INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2024) 1–30
DOI: 10.1111/itor.13564

Fast heuristics for the time-constrained immobile server problem

Adam Q. Colley[a] and Eli V. Olinick[b,*]

[a]*Boundless Integrations, Florence, AL 35630, USA*
[b]*Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, TX 75205, USA*
E-mail: aqcolley@boundlessintegrations.com[Colley]; olinick@smu.edu[Olinick]

Received 7 March 2024; received in revised form 24 July 2024; accepted 9 September 2024

Abstract

We propose easy-to-implement heuristics for time-constrained applications of a problem referred to in the literature as the facility location problem with immobile servers, stochastic demand, and congestion, the service system design problem, or the 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. The expected-waiting-time cost results in a nonlinear 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 straightforward, greedy heuristics can rapidly find high-quality solutions. In addition to filling a gap in the literature on ISP heuristics, new stopping criteria for an existing cutting plane algorithm are proposed and tested, and a new mixed-integer linear model requiring no iterating algorithm is developed. In many cases, our heuristic approach finds solutions of the same or better quality than those found by exact methods implemented with expensive, state-of-the-art mathematical programming software, in particular a commercial nonlinear mixed-integer linear programming solver, given a five-minute time limit.

Keywords: heuristics; service system design; facility location; immobile servers; stochastic demand; congestion

1. Introduction

We propose easy-to-implement heuristics for time-constrained applications of a problem referred to in the literature as the facility location problem with immobile servers, stochastic demand, and congestion (Elhedhli, 2006), the service system design problem (Amiri, 1997, 1998, 2001),

*Corresponding author.

or the immobile server problem (ISP). The ISP 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, Amiri (1997) cites applications to warehouse location (Amiri and Pirkul, 1997), and waste collection and disposal (e.g., Agnihothri et al., 1990 and Riccio, 1984). Motivated by manufacturing applications, Rajagopalan and Yu (2001) 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.

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 PBXs—at existing facilities, airport security to determine the number of agents for each lane and content delivery networks (Peng, 2019)—which need the capability to quickly adapt to changing demands. Our study is motivated by these kinds of applications as well as cases where a series of related ISPs are solved to evaluate alternatives in a simulation-optimization process.

The elapsed time is important for content delivery networks as unexpected or abnormally large spikes in traffic can occur. For example, an abnormally large spike in traffic over a wide geographical area could be sparked by an influencer on social media recommending a particular show provided by the streaming service utilizing the content delivery network. If the content delivery network does not have the ability to be rapidly adjusted by copying the show to the appropriately placed geographic locations, bottlenecks can occur on the few servers currently hosting the show—affecting all traffic on those servers.

Colley (2022) describes another dynamic telecommunications application in a commercial setting in which business processes (customers in the ISP) are placed into priority queues which are assigned a specific number of threads in processes running on back-end servers (capacity in the ISP). This application requires frequent turning (reallocation of threads to processes and customers to queues), especially in reaction to unexpected or abnormally large spikes in traffic. 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 adjust 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. The commercial requirement for this application is for a subsecond response time for tuning (i.e., resolving the ISP).

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. (2002a) develop heuristic solution procedures that are tested on bank-location data. Li et al. (1999) apply a similar model to determine the optimal placement of proxy websites. Castillo et al. (2009) 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 (2002) 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 Boffey et al. (2007) and Berman and Krass (2015) catalog numerous other variations on the standard ISP such as models/applications with finite queues.

Since ISP is $\mathcal{NP}$-hard (Agnihothri et al., 1990; Amiri, 1997), heuristic solution procedures are common in the literature. Citing the computational complexity of the standard problem, Amiri (1997) proposes two heuristics solution procedures based on Lagrangian 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 (2006) 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 of iterations and produce a solution that is optimal for the original, nonlinear 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 Elhedhli (2006) 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.

1.1. Exact methods

Vidyarthi and Jayaswal (2014) apply a similar approach to the one in Elhedhli (2006) 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.

Stiermaier (2010) considers a budget for the number of servers assigned, but the objective function does not have the capacity cost term (Ahmadi-Javid and Hoseinpour, 2022). Elhedhli et al. (2018) 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 nonlinear 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 (2022) and Aboolian et al. (2022) generalize the model in Elhedhli et al. (2018) to consider a general nondecreasing concave cost function for capacity. A key distinction between these two studies is that the model in Hoseinpour (2022) allows the decisionmaker to assign a customer to any open facility whereas Aboolian et al. (2022) address the consumer-choice scenario in which customers go to the nearest open facility.

The content delivery network 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. (2012a) 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. (2010) 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, Marianov et al. (2009) consider an $M/E_r/n/N$ model in which given service-level requirements are enforced by constraints on server utilization.

