2's capacity increases to 30 units as that is the smallest feasible capacity level after assigning Customer 5 to Faciltity 2. The solution cost is now \$2,490.00.



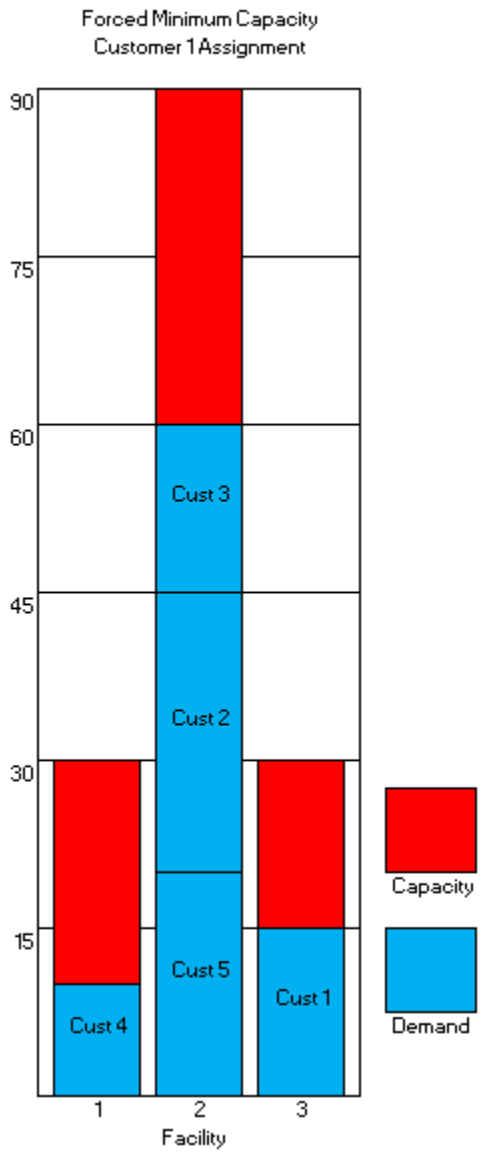
Customer 2 is assigned to Faciltity 2. Because Faciltity 2 had an available capacity of 10 units and Customer 2 has 25 units of demand, Faciltity 2's capacity increases to 60 units as that is the smallest feasible capacity level after assigning Customer 2 to Faciltity 2. The solution cost is now \$3,830.00.



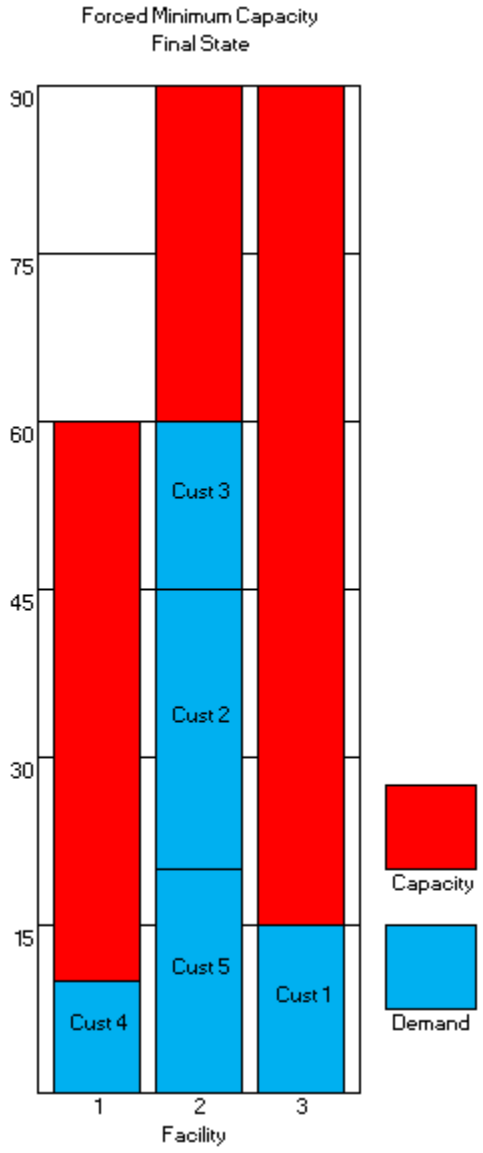
Customer 4 is assigned to Faciltity 1. Because Faciltity 1 had no capacity and Customer 4 has 10 units of demand, Faciltity 1's capacity increases to 30 units as that is the smallest feasible capacity level after assigning Customer 4 to Faciltity 1. The solution cost is now \$4,670.00.



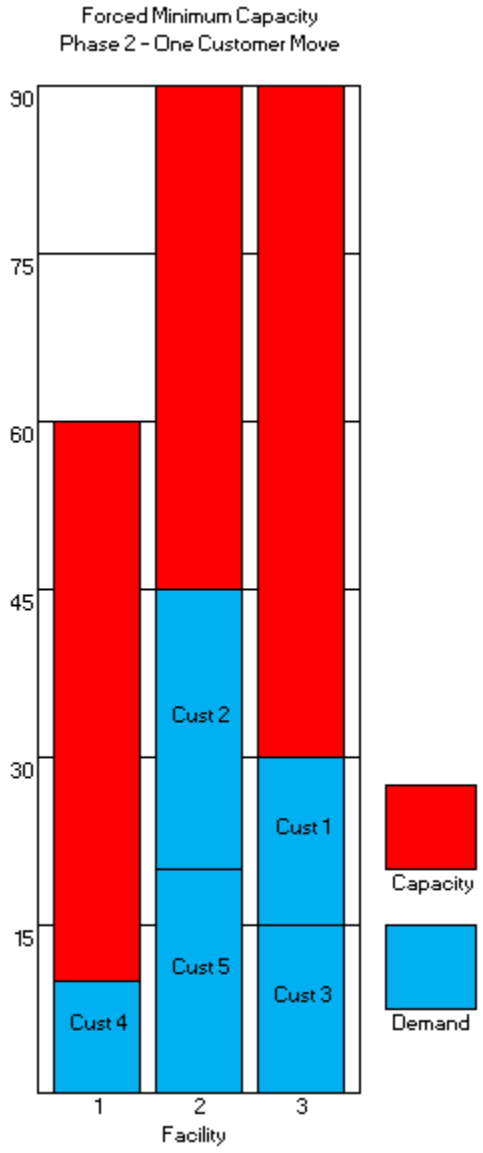
Customer 3 is assigned to Faciltity 2. Because Faciltity 2 had an available capacity of 15 units and Customer 3 has 15 units of demand, Faciltity 2's capacity increases to 90 units as that is the smallest feasible capacity level after assigning Customer 3 to Faciltity 2. The solution cost is now \$4,2100.00.



Customer 1 is assigned to Faciltity 3. Because Faciltity 3 had no capacity and Customer 1 has 15 units of demand, Faciltity 3's capacity increases to 30 units as that is the smallest feasible capacity level after assigning Customer 1 to Faciltity 3. The solution cost is now \$5,525.00.



Because the Phase 1 capacity option is to force a minimum capacity level, Phase 1 evaluates each facility to determine if increasing the capacity level at the facility will reduce the solution cost. In this illustration, Facility 1 is increased to 60 units of capacity and Facility 3 is increased to 90 units of capacity. The solution cost is now $4,695.00 and Phase 1 has completed.



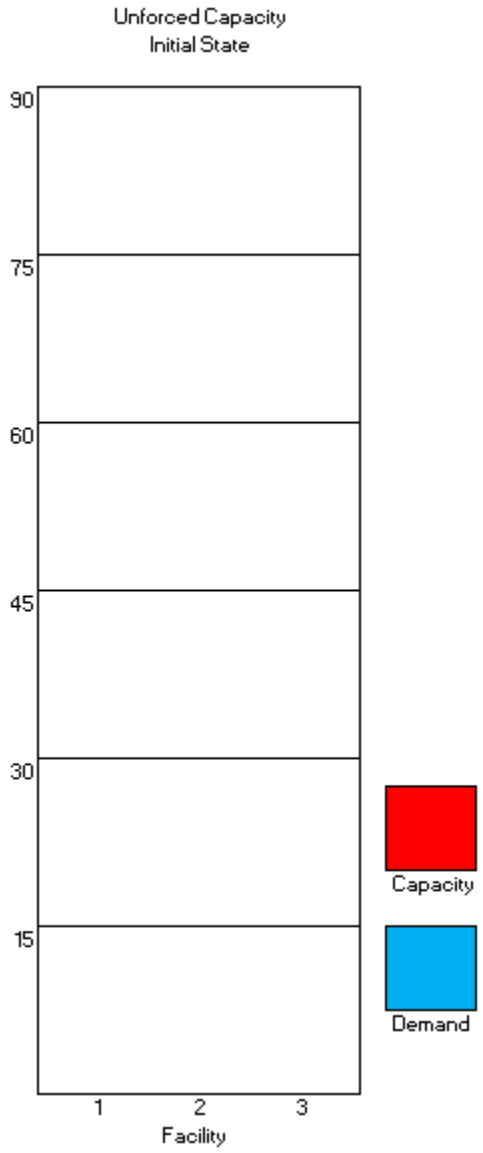
Phase 2 begins by checking if moving any single customer from one facility to another will improve the solution cost. This step found that moving Customer 3 from Faciltity 2 to Faciltity 3 and maintaining the same capacity levels was the best improvement in the solution cost. The solution cost is now \$3,770.00. Phase 2 will check for additional one-customer moves and will not find any. It will then check for a two-customer move and not find any improvement. It then checks to see if

swapping two customers will improve the solution cost and again does not find any improvement.

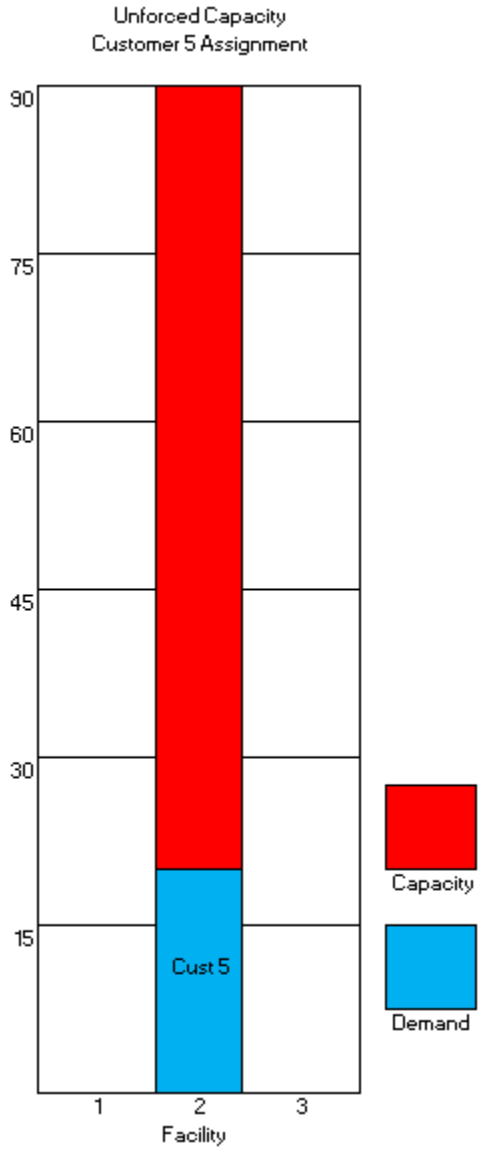
Because Phase 2 improved the solution cost in the previous round of checks, it starts checking again. Phase 2 will not find any one-customer moves. It will then check for a two-customer move and not find any improvement. It then checks to see if swapping two customers will improve the solution cost and again does not find any improvement.

Because Phase 2 did not find any improvements to the solution cost in the previous round of checks, it exits the heuristic with a solution cost of $3,770.00.

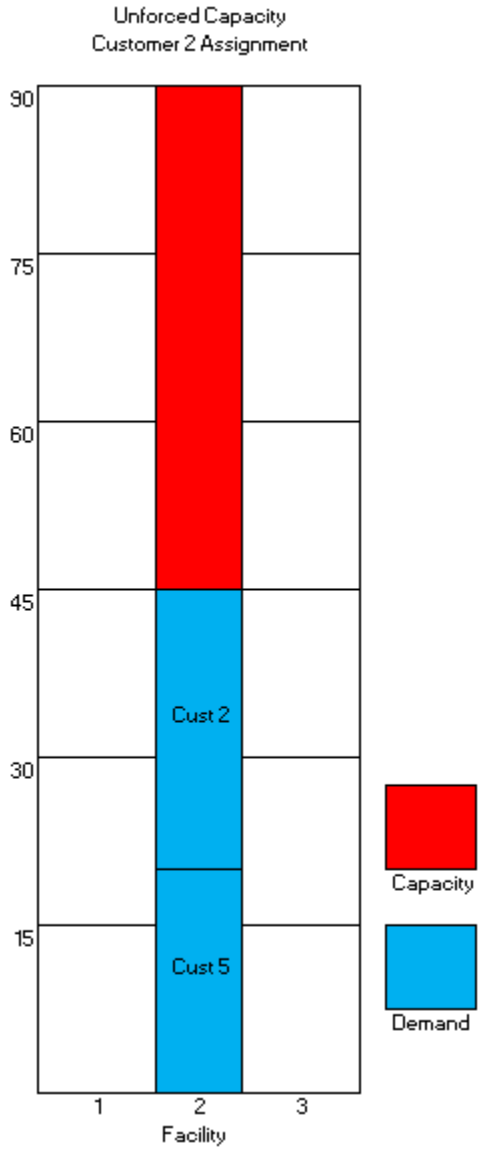
4.2.2. Unforced Capacity Illustration



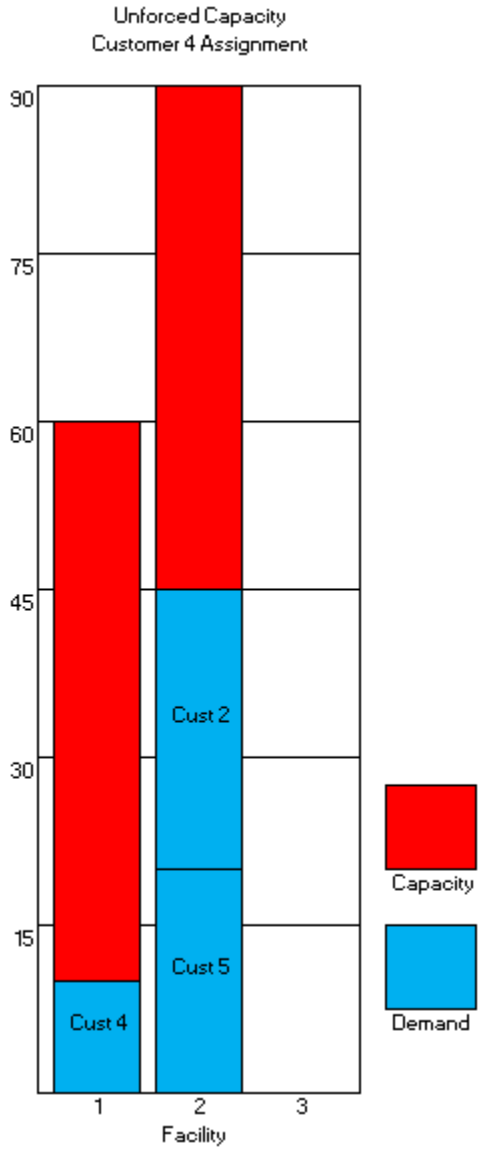
Unforced Capacity starts with no capacity assigned to any facility. The solution cost starts at $0.00.



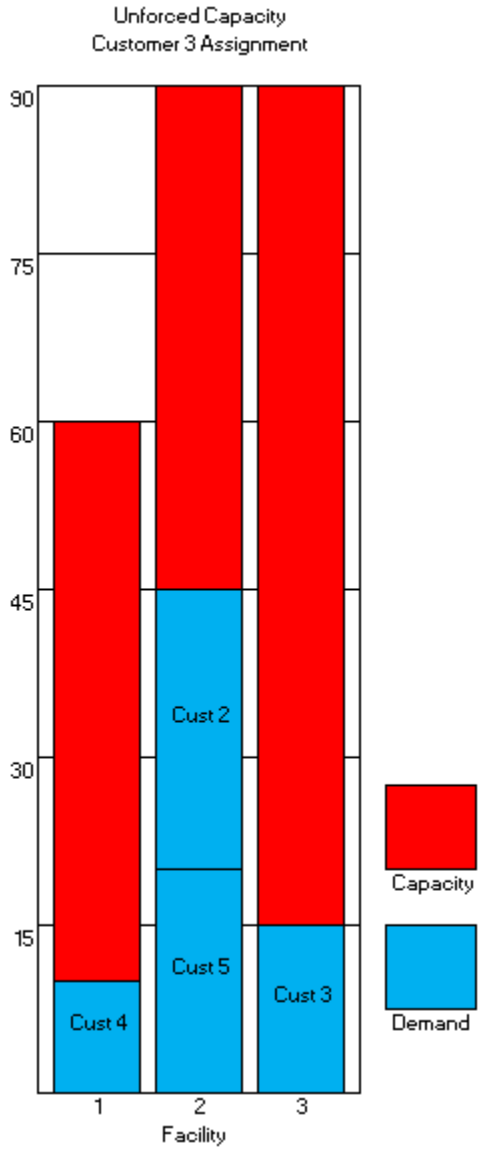
Customer 5 is assigned to Faciltity 2. With the addition of 20 units of demand, Faciltity 2's capacity increases to 90 units as that is the smallest increase in the solution cost. The solution cost is now \$955.71.



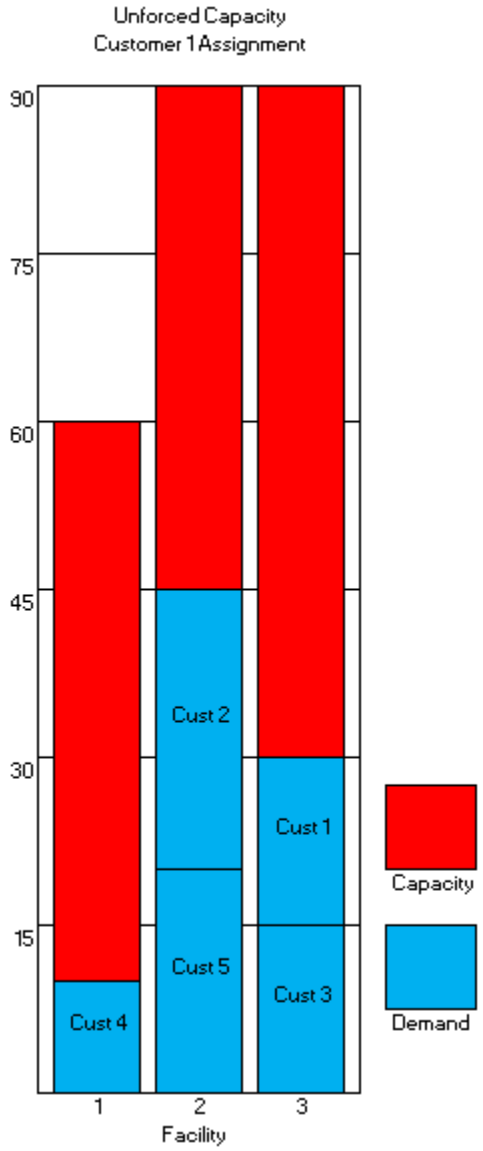
Customer 2 is assigned to Faciltity 2. Faciltity 2's capacity is already at the maximum available capacity, so it does not change. The solution cost is now $1,920.00.