1.2. Metaheuristics

Pasandideh and Chambari (2010) and Chambari et al. (2011) adapt standard genetic algorithms (GAs) from the literature—nondominated ranked GA for solving multiobjective optimization problems (NRGA) (Al Jadaan et al., 2008), and the fast elitist nondominated sorting GA for multiobjective optimization (NSGA-II) (Deb et al., 2000)—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 chromosomes for individuals in the population. The initial population of solutions are essentially random assignments of customers to facilities and random allocations of 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 six to 22 customers and five 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 (2012) 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 Chambari et al. (2011) and Pasandideh and Chambari (2010). The GA uses a desirability function (Derringer and Suich, 1980) 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 Pasandideh and Chambari (2010) and Chambari et al. (2011), 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 three to 150 and four 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 dataset, 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. (2013) further extend the model by including a third objective, cost, and treating each service facility as an $M/M/s$ queue. They propose a multiobjective version of the harmony search metaheuristic (MOHS) framework (Geem et al., 2001) and test it against GA's based on NRGA and NSGA-II on a set of 20 problem instances with up to 3500 customers, and 700 servers distributed among 1100 potential facility locations. Like a GA, 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 one generation of the population to the next. The algorithms in Rahmati et al. (2013) were implemented in MATLAB on a laptop with a 2 GHz 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 GAs in measures of the quality of the Pareto frontier (e.g., diversity and number of Pareto-optimal solutions).

Hajipour et al. (2014) extend the model even further by introducing an additional budget for capacity. They compare genetic and harmony search algorithms similar to those proposed by Rahmati et al. (2013) and a multiobjective simulated annealing algorithm. As in Rahmati et al. (2013), the algorithms were implemented in MATLAB on a laptop with a 2 GHz CPU and 8 GB of RAM and tested on a set of 25 problem instances with up to 5500 customers and 1800 facilities. The harmony search algorithm solved all problem instances within a minute of CPU time.

Arkat and Jafari (2016) address an 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 GA for the problem. The solution approaches are tested on problems with 10–25 customers and 3–15 facilities.

Zamani et al. (2021) 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 GA and an Ant Lion heuristic (Mirjalili, 2015) to solve this complex variation of the fundamental problem as well as a nonlinear mixed integer program that they attempt to solve with Baron (Kılınç and Sahinidis, 2018). In their numerical testing, they allow Baron up to three hours of CPU time to solve a given problem instance. Testing instances with 50–200 customers and 10–30 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 of 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 evaluate their heuristic solution approaches with a comparison to an exact method.

The heuristics for the ISP surveyed in this section are fundamentally different than the one developed in this paper. 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 making the algorithm effective, but it does not 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 multiobjective 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 paper has a single cost function in which the user-defined trade-offs between competing objectives are specified in the data.

1.3. 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. (1990) 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 (1997, 1998, 2001) describes a similar heuristic for attempting to construct a primal solution at each step of a Lagrangian relaxation procedure.

Stiermaier (2010) proposes metaheuristics for a special case of the ISP proposed by Aboolian et al. (2008) 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 Aboolian et al. (2008). 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).

1.4. Greedy heuristics

We found only two examples of stand-alone greedy heuristics for ISPs in the literature. Wang et al. (2002a) 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 has the least cost 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 for the uncapacitated facility location (UFL) problem. To improve solution quality, the GD heuristic is embedded in a Tabu search (TS) (Glover and Laguna, 1997) framework. Both heuristics are compared to a Lagrangian relaxation scheme, which is shown to converge to solutions with a given optimality tolerance, in a computational 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 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 (16–40 minutes) in some cases.

In Agnihothri et al. (1990), 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 postprocessing/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. (2002a). This type of heuristic is the most similar we have found in the literature to the heuristic developed in this paper. One important distinction is that unlike the problem studied in this paper, the ISPs considered by Agnihothri et al. (1990) and Wang et al. (2002a) do not involve deciding how much capacity to allocate to the servers. The heuristic developed in this paper is novel in that it employs multiple capacity allocation mechanisms and multiple greedy customer-assignment strategies.

2. Contributions

This paper develops a fast, effective heuristic for the ISP 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 paper 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 datasets have been made available to share with other researchers (Colley, 2021). 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 content delivery networks. The heuristic was also shown to improve the performance of the exact methods by quickly finding high-quality incumbent solutions.

The rest of this paper is structured as follows. In Section 3, we formally state the ISP as a nonlinear mixed integer program. In Section 4, we briefly review Elhedhli's cutting plane algorithm for the ISP and introduce a new mixed integer linear programming formulation, the Colley model, in Section 5. In Section 6, we compare the relative sizes of the formulations described in Sections 3–5. We describe our heuristic framework for solving ISP in Section 7 and testbed of problem instances

in Section 8. We summarize an extensive computational experiment with the exact and heuristic approaches in Section 9. In Section 10, we draw conclusions from the results presented in Section 9 and suggest directions for further investigation.

3. Immobile server problem

The input for the problem consists of a set of traffic streams (customers) denoted by $i \in \{1, 2, \ldots, m\}$, single-server queues (facilities) denoted by $j \in \{1, 2, \ldots, n\}$, and server-capacity levels denoted by $k \in \{1, 2, \ldots, K\}$, 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. Using binary decision variables x_{ij} to indicate that customer i is assigned to queue j and y_{jk} to indicate that queue j is operating at capacity level k, the base nonlinear ISP is

minimize

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

subject to

$$\sum_{i=1}^{m}\lambda_i x_{ij} - \sum_{k=1}^{K}\mu_{jk}y_{jk} \leq 0 \qquad j \in \{1, \ldots, n\}, \tag{2}$$

$$\sum_{k=1}^{K} y_{jk} \leq 1 \qquad j \in \{1, \ldots, n\}, \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1, \ldots, m\}, \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad i \in \{1, \ldots, m\},\ j \in \{1, \ldots, n\}, \tag{5}$$

$$y_{jk} \in \{0, 1\} \qquad j \in \{1, \ldots, n\},\ k \in \{1, \ldots, K\}. \tag{6}$$

The objective function (1) is to minimize the facility assignment cost, the customer waiting-time cost, and the facility capacity cost. Constraint (2) forces there to be sufficient capacity assigned to the facility to meet the demand assigned to the facility. Constraint (3) specifies that a facility may only have one capacity level assigned (as y_{jk} is binary from constraint (6)) but may also have no capacity level assigned—signifying the facility is unused (not opened). Constraint (4) specifies that a customer must be assigned to exactly one facility (as x_{jk} is binary from constraint (5)).

4. Elhedhli cutting plane algorithm

Elhedhli (2006) proposed a cutting-plane algorithm that iteratively solves a sequence of linearized formulations of the ISP given by

minimize

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\lambda_i x_{ij} + t\sum_{j=1}^{n} R_j + \sum_{j=1}^{n}\sum_{k=1}^{K} f_{jk}y_{jk} \tag{7}$$

subject to

$$\sum_{i=1}^{m} \lambda_i x_{ij} - \sum_{k=1}^{K} \mu_{jk} z_{jk} = 0 \qquad j \in \{1,\ldots,n\}, \tag{8}$$

$$\sum_{k=1}^{K} y_{jk} \le 1 \qquad j \in \{1,\ldots,n\}, \tag{9}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1,\ldots,m\}, \tag{10}$$

$$z_{jk} - y_{jk} \le 0 \qquad j \in \{1,\ldots,n\}, k \in \{1,\ldots,K\}, \tag{11}$$

$$z_{jk} - \frac{R_j}{(1+r_{jh})^2} \le \frac{r_{jh}^2}{(1+r_{jh})^2} \qquad j \in \{1,\ldots,n\}, k \in \{1,\ldots,K\}, h \in \{1,\ldots,H\}, \tag{12}$$

$$x_{ij} \in \{0,1\} \qquad i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}, \tag{13}$$

$$y_{jk} \in \{0,1\} \qquad j \in \{1,\ldots,n\}, k \in \{1,\ldots,K\}, \tag{14}$$

$$0 \le z_{jk} \le 1 \qquad j \in \{1,\ldots,n\}, k \in \{1,\ldots,K\}, \tag{15}$$

$$0 \le R_j \qquad j \in \{1,\ldots,n\}. \tag{16}$$

Here H is an index set of cutting planes that is initially empty. At the end of iteration h, the r_{jh} values (waiting times at the facilities based on the current solution) are set using the following equation:

$$r_{jh} = \frac{\sum_{i=1}^{m} \lambda_i x_{ij}}{\sum_{k=1}^{K} \mu_{jk} y_{jk} - \sum_{i=1}^{m} \lambda_i x_{ij}} \qquad j \in \{1,\ldots,n\}. \tag{17}$$

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. At the end of any iteration, the value of the objective function (7) for the current solution provides a lower bound on the cost of an optimal solution to the given ISP instance and an upper bound is derived from evaluating (1) with the current values of the x and y variables. Elhedhli (2006) recommends a stopping condition where the gap between the upper bound and lower bound converges to a targeted value and used 10^{-3} in his computational experiments. For this study, we

considered three other possible stopping criteria for the algorithm: (1) stop when the same solution has been found twice in a row (indicating no new cutting planes will be created), (2) the gap in solution cost between iterations has converged to a specified target, and (3) the gap between the successive values for R_j have converged to a specific target. In preliminary experiments, we found that using the first two new criteria found the same or slightly better solutions than using the stopping criterion proposed by Elhedhli (2006) while the results with the third criterion were mixed.