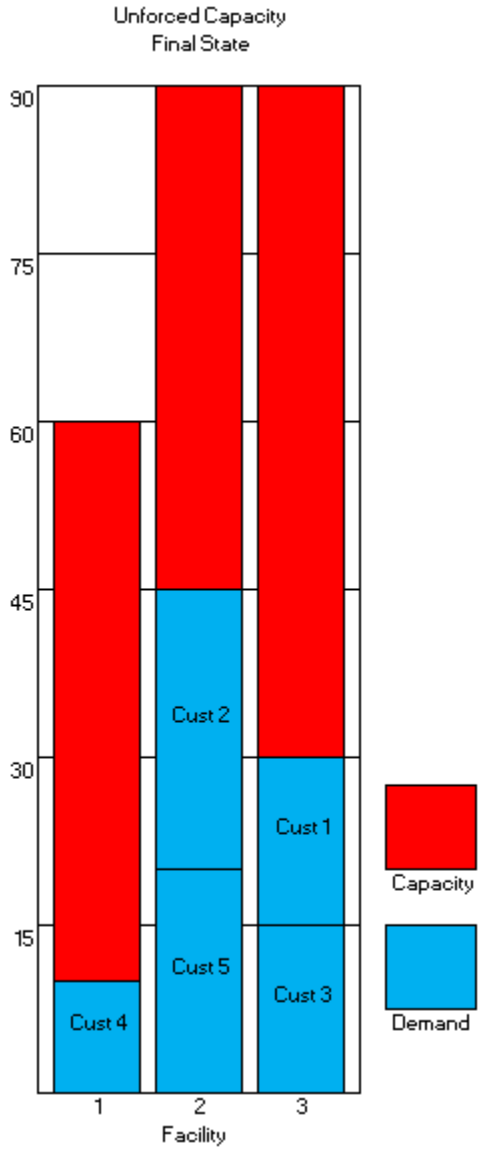
Customer 4 is assigned to Faciltity 1. With the addition of 10 units of demand, Faciltity 1's capacity increases to 60 units as that is the smallest increase in the solution cost. The solution cost is now $2,550.00.



Customer 3 is assigned to Faciltity 3. With the addition of 15 units of demand, Faciltity 3's capacity increases to 90 units as that is the smallest increase in the solution cost. The solution cost is now \$3,245.00.



Customer 1 is assigned to Faciltity 3. Faciltity 3's capacity is already at the maximum available capacity, so it does not change. The solution cost is now $3,770.00.



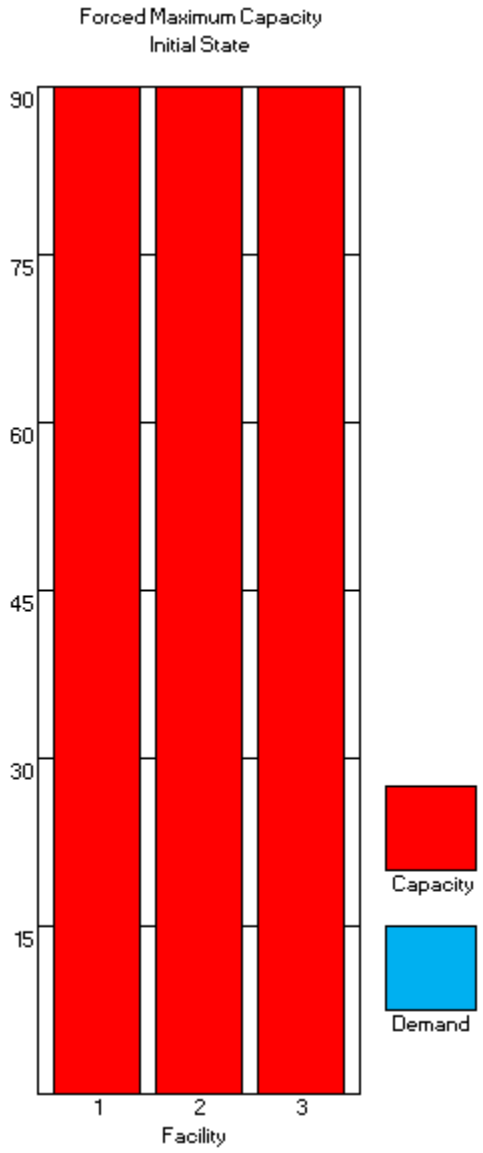
Because the Phase 1 capacity option is to update the capacity level based on the best cost at the time the customer is assigned, Phase 1 does not evaluate the solution for any capacity changes. The solution cost remains $3,770.00 and Phase 1 has completed.

Phase 2 will not find any one-customer moves. It will then check for a two-customer move and not find any improvement. It then checks to see if swapping two

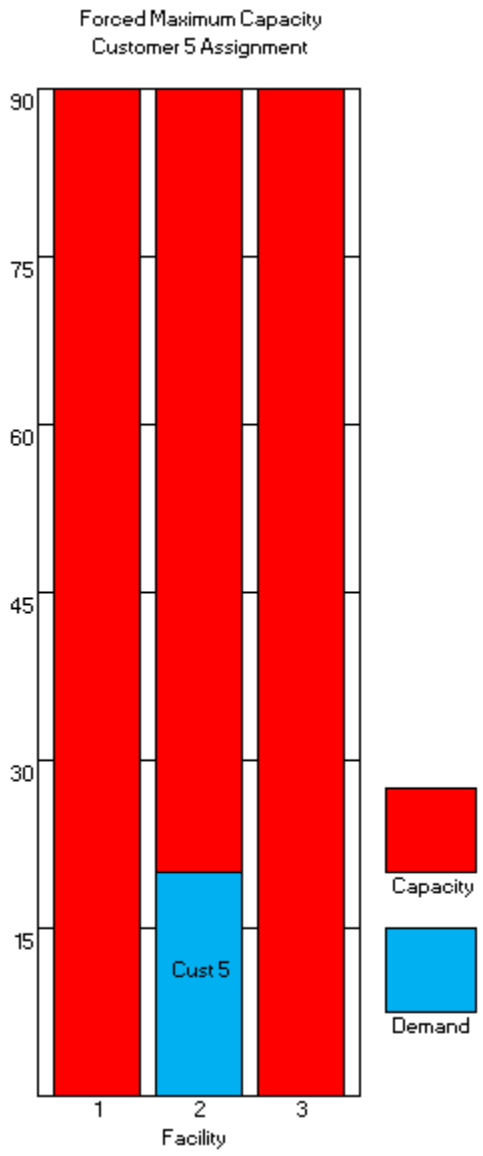
customers will improve the solution cost and again does not find any improvement.

Because Phase 2 did not find any improvements to the solution cost in the previous round of checks, it exits the heuristic with a solution cost of $3,770.00.

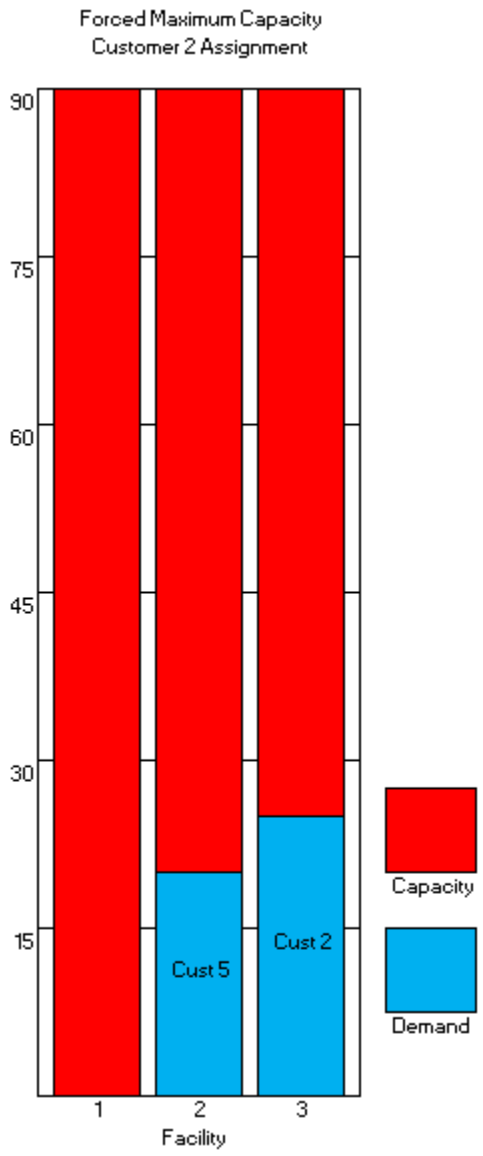
4.2.3. Forced Maximum Capacity Illustration



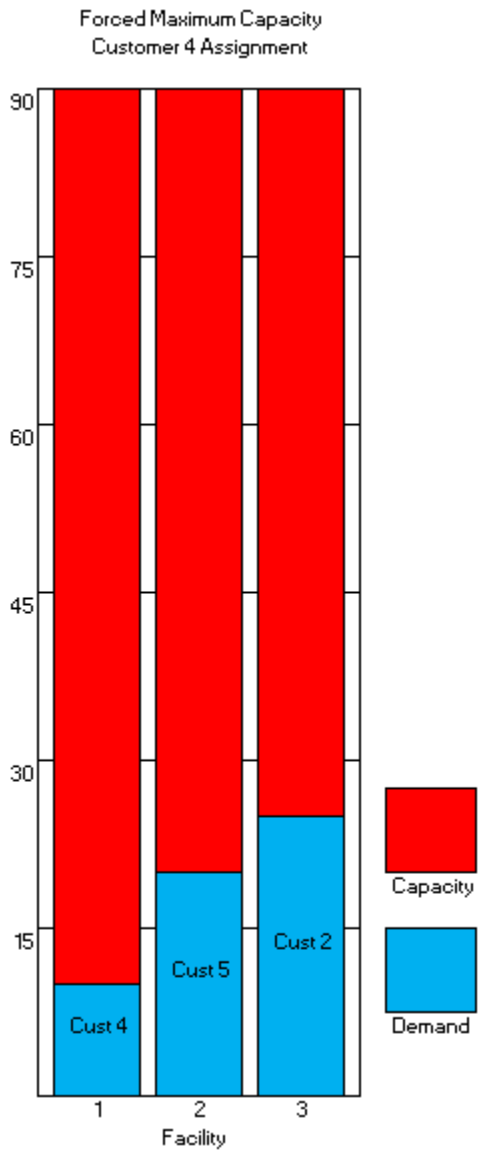
Forced Maximum Capacity starts with maximum capacity assigned to all facilities. The solution cost starts at $810.00.



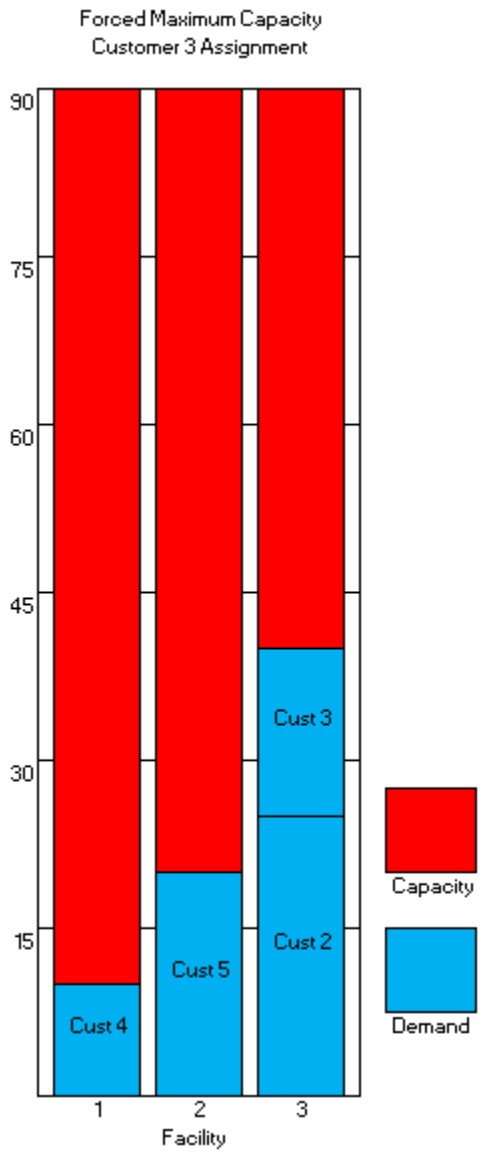
Customer 5 is assigned to Faciltity 2. The solution cost is now $1,495.71.



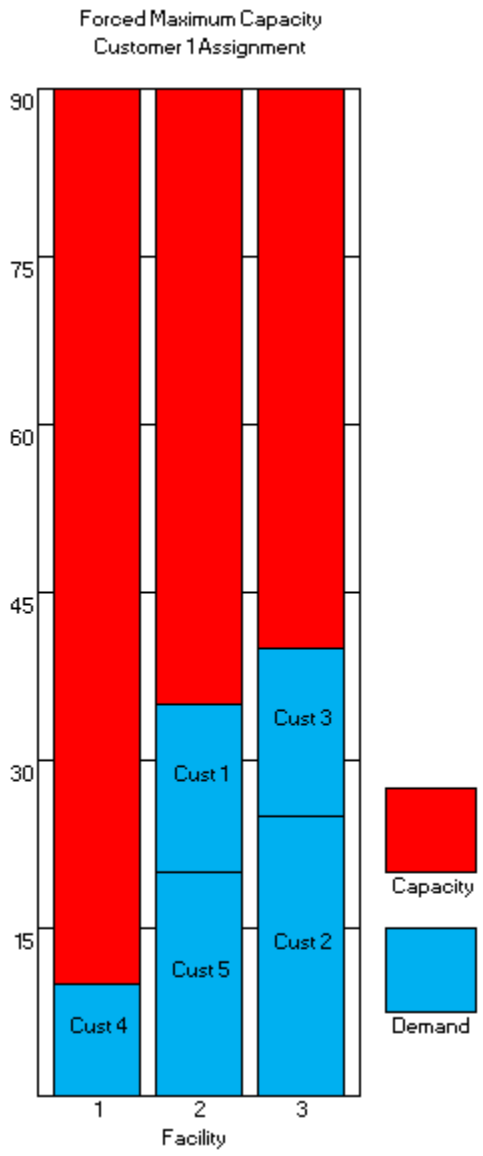
Customer 2 is assigned to Faciltity 3. The solution cost is now $2,380.33.



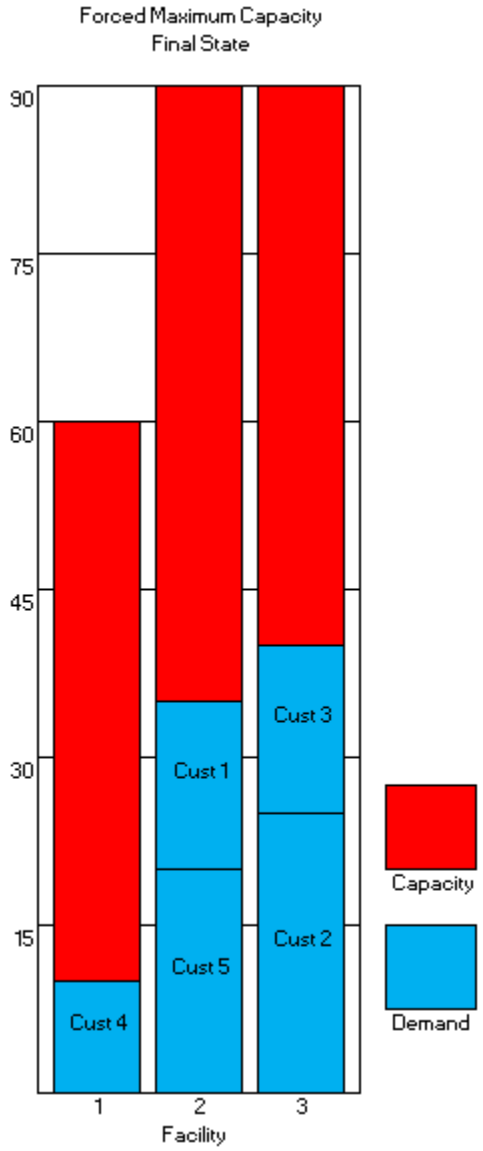
Customer 4 is assigned to Faciltity 1. The solution cost is now $2,755.33.



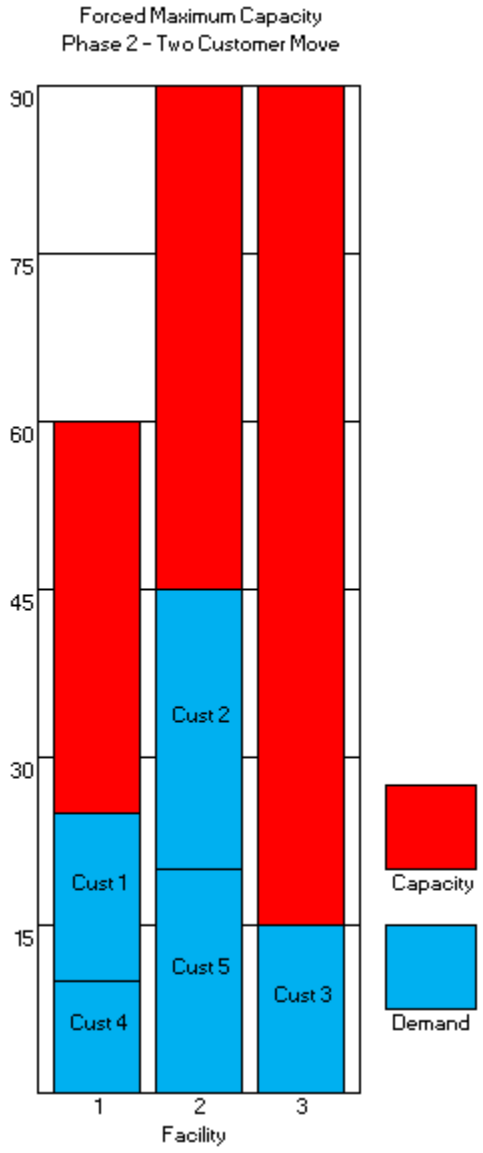
Customer 3 is assigned to Faciltity 3. The solution cost is now $3,395.71.



Customer 1 is assigned to Faciltity 2. The solution cost is now $3,896.36.

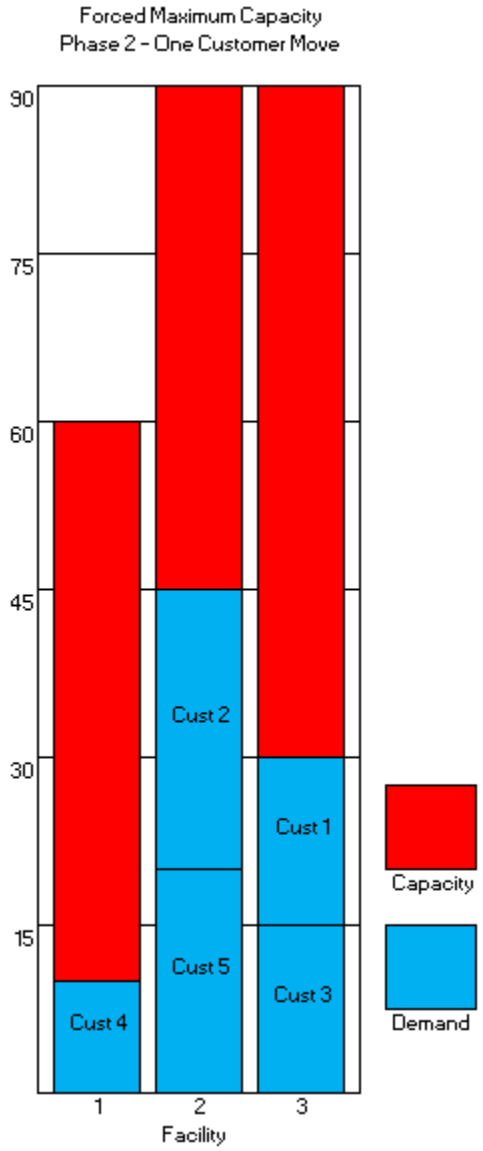


Because the Phase 1 capacity option is to force a maximum capacity level, Phase 1 evaluates each facility to determine if decreasing the capacity level at the facility will reduce the solution cost. In this illustration, Facility 1 is decreased to 60 units of capacity. The solution cost is now \$3,881.36 and Phase 1 has completed.



Phase 2 begins by checking if moving any single customer from one facility to another will improve the solution cost. This step found that moving Customer 1 to Facility 1 was the lowest cost move, but was still higher than current solution cost and thus did not update the solution. It will then check for a two-customer move and based on Customer 1 moving to Facility 1 being the lowest increase in cost, found that also moving Customer 2 to Facility 2 would produce a net reduction in the

solution cost. The solution cost is now $3,819.62. It then checks to see if swapping two customers will improve the solution cost and does not find any improvement.



Because Phase 2 improved the solution cost in the previous round of checks, it starts checking again. Phase 2's one-customer move step finds that moving Customer 1 from Faciltity 1 to Faciltity 3 and maintaining the same capacity levels was the best improvement in the solution cost. The solution cost is now $3,770.00. Phase

2 will check for additional one-customer moves and will not find any. It will then check for a two-customer move and not find any improvement. It then checks to see if swapping two customers will improve the solution cost and again does not find any improvement.