5. Colley mixed-integer linear model

In this section, we present the Colley model, which is a new mixed-integer linear programming (MIP) for the ISP that can be solved with a straightforward application of an MIP solver. In order to linearize the objective function (1), the model introduces a continuous decision variable q_{jk}, which represents 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 $t \sum_{j=1}^{n} \sum_{k=1}^{K} q_{jk}$. The model also uses continuous decision variable v_{ij} that takes the value $1 + \sum_{k=1}^{K} q_{jk}$ 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 at least one plus the largest possible average waiting time resulting from any feasible solution to the ISP instance. The smallest possible value for M can be found by solving a knapsack problem or estimated by taking the largest μ_{jk} value if all input parameters are integers. The Colley model for the ISP is the following mixed-integer linear program:

minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \lambda_i x_{ij} + t \sum_{j=1}^{n} \sum_{k=1}^{K} q_{jk} + \sum_{j=1}^{n} \sum_{k=1}^{K} f_{jk} y_{jk} \tag{18}$$

subject to

$$\sum_{i=1}^{m} \lambda_i v_{ij} - \sum_{k=1}^{K} \mu_{jk} q_{jk} = 0 \qquad j \in \{1, \ldots, n\}, \tag{19}$$

$$\sum_{k=1}^{K} y_{jk} \leq 1 \qquad j \in \{1, \ldots, n\}, \tag{20}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1, \ldots, m\}, \tag{21}$$

$$q_{jk} \leq M y_{jk} \qquad j \in \{1, \ldots, n\}, k \in \{1, \ldots, K\}, \tag{22}$$

$$v_{ij} \geq \left(1 + \sum_{k=1}^{K} q_{jk}\right) - M\left(1 - x_{ij}\right) \qquad i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}, \tag{23}$$

$$x_{ij} \in \{0, 1\} \qquad i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}, \tag{24}$$

$$y_{jk} \in \{0, 1\} \qquad j \in \{1, \ldots, n\}, k \in \{1, \ldots, K\}, \tag{25}$$

$$0 \leq q_{jk} \qquad j \in \{1, \ldots, n\}, k \in \{1, \ldots, K\}, \tag{26}$$

$$0 \leq v_{ij} \qquad i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}. \tag{27}$$

Intuitively, (19) and (23) together with the objective function ensure that the q and v variables have the intended values. If facility j is closed, the objective function will drive y_{jk} and q_{jk} to zero for all k; and the solution will have $v_{ij} = 0$ for all i to satisfy (19). Now suppose that facility j is open. If customer i is assigned to the facility, then (23) forces $v_{ij} \geq 1 + \sum_{k=1}^{K} q_{jk}$ and the objective function will drive v_{ij} to be exactly equal to $1 + \sum_{k=1}^{K} q_{jk}$. If customer i is not assigned to the facility, then (23) is nonbinding and the objective function and (19) will drive v_{ij} to zero. Hereinafter, we say that a feasible solution to the Colley model is *locally optimal* if the v and q variables are as small as possible for the given values of the x and y variables.

Next, we argue that the model correctly determines the average waiting times in locally optimal solutions. For ease of exposition, we introduce some additional notation related to given binary values for the x and y variables that satisfy the capacity and demand-assignment constraints, (20) and (21). We introduce $I_j \subseteq \{i \in 1, \ldots, m : x_{ij} = 1\}$ to denote the set of customers assigned to facility j, and $\hat{\lambda}_j = \sum_{i=1}^{m} \lambda_i x_{ij} = \sum_{i \in I_j} \lambda_i$ and $\hat{\mu}_j = \sum_{k=1}^{K} \mu_{jk} y_{jk}$ to denote the arrival rate and capacity at the facility, respectively. For each facility j, we introduce parameter $\hat{q}_j$ and set $\hat{q}_j = \sum_{k=1}^{K} q_{jk}$ if j is open and $\hat{q}_j = 0$ if it is not open.

Lemma 1. *In a locally optimal solution to the Colley model, $\hat{q}_j$ is equal to the average waiting time at facility j if the facility is open and zero otherwise.*

Proof. There are two cases to consider for facility j:

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

$$\sum_{i=1}^{m} \lambda_i v_{ij} = \sum_{k=1}^{K} \mu_{jk} q_{jk} \qquad \Rightarrow \tag{28}$$

$$\sum_{i \in I_j} \lambda_i v_{ij} = \sum_{k=1}^{K} \mu_{jk} q_{jk} \qquad \Rightarrow \tag{29}$$

$$\sum_{i \in I_j} \lambda_i v_{ij} = \hat{\mu}_j \hat{q}_j \qquad \Rightarrow \tag{30}$$

$$\sum_{i \in I_j} \lambda_i (1 + \hat{q}_j) = \hat{\mu}_j \hat{q}_j \qquad \Rightarrow \tag{31}$$

$$\sum_{i \in I_j} \lambda_i + \sum_{i \in I_j} \lambda_i \hat{q}_j = \hat{\mu}_j \hat{q}_j \qquad \Rightarrow \tag{32}$$

$$\hat{\lambda}_j + \hat{\lambda}_j \hat{q}_j = \hat{\mu}_j \hat{q}_j \qquad \Rightarrow \tag{33}$$

$$\hat{q}_j = \frac{\hat{\lambda}_j}{\hat{\mu}_j - \hat{\lambda}_j}. \tag{34}$$

Table 1
Mathematical model size comparison

Model	Constraints	Decision variables
Nonlinear	$2n + m + mn + nK$	$mn + nK$
Elhedhli	$2n + m + mn + 2nK + nKH$	$mn + 2nK + n$
Colley	$2n + m + 2mn + 2nK$	$2mn + 2nK$

Equation (29) follows from Equation (28) (constraint (19)) because the solution is locally optimal. Equation (30) follows from Equation (29) because constraints (20) and (25) imply that $q_{jk} = 1$ for at most one capacity level k. Equation (31) follows from Equation (30) by the definition of $\hat{q}$, and Equation (33) follows from Equation (32) by the definition of $\hat{\lambda}$.