Because Phase 2 improved the solution cost in the previous round of checks, it starts checking again. Phase 2 will not find any one-customer moves. It will then check for a two-customer move and not find any improvement. It then checks to see if swapping two customers will improve the solution cost and again does not find any improvement.

Because Phase 2 did not find any improvements to the solution cost in the previous round of checks, it exits the heuristic with a solution cost of $3,770.00.

4.3. Multiple Use Routines

For the heuristic, calculating the cost is an important routine used throughout. The cost calculation (4.2) verifies the input solution of $\bar{x}$ and $\bar{y}$ represents a feasible solution (internally with v being set to 1 if feasible). Lines 5 through 20 check feasibility and the value of z is calculated if the solution is feasible, otherwise z is set to ∞.

Algorithm 4.2 *Calculate Cost*

Input: $\bar{x}, \bar{y}$

Output: z

1: $z \leftarrow \infty$

2: $v \leftarrow 1$

3: $U_j \leftarrow \sum_{k=1}^{K} \mu_{jk} y_{jk} \ \forall \ j \in \{1..n\}$ */* Calculate capacity by facility */*

4: $\Lambda_j \leftarrow \sum_{i=1}^{m} \lambda_i x_{ij} \ \forall \ j \in \{1..n\}$ */* Calculate demand by facility */*

/ Check Feasibility */*

5: ***for*** $j = 1$ ***to*** n ***do***

6: ***if*** $\Lambda_j > 0$ *and* $\Lambda_j \geq U_j$ ***then***

7: $v \leftarrow 0$

8: $j \leftarrow n + 1$

9: ***end if***

10: ***if*** $\sum_{k=1}^{K} y_{jk} > 1$ ***then***

11: $v \leftarrow 0$

12: $j \leftarrow n + 1$

13: ***end if***

14: ***end for***

15: ***for*** $i = 1$ ***to*** m ***do***

16: ***if*** $\sum_{j=1}^{n} x_{ij} \neq 1$ ***then***

17: $v \leftarrow 0$

18: $i \leftarrow m+1$

19: ***end if***

20: ***end for***

/ If feasible, calculate cost for all facilities having demand */*

21: ***if*** $v = 1$ ***then***

22: $z \leftarrow 0$

23: ***for*** $j = 1$ ***to*** n ***do***

24: ***if*** $\Lambda_j > 0$ ***then***

25: $z \leftarrow z + (\sum_{i=1}^{m} \lambda_i c_{ij} x_{ij}) + \left(t \frac{\Lambda_j}{U_j - \Lambda_j}\right) + \left(\sum_{k=1}^{K} f_{jk} y_{jk}\right)$

26: ***end if***

27: ***end for***

28: ***end if***

For Phase 2, the need to calculate the lowest cost movement of a customer from one facility to another is needed in multiple places. In this routine (4.3), the input includes the current solution ($\bar{x}$ and $\bar{y}$) along with a targeted customer (i) and the output includes the best solution ($\bar{x}$ and $\bar{y}$) with its cost (δ) when the targeted customer is assigned to a different facility. On line 10, the capacity for the new facility can only stay the same or increase as the demand at the new facility is increasing. Likewise on line 11, the capacity at the old facility can only stay the same or decrease as the demand at the old facility is decreasing. The capacity at the old facility is allowed to decrease to 0 to indicate the facility is no longer utilized.

Algorithm 4.3 *Calculate Move*

Input: $\bar{x}, \bar{y}, i$

Output: $\bar{x}, \bar{y}, \delta$

1: $\delta \leftarrow \infty$

2: $J \leftarrow \sum_{j=1}^{n} j x_{ij}$ */* Current facility for target customer */*

3: $V \leftarrow \sum_{k=1}^{K} k y_{Jk}$ */* Current capacity index of current facility */*

```
4: for j = 1 to n do
5:   if j ≠ J then
6:     ν ← Σ_{k=1}^{K} k y_{jk} /* Current capacity index of new facility */
       /* Update to new capacity level 1 if first use of new facility */
7:     if ν = 0 then
8:       ν ← 1
9:     end if
10:    for k = ν to K do
11:      for κ = V to 0 do
12:        χ̄ ← x̄
13:        γ̄ ← ȳ
14:        χ_{iJ} ← 0
15:        χ_{ij} ← 1
16:        γ_{JV} ← 0
17:        if κ ≠ 0 then
18:          γ_{Jκ} ← 1 /* Old facility still has capacity */
19:        end if
20:        γ_{jν} ← 0
21:        γ_{jk} ← 1
22:        Δ ← CalculateCost 4.2 (χ̄, γ̄)
           /* Update solution if new cost is lower */
23:        if Δ ≠ ∞ then
24:          if δ > Δ then
25:            x̄ ← χ̄
26:            ȳ ← γ̄
27:            δ ← Δ
28:          end if
29:        end if
```

30: ***end for***

31: ***end for***

32: ***end if***

33: ***end for***

4.4. Phase 1 Routines

For the Forced Minimum Capacity routine (4.4), lines 6 through 46 assign the customer to a facility, only increaing the capacity at the facility if required to stay feasible. Lines 47 through 62 increase the capacity at the facility after all customers have been assigned if increasing the capacity will reduce the solution cost.

Algorithm 4.4 *Forced Minimum Capacity*

Input: $\bar{L}$

Output: $\bar{x}, \bar{y}$

1: $x_{ij} \leftarrow 0 \ \forall \ i \in \{1..m\}, j \in \{1..n\}$

2: $y_{jk} \leftarrow 0 \ \forall \ j \in \{1..n\}, k \in \{1..K\}$

3: $\Lambda_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

4: $U_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

5: $F_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

/ Process each customer from sorted* $\bar{L}$ *list */*

6: ***for all*** $I \in \bar{L}$ ***do***

7: $J \leftarrow 0$

8: $V \leftarrow 0$

9: $z \leftarrow \infty$

10: ***for*** $j = 1$ ***to*** n ***do***

11: $l \leftarrow \Lambda_j + \lambda_I$ */* New demand at facility */*

12: $\gamma \leftarrow 0$

13: $\Omega \leftarrow U_j$ */* Current capacity of facility */*

14: $\nu \leftarrow \sum_{k=1}^{K} k y_{jk}$ /* *Current capacity index of facility* */

/* *Update new capacity variables if new demand is infeasible* */

15: ***if*** $l \geq \Omega$ ***then***

16: ***for*** $k = \nu + 1$ ***to*** K ***do***

17: ***if*** $l < \mu_{jk}$ ***then***

18: $\nu \leftarrow k$

19: $\gamma \leftarrow f_{jk} - F_j$ /* *Change in fixed cost* */

20: $\Omega \leftarrow \mu_{jk}$

21: $k = K + 1$

22: ***end if***

23: ***end for***

/* *If feasible, calculate increase in solution cost* */

24: ***if*** $l < \Omega$ ***then***

25: $\Delta \leftarrow \left(t\frac{l}{\Omega - l}\right)$ /* *New wait time cost* */

26: ***if*** $\Lambda_j > 0$ ***then***

27: $\Delta \leftarrow \Delta - \left(t\frac{\Lambda_j}{U_j - \Lambda_j}\right)$ /* *Old wait time cost* */

28: ***end if***

29: $\delta \leftarrow \lambda_I c_{Ij} + \Delta + \gamma$

/* *Update planned facility assignment if incrmental cost is lower* */

30: ***if*** $z > \delta$ ***then***

31: $J \leftarrow j$

32: $V \leftarrow \nu$

33: $z \leftarrow \delta$

34: ***end if***

35: ***end if***

36: ***end if***

37: ***end for***

/* *If feasible assignment found, update solution and facility-level variables* */

38: **if** $J > 0$ **then**

39: $x_{IJ} \leftarrow 1$

40: $y_{Jk} \leftarrow 0 \ \forall \ k \in \{1..K\}$

41: $y_{JV} \leftarrow 1$

42: $\Lambda_J \leftarrow \Lambda_J + \lambda_I$

43: $U_J \leftarrow \mu_{JV}$

44: $F_J \leftarrow f_{JV}$

45: **end if**

46: **end for**

47: $z \leftarrow CalculateCost\ 4.2\ (\bar{x}, \bar{y})$

/* *If feasible, update any facility capacity level to reduce cost* */

48: **if** $z \neq \infty$ **then**

49: **for** $j = 1$ **to** n **do**

50: $\nu \leftarrow \sum_{k=1}^{K} k y_{jk}$ /* *Current capacity index of facility* */

51: $\bar{Y} \leftarrow \bar{y}$

52: **for** $k = \nu + 1$ **to** K **do**

53: $Y_{jV} \leftarrow 0 \ \forall \ V \in \{1..K\}$

54: $Y_{jk} \leftarrow 1$

55: $\delta \leftarrow CalculateCost\ 4.2\ (\bar{x}, \bar{Y})$

/* *If solution cost decreased, update solution* */

56: **if** $z > \delta$ **then**

57: $\bar{y} \leftarrow \bar{Y}$

58: $z \leftarrow \delta$

59: **end if**

60: **end for**

61: **end for**

62: **end if**

For the Unforced Capacity routine (4.5), lines 6 through 40 assign the customer

to a facility and increase the capacity at the facility if needed to minimize the new solution cost.

Algorithm 4.5 *Unforced Capacity*

Input: $\bar{L}$

Output: $\bar{x}, \bar{y}$

1: $x_{ij} \leftarrow 0 \ \forall \ i \in \{1..m\}, j \in \{1..n\}$

2: $y_{jk} \leftarrow 0 \ \forall \ j \in \{1..n\}, k \in \{1..K\}$

3: $\Lambda_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

4: $U_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

5: $F_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

/ Process each customer from sorted $\bar{L}$ list */*

6: ***for all*** $I \in \bar{L}$ ***do***

7: $J \leftarrow 0$

8: $V \leftarrow 0$

9: $z \leftarrow \infty$

10: ***for*** $j = 1$ ***to*** n ***do***

11: $l \leftarrow \Lambda_j + \lambda_I$ */* New demand at facility */*

12: $\nu \leftarrow \sum_{k=1}^{K} k y_{jk}$ */* Current capacity index of facility */*

/ Update to capacity level 1 if first use of facility */*

13: ***if*** $\nu = 0$ ***then***

14: $\nu \leftarrow 1$

15: ***end if***

16: ***for*** $k = \nu$ ***to*** K ***do***

17: $\gamma \leftarrow f_{jk} - F_j$ */* Change in fixed cost */*

/ If feasible, calculate increase in solution cost */*

18: ***if*** $l < \mu_{jk}$ ***then***

19: $\Delta \leftarrow \left(t \frac{l}{\mu_{jk} - l} \right)$ */* New wait time cost */*

20: ***if*** $\Lambda_j > 0$ ***then***

21: $\Delta \leftarrow \Delta - \left(t\frac{\Lambda_j}{U_j - \Lambda_j}\right)$ /* *Old wait time cost* */

22: ***end if***

23: $\delta \leftarrow \lambda_I c_{Ij} + \Delta + \gamma$

/* *Update planned facility assignment if incrmental cost is lower* */

24: ***if*** $z > \delta$ ***then***

25: $J \leftarrow j$

26: $V \leftarrow k$

27: $z \leftarrow \delta$

28: ***end if***

29: ***end if***

30: ***end for***

31: ***end for***

/* *If feasible assignment found, update solution and facility-level variables* */

32: ***if*** $J > 0$ ***then***

33: $x_{IJ} \leftarrow 1$

34: $y_{Jk} \leftarrow 0 \ \forall \ k \in \{1..K\}$

35: $y_{JV} \leftarrow 1$

36: $\Lambda_J \leftarrow \Lambda_J + \lambda_I$

37: $U_J \leftarrow \mu_{JV}$

38: $F_J \leftarrow f_{JV}$

39: ***end if***

40: ***end for***

For the Forced Maximum Capacity routine (4.6), lines 5 through 26 assign the customer to a facility where the facilities are already assigned their maximum capacities. Lines 27 through 45 decrease the capacity at the facility after all customers have been assigned if decreasing the capacity will reduce the solution cost.

Algorithm 4.6 *Forced Maximum Capacity*

Input: $\bar{L}$

Output: $\bar{x}, \bar{y}$

1: $x_{ij} \leftarrow 0 \ \forall \ i \in \{1..m\}, j \in \{1..n\}$

2: $y_{jk} \leftarrow 0 \ \forall \ j \in \{1..n\}, k \in \{1..K-1\}$

3: $y_{jK} \leftarrow 1 \ \forall \ j \in \{1..n\}$ */* Set all facilities to maximum capacity */*

4: $\Lambda_j \leftarrow 0 \ \forall \ j \in \{1..n\}$

/ Process each customer from sorted $\bar{L}$ list */*

5: ***for all*** $I \in \bar{L}$ ***do***

6: $J \leftarrow 0$

7: $z \leftarrow \infty$

8: ***for*** $j = 1$ ***to*** n ***do***

9: $l \leftarrow \Lambda_j + \lambda_I$ */* New demand at facility */*

/ If feasible, calculate increase in solution cost */*

10: ***if*** $l < \mu_{jK}$ ***then***

11: $\Delta \leftarrow \left(t \frac{l}{\mu_{jK} - l} \right)$ */* New wait time cost */*

12: ***if*** $\Lambda_j > 0$ ***then***

13: $\Delta \leftarrow \Delta - \left(t \frac{\Lambda_j}{\mu_{jK} - \Lambda_j} \right)$ */* Old wait time cost */*

14: ***end if***

15: $\delta \leftarrow \lambda_I c_{Ij} + \Delta$

/ Update planned facility assignment if incrmental cost is lower */*

16: ***if*** $z > \delta$ ***then***

17: $J \leftarrow j$

18: $z \leftarrow \delta$

19: ***end if***

20: ***end if***

21: ***end for***

/ If feasible assignment found, update solution and facility-level variables */*

```
    if J > 0 then
      x_IJ ← 1
      Λ_J ← Λ_J + λ_I
    end if
  end for
  z ← CalculateCost 4.2 (x̄, ȳ)
  /* If feasible, update any facility capacity level to reduce cost */
  if z ≠ ∞ then
    for j = 1 to n do
      if Λ_j = 0 then
        y_jK ← 0 /* Remove capacity from facility if facility has no demand */
      else
        Ȳ ← ȳ
        for k = K − 1 to 1 do
          Y_jV ← 0 ∀ V ∈ {1..K}
          Y_jk ← 1
          δ ← CalculateCost 4.2 (x̄, Ȳ)
          /* If solution cost decreased, update solution */
          if z > δ then
            ȳ ← Ȳ
            z ← δ
          end if
        end for
      end if
    end for
  end if
```

4.5. Phase 2 Routines

The One Customer Move routine (4.7) takes an initial list of all customers ($\bar{L}$) and analyzes the list for the best customer to reassign to a new facility. The best customer for that iteration is removed from the list until the list is empty or no feasible reassignment can be made. If the best customer move increases the solution cost, all customers with the same demand, facility, and cost are removed from the list (lines 26 to 32) and the internal working soltuon of $\bar{X}$ and $\bar{Y}$ are updated but the output solution is not updated unless future iterations drop the new solution cost below the current best solution cost.

Algorithm 4.7 *One Customer Move*

Input: $\bar{x}, \bar{y}$

Output: $\bar{x}, \bar{y}$

/ $\bar{X}$ and $\bar{Y}$ represent working solution, not output solution */*

1: $\bar{X} \leftarrow \bar{x}$

2: $\bar{Y} \leftarrow \bar{y}$

3: $\bar{L} \leftarrow \{1..m\}$ */* Initialize list $\bar{L}$ to all customers */*

4: ***while*** $\bar{L} \neq \emptyset$ ***do***

5: $I \leftarrow 0$

6: $Z \leftarrow CalculateCost$ *4.2* $(\bar{x}, \bar{y})$

7: $z \leftarrow \infty$

8: $\bar{\chi} \leftarrow \bar{X}$

9: $\bar{\gamma} \leftarrow \bar{Y}$

/ $\bar{L}$ shrinks as customers with best solution are removed from list */*

10: ***for all*** $i \in \bar{L}$ ***do***

11: $(\bar{\rho}, \bar{\sigma}, \delta) \leftarrow CalculateMove$ *4.3* $(\bar{X}, \bar{Y}, i)$

/ Store best customer and solution across all customers still in $\bar{L}$ */*

12: ***if*** $\delta \neq \infty$ ***then***

13: **if** $z > \delta$ **then**

14: $I \leftarrow i$

15: $z \leftarrow \delta$

16: $\bar{\chi} \leftarrow \bar{\rho}$

17: $\bar{\gamma} \leftarrow \bar{\sigma}$

18: **end if**

19: **end if**

20: **end for**

/* If feasible solution found, store as new working solution */

21: **if** $z \neq \infty$ **then**

22: $\bar{L} \leftarrow \bar{L} \setminus \{I\}$

23: $\bar{X} \leftarrow \bar{\chi}$

24: $\bar{Y} \leftarrow \bar{\gamma}$

/* If working solution is higher, just remove similar customers from $\bar{L}$ */

25: **if** $z \geq Z$ **then**

26: $J \leftarrow \sum_{p=1}^{n} pX_{Ip}$ /* Current facility for best customer */

/* Check all customers */

27: **for** $o = 1$ **to** m **do**

28: $j \leftarrow \sum_{p=1}^{n} pX_{op}$ /* Current facility for checked customer */

/* Remove checked customer if same demand, facility, and cost */

29: **if** $\lambda_o = \lambda_I$ *and* $j = J$ *and* $c_{oj} = c_{IJ}$ **then**

30: $\bar{L} \leftarrow \bar{L} \setminus \{o\}$

31: **end if**

32: **end for**

33: **else**

34: $\bar{x} \leftarrow \bar{X}$ /* Save working solution as output solution if cost decreased */

35: $\bar{y} \leftarrow \bar{Y}$

36: **end if**

37: **else**

38: $\bar{L} \leftarrow \emptyset$

39: **end if**

40: **end while**

The Two Customer Move routine (4.8) finds the best customer to be reassigned to a new facility (lines 8 to 18) and the uses that solution to find the best customer to reassign to a new facility—that is not the same customer (lines 25 to 34)—to see if the new overall solution cost is less than the current solution cost in an attempt to get out of a potentially local optimum [16]. If a better solution is found, the routine terminates. Otherwise, the previously best customer from the first loop is removed from the customer list and the routine tries again.

Algorithm 4.8 *Two Customer Move*

Input: $\bar{x}, \bar{y}$

Output: $\bar{x}, \bar{y}$

1: $Z \leftarrow CalculateCost\ 4.2\ (\bar{x}, \bar{y})$

2: $\bar{L} \leftarrow \{1..m\}$ /* *Initialize list* $\bar{L}$ *to all customers* */

3: **while** $\bar{L} \neq \emptyset$ **do**

4: $I \leftarrow 0$

5: $z \leftarrow \infty$

6: $\bar{\chi} \leftarrow \bar{x}$

7: $\bar{\gamma} \leftarrow \bar{y}$

/* $\bar{L}$ *shrinks as customers with best solution are removed from list* */

8: **for all** $i \in \bar{L}$ **do**

9: $(\bar{\rho}, \bar{\sigma}, \delta) \leftarrow CalculateMove\ 4.3\ (\bar{x}, \bar{y}, i)$

/* *Store best customer and solution across all customers still in* $\bar{L}$ */

10: **if** $\delta \neq \infty$ **then**

11: **if** $z > \delta$ **then**

12: $I \leftarrow i$

13: $z \leftarrow \delta$

14: $\bar{\chi} \leftarrow \bar{\rho}$

15: $\bar{\gamma} \leftarrow \bar{\sigma}$

16: ***end if***

17: ***end if***

18: ***end for***

/ If feasible, do same steps on all other customers */*

19: ***if*** $z \neq \infty$ ***then***

20: $\bar{L} \leftarrow \bar{L} \setminus \{I\}$

21: $z \leftarrow \infty$

22: $\bar{X} \leftarrow \bar{\chi}$

23: $\bar{Y} \leftarrow \bar{\gamma}$

24: ***for*** $i = 1$ ***to*** m ***do***

25: ***if*** $I \neq i$ ***then***

26: $(\bar{\rho}, \bar{\sigma}, \delta) \leftarrow CalculateMove\ 4.3\ (\bar{\chi}, \bar{\gamma}, i)$

27: ***if*** $\delta \neq \infty$ ***then***

28: ***if*** $z > \delta$ ***then***

29: $z \leftarrow \delta$

30: $\bar{X} \leftarrow \bar{\rho}$

31: $\bar{Y} \leftarrow \bar{\sigma}$

32: ***end if***

33: ***end if***

34: ***end if***

35: ***end for***

/ If feasible and has a lower cost, store solution and stop */*

36: ***if*** $z \neq \infty$ ***then***

37: ***if*** $z < Z$ ***then***

38: $\bar{x} \leftarrow \bar{X}$

39: $\bar{y} \leftarrow \bar{Y}$

40: $\bar{L} \leftarrow \emptyset$

41: ***end if***

42: ***end if***

43: ***else***

44: $\bar{L} \leftarrow \emptyset$

45: ***end if***

46: ***end while***

The Swap Customers routine (4.9) loops through all customer combinations (lines 7 and 9) where the customers are in different facilities (line 11) to determine if swapping those customers would decrease the cost. The overall routine continues to loop as long as the solution cost is decreasing and a feasible swap is found.

Algorithm 4.9 *Swap Customers*

Input: $\bar{x}, \bar{y}$

Output: $\bar{x}, \bar{y}$

1: $Z \leftarrow CalculateCost\ 4.2\ (\bar{x}, \bar{y})$

2: $z \leftarrow 0$

/ Loop while improving solution */*

3: ***while*** $z < Z$ ***do***

4: $z \leftarrow \infty$

5: $\bar{\chi} \leftarrow \bar{x}$

6: $\bar{\gamma} \leftarrow \bar{y}$

/ Every customer combination, $I > i$ */*

7: ***for*** $i = 1$ ***to*** m ***do***

8: $j \leftarrow \sum_{p=1}^{n} p x_{ip}$ */* Current facility for customer i */*

9: ***for*** $I = i + 1$ ***to*** m ***do***

10: $J \leftarrow \sum_{p=1}^{n} px_{Ip}$ /* *Current facility for customer I* */

/* *Only check if customers are assigned to different facilities* */

11: ***if*** $j \neq J$ ***then***

12: ***for*** $k = 1$ ***to*** K ***do***

13: ***for*** $\nu = 1$ ***to*** K ***do***

14: $\bar{X} \leftarrow \bar{x}$

15: $\bar{Y} \leftarrow \bar{y}$

/* *Swap customers and set capacity levels being tested* */

16: $X_{ij} \leftarrow 0$

17: $X_{iJ} \leftarrow 1$

18: $X_{IJ} \leftarrow 0$

19: $X_{Ij} \leftarrow 1$

20: $Y_{jq} \leftarrow 0 \ \forall \ q \in \{1..K\}$

21: $Y_{jk} \leftarrow 1$

22: $Y_{Jq} \leftarrow 0 \ \forall \ q \in \{1..K\}$

23: $Y_{J\nu} \leftarrow 1$

24: $\delta \leftarrow CalculateCost\ 4.2\ (\bar{X}, \bar{Y})$

/* *Store best solution across all customer combinations* */

25: ***if*** $\delta \neq \infty$ ***then***

26: ***if*** $\delta < z$ ***then***

27: $z \leftarrow \delta$

28: $\bar{\chi} \leftarrow \bar{X}$

29: $\bar{\gamma} \leftarrow \bar{Y}$

30: ***end if***

31: ***end if***

32: ***end for***

33: ***end for***

34: ***end if***

35: ***end for***

36: ***end for***

/ If feasible and has a lower cost, store solution and try again */*

37: ***if*** $z \neq \infty$ ***then***

38: ***if*** $z < Z$ ***then***

39: $\bar{x} \leftarrow \bar{\chi}$

40: $\bar{y} \leftarrow \bar{\gamma}$

41: $Z \leftarrow z$

42: $z \leftarrow 0$

43: ***end if***

44: ***else***

45: $z \leftarrow Z$

46: ***end if***

47: ***end while***

4.6. Heuristic Complexity Analysis

For the complexity analysis, the number of customers is assumed to be larger than the number of facilities—which is assumed to be larger than the number of capacity levels. While this is not true for every data set, it is the most common scenario. Given this assumption, the complexity for Phase 1 of the heuristic was calculated as $O\left(mn^2K\right)$ and the complexity for Phase 2 of the heuristic was calculated as $O\left(m^3n^2K^2\right)$.

Chapter 5

Data Sets

For comparison purposes, the 695 data sets being tested were broken into 6 logical groupings. The groupings either pull from the same data source, or differ based on how the costs were structured. This assists with determining if certain types of data sets increase or decrease the difficulty of finding a globally optimal solution with the solution approaches described in Chapters 3 and 4. We also consider the number of solutions that would need to be evaluated to find an optimal solution for a given data set using an enumeration or brute-force approach.

The calculation for the brute-force solution size is based on multiplying the total possible facility assignment solutions (n^m) and the total possible facility capacity settings ($(K+1)^n$—from 0 to K as a facility could be unused) to get $n^m (K+1)^n$. Given the size of some brute-force solution spaces, the detailed data shows the value as a power of 10—as some of the values were too large for Excel to calculate.

5.1. Holmberg

The Holmberg data sets come from Holmberg et al. in 1999 [28] and were provided for research here as a set of data that has been analyzed using various other methodologies previously. It consists of thirty distinct data points of four subsets of data. Each data set within the subsets varies t using $\beta \in \{0.1, 1.0, 10.0\}$, where t is calculated as the maximum $c_{ij}\lambda_i$ value.

The first subset of data contains 4 data sets of 50 customers, 10 facilities, and 3 capacity levels that are varied by μ_{jk} between the 4 data sets. They represent a

data set that could have 10^{50} possible facility assignments. Since only μ_{jk} varies, the t value would be consistent across all four data sets—and then modified by β. This block of twelve data sets represents medium sized set of customers and how t and μ_{jk} affect difficulty.

The second subset of data contains 1 data set of 50 customers, 20 facilities, and 3 capacity levels. It represent a data set that could have 20^{50} possible facility assignments. The t value is modified by β. This block of three data sets represents medium sized set of customers with a large number of facilities and how t affects difficulty.

The third subset of data contains 3 data sets of 90 customers, 10 facilities, and 3 capacity levels that are varied by λ_i between the 3 data sets. They represent a data set that could have 10^{90} possible facility assignments. Since only λ_i varies, the t value would also vary across all three data sets—and then modified by β. This block of nine data sets represents large sized set of customers and how t and λ_i affect difficulty.

The fourth subset of data contains 2 data sets of 100 customers, 10 facilities, and 3 capacity levels that are varied by λ_i between the 2 data sets. They represent a data set that could have 10^{100} possible facility assignments. Since only λ_i varies, the t value would also vary across all three data sets—and then modified by β. This block of six data sets represents extra-large sized set of customers and how t and λ_i affect difficulty.

The brute-force solution space size for the Holmberg group ranges from $10^{56.02}$ to $10^{106.02}$. The Phase 1 Complexity (mn^2K) ranges from $15,000$ to $60,000$. The Phase 2 Complexity ($m^3n^2K^2$) ranges from $112,500,000$ to $900,000,000$.

5.2. Small, Varying Costs

The Small, Varying Costs data sets were a single data set of 25 customers, 5 facilities, and 3 capacity levels that are varied in 3 subsets to get 36 data points. It

represents a data set that could have 5^{25} possible facility assignments. The value for t is fixed to 600 and varied by $\beta \in \{0.1, 1.0, 10.0, 100.0\}$. The λ_i and c_{ij} values were specifically chosen to provide multiple groupings of customers with similar facility assignment costs.

The first subset of data contains 3 data sets that are varied by μ_{jk}. This block of twelve data sets shows a small sized set of customers and how t and μ_{jk} affect difficulty.

The second subset of data contains 3 data sets that are varied by f_{jk}. This block of twelve data sets shows a small sized set of customers and how t and f_{jk} affect difficulty.

The third subset of data contains 3 data sets that are varied by c_{ij}. This block of twelve data sets shows a small sized set of customers and how t and c_{ij} affect difficulty.

The brute-force solution space size for the Small, Varying Costs group is $10^{20.48}$. The Phase 1 Complexity is $1,875$. The Phase 2 Complexity is $3,515,625$.

5.3. Various, Linear Costs

The Various, Linear Costs data sets represent actual client data from six companies of various sizes. The identifying information was scrubbed from the data and they were labeled A through F based on size from smallest to largest in terms of application usage ($\sum \lambda_i$).

Given that the number of "customers" (e.g. business processes) could vary, and wanting a consistent way to build each data set, the business processes were grouped into 18, 36, or 72 "customers" based on similarities. B2Bi can have up to nine priority queues used by the business processes, so the "facilities" were set to either three, six, or nine. The capacity levels are five, seven, and nine—which represent the number

of threads to assign to the priority queue times a multiplier for the client. The value for t was arbitrarily set to 600 and varied by $\beta \in \{0.1, 1.0, 10.0\}$.

Since that would give 81 possible combinations of settings per client, a Taguchi L9 Orthogonal Array ([43]) was used to get to a set of 9 combinations per client—giving a total of 54 data points in this group. For each client, the L9 array is:

Table 5.1. Various, Linear Cost Taguchi L9 Orthogonal Array

L	**Customers**	**Facilities**	**Capacities**	β
1	72	9	5	0.1
2	72	6	7	1.0
3	72	3	9	10.0
4	36	9	7	10.0
5	36	6	9	0.1
6	36	3	5	1.0
7	18	9	9	1.0
8	18	6	5	10.0
9	18	3	7	0.1

The c_{ij} values were calculated as 0 at the preferred facility and then an additional 25 as the customer moves further away from the preferred facility. This was to keep the cost linear while trying to keep the customer close to its preferred facility. The μ_{jk} values were based on 1500 times the multiplier for the client and then linearly incremented for each capacity level.

The f_{jk} values started with 0 at the first capacity level and then incremented linearly based on a value for that facility. The base value for each facility favored "higher" facilities with lower f_{jk} values, making it easier to add capacity to those facilities. A Fibonacci Sequence ([34]) was used to calculate these values so that the

three, six, or nine facilities used for the data point scaled appropriately. The f_{jk} values were not varied by client as more critical business processes had a "higher" preferred facility.

The brute-force solution space size for the Various, Linear Costs group ranges from $10^{11.30}$ to $10^{75.71}$. The Phase 1 Complexity ranges from $1,134$ to $29,160$. The Phase 2 Complexity ranges from $2,571,912$ to $755,827,200$.

5.4. Various, Nonlinear Costs

The same set of client data from Various, Linear Costs was used to get an additional set of 54 data points using nonlinear costs. Customers were still grouped into 18, 36, or 72 and t still varied using the same β values. However, the number of facilities was set to 9 and the number of capacity levels was set to 5. This provides an explicit nine data points per client. The goal was to determine how nonlinear costs effected the solution times.

The c_{ij} values were calculated as 25 at the preferred facility and then doubled moving away to a "lower" facility and tripled moving away to a "higher" facility—making the model prefer to send a customer to a "lower" facility over a "higher" facility, which had been equally likely in the linear costs.

The μ_{jk} values began with the same capacity as before, but doubled at each step instead of increasing linearly. This matches the default standard of assigning powers of 2 as the number of threads to the priority queues in B2Bi. Likewise, the f_{jk} values were similarly scaled for consistency.

The brute-force solution space size for the Various, Nonlinear Costs group ranges from $10^{24.18}$ to $10^{75.71}$. The Phase 1 Complexity ranges from $7,290$ to $29,160$. The Phase 2 Complexity ranges from $11,809,800$ to $755,827,200$.

5.5. Sizing

The Sizing data sets were generated to determine how well the heuristic and mathematical models could handle larger data sets. The number of customers was in the set $\{250, 500, 750, 1000, 2500, 5000, 7500, 10000\}$. The number of facilities was in the set $\{25, 50, 75, 100\}$. The number of capacity levels was in the set $\{5, 10, 15, 20\}$. The value for t was arbitrarily set to 600 and varied by $\beta \in \{0.1, 1.0, 10.0, 100.0\}$. This generated 512 data sets.

For each customer size, facility size, and capacity level size, the λ_i for each customer was calculated as a random value from 5 to 50. The sum of the demand was used to calculate an average needed capacity per facility ($\bar{\mu} = \frac{\sum_{i=1}^{m} \lambda_i}{n}$). For each facility, the actual maximum capacity was calculated as the average needed capacity times a random value ($\hat{r}$) between 1.5 and 2.0 ($\mu_{jK} = \bar{\mu}\hat{r}$) and rounded up to the nearest multiple of 60 so it would be evenly divisible by the number of capacity levels. The capacity levels were then linearly distributed. This calculation ensured the existence of a feasible solution.

The facility assignment cost was calculated by randomly distributing all facilities and customers on a 1000x1000 grid and calculating the next highest integer of the distance between the customers and facilities plus 1 (to ensure no zero costs). The faciltity capacity cost was calculated with a base facility opening cost of a random value ($\bar{r}$) between 200 and 400 plus a concave cost function of the capacity ($\bar{r}+5\sqrt{\mu_{jk}}$).

The brute-force solution space size for the Sizing group ranges from $10^{368.94}$ to $10^{20,132.22}$. The Phase 1 Complexity ranges from $781,250$ to $2,000,000,000$. The Phase 2 Complexity ranges from $244,140,625,000$ to $4,000,000,000,000,000,000$.

5.6. Saw-Tooth

The Saw-Tooth data sets were derived from a single problem instance with 5 customers, 3 facilities, and 3 capacity levels that are varied to get 9 data sets. This ultra-small data set was created for the sole purpose of demonstrating how the customer wait-time cost affects the difficulty of solving even a small problem. A brute-force analysis was done in Excel. The term "Saw-Tooth" derives from the visualization of the solution space with the solutions initially sorted based on the facility assignment cost.

A problem instance in this data set that could have 243 possible facility assignments and 64 facility capacities for a total of $15,552$ total solutions—including infeasible solutions. The Phase 1 Complexity is 135 and the Phase 2 Complexity is $10,125$. The value for t is fixed to 1000 and varied by $\beta \in \{0.1, 1.0, 10.0\}$. The capacity distributions were varied using the set $\{\{30, 60, 90\}, \{40, 80, 120\}, \{30, 60, 120\}\}$ to try and determine if linear versus nonlinear capacity levels will make a difference in the practical (as opposed to theoretical) difficulty of the problem. In the example solution times below, the Colley Model was used for timing to avoid the potential issue of variability of the number of iterations in the Elhedhli Models.

The Linear-90 capacity distribution has capacity levels of 30, 60, and 90 for each faciltity. This represents a more constrained data set as there are more infeasible points in the solution space. When the adjusted value for the customer wait-time cost is lowest (i.e $\beta = 0.1 \Rightarrow t = 100$), the average solution time is 0.057 seconds. As the value for t increases by a factor of 10, the average solution time increases to 0.083 seconds and then 0.117 seconds. While all of these times are subsecond, that is due mainly to the ultra-small nature of the data. The increases in solution time as a factor of the initial solution time are relatively large: 1.44 and 2.04.

The Linear-120 capacity distribution has capacity levels of 40, 80, and 120 for each faciltity. This represents a less constrained data set as there are fewer infeasible points in the solution space. When the adjusted value for the customer wait-time cost is lowest (i.e $\beta = 0.1 \Rightarrow t = 100$), the average solution time is 0.060 seconds. As the value for t increases by a factor of 10, the average solution time increases to 0.074 seconds and then 0.105 seconds. While all of these times are subsecond, that is due mainly to the ultra-small nature of the data. The increases in solution time as a factor of the initial solution time are again relatively large: 1.22 and 1.75.

The Nonlinear-120 capacity distribution has capacity levels of 30, 60, and 120 for each faciltity. This demonstrates a doubling of the capacity level which adds some infeasible points back to the solution space. When the adjusted value for the customer wait-time cost is lowest (i.e $\beta = 0.1 \Rightarrow t = 100$), the average solution time is 0.061 seconds. As the value for t increases by a factor of 10, the average solution time increases to 0.067 seconds and then 0.123 seconds. While all of these times are subsecond, that is due mainly to the ultra-small nature of the data. The increases in solution time as a factor of the initial solution time are again relatively large: 1.11 and 2.02.

From observation of the solutions times versus the capacity distribution and t values, a t value 10 times larger has more impact on the solution time for a more contrainsted capacity distribution while a t value 100 times larger has a noticeable impact on solution time for all capacity distributions.

5.7. Data Sets in Existing Literature

The solution techniques described in Chapters 3 and 4 were tested on **695** data sets with up to **10,000** customers, **100** facilities, and **20** capacity levels. To put this in context, Table 5.2 summarizes the scale of other computational experiments in the ISP literature. For each work cited, the table lists the number of problem instances solved and the maximum values for the number of facilities, customer locations, and capacity levels in the data. Note that "N/A" in the capacity-level column indicates a case in which capacity level is not a decision variable in the ISP variant being studied. For example, the facilities in [39] are $M/M/s$ queues and the decision is to decide how many identical servers to assign to each facility.

Table 5.2. Problem Instance Data from the ISP Literature

		Maximum Number of		
Reference	Data Sets (Problem Instances)	Customers	Facilities	Capacity Levels
[3]	40	800	200	1
[4]	40	100	100	3
[5]	270	20	300	1
[7]	150	500	40	5
[8]	90	300	20	5
[9]	150	200	30	5
[12]	10	15	25	1
[18] [36]	15	20	22	1
[22]	55	100	20	3
[23]	55	150	30	N/A
[27]	25	5,500	1,800	N/A
[29]	27	150	30	N/A
[37]	12	65	150	1
[39]	20	3,500	700	N/A
[40]	10	30	15	N/A
[45]	216	400	25	5
[46]	90	459	84	3
[47]	71	200	30	1

Chapter 6

Analysis of Results

All models were tested on the same server (an HP Server model DL380 with Dual 14 Core Intel Xeon@2.6Ghz and 380GB of RAM). A pause of one second between each AMPL (version 20200810 [11]) or Java (1.6.0 [2]) call was performed to allow for the system to return to an idle state.

Each of the 695 data sets were first processed by the heuristic using Java for comparison and possibly for use as the starting solution for the other models. The Elhedhli linear model with a version for each stopping condition, Colley linear model, and nonlinear model were executed for each data set within a given set of run parameters. Metrics and results were gathered during each execution.

Three runs were performed for each data set and model using the following run parameters: whether to use the heuristic solution as the starting point (no starting point, best Phase 1 solution as a starting point, or best Phase 2 solution as a starting point), whether to limit the overall system capacity, and whether to limit the number of facilities. This provided a set of twelve run parameters with three runs each so that solution time could be averaged for each data set, model, and run parameters. All models and the heuristic were given a five minute time limit. Runs that terminated due to reaching the limit are said to have timed out.

There are four ISPs associated with any given data set: (1) the base case described in Chapter 2, (2) the base case with a constraint limiting the total number of facilities opened, (3) the base case with a constraint limiting the total capacity allocated across all facilities, and (4) the base case with constraints limiting both the total

capacity allocated and then number of facilities opened. For any particular ISP, we use the terms best solution and "optimal" solution to refer to a solution that has the minimum cost among all a feasible solutions found for that instance. In some cases, the "optimal" solution is actually a provably optimal solution found by Baron or Gurobi; in other cases none of the exact methods terminated with an optimality gap of 0% and so the "optimal" solution is really just the best solution we were able to find. Note that in many cases multiple solution approaches found the "optimal" solution for a given ISP instance.

6.1. Nonlinear Model Issues

The Nonlinear Model was solved with Baron and timed out on 98.41% of the data set and run parameter scenarios. There were instances where the Nonlinear Model did not find a feasible solution within the time limit. This increased dramatically as the size of the data set increased.

The vast number of timeouts resulted in the Nonlinear Model finding the best solution for the data set and run parameters only 13.69% of the time compared to 44.83% of the time for the Colley Linear Model and 45.68% of the time for the Elhedhli Linear Model using the solution cost as the stopping criteria.

Overall, the results indicate solving the non-linear model directly is not practical. The heuristic and the exact approaches based on linearized models performed significantly better than Baron in the experiments for this dissertation.