Case 2. Facility j is not open (i.e., $\sum_{i=1}^{m} x_{ij} = 0$):

From constraint (23), all v_{ij} for facility j can be equal to zero, allowing $\hat{q}_j$ to also equal 0 from constraint (19). □

6. 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 1.

While the Elhedhli model grows with each iteration, the formula listed above would be applicable to the final iteration which has the largest number of constraints. The size of the model depends on the problem instance and the number of iterations required, which in turn can be affected by whether or not an initial solution is given. The more iterations required, the larger the model becomes and the more time is required for solving each additional iteration. Even with this, we found that problem instances that were solved to provable optimality typically were solved fastest with the Elhedhli's cutting plane algorithm.

7. Heuristic

The heuristic developed in this paper is designed for applications with stringent solution-time requirements such as rapidly adapting to demand spikes in content delivery networks. This section provides an overview of the heuristic. (See Colley, 2022, for a detailed description with pseudocode and illustrated examples.) The heuristic has two phases, the first of which comprises multiple variations of a greedy constructive algorithm (Burke and Kendall, 2014). The second phase of the heuristic applies local search procedures to attempt to improve on the best solution found in the first phase.

7.1. Phase 1

In simplest terms, the greedy constructive algorithm makes an initial capacity allocation to the facilities according to a given rule and then assigns customers to facilities one at a time according

to a given sort order. In a generic step of the algorithm, the next customer on the sorted list is assigned to a facility that minimizes the incremental increase in total cost given the existing customer assignments. Assuming that the capacities are fixed at their initial values, the incremental increase in cost is calculated as follows.

Suppose that the capacity allocated to facility j is μ_{jk} and that the total arrival rate of the customers currently assigned to the facility is $\hat{\lambda}_j$. If the next customer, i, is assigned to facility j, then the incremental increase in cost is $c_{ij}\lambda_i + t(\frac{\hat{\lambda}_j+\lambda_i}{\mu_{jk}-(\hat{\lambda}_j+\lambda_i)} - \frac{\hat{\lambda}_j}{\mu_{jk}-\hat{\lambda}_j})$. The greedy constructive algorithm assigns customer i to a facility j that has sufficient residual capacity to accommodate the addition of customer i (i.e., $\mu_{jk} - \hat{\lambda}_j > \lambda_i$) and minimizes the incremental increase in cost.

Each variation of the greedy constructive algorithm applied in this study used one of three capacity-allocation rules: (1) forced maximum capacity, (2) forced minimum capacity, and (3) unforced capacity.

7.1.1. Forced maximum capacity

Using the forced maximum capacity rule, the algorithm proceeds as described above and each facility is initially allocated the maximum possible capacity (i.e., the capacity allocated to facility j is the largest μ_{jk} value). After the last customer is assigned to a facility, variations of the algorithm using this capacity-allocation rule evaluate each facility and, if feasible, reduce the capacity levels if the decrease in capacity cost is larger than the resulting increase in waiting-time cost.

7.1.2. Forced minimum capacity

Using the forced minimum capacity rule, the capacity allocation can change with the customer assignments. Initially, all facilities are closed (unused). If customer i is the first customer assigned to a closed facility j, then facility j is opened and allocated the smallest capacity level k such that $\mu_{jk} > \lambda_i$. If customer i is assigned to a facility j that is already open then there are two cases to consider:

1. $\mu_{jk} > \hat{\lambda}_j + \lambda_i$.
   In this case, the incremental increase in cost is $c_{ij}\lambda_i + t(\frac{\hat{\lambda}_j+\lambda_i}{\mu_{jk}-(\hat{\lambda}_j+\lambda_i)} - \frac{\hat{\lambda}_j}{\mu_{jk}-\hat{\lambda}_j})$.
2. $\mu_{jk} \leq \hat{\lambda}_j + \lambda_i$.
   In this case, the heuristic increases the capacity allocation to facility j to the smallest k' such that $\mu_{jk'} > \hat{\lambda}_j + \lambda_i$ making the incremental cost $c_{ij}\lambda_i + t(\frac{\hat{\lambda}_j+\lambda_i}{\mu_{jk'}-(\hat{\lambda}_j+\lambda_i)} - \frac{\hat{\lambda}_j}{\mu_{jk}-\hat{\lambda}_j}) + f_{jk'} - f_{jk}$.

After the last customer is assigned to a facility, variations of the algorithm using this capacity-allocation rule evaluate each facility and, if feasible, increase the capacity level if the increase in capacity cost is less than the resulting decrease in waiting-time cost.