6.2. Elhedhli Linear Model Issues

The Elhedhli Linear Model was solved with Gurobi 9.1.2 ([26]) and was tested with three different stopping criteria: solution, cost, and coverging of the lower bound of the ratio. Using a stopping condition of the solution and cost had similar—but not

identical—results. Using a stopping condition of the coverging of the lower bound of the ratio timed out significantly more often (84.17% of the time versus about 67.00% of the time with the other stopping conditions). The first two stopping criteria found a solution in subsecond time just under 22.00% of the time compared to the Colley Linear Model of 19.69% of the time (see Table 6.1).

The Elhedhli Linear Model also had issues of moving away from a good starting point (using the heuristic solution) to a worse solution just over 38.00% of the time. This again is due to the use of the lower bound used to linearize the objective function. The heuristic starting point provides a starting lower bound which influences the Elhedhli Linear Model to find what it perceives to be a lower cost solution because of how the lower bound is calculated, but is really moving towards a higher cost solution when that solution is plugged into the nonlinear objective function. For example, when the model was applied to Data Set E6 in the Base Scenario with a given a starting solution with a cost of 664,267.62 it moved to a solution with a cost of 664,292.74.

6.3. Colley Linear Model Issues

The Colley Linear Model was solved with Gurobi 9.1.2 ([26]) and timed out 75.69% of the time but also completed in one second or less 19.69% of the time (see Table 6.1). This shows that the solution time is highly dependent on the data set and run parameters as it is the mathematical model with the greatest number of constraints between the three models analyzed.

The primary issue with the Colley Linear Model is the need to set the integer feasible tolerance and mixed-integer solution tolerance as small as possible. the default settings for Gurobi is an integer feasible tolerance of 1e-05 and a mixed-integer solution tolerance of 1e-04. By adjusting these tolerances down, the chance that the

solver stops before reaching the true optimal solution increases. The size of the data set greatly impacts the effectiveness of the model as it timed out on most data sets with 1,000 or more customers.

6.4. Heuristic Issues

The Heuristic Model generally performed Phase 1 in under one quater of a second until the number of customers in the data set reached 250 and then grew gradually from 3 seconds to 8 seconds based on the number of customers. There was no improvement in Phase 2 56.80% of the time (65.04% of the time for just the base constraints). Of the instances where Phase 2 did improve the solution quality, the largest improvement was 45.53% (11.61% for just the base constraints). However, utilizing Phase 2 dramatically increased the solution time depending on the number of customers in the data set with the longest solution time being over the time limit.

By including Phase 2, the likelihood of the heuristic finding the same best solution (or better) compared to the mathematical models went from 18.49% to 23.24% (25.47% to 29.93% for just the base constraints) with the additional 512 sizing data sets. Without the additional 512 sizing data sets, the Phase 1 solution to Phase 2 solution quality changed from 63.39% to 80.46% (84.70% to 98.36% for just the base constraints). This shows that the addition of the overall system capacity and maximum facilities constraints along with the additional 512 data sets with customer sizes ranging from 250 to 10,000 significantly impacted the performance of the heuristic.

On the Various, Nonlinear Costs data set group for the base constraints, the heuristic solution matched the best solution from the mathematical models 100.00% of the time using only the Phase 1 solution for all 54 data sets within the group. For this data set group, the Phase 1 solution time averaged 0.186 seconds while the Colley Linear Model averaged 0.480 seconds, the Elhedhli Linear Model averaged under one

quarter of a second, and the Nonlinear Model timed out 100.00% of the time.

Table 6.1. Subsecond Performance

Algorithm	**Base Scenario**	**Overall**
Phase 1	188 (27.05%)	749 (26.94%)
Phase 1 + Phase 2	79 (11.37%)	313 (11.26%)
Colley	153 (22.01%)	511 (18.38%)
Colley w/Phase 1 incumbent	154 (22.16%)	545 (19.60%)
Colley w/Phase 2 incumbent	57 (8.20%)	204 (7.34%)
Baron	0 (0.00%)	0 (0.00%)
Baron w/Phase 1 incumbent	2 (0.29%)	8 (0.29%)
Baron w/Phase 2 incumbent	0 (0.00%)	0 (0.00%)
Elhedhli Solution	159 (22.88%)	600 (21.58%)
Elhedhli Solution w/Phase 1 incumbent	159 (22.88%)	593 (21.33%)
Elhedhli Solution w/Phase 2 incumbent	59 (8.49%)	227 (8.17%)
Elhedhli Cost	157 (22.59%)	588 (21.15%)
Elhedhli Cost w/Phase 1 incumbent	159 (22.88%)	589 (21.19%)
Elhedhli Cost w/Phase 2 incumbent	60 (8.63%)	222 (7.99%)
Elhedhli Rr	147 (21.15%)	294 (10.58%)
Elhedhli Rr w/Phase 1 incumbent	150 (21.58%)	297 (10.68%)
Elhedhli Rr w/Phase 2 incumbent	56 (8.06%)	115 (4.14%)

6.5. Summary of Results

The results were analyzed three different ways [1]: Independent (with no interaction between the mathematical models (i.e. the exact methods) and the heuristic),

[1]Detailed results are available on SMU Scholar ([19]).

Interactive (with only Phase 1 of the heuristic as a starting point for the mathematical models), and Complete (with both Phase 1 and Phase 2 as a starting point for the mathematical models). With each analysis method, the overall best solution was found for each data set across three runs of the mathematical models and the heuristic. To be considered the top performer for the data set, the algorithm had to have the best solution cost with the shortest time.

Because there were three runs for each data set and run parameters, the time was analyzed running concurrently (maximum time of the three runs) and sequentially (total time for the three runs). The time was also adjusted appropriately for using the heuristic starting point. For example, the Colley linear model applied to the E6 data set starting with the Phase 1 heuristic solution and with no additional constraints had a maximum solution time of 0.138 seconds and the Phase 1 heuristic took 0.121 seconds giving an adjusted time of 0.259 seconds. Without being given a starting point it had a maximum solution time of 0.434 seconds and would not have solved the problem as quickly as it did without the time needed for the heuristic to solve Phase 1.

Given that the solution time was measured in milliseconds, a unqiue top performer was found for each data set, run parameter combination, and analysis method. While solution time is important in this dissertation, analysis was performed to find "cohorts" of the top performer. For this dissertation, a cohort was defined as another algorithm within the same analysis method with the same solution cost but within a margin of 1 second per run. For concurrent analysis it's within 1 second of the maximum solution time of the top performer and for sequential analysis it's within 3 seconds of the total time of the top performer. As the cohort analysis cannot be reasonably grouped, cohort tables have been included in this summary for only the Various, Linear group (example shown in Figure 6.1) and the Holmberg group (only

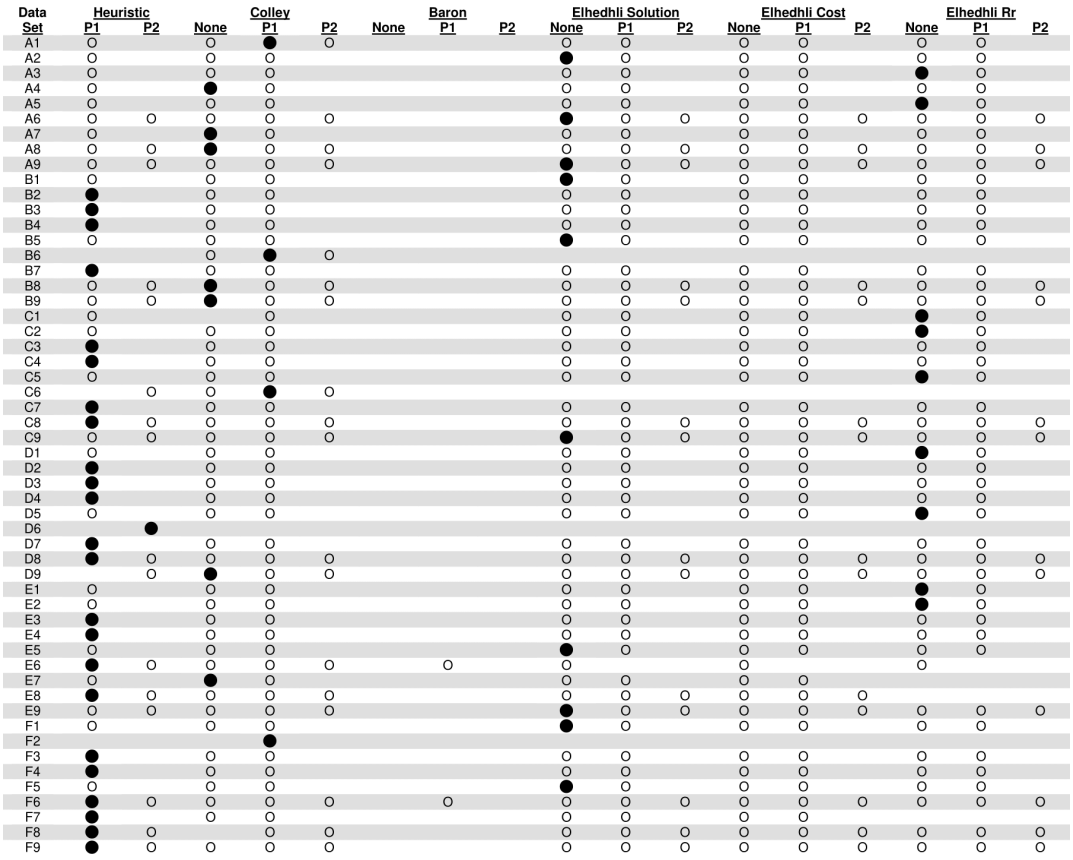


Figure 6.1. Cohorts for Base Scenario Various, Linear Group

available in the detailed results) for the base scenario in each analysis method with a solid circle for the top performer and an open circle for a cohort. The exhaustive cohort analysis is available in the detailed results.

For example, in Figure 6.1, Data Set F9's top performer was the Heuristic Phase 1 solution with cohorts in every other solver except for Baron while Data Set F2's top performer was the Colley Linear Model given the Heuristic Phase 1 solution as the incumbent solution and there were no cohorts.

For analyzing the top performers, tables are provided in the detailed results for each analysis method that include the overall group of 2,780 instances (695 data sets

for all 4 run parameters), a table for each of 695 data sets for each run parameter (base constraints, maximum facility constraint, maximum capacity constraint, and both the maximum facility and maximum capacity constraints), and tables for each data set group within each run parameter. The tables have several sections: counts for instances that did not time out, counts for instances that did time out, the number and percentage of instances that found a solution, the number and percentage of instances that found the best solution, the number and percentage of top performers based on concurrent runs, and the number and percentage of top performers based on sequential runs. The counts in the finished within the time limit section and the timed out section are broken into counts for having the best solution, not having the best solution, and not having a solution at all. A subset of the top performer tables with selected columns are included in this dissertation.

The "Phase 1" and "Phase 1 + Phase 2" algorithms are the solutions from just the heuristic. "Phase 1 + Phase 2" was not included in the Independent and Interactive analysis methods as they are focusing on solution quality and speed while the Complete analysis method is focusing more on solution quality.

The "Colley" set of algorithms are for the Colley Linear Model. The "Baron" set of algorithms are for the Nonlinear Model. The "Elhedhli Solution" set of algorithms are for the Elhedhli Linear Model using the same solution twice in a row as the stopping criteria. The "Elhedhli Cost" set of algorithms are for the Elhedhli Linear Model using the converging of the solution cost as the stopping criteria. The "Elhedhli Rr" set of algorithms are for the Elhedhli Linear Model using the converging of the lower bound of the ratios as the stopping criteria.

See Table 6.2 for abbreviations of the full, descriptive names of the 17 algorithms which are used in other tables and figures. Only some of them are included in the Independent and Interactive analysis methods as seen in Table 6.3.

Table 6.2. Algorithm Abbreviations

Algorithm	**Abbreviation**
Phase 1	P1
Phase 1 + Phase 2	P2
Colley	Colley - None
Colley w/Phase 1 incumbent	Colley - P1
Colley w/Phase 2 incumbent	Colley - P2
Baron	Nonlinear - None
Baron w/Phase 1 incumbent	Nonlinear - P1
Baron w/Phase 2 incumbent	Nonlinear - P2
Elhedhli Solution	Linear_Sol - None
Elhedhli Solution w/Phase 1 incumbent	Linear_Sol - P1
Elhedhli Solution w/Phase 2 incumbent	Linear_Sol - P2
Elhedhli Cost	Linear_Cost - None
Elhedhli Cost w/Phase 1 incumbent	Linear_Cost - P1
Elhedhli Cost w/Phase 2 incumbent	Linear_Cost - P2
Elhedhli Rr	Linear_Rr - None
Elhedhli Rr w/Phase 1 incumbent	Linear_Rr - P1
Elhedhli Rr w/Phase 2 incumbent	Linear_Rr - P2

Table 6.3. Algorithms in each Analysis Method

	Analysis Method		
Algorithm	**Independent**	**Interactive**	**Complete**
P1	√	√	√
P2			√
Colley - None	√	√	√
Colley - P1		√	√
Colley - P2			√
Nonlinear - None	√	√	√
Nonlinear - P1		√	√
Nonlinear - P2			√
Linear_Sol - None	√	√	√
Linear_Sol - P1		√	√
Linear_Sol - P2			√
Linear_Cost - None	√	√	√
Linear_Cost - P1		√	√
Linear_Cost - P2			√
Linear_Rr - None	√	√	√
Linear_Rr - P1		√	√
Linear_Rr - P2			√

Recall that "optimal" solution really means the "best" solution found for each data set and run parameter combination. This could change between analysis methods. However, all groups except for the Sizing group had a maximum decrease in solution cost between analysis methods of less than 1e-06. When changing from the Independent analysis method to the Interactive analysis method, the sizing group had an average decrease in solution cost of 2.47% and a maximum descrease of 37.30% overall and an average decrease in solution cost of 0.96% and a maximum descrease of 6.47% for just the base scenario. When changing from the Interactive analysis method to the Complete analysis method, the sizing group had an average decrease in solution cost of 1.25% and a maximum descrease of 40.03% overall and an average decrease in solution cost of 0.09% and a maximum descrease of 2.45% for just the base scenario.

For example, in Table 6.4, the models were analyzed as if they had been executed independently. When analyzed this way, the Heuristic Phase 1 solution was the top performer over 60% of the instances with the Elhedhli Linear Model utilizing the same solution twice in a row as the stopping criteria had the next highest percentage of top performers of over 15%. However, Table 6.9 shows that by interacting the solvers together and providing each mathematical model with the Heuristic P1 solution as an incumbent solution, the Colley Linear Model using the Heuristic Phase 1 solution as the incumbent solution is now the top performer over 37% of the instances and the Heuristic Phase 1 solver by itself drops to being the top performer just under 12% of the instances.

6.5.1. Independent Analysis

The independent analysis method summarized in Table 6.4 makes a very strong case for using Phase 1 of the heuristic over the five exact methods. As indicated by the "Found" column in the table, Phase 1 of the heuristic found feasible solutions to

all 2,780 problem instances. The Colley and non-linear models found feasible solutions to 1,085 (39.03%) and 451 (16.22%) problem instances, respectively. The best performing variants of the Elhedhli model, which were proposed in this dissertation, each found feasible solutions to 2,520 (90.65%) of the problem instances. Only 28.67% of the problems were solved using the Rr stopping criterion. As indicated in the "Optimal" column of the table, Phase 1 was also the leader in terms of solution quality; it found "optimal" solutions to 1,872 (67.34%) of the problem instances as opposed to the next best performer, Elhedhli with the cost-based stopping criterion, which found "optimal" solutions to 1,237 (44.5%) of the problem instances. The Colley and non-linear models found "optimal" solutions to 795 (28.60%) and 135 (4.86%) of the problem instances, respectively. As indicated in the "Top Performer" columns, Phase 1 was the top performer on 1,689 (60.76%) of the problem instances when the three runs were made concurrently, and on 1,680 (60.43%) of the problem instances when the runs were made sequentially. The next best performer was the Elhedhli model with the solution-based stopping criterion, which was a top performer on 452 (16.26%) and 441 (15.86%) of the problem instances when the runs were made concurrently and sequentially, respectively.

Focusing on just the base scenario (see Table 6.5), there is more "diversity" amoung the top performers. As shown in Table 6.5, the Colley model was a top performer 12.37% time when the runs were concurrent and 12.23% of the time when they were sequential; the Elhedhli model with the solution-based stopping criterion was a top performer 17.12% time when the runs were concurrent and 15.68% of the time when they were sequential. However, Phase 1 was a top performer 50.07% and 49.93% of the time when the runs were made concurrently and sequentially, respectively. Considering only the maximum-capacity scenarios (see Table 6.6) also shows a diversity of top performers, but again Phase 1 dominates; it was a top performer

52.66% and 51.94% of the time compared to the second-best performer, the Elhedhli model with the solution-based stopping criterion, which was a top performer 16.40% of the time and 15.97% of the time. Focusing on just the maximum-number-of-facilities scenario (see Table 6.7), the independent method of analysis shows a similar trend to the overall trend shown in Table 6.4: Phase 1 was a top performer 70.50% and 70.07% of the time, and the Elhedhli model with the solution-based stopping criterion was a top performer 15.68% and 15.25%, depending on how the runs were made. Finally, independent analysis of the scenario with limits on both system capacity and the number of facilities (see Table 6.8) shows that Phase 1 was the top performer 69.78% of time with both concurrent and sequential runs while the next-best performer, the Elhedhli model with the solution-based stopping criterion was a top performer 15.83% time when the runs were concurrent and 16.55% of the time when they were sequential.

Figure 6.2 shows the average solution time for the algorithms included in this analysis method. The Nonlinear model solved using Baron almost immediately hits the 300 second time-limit. All of the other algorithms hit the time limit once the number of customers surpasses 100 except for the Phase 1 heuristic—which never exceeds 8 seconds. Because the Elhedhli models are iterating, they can exceed the 300 second time-limit which was imposed on each iteration of the Gurobi solver and also checked cumulatively between iterations. For example, a problem instance might have accumulated 100 seconds of solve time before Gurobi began solving the next iteration. Because Gurobi had a 300 second time-limit, it ran for 300 seconds before stopping. Thus, the total solve time was 400 seconds and the model stopped before starting the next iteration.

Table 6.4. Independent Top Performer Table

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
P1	100.00%	67.34%	60.76%	60.43%
Colley - None	39.03%	28.60%	8.13%	8.31%
Nonlinear - None	16.22%	4.86%	0.07%	0.07%
Linear_Sol - None	90.65%	43.99%	16.26%	15.86%
Linear_Cost - None	90.65%	44.50%	9.57%	9.06%
Linear_Rr - None	28.67%	16.65%	5.22%	6.26%

Table 6.5. Independent Top Performer for All Base Scenario Data Sets

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
P1	100.00%	63.45%	50.07%	49.93%
Colley - None	52.23%	31.94%	12.37%	12.23%
Nonlinear - None	26.19%	9.06%	0.00%	0.00%
Linear_Sol - None	100.00%	51.08%	17.12%	15.86%
Linear_Cost - None	100.00%	51.37%	9.21%	8.63%
Linear_Rr - None	61.29%	34.24%	11.22%	13.53%

Table 6.6. Independent Top Performer for All Maximum Capacity Scenario Data Sets

Algorithm	**Found**	**"Optimal"**	**Top Performer**	
			Concurrent	**Sequential**
P1	100.00%	61.87%	52.66%	51.94%
Colley - None	45.04%	30.79%	10.07%	10.22%
Nonlinear - None	24.60%	7.05%	0.14%	0.14%
Linear_Sol - None	100.00%	48.78%	16.40%	15.97%
Linear_Cost - None	100.00%	49.78%	12.09%	11.37%
Linear_Rr - None	48.20%	28.06%	8.63%	10.36%

Table 6.7. Independent Top Performer for All Maximum Facility Scenario Data Sets

Algorithm	**Found**	**"Optimal"**	**Top Performer**	
			Concurrent	**Sequential**
P1	100.00%	72.37%	70.50%	70.07%
Colley - None	29.50%	25.90%	4.75%	5.18%
Nonlinear - None	6.76%	1.73%	0.14%	0.14%
Linear_Sol - None	81.58%	37.55%	15.68%	15.25%
Linear_Cost - None	81.58%	38.13%	8.35%	8.78%
Linear_Rr - None	2.59%	2.16%	0.58%	0.58%

Table 6.8. Independent Top Performer for All Both Constraints Scenario Data Sets

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
P1	100.00%	71.65%	69.78%	69.78%
Colley - None	29.35%	25.76%	5.32%	5.61%
Nonlinear - None	7.34%	1.58%	0.00%	0.00%
Linear_Sol - None	81.01%	38.56%	15.83%	16.55%
Linear_Cost - None	81.01%	38.71%	8.63%	7.48%
Linear_Rr - None	2.59%	2.16%	0.43%	0.58%

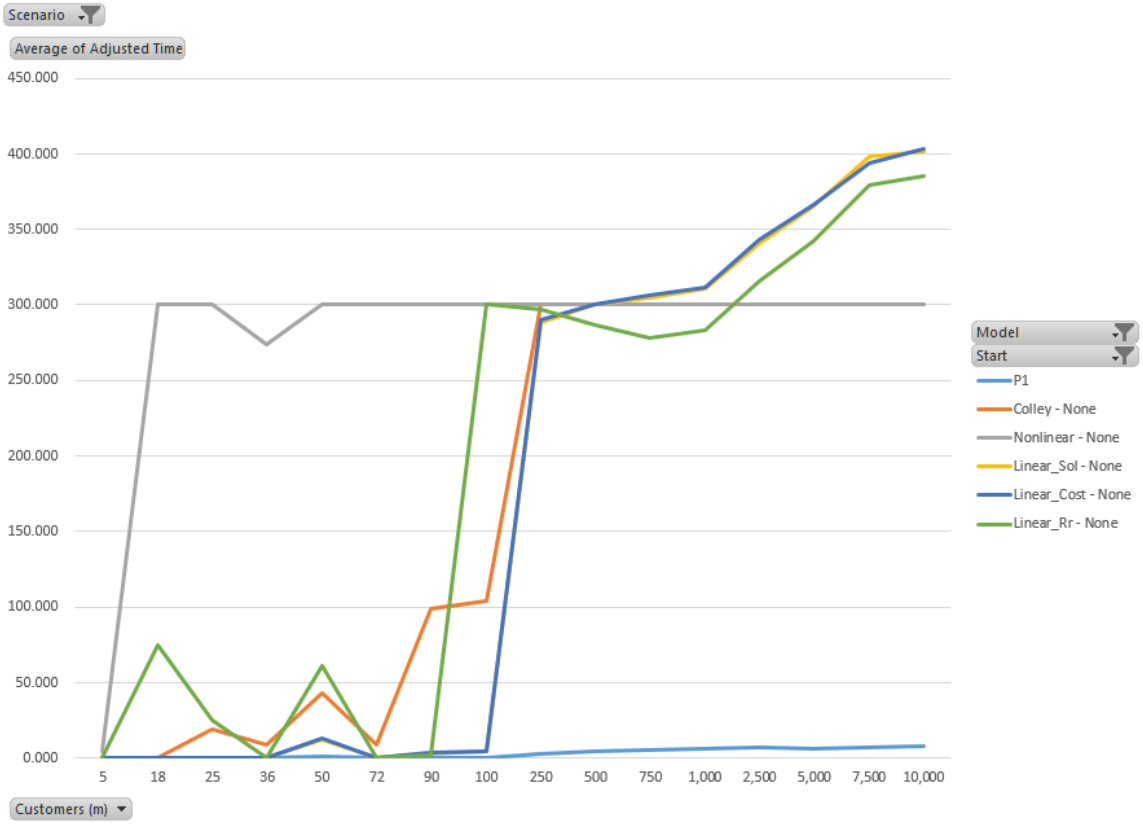


Figure 6.2. Independent Average Solution Time

6.5.2. Interactive Analysis

Table 6.9 summarizes the results using the Phase 1 solution as an initial incumbent solution when solving the ISP with one of the exact methods. The results show that the exact methods benefit significantly from starting with a good incumbent solution. For example, the Colley model found "optimal" solutions to 1,781 (64.06%) instances when given the Phase 1 incumbent as opposed to only 721 (25.94%) of instances when solved without an incumbent. As noted earlier, the Elhedhli model tended to move away from the incumbent solution. Nevertheless, the Elhedhli model with the solution-based and cost-based stopping criteria found "optimal" solutions to 1,535 (55.22%) and 1,555 (55.94%) of the problem instances when given the Phase 1 incumbent as opposed to 890 (32.01%) and 903 (32.48%) of instances without the incumbent. The incumbent solution was the least helpful when used with the nonlinear model. Without the incumbent, Baron found "optimal" solutions to 135 (4.86%) of the problem instances; with the incumbent, that number increased to 454 (16.33%). As expected, the exact methods were more likely than the heuristic to be top performers in this experiment. It is interesting to note that among the exact methods, the Colley model dominated in terms of being a top performer. In this experiment the Colley model was the top performer approximately 37% of the time; the next-best performer, the Elhedhli model with the solution-based stopping criterion, was the top performer approximately 15% of the time. Overall, these results show that if one is going to solve the ISP with an exact method then it is worth the additional time to first find an incumbent solution with Phase 1 of the heuristic.

Figure 6.3 shows the average solution time for the algorithms included in this analysis method. Given the speed of the Phase 1 heuristic, the exact methods starting with the incumbent solution did not incur a significant increase in solution time and tended to trend similarly to the Independent Analysis Method.

Table 6.9. Interactive Top Performer Table

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
P1	100.00%	18.49%	11.91%	11.58%
Colley - None	39.03%	25.94%	4.78%	4.96%
Colley - P1	100.00%	64.06%	37.19%	37.30%
Nonlinear - None	16.22%	4.86%	0.04%	0.04%
Nonlinear - P1	64.06%	16.33%	0.04%	0.04%
Linear_Sol - None	90.65%	32.01%	7.52%	6.98%
Linear_Sol - P1	100.00%	55.22%	15.47%	15.25%
Linear_Cost - None	90.65%	32.48%	2.45%	2.19%
Linear_Cost - P1	100.00%	55.94%	12.45%	12.70%
Linear_Rr - None	28.67%	12.99%	2.30%	2.91%
Linear_Rr - P1	32.73%	22.81%	5.86%	6.04%

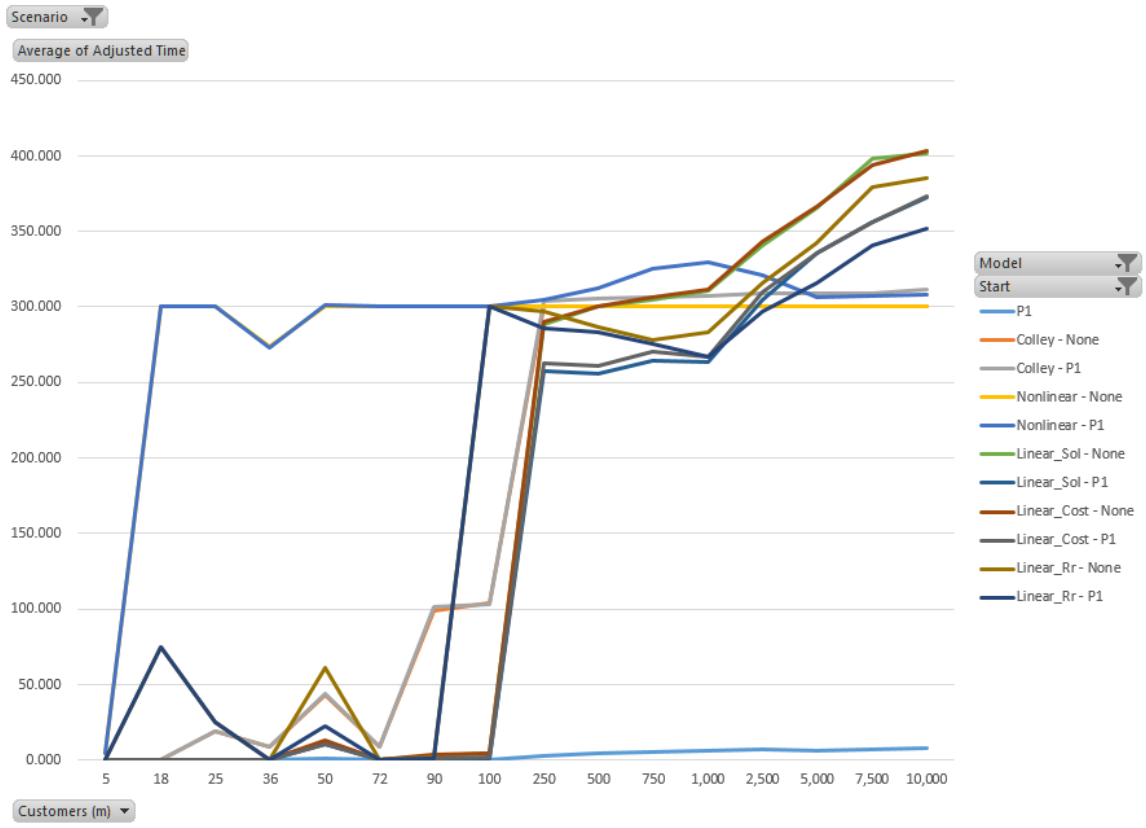


Figure 6.3. Interactive Average Solution Time

6.5.3. Complete Analysis

Table 6.10 summarizes the complete analysis. Continuing the heuristic to Phase 2 improved the quality of the solutions; it found "optimal" solutions to 514 (18.49%) instances using only Phase 1 and 646 (23.44%) using Phase 2. As noted earlier, Phase 2 significantly increased the running time of the heuristic. Comparing the first two rows of the table we see that continuing to Phase 2 caused the heuristic to time out on 2,040 of the 2,780 problem instances. When stopping after Phase 1, the heuristic only timed out once.

Using the Phase 2 solution as an incumbent also improved the quality of the solutions found by the exact methods. For example, the Colley model found "optimal" solutions to 1329 (47.81%) problem instances in this experiment when it was given the Phase 1 incumbent; that increased to 1698 (61.08%) with the Phase 2 incumbent. The improvements for the other exact methods were more modest. For example, the Elhedhli model with the cost-based stopping criterion found "optimal" solutions to 1437 (51.69%) problem instances in this experiment when it was given the Phase 1 incumbent; that increased to 1499 (53.92%) with the Phase 2 incumbent. Taking solution time into account, the results show that in most cases any given exact method was a top performer more often when starting with the Phase 1 incumbent than with the Phase 2 incumbent. For example, the Colley model with the Phase 1 incumbent was a top performer 20.86% of the time and a top performer with the Phase 2 incumbent 16.08% of the time. Overall, these results suggest that the potential improvement in solution quality gained by starting with the Phase 2 incumbent is not worth the additional time requirement.

Figure 6.4 shows the average solution time for the algorithms included in this analysis method. When starting with the Phase 2 heuristic solution, a noticeable increase in solution time is realized as the 300 second time-limit for the heuristic and

the 300 second time-limit for the exact methods were additive to get a total time-limit of 600 seconds.

Table 6.10. Complete Top Performer Table

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
P1	100.00%	18.49%	11.91%	11.58%
P2	100.00%	23.24%	0.83%	0.94%
Colley - None	39.03%	25.61%	4.39%	4.57%
Colley - P1	100.00%	47.81%	20.86%	20.97%
Colley - P2	100.00%	61.08%	16.08%	16.08%
Nonlinear - None	16.22%	4.86%	0.00%	0.00%
Nonlinear - P1	64.06%	16.33%	0.00%	0.00%
Nonlinear - P2	64.06%	19.89%	0.00%	0.00%
Linear_Sol - None	90.65%	31.12%	7.05%	6.40%
Linear_Sol - P1	100.00%	50.97%	13.88%	13.56%
Linear_Sol - P2	100.00%	53.06%	2.52%	2.30%
Linear_Cost - None	90.65%	31.44%	1.62%	1.37%
Linear_Cost - P1	100.00%	51.69%	9.96%	10.36%
Linear_Cost - P2	100.00%	53.92%	2.88%	3.02%
Linear_Rr - None	28.67%	12.91%	2.23%	2.88%
Linear_Rr - P1	32.73%	21.94%	5.25%	5.36%
Linear_Rr - P2	32.63%	22.66%	0.54%	0.61%

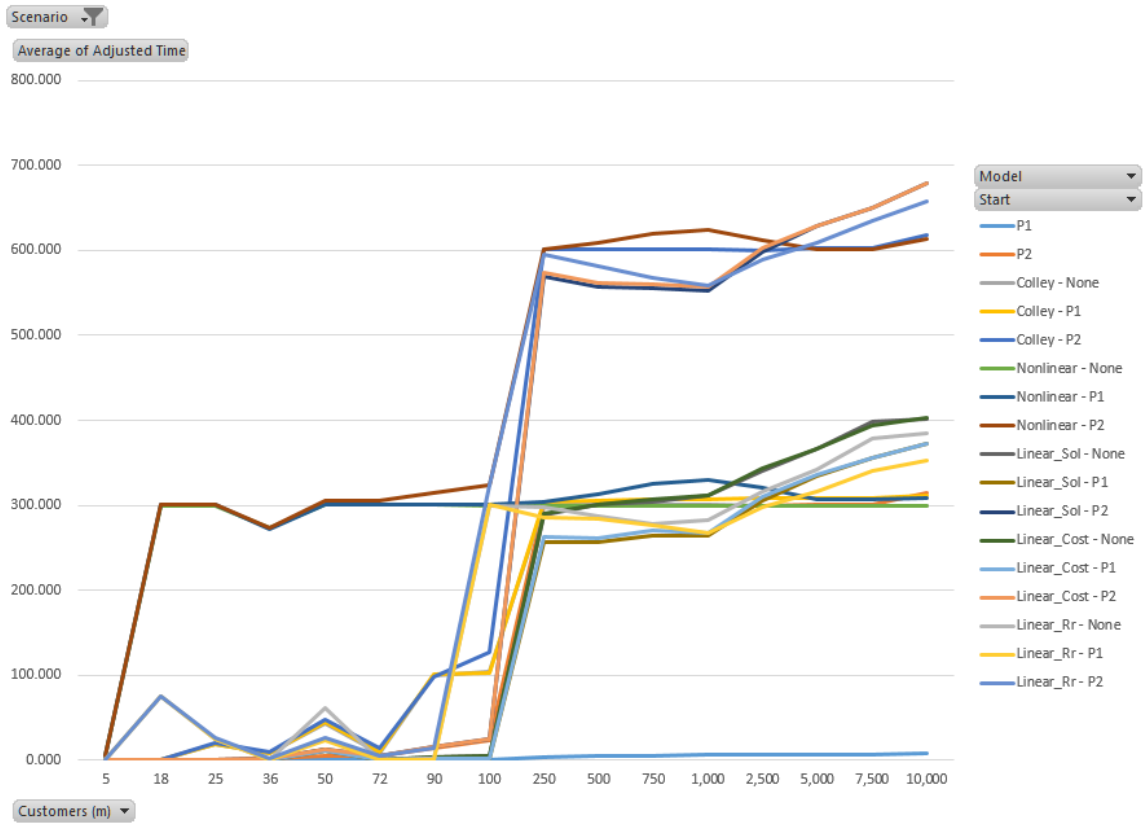


Figure 6.4. Complete Average Solution Time

6.6. Analysis of Variance

An analysis of variance (ANOVA) was also performed on the data to attempt to find an equation for predicting the length of time it will take to solve the problem. Initially, all individual runs that did not time out were analyzed (as the instances over the time limit could skew the equation). A total of 28 ANOVA tests were done with varying factors and covariate combinations. The best R^2 value of 77.02% for the derived equation was using the data set (Figure 6.5) as the factor indicating that data sets in their entirety must be used. The next best result was an R^2 value of 73.07% with the scenario as the factor and the number of customers, number of facilities, number of capacity levels, and the value of t as covariates (Figure 6.6). The ANOVA returned a regression equation for each scenario. For example, the regression equation for the Base Scenario was $2.305 - 0.00847m - 0.2420n - 1.662K - 0.002939t + 0.000784mn + 0.002787mK + 0.000073mt + 0.22334nK - 0.000159nt + 0.000973Kt - 0.000105mnK - 0.000003mnt - 0.000025mKt + 0.000052nKt + 0.000001mnKt$.

With the scenario being an important factor, the data was filtered to only the base scenario to do further ANOVA tests. Changing the factor to starting point (and using the same covariates), the R^2 was 84.05% (Figure 6.7). The data was then filtered to just instances with no starting point. The ANOVA test was updated to use model as the factor resulting in an R^2 value of 86.78% (Figure 6.8). The data was then filtered to just instances with the Elhedhli Linear Model using Solution Cost as the stopping point. The ANOVA test was updated to use group as the factor resulting in an R^2 value of 90.08% (Figure 6.9). Finally, the data was filtered one last time to only the Sizing group and an ANOVA test with no factor resulted in an R^2 value of 88.46% (Figure 6.10).

Additional ANOVA tests were performed on the final set of filtered data to determine if the number of customers, number of facilities, number of capacity levels, and

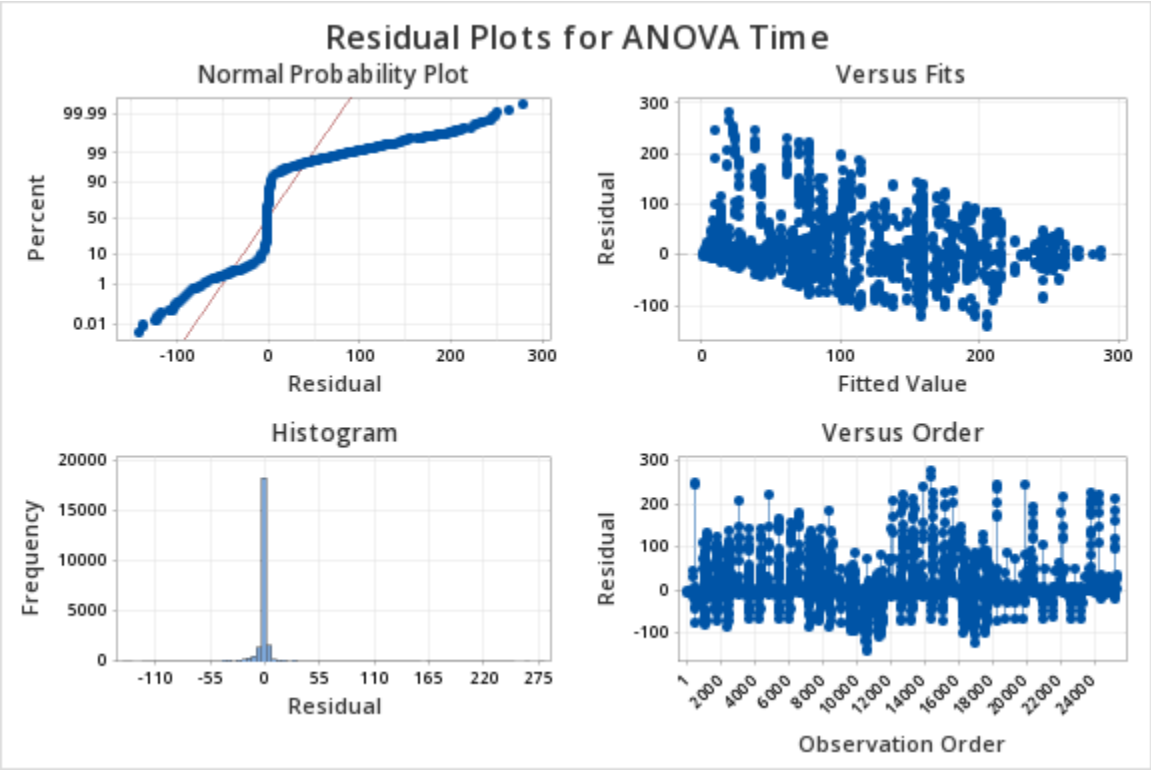


Figure 6.5. ANOVA Analysis by Data Set

the value of t were still the best covariates. All other covariate tests only resulted in an R^2 value under 20.00%. All of the ANOVA test results can be found in the detailed results.

On the figures referenced above, the Versus Fits chart shows the residuals on the y-axis and the fitted values on the x-axis and the Versus Order chart shows the residuals in the order that the data were collected.