7.1.3. Unforced capacity

Variations of the algorithm using the unforced capacity rule follow the same logic as those using forced minimum capacity except that when the capacity level at facility j is increased it can be increased to any k' such that $\mu_{jk'} > \hat{\lambda}_j + \lambda_i$ and the capacity level is selected to minimize the

incremental increase in cost. Because these variations update the capacity level based on the best cost at the time the customer is assigned, they do not evaluate the solution for any capacity changes after the last customer has been assigned to a facility.

7.1.4. Sort orders

Each variant of the greedy constructive algorithm applies a different sort order to the customers. We used 20 sort orders in our computational experiments. The sort orders were derived from eight fundamental types. Seven of the types were calculated from the data to provide a deterministic search (Burke and Kendall, 2014). The seven types derived from the data used the following sort criteria: λ_i, c_{min}, c_{avg}, c_{max}, $\lambda_i c_{min}$, $\lambda_i c_{avg}$, and λ_i, c_{max}, where c_{min}, c_{avg}, and c_{max} are the minimum, average, and maximum of the c_{ij} values for customer i. The first seven sort orders correspond to applying each of the criteria in nondecreasing order, and the next seven correspond to applying them in nonincreasing order.

The eighth type of sort order comprised three random orders that were then reversed for the final three sort orders. The random orders were included to allow for a potentially better solution to be found. In preliminary experiments, Phase 1 found a better solution because of the randomized lists than it would have found without it. Additionally, we found that including the randomized orderings improved the solutions returned by Phase 2 for 54.35% of the problem instances.

Altogether, we used a combination of three capacity-allocation rules and twenty sort orders for a total of 60 different variations of the greedy constructive algorithm in Phase 1 of the heuristic. Using multithreading, these variations were executed in parallel and the best feasible solution was returned as the result. In cases where two or more solutions are tied for the minimum cost, the heuristic returns the one whose thread finished first.

7.2. Phase 2

Given the solution from Phase 1, Phase 2 of the heuristic continues to loop for a user-specified time limit or until the solution no longer changes. Within this outer loop, three subroutines are executed in the sequence of one customer move, two customer move, and swap customers.

The one customer move routine takes an initial list of all customers and analyzes the list for the best customer to reassign to a new facility (i.e., the customer that if reassigned to another facility would decrease the cost the most or increase it the least). If the best customer move increases the solution cost, that customer, i, and all similar customers are removed from the list where customers i and i' are similar if they are assigned to the same facility j, $\lambda_i = \lambda_{i'}$ and $c_{ij} = c_{i'j}$. Otherwise, the best customer is reassigned and removed from the list. This process is repeated until the list is empty or no feasible reassignment can be made.

The two customer move routine finds the best customer to be reassigned to a new facility, as in one customer move, and then uses that solution to find the second best customer to reassign to a new facility 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. There are two cases to consider:

1. If a better solution is found, the routine terminates early and Phase 2 returns to the beginning of the sequence; that is, it applies the one customer move to the new solution.

2. If a better solution is not found, the previously best customer from the first loop is removed from the customer list and the routine tries again.

If one customer move terminates without finding a better solution, Phase 2 moves on to the swap customer routine. The swap customers routine loops through all two-customer combinations where the customers are in different facilities 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.

7.3. 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 problem instance, it is the most common scenario. Given this assumption, the complexity for Phase 1 of the heuristic can be shown to be $O\left(mn^2K\right)$ and the complexity for Phase 2 of the heuristic can be shown to be $O\left(m^3n^2K^2\right)$ (Colley, 2022).

8. Problem instances

For comparison purposes, the 695 problem instances being tested were broken into six logical groupings. The groupings either pull from the same data source or differ based on how the costs were structured. We also consider the number of solutions that would need to be evaluated to find an optimal solution for a given problem instance 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 show the value as a power of 10.

8.1. Holmberg problem instances

The Holmberg problem instances come from Holmberg et al. (1999) and were provided for research here as a set of data that has been analyzed using various other methodologies previously (e.g., Elhedhli, 2006; Ahmadi-Javid and Hoseinpour, 2022). It consists of 30 distinct data points of four subsets of data. Each problem instance within the subsets varies t using $\beta \in \{0.1, 1.0, 10.0\}$, where t is calculated as β times the maximum $c_{ij}\lambda_i$ value.

The first subset of data contains four problem instances of 50 customers, 10 facilities, and three capacity levels that are varied by μ_{jk} between the four problem instances. They represent problem instances that could have as many as 10^{50} possible facility assignments. Since only μ_{jk} varies, the t value is consistent across all four problem instances and is modified by β. This block of 12 problem instances represents a medium-sized set of customers.

The second subset of data contains one problem instance of 50 customers, 20 facilities, and three capacity levels. It represents a problem instance that could have 20^{50} possible facility assignments. The t value is modified by β.

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

The fourth subset of data contains two problem instances of 100 customers, 10 facilities, and three capacity levels that are varied by λ_i between the two problem instances. They represent a problem instance that could have 10^{100} possible facility assignments. Since only λ_i varies, the t value also varies across all three problem instances. This block of six problem instances represents anextra-large sized set of customers.