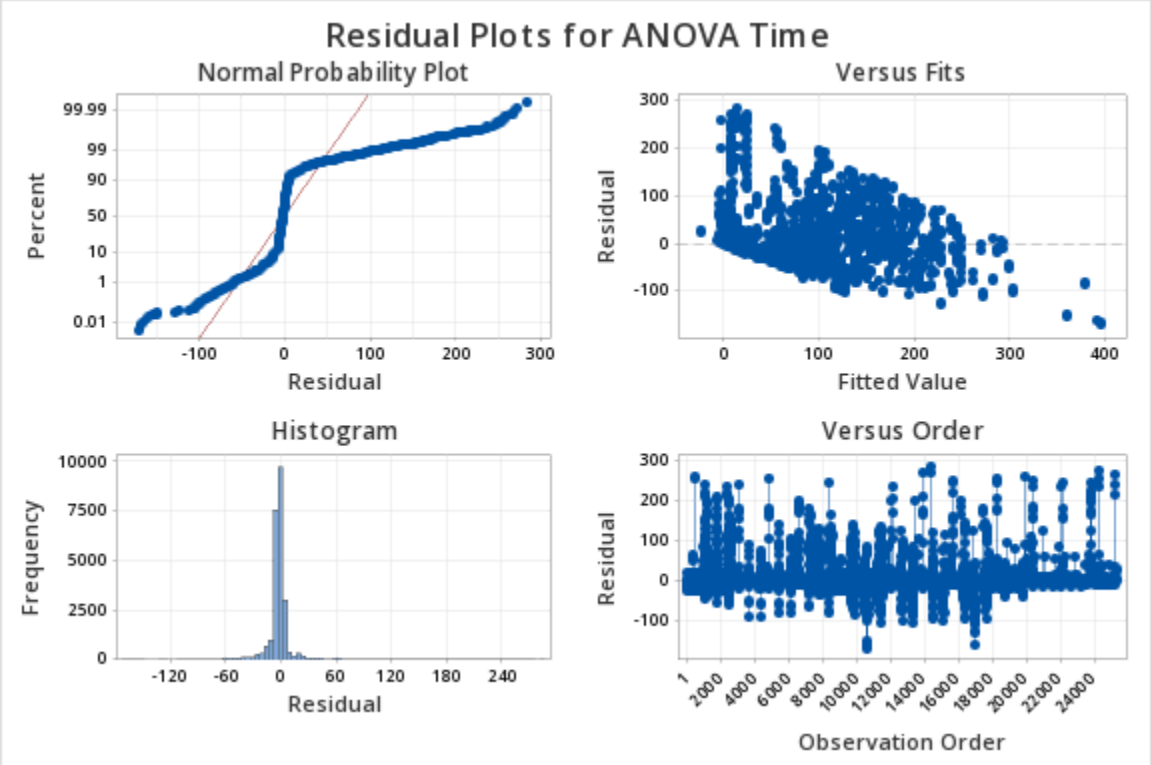


Figure 6.6. ANOVA Analysis by Scenario and m, n, K, t

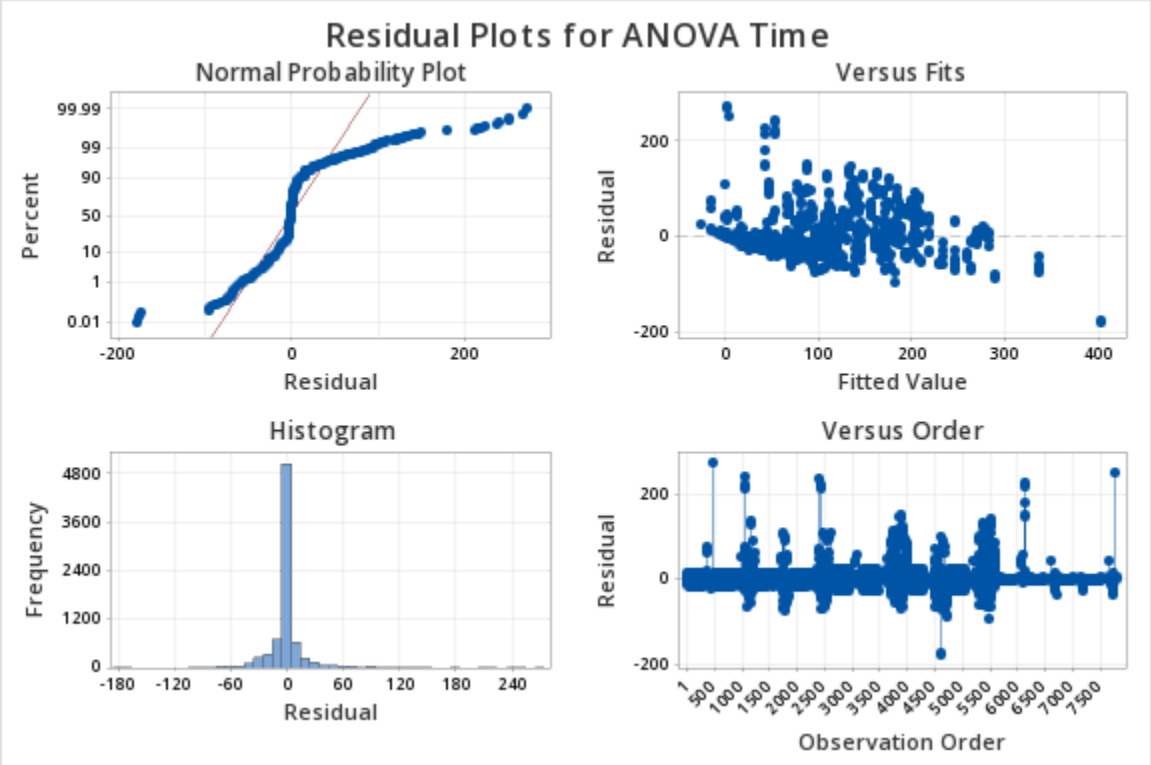


Figure 6.7. ANOVA Analysis Base by Starting Point and m, n, K, t

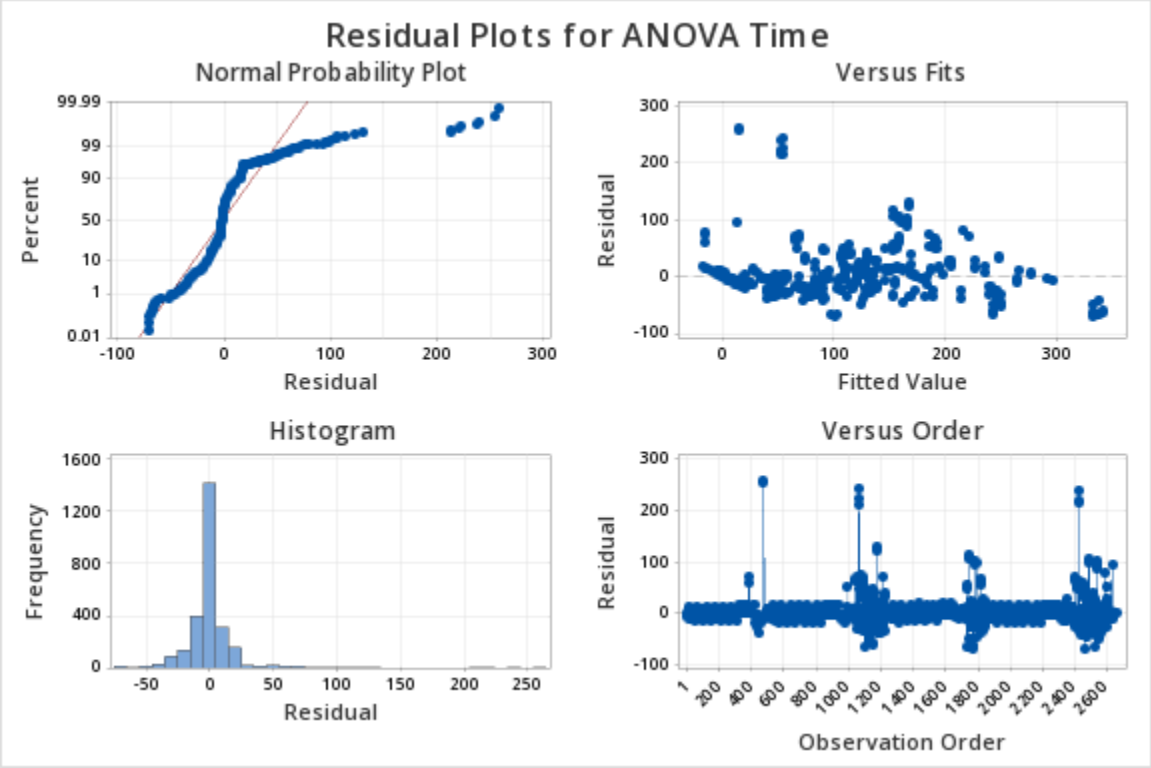


Figure 6.8. ANOVA Analysis Base (None) by Model and m, n, K, t

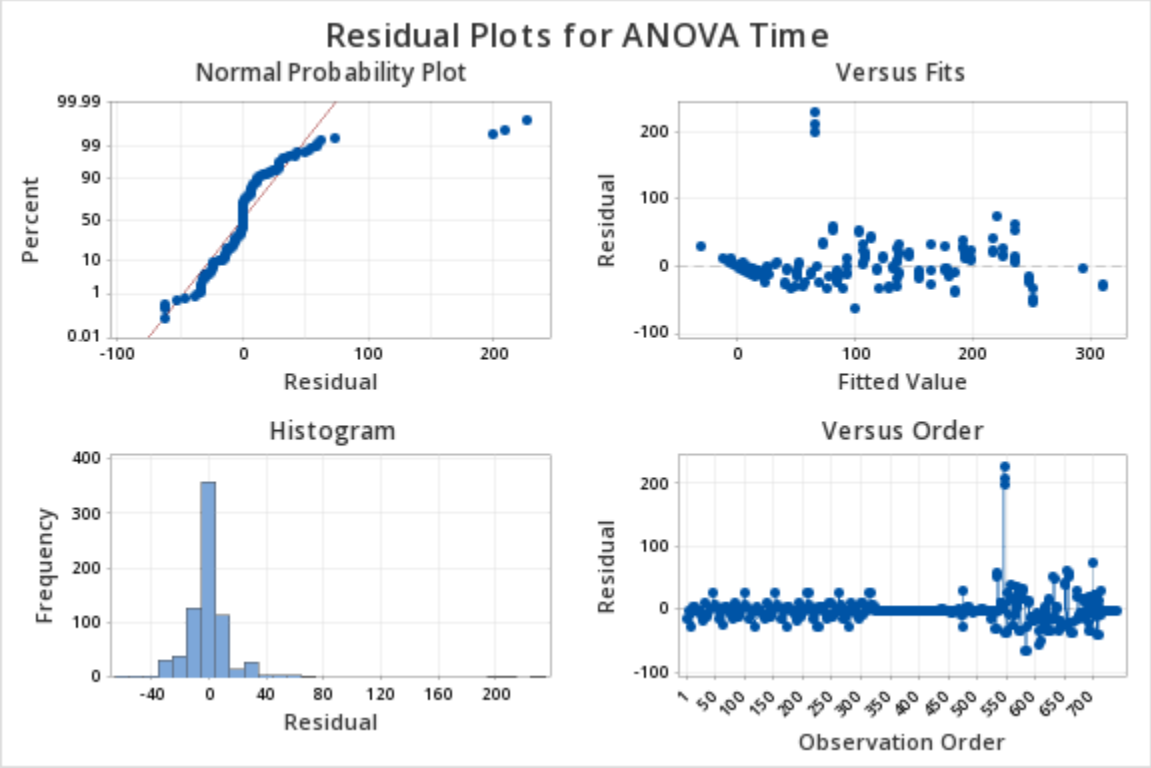


Figure 6.9. ANOVA Analysis Base (None) Elhedhli Cost by Group and m, n, K, t

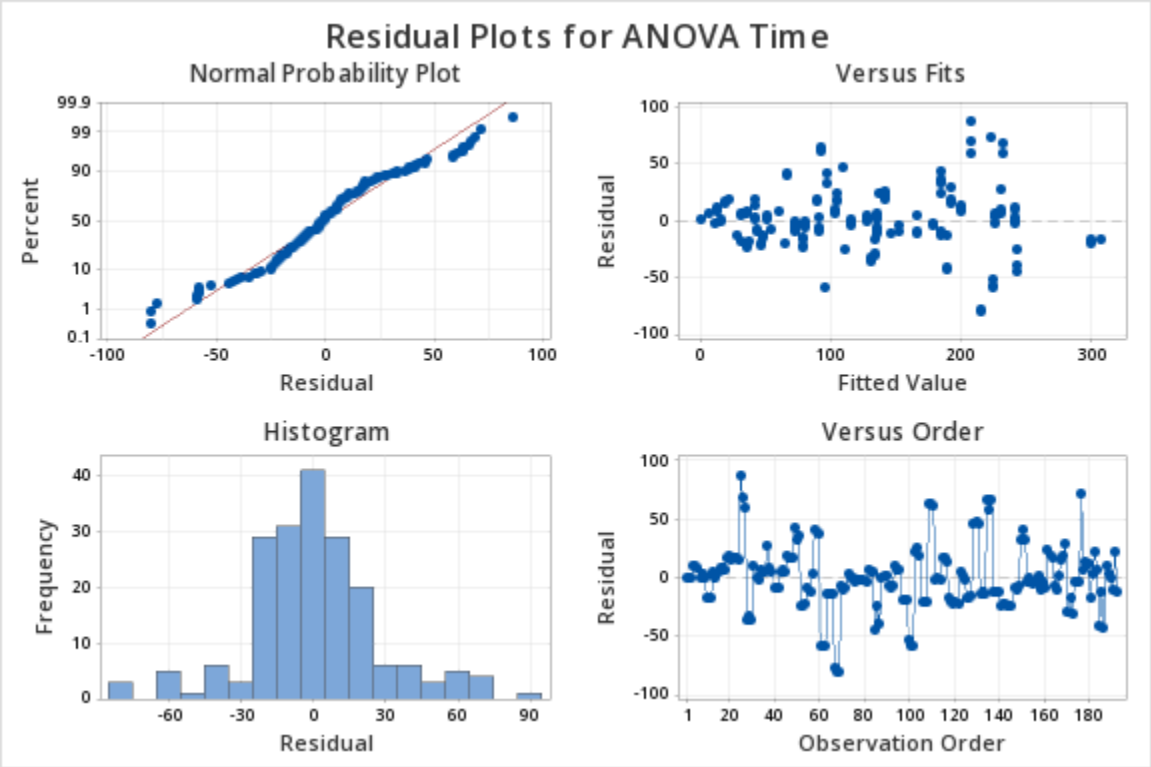


Figure 6.10. ANOVA Analysis Sizing Group Base (None) Elhedhli Cost by m, n, K, t

6.7. Extended Research

In addition to the exhaustive amount of data provided in this dissertation and the detail results, additional tests were being performed that did not complete. In those tests, the Heuristic was executed with only Phase 1 and with ten randomly sorted customer lists (no deterministically sorted customer lists) and the Elhedhli Model was executed without a time limit using the solution cost stopping criteria and no incumbent solution. Both algorithms were being executed with only the base constraints. Based on the solution times, only 78 of the 695 data sets could be executed within the available time—even with them executing on 8 servers.

Of the 78 instances of the Heuristic, the Phase 1 solution improved nine times (11.54%) and found a better solution than the current Complete Analysis Top Performer twice (2.56%)—with the remaining 67 (85.90%) instances finding a worse solution. The longest running Elhedhli instance took over 48 days. The Gurobi processes were unable to find a solution (insufficient system resources to continue processing) for 32 (41.03%) of the instances.

Of the remaining 46 Elhedhli instances, 29 (63.04%) instances found a better solution than the current Complete Analysis Top Performer. The largest improvement was 4.29% and took 3 days to execute. The remaining improvements were all under 1% with 4 (8.70%) finding a worse solution.

Chapter 7

Using M/M/s Model

Given that the capacity levels are not continuous variables but rather discrete break points, the capacity could be the result of adding more servers to a facility. Therefore, the current models—which are using the M/M/1 formula for the customer wait-time cost—are assuming one queue per server at the facility, where a customer picks a queue upon arrival and does not change to a different queue. Essentially, this formulates to s distinct M/M/1 queues.

Assuming a single queue, instead of one queue per server, the M/M/s formulas would give a more accurate value for the customer wait-time cost—thus more accurately model assigning threads in B2Bi. However, the need to use factorials prevents this formula from being used in a mathematical programming model. Therefore, the use of the M/M/s formulas would be restricted solely to heuristic algorithms.

7.1. The Current M/M/s Formulas

Given λ represents the average demand for the queue, μ represents the average capacity per server, and s represents the number of servers in the queue, the following M/M/s formulas can be obtained ([35]):

Average work being performed by the system:

$$\alpha = \frac{\lambda}{\mu} \tag{7.1}$$

Average work being performed by a single server:

$$\rho = \frac{\alpha}{s} \qquad 0 < \rho < 1 \tag{7.2}$$

Probability the system is empty:

$$P_0 = \left[\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} \right) + \left(\frac{\alpha^s}{s!} \cdot \frac{1}{1 - \frac{\alpha}{s}} \right) \right]^{-1} \tag{7.3}$$

Probability the system has k customers:

$$P_k = \begin{cases} P_0 \frac{\alpha^k}{k!}; & 0 \leq k \leq s \\ P_0 \frac{\alpha^k}{s^{k-s} s!}; & k > s \end{cases} \tag{7.4}$$

Average length of the queue (i.e. average number of waiting customers):

$$L_q = P_0 \frac{\rho \left(s\rho \right)^s}{\left(1 - \rho \right)^2 s!} \tag{7.5}$$

Average wait time in the queue for a single customer:

$$W_q = \frac{L_q}{\lambda} \tag{7.6}$$

Average length of the system (i.e. average number of customers waiting and being served):

$$L_s = L_q + \alpha \tag{7.7}$$

Average time in the system for a single customer:

$$W_s = \frac{L_s}{\lambda} \tag{7.8}$$

Average number of idle servers:

$$s_i = \sum_{k=0}^{s-1} (s - k) P_k \tag{7.9}$$

7.2. Accuracy Differences Using M/M/1 Formula

If each break point in the capacity level is due to adding a new server, the M/M/1 formula for the customer wait-time cost can be updated to use the number of servers at each capacity level. A new s_{jk} parameter value could be used to represent the number of servers for the facility at the capacity level and μ could be the average capacity per server. Thus, the customer wait-time cost could be rewritten as:

$$t \sum_{j=1}^{n} \frac{\sum_{i=1}^{m} \lambda_{ij} x_{ij}}{\sum_{k=1}^{K} s_{jk} \mu y_{jk} - \sum_{i=1}^{m} \lambda_{ij} x_{ij}} \tag{7.10}$$

If the M/M/1 result is then compared to the M/M/s result for L_q, it can be proved that the M/M/1 formula 7.10 always returns a higher value than the M/M/s formula (see Appendix C). This means the base formula 7.10 is overweighting the customer wait-time cost if the capacity break points are due to the addition of more servers in the same queue for customers to arrive—which is definitely the case for the B2Bi application as each facility represents a single priority queue.

7.3. Impact on Heuristic Using M/M/s Formula

The impact of using the M/M/s formula 7.5 on the heuristic would not necessarily be in complexity as the L_q value can be found in polynomial time ($O(s)$). It can also be performed as a subroutine to mitigate any redundant code. However, more research is needed to determine what the impact is on the solve time. The primary goal of the heuristic is to have a result in no longer than one second of elapsed time.

7.4. Additional Cost Options Using M/M/s

By changing the base model to work in terms of number of servers instead of strictly capacity, other costs from the M/M/s formulas could also be used. The more

complex formulas could not be utilized in a mathematical programming software due to the need to perform factorial calculations, but could be used in the heuristic.

Another cost that appears complex at first is the average number of idle servers. This formula can be simplified into $s_i = s - \alpha$ (see Appendix B). Currently, the data sets derived from the client data is using the fixed cost (f_{jk}) to help drive more capacity to "higher" facilities. The idle server cost could be used instead. The goal of this cost would be to increase capacity at facilities where the availability of a server when a customer arrives is greater compared to other facilities.

Using the heuristic could also open up more complex cost calculations, such as targeting the probability that there are sufficient servers to handle new customers arriving a certain percentage of the time ($\sum_{k=0}^{s} P_k$). Depending on the need, this could be more reliable than the average number of idle servers. It could also be converted into a constraint instead of a cost depending on the need. However, this could potentially lead to an infeasible problem if the target was too stringent.

Chapter 8

CONCLUSION

8.1. Contributions

This dissertation develops a fast, effective heuristic for the immobile server problem that is easy to implement. This fills a gap between computationally intensive solution approaches in the literature: exact methods, that require specialized optimization software to implement, and meta-heuristics. The exact methods in the literature involve iteratively solving sequences of mixed integer programs. Another contribution of this dissertation is a stand-alone mathematical programming model that can be solved with a single, straightforward application of a commercial solver. A relatively large testbed of new problem instances, many of which are orders of magnitude larger than those in the literature, was generated to evaluate the new solution approaches. These data sets have been made available to share with other researchers on the SMU Scholar website. In the computational study with the testbed data and data from the literature, the heuristic was shown to be very effective for time-limited use cases such as reconfiguring resources for B2Bi clients and content delivery networks. The heuristic was also shown to improve the performance of the exact methods by quickly finding high-quality incumbent solutions. Through statistical and sensitivity analysis, the dissertation also provides a better understanding of the effect of customer waiting time cost and capacity budgets on solution with exact methods.

8.2. Alternate Use Cases

Given the opportunity (such as with the B2Bi use case), using nonlinear costs with just the base model appears to work best with the heuristic using only Phase 1 as the results are consistent with the mathematical models but are obtained significantly faster and without the need to purchase optimization software. Even with using linear costs (which is most use cases where costs are provided), the heuristic performed well under most circumstances in terms of the error in relation to the mathematical models by including Phase 2 at the cost of increased solution time and not meeting the goal of finding at least one solution per second.

For any of the models, keeping the average value for the customer wait-time cost ($t\frac{\hat{\lambda}}{\hat{\mu}-\hat{\lambda}}$ where $\hat{\lambda}$ is the total system demand and $\hat{\mu}$ is the total system capacity) at or below the average cost for the facility assigment cost ($\frac{\sum_{i=1}^{m}\hat{c}_i\lambda_i}{m}$ where $\hat{c}_i$ is the average facility assign cost for customer i) appears to correlate to an easier problem for the models to solve. Using nonlinear costs appears to significantly help with reducing the size of the problem—as fewer capacity levels might be required.

8.3. Future Research

Using the M/M/s formulas with the heuristic is planned for future research. This could create an additional benefit to the solution by having a more accurate cost calculation or addition of constraints that cannot be described accurately using the M/M/1 formulas. This could give the heuristic an advantage over the other models for specific needs.

The heuristic can also be studied further with additional options for Phase 1—such as sorting by the standard deviation or providing Phase 2 with an optional starting point that does not include the customer wait-time cost. Additional subroutines for Phase 2 could also be developed. By incorporating three randomized lists, the

importance of finding additional sorting options was showcased in that 38.06% of the instances the Heuristic Phase 1 found a better solution because of the randomized lists than it would have found without it. Additionally, Phase 2 had a better solution 54.35% of the instances because of Phase 1 providing additional starting points. The best case scenario would be to use the best solution for each instance, work backwards, and find an algorithm to sort the customers so that Phase 1 will return the best solution every time.

Appendix A

Detailed Results

Detailed results will be reviewed outside of this document for the dissertation. A zip file containing all inputs and outputs can be found on SMU Scholar ([19]). There are two compressed files on the site. The main file (Colley Dissertation Analysis.zip) and the supplemental file (UNIX AMPL and Script Output Files.zip) are both available on the SMU Scholar site. The supplemental file is approximately 10.5 GB in size and includes all AMPL and Script output files from running each of the 141,780 results. The files are stored in folders based on the type of output (AMPL or Script), incumbent solution used, scenario, and data-set group. The main file contents are outlined in Table A.1.

Table A.1. Main Detailed Results Compressed File Contents

File / File Type	**Desription**
AMPL Files	Directory containing files used by AMPL.
Data Sets	Directories containing 695 data sets by data-set group.
Analysis.mpx	MiniTab Project File used for ANOVA testing.
Analysis.xlsx	Excel Spreadsheet used to analyze the data.
Cohort Tables.pdf	PDF of selected Cohort Tables.
Minitab Results.docx	Word Document containing results of ANOVA testing.
Overall Results.xlsx	Excel Spreadsheet containing Top Performer Tables.
preResults.csv	CSV File containing heuristic results.
readme.txt	Read Me file explaining the contents of the compressed file.
Result.csv	CSV File containing AMPL results.
SawTooth.xlsx	Excel Spreadsheet containing the results of the brute-force analysis.

Appendix B

Idle Server Calculation Proof

Proposition:

$$s_i = s - \alpha \tag{B.1}$$

Given:

$$P_0 = \left[\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} \right) + \left(\frac{\alpha^s}{s!} \cdot \frac{1}{1 - \frac{\alpha}{s}} \right) \right]^{-1} \tag{B.2}$$

$$P_k = \begin{cases} P_0 \frac{\alpha^k}{k!}; & 0 \leq k \leq s \\ P_0 \frac{\alpha^k}{s^{k-s} s!}; & k > s \end{cases} \tag{B.3}$$

$$s_i = \sum_{k=0}^{s-1} (s-k) P_k \tag{B.4}$$

Proof

Start with the equality:

$$\frac{\alpha^s}{(s-1)!} = \frac{\alpha^s}{(s-1)!} \tag{B.5}$$

Add and subtract a summation that can incorporate one side of the equality:

$$\frac{\alpha^s}{(s-1)!} + \sum_{k=0}^{s-2} \frac{\alpha^{k+1}}{k!} - \sum_{k=0}^{s-2} \frac{\alpha^{k+1}}{k!} = \frac{\alpha^s}{(s-1)!} \tag{B.6}$$

$$\sum_{k=0}^{s-1} \frac{\alpha^{k+1}}{k!} - \sum_{k=0}^{s-2} \frac{\alpha^{k+1}}{k!} = \frac{\alpha^s}{(s-1)!} \tag{B.7}$$

$$- \sum_{k=0}^{s-2} \frac{\alpha^{k+1}}{k!} = - \sum_{k=0}^{s-1} \frac{\alpha^{k+1}}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.8}$$

Add a new summation to each side:

$$\sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=0}^{s-2} \frac{\alpha^{k+1}}{k!} = \sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=0}^{s-1} \frac{\alpha^{k+1}}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.9}$$

Normalize all summation ranges to start with $k = 0$ and end with $s - 1$:

$$\sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=1}^{s-1} \frac{\alpha^k}{(k-1)!} = \sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=0}^{s-1} \frac{\alpha^{k+1}}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.10}$$

$$\sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=1}^{s-1} \frac{k\alpha^k}{k!} = \sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=0}^{s-1} \frac{\alpha^{k+1}}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.11}$$

$$\sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=0}^{s-1} \frac{k\alpha^k}{k!} = \sum_{k=0}^{s-1} \frac{s\alpha^k}{k!} - \sum_{k=0}^{s-1} \frac{\alpha^{k+1}}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.12}$$

Combine summations:

$$\sum_{k=0}^{s-1} \frac{s\alpha^k - k\alpha^k}{k!} = \sum_{k=0}^{s-1} \frac{s\alpha^k - \alpha^{k+1}}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.13}$$

$$\sum_{k=0}^{s-1} \frac{(s-k)\,\alpha^k}{k!} = \sum_{k=0}^{s-1} \frac{(s-\alpha)\,\alpha^k}{k!} + \frac{\alpha^s}{(s-1)!} \tag{B.14}$$

Separate out $s - \alpha$ from one side and simplify:

$$\sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = (s-\alpha)\left[\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} + \frac{\alpha^s}{(s-1)!\,(s-\alpha)}\right] \tag{B.15}$$

$$\sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = (s-\alpha)\left[\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} + \frac{s\alpha^s}{s!\,(s-\alpha)}\right] \tag{B.16}$$

$$\sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = (s-\alpha)\left[\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} + \frac{\alpha^s}{s!\frac{s-\alpha}{s}}\right] \tag{B.17}$$

$$\sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = (s-\alpha)\left[\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} + \frac{\alpha^s}{s!\left(1-\frac{\alpha}{s}\right)}\right] \tag{B.18}$$

$$\sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = \frac{(s-\alpha)}{\left[\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!}\right) + \left(\frac{\alpha^s}{s!} \cdot \frac{1}{1-\frac{\alpha}{s}}\right)\right]^{-1}} \tag{B.19}$$

Substitute known formulas:

$$\sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = \frac{(s-\alpha)}{P_0} \tag{B.20}$$

$$P_0 \sum_{k=0}^{s-1} (s-k)\,\frac{\alpha^k}{k!} = s-\alpha \tag{B.21}$$

$$\sum_{k=0}^{s-1} (s-k)\,P_0\frac{\alpha^k}{k!} = s-\alpha \tag{B.22}$$

$$\sum_{k=0}^{s-1} (s-k)\,P_k = s-\alpha \tag{B.23}$$

$$s_i = s-\alpha \tag{B.24}$$

■

Appendix C

Proof of M/M/1 always higher than M/M/s

Proposition:

$$L_q^{M/M/s} < L_q^{M/M/1} \tag{C.1}$$

Given:

$$\alpha = \frac{\lambda}{\mu} \tag{C.2}$$

$$\rho = \frac{\alpha}{s} \qquad 0 < \rho < 1 \tag{C.3}$$

$$\mu^{M/M/1} = s\mu \tag{C.4}$$

$$P_0 = \left[\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} \right) + \left(\frac{\alpha^s}{s!} \cdot \frac{1}{1 - \frac{\alpha}{s}} \right) \right]^{-1} \tag{C.5}$$

$$L_q^{M/M/1} = \frac{\lambda}{\mu^{M/M/1} - \lambda} \tag{C.6}$$

$$L_q^{M/M/s} = P_0 \frac{\rho \left(s\rho\right)^s}{\left(1 - \rho\right)^2 s!} \tag{C.7}$$

Proof

Start with the assumption and simplify into terms of s and α:

$$L_q^{M/M/s} < L_q^{M/M/1} \tag{C.8}$$

$$P_0 \frac{\rho\left(s\rho\right)^s}{\left(1-\rho\right)^2 s!} < \frac{\lambda}{\mu^{M/M/1} - \lambda} \tag{C.9}$$

$$P_0 \frac{\frac{\alpha}{s}\left(\alpha\right)^s}{\left(1-\frac{\alpha}{s}\right)^2 s!} < \frac{\lambda}{s\mu - \lambda} \tag{C.10}$$

$$P_0 \frac{\alpha^{s+1}}{s\left(\frac{s-\alpha}{s}\right)^2 s!} < \frac{\lambda}{s\mu - \alpha\mu} \tag{C.11}$$

$$P_0 \frac{\alpha^{s+1}}{\frac{(s-\alpha)^2}{s} s!} < \frac{\lambda}{\mu\left(s-\alpha\right)} \tag{C.12}$$

$$P_0 \frac{\alpha^{s+1}}{\left(s-\alpha\right)^2 \left(s-1\right)!} < \frac{\alpha}{s-\alpha} \tag{C.13}$$

Cancel terms and explode P_0, then simplify:

$$\frac{\alpha^s}{(s-1)!} < \frac{s-\alpha}{P_0} \tag{C.14}$$

$$\frac{\alpha^s}{(s-1)!} < \left(s-\alpha\right)\left(\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!}\right) + \left(\frac{\alpha^s}{s!} \cdot \frac{1}{1-\frac{\alpha}{s}}\right)\right) \tag{C.15}$$

$$\frac{\alpha^s}{(s-1)!} < \left(s-\alpha\right)\left(\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!}\right) + \left(\frac{\alpha^s}{s!\left(\frac{s-\alpha}{s}\right)}\right)\right) \tag{C.16}$$

$$\frac{\alpha^s}{(s-1)!} < \left(s-\alpha\right)\left(\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!}\right) + \left(\frac{\alpha^s}{(s-1)!\left(s-\alpha\right)}\right)\right) \tag{C.17}$$

$$\frac{\alpha^s}{(s-1)!} < \left(s-\alpha\right)\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!}\right) + \frac{\alpha^s}{(s-1)!} \tag{C.18}$$

$$0 < \left(s-\alpha\right)\left(\sum_{k=0}^{s-1} \frac{\alpha^k}{k!}\right) \tag{C.19}$$

Because $s > \alpha > 0$, the following are always true:

$$s - \alpha > 0 \tag{C.20}$$

$$\sum_{k=0}^{s-1} \frac{\alpha^k}{k!} > 0 \tag{C.21}$$

Therefore, the assumption is confirmed that the value for the average length of the queue using the M/M/1 formula is always greater than the value for the average length of the queue using the M/M/s formula.

■

REFERENCES

[1] IBM Sterling B2B Integrator. https://bridgesgi.com/wp-content/uploads/2014/10/Bridge-Solutions-B2B-Integrator.pdf.

[2] Java 1.6. https://www.oracle.com/java/technologies/javase/webnotes.html.

[3] Aboolian, R., Berman, O., and Drezner, Z. Location and allocation of service units on a congested network. *IIE Transactions 40* (2008), 422–433.

[4] Aboolian, R., Berman, O., and Krass, D. Profit maximizing distributed service system design with congestion and elastic demand. *Transportation Science 46* (2012), 153–295.

[5] Agnihothri, S., Narasimhan, S., and Pirkul, H. An assignment problem with queueing time cost. *Naval Research Logistics - NAV RES LOG 37* (1990), 231–244.

[6] Al Jadaan, O., Rajamani, L., and Rao, C. R. Non-dominated ranked genetic algorithm for solving multi-objective optimization problems: NRGA. *Journal of Theoretical and Applied Information Technology 2* (2008), 60–67.

[7] Amiri, A. Solution procedures for the service system design problem. *Computers & Operations Research 24*, 1 (1997), 49–60.

[8] Amiri, A. The design of service systems with queueing time cost, workload capacities and backup service. *European Journal of Operational Research 104*, 1 (1998), 201–217.

[9] Amiri, A. The multi-hour service system design problem. *European Journal of Operational Research 128*, 3 (2001), 625–638.

[10] Amiri, A., and Pirkul, H. New formulation and relaxation to solve a concave-cost network flow problem. *Journal of the Operational Research Society 48* (1997), 278287.

[11] AMPL. *AMPL Version 10.6.16*. AMPL Optimization LLC, 2009.

[12] Arkat, J., and Jafari. Network location problem with stochastic and uniformly distributed demands. *International Journal of Engineering 29*, 5 (2016), 654–662.

[13] Berman, O., and Krass, D. Stochastic location models with congestion. In *Location Science*, G. Laporte, S. Nickel, and F. S. da Gama, Eds. Springer, Berlin, 2015, pp. 443–486.

[14] Boffey, B., Galvao, R., and Espejo, L. A review of congestion models in the location of facilities with immobile servers. *European Journal of Operational Research 178* (2007), 643–662.

[15] Boffey, B., Galvão, R. D., and Marianov, V. Location of single-server immobile facilities subject to a loss constraint. *Journal of the Operational Research Society 61*, 6 (2010), 987–999.

[16] Burke, E. K. and Kendall G. *Search Methodologies*. Springer, New York, NY, 2014. https://doi.org/10.1007/978-1-4614-6940-7.

[17] Castillo, I., Ingolfsson, A., and Sim, T. Socially Optimal Location of Facilities with Fixed Servers, Stochastic Demand and Congestion. `http://www.optimization-online.org/DB_HTML/2002/12/578.html`, 07 2002. [Online; accessed 7-April-2021].

[18] Chambari, A., Rahmaty, S. H., Hajipour, V., and Karimi, A. A bi-objective model for location-allocation problem within queuing framework. *World Academy of Science. Engineering and Technology 78* (2011), 138–145.

[19] Colley, A. Q. Customer and Capacity Assignment for a Set of Immobile Servers using Heuristics. `https://scholar.smu.edu/engineering_management_research/3/`, 2021. [Online;].

[20] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *Parallel Problem Solving from Nature PPSN VI* (Berlin, Heidelberg, 2000), M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds., Springer Berlin Heidelberg, pp. 849–858.

[21] Derringer, G., and Suich, R. Simultaneous optimization of several response variables. *Journal of quality technology 12*, 4 (1980), 214–219.

[22] Elhedhli, S. Service system design with immobile servers, stochastic demand, and congestion. *Manufacturing & Service Operations Management 8*, 1 (2006), 92–97. http://dx.doi.org/10.1287/msom.1050.0094.

[23] Elhedhli, S., Wang, Y., and Saif, A. Service system design with immobile servers, stochastic demand and concave-cost capacity selection. *Computers & Operations Research 94* (2018), 65–75.

[24] Geem, Z. W., Kim, J. H., and Loganathan, G. V. A new heuristic optimization algorithm: harmony search. *simulation 76*, 2 (2001), 60–68.

[25] Glover, F. and Laguna M. *Tabu Search*. Springer, New York, NY, 1997. https://doi.org/10.1007/978-1-4615-6089-0.

[26] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[27] Hajipour, V., Rahmati, S. H. A., Pasandideh, S. H. R., and Niaki, S. T. A. A multi-objective harmony search algorithm to optimize multi-server location–allocation problem in congested systems. *Computers & Industrial Engineering 72* (2014), 187–197.

[28] Holmberg, K., Ronnqvist, M., and Yuan, D. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research 113*, 3 (1999), 544–559.

[29] Hoseinpour, P. An economies-of-scale service system design problem. `https://arxiv.org/abs/2006.07851`, 2020. [Online; accessed 7-April-2021].

[30] Li, B., Golin, M. J., Italiano, G. F., Deng, X., and Sohraby, K. On the optimal placement of web proxies in the internet. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)* (1999), vol. 3, pp. 1282–1290 vol.3.

[31] Marianov, V., Boffey, T. B., and Galvão, R. D. Optimal location of multi-server congestible facilities operating as m/e r/m/n queues. *Journal of the Operational Research Society 60*, 5 (2009), 674–684.

[32] Marianov, V., and Serra, D. Locationallocation of multiple-server service centers with constrained queues or waiting times. *Annals of Operations Research 111*, 1 (2002), 35–50.

[33] Mirjalili, S. The ant lion optimizer. *Advances in Engineering Software 83* (2015), 80–98.

[34] P. Chandra, E. W. W. Fibonacci number. *MathWorld–A Wolfram Web Resource*. https://mathworld.wolfram.com/FibonacciNumber.html.

[35] P. Harrison, N. M. P. *Performance Modelling of Communication Networks and Computer Architectures*. AddisonWesley, 1992.

[36] Pasandideh, S. H. R., and Chambari, A. A new model for location-allocation problem within queuing framework. *Journal of Optimization in Industrial Engineering*, 6 (2010), 53–61.

[37] Pasandideh, S. H. R., and Niaki, S. Genetic application in a facility location problem with random demand within queuing framework. *Journal of Intelligent Manufacturing 23* (2012), 651–659.

[38] Peng, Y. Resource allocation and task scheduling optimization in cloud-based content delivery networks with edge computing. `https://scholar.smu.edu/engineering_managment_etds/6`, 2019. [Online;].

[39] Rahmati, S. H. A., Hajipour, V., and Niaki, S. T. A. A soft-computing pareto-based meta-heuristic algorithm for a multi-objective multi-server facility location problem. *Applied soft computing 13*, 4 (2013), 1728–1740.

[40] Rajagopalan, S., and Yu, H.-L. Capacity planning with congestion effects. *European Journal of Operational Research 134*, 2 (2001), 365–377. Financial Modelling.

[41] Riccio, L. J. Management science in New York's department of sanitation. *INFORMS Journal on Applied Analytics 14*, 2 (1984), 1–13.

[42] Stiermaier, S. *Facility location and allocation problems with stochastic customer demand and immobile servers*. PhD thesis, 2010.

[43] Taguchi, G. Orthogonal arrays and linear graphs. *American Supplier Institute, Inc* (1986).

[44] The Optimization Firm. BARON. `https://minlp.com/baron`, 2021. [Online; accessed 10-April-2021].

[45] Vidyarthi, N., and Jayaswal, S. Efficient solution of a class of locational-location problems with stochastic demand and congestion. *Computers & Operations Research 48* (2014), 20–30.

[46] Wang, Q., Batta, R., and Rump, C. M. Algorithms for a facility location problem with stochastic customer demand and immobile servers. *Annals of Operations Research 111* (2002), 1734.

[47] Zamani, S., Arkat, J., Niaki, S. T. A., and Ahmadizar, F. Locations of congested facilities with interruptible immobile servers. *Computers & Industrial Engineering 156* (2021), 107220.