The brute-force solution space size for the Holmberg problem instances ranges from $10^{56.02}$ to $10^{106.02}$. The Phase 1 complexity bound on the number of basic operations (mn^2K) ranges from 15,000 to 60,000. The Phase 2 complexity bound ($m^3n^2K^2$) ranges from 112,500,000 to 900,000,000.

8.2. Small, varying costs problem instances

The small, varying costs problem instances were a single problem instance of 25 customers, five facilities, and three capacity levels that are varied in three subsets to get 36 data points. It represents a problem instance 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 three problem instances that are varied by μ_{jk}. This block of 12 problem instances shows a small-sized set of customers and how t and μ_{jk} affect sensitivity. The second subset of data contains three problem instances that are varied by f_{jk}. This block of 12 problem instances shows a small-sized set of customers and how t and f_{jk} affect sensitivity. The third subset of data contains three problem instances that are varied by c_{ij}. This block of 12 problem instances shows a small-sized set of customers and how t and c_{ij} affect sensitivity.

The brute-force solution space size for the small, varying costs group is $10^{20.48}$. The Phase 1 complexity bound is 1875. The Phase 2 complexity bound is 3,515,625.

8.3. Various, linear costs

The various, linear costs problem instances 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 could vary, and wanting a consistent way to build each problem instance, the customers were grouped into 18, 36, or 72 based on similarities. Facilities were set to either three, six, or nine. The capacity levels are five, seven, and nine. The value for t was arbitrarily set to 600 and varied by $\beta \in \{0.1, 1.0, 10.0\}$.

Table 2
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

Since that would give 81 possible combinations of settings per client, a Taguchi L9 orthogonal array (Taguchi, 1986) was used to get to a set of nine combinations per client—giving a total of 54 data points in this group. For each client, the L9 array is in Table 2.

The c_{ij} values were calculated as zero 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 a multiplier for the client and then linearly incremented for each capacity level.

The f_{jk} values started with zero 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 (Chandra and Weisstein, 2021) 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 a 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 bound ranges from 1134 to 29,160. The Phase 2 complexity bound ranges from 2,571,912 to 755,827,200.

8.4. 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 nine, and the number of capacity levels was set to five. This provides an explicit nine data points per client. The goal was to determine how nonlinear costs affected 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 provided a nonlinear scaling of the capacity to decrease the number of capacity

levels that would have been required. If done linearly, it would have represented 32 capacity levels instead of five capacity levels. 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 bound ranges from 7290 to 29,160. The Phase 2 complexity bound ranges from 11,809,800 to 755,827,200.

8.5. Sizing

The sizing problem instances were generated to determine how well the heuristic and mathematical models could handle larger problem instances. 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 problem instances.

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 1000×1000 grid and calculating the next highest integer of the distance between the customers and facilities plus 1 (to ensure no zero costs). The facility 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 bound ranges from 781,250 to 2,000,000,000. The Phase 2 complexity bound ranges from 244,140,625,000 to 4,000,000,000,000,000,000.

8.6. Saw-tooth

The saw-tooth problem instances were derived from a single problem instance with five customers, three facilities, and three capacity levels that are varied to get nine problem instances. This ultra-small problem instance was created for the sole purpose of demonstrating how the customer waiting-time cost affects the sensitivity of solving even a small problem. 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 group 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 bound is 135, and the Phase 2 complexity bound 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

Table 3
Problem instance data from the ISP literature

Reference	Problem instances	Maximum number of		
		Customers	Facilities	Capacity levels
Aboolian et al. (2008)	40	800	200	1
Aboolian et al. (2012a)	40	100	100	3
Agnihothri et al. (1990)	270	20	300	1
Amiri (1997)	150	500	40	5
Amiri (1998)	90	300	20	5
Amiri (2001)	150	200	30	5
Arkat and Jafari (2016)	10	15	25	1
Chambari et al. (2011)	15	20	22	1
Pasandideh and Chambari (2010)	15	20	22	1
Elhedhli (2006)	55	100	20	3
Elhedhli et al. (2018)	55	150	30	N/A
Hajipour et al. (2014)	25	5,500	1,800	N/A
Hoseinpour (2022)	27	150	30	N/A
Pasandideh and Niaki (2012)	12	65	150	1
Rahmati et al. (2013)	20	3,500	700	N/A
Rajagopalan and Yu (2001)	10	30	15	N/A
Vidyarthi and Jayaswal (2014)	216	400	25	5
Wang et al. (2002a)	90	459	84	3
Zamani et al. (2021)	71	200	30	1

{{30, 60, 90}, {40, 80, 120}, {30, 60, 120}} to try and determine if linear versus nonlinear capacity levels make a difference in the practical (as opposed to theoretical) sensitivity of the problem.

8.7. Problem instances in existing literature

The solution techniques described in Sections 5 and 7 were tested on 695 problem instances with up to 10,000 customers, 100 facilities, and 20 capacity levels. To put this in context, Table 3 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 Rahmati et al. (2013) are $M/M/s$ queues and the decision is to decide how many identical servers to assign to each facility.

9. Summary of results

We tested the 17 algorithms for the ISP listed in Table 4. The first group of algorithms listed in the table, Phase 1 and Phase 1 + 2, use the heuristic described in Section 7. We refer to the process of stopping the heuristic after Phase 1 (i.e., using the best greedy solution) as the Phase 1 algorithm

Table 4
Algorithm abbreviations and inclusion in each analysis method

Algorithm	Abbreviation	Analysis method Independent	Interactive	Complete
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			√

(or Phase 1 for short), and running the complete heuristic as the Phase 1 + Phase 2 algorithm (or Phase 1 + Phase 2). The second group of algorithms listed in Table 4 uses the Colley model described in Section 5. We refer to solving the Colley model with a straightforward application of the Gurobi solver as the Colley algorithm. The Colley w/Phase 1 incumbent algorithm consists of first running the Phase 1 algorithm and then running the Colley algorithm with the solution returned by the heuristic as an initial incumbent solution in the branch-and-bound process. The Colley w/Phase 2 incumbent algorithm uses the solution returned by Phase 2 of the heuristic as the incumbent. The third group of algorithms in Table 4 is analogous to the third group, but they use the Baron nonlinear mixed integer solver to solve the nonlinear formulation of the ISP given in Section 3.

The last three groups of algorithms in Table 4 use Elhedhli's cutting plane algorithm to solve the ISP. The terms "Solution," "Cost," and "Rr" in the algorithm names refer to stopping criteria. The Elhedhli solution algorithms stop when the values of the x and y variables at the end of one iteration are the same as the values at the end of the previous iteration. The Elhedhli cost algorithms stop when the absolute difference between the solution costs (7) in two consecutive iterations is within 1e−08. The Elhedhli Rr algorithms stop at the end of iteration h if the absolute difference between R_j and r_{jh} for each facility j is within 1e−03.

Our experiments were performed on an HP Server model DL380 with Dual 14 Core Intel Xeon@2.6 GHz processors and 380 GB of RAM. The mathematical programming models were implemented in AMPL version 20200810 (AMPL, 2009). The Colley and Elhedhli models were solved with Gurobi version 9.1.2 (Gurobi Optimization, LLC, 2021), and the nonlinear formulation was solved with Baron version 21.1.13 (Kılınç and Sahinidis, 2018). The heuristic was implemented in Java 1.6.0 (Oracle, 2003). A pause of one second between each AMPL or Java call was performed to

allow for the system to return to an idle state. All algorithms were given a five-minute (300-second) time limit in our experiments. The five-minute time limit was selected as a relatively long, but acceptable run time given the motivating applications from Colley (2022). Runs that terminated due to reaching the limit are said to have *timed out*. We observed slight variations in running time between multiple runs of the same combination of algorithm and problem instance. This is due in part to the nature of multithreading and in part because we ran our experiments on a shared computer. The random sort orders in Phase 1 also contributed to the variation. Therefore, the experiments described in this section included three runs of each algorithm-instance combination.

The results are analyzed in three ways, each representing a different use case. First, we present independent analysis considering only the most straightforward algorithms in which there is no interaction between the heuristic and exact methods. That is, we compare the Phase 1, Barron, Colley, Elhedhli solution, Elhedhli cost, and Elhedhli Rr algorithms as stand-alone solution approaches. Next, we present interactive analysis, which includes the "w/Phase 1" algorithms. Finally, the complete analysis considers all 17 algorithms.

Because there were three runs for each combination of problem instance and algorithm, 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 model applied to a particular problem instance starting with the Phase 1 heuristic solution 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 complete Phase 1.

Given that the solution time was measured in milliseconds, a unique *top performer* was identified for each combination of problem instance and analysis method. To be considered the top performer for the combination, the algorithm had to have the best solution cost with the shortest time. For convenience, we refer to the best solution found for a given problem instance as "optimal" even if the run that produced it timed out. In some cases, other algorithms within the same analysis method found solutions with the same solution cost as the one found by the top performer within a margin of one second per run. Interested readers are referred to Colley (2022) for an exhaustive cohort analysis of such cases.

9.1. Independent analysis

The independent analysis method summarized in Tables 5 and 6 makes a very strong case for using Phase 1 of the heuristic over the five exact methods as the heuristic Phase 1 found its best solution in under one second for all datasets in the Holmberg group. As indicated by the "Found" column in the table, Phase 1 of the heuristic found feasible solutions to all 695 problem instances. The Colley and Baron algorithms found feasible solutions to 363 (52.23%) and 182 (26.19%) problem instances, respectively. The best performing variants of the Elhedhli algorithm, which were proposed in this paper, each found feasible solutions to 695 (100.00%) of the problem instances. Only 61.29% 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 441 (63.45%) of the problem instances as opposed to the next best performer, Elhedhli cost, which found

Table 5
Independent top performer table

Algorithm	Found	"Optimal"	Top performer	
			Concurrent	Sequential
Phase 1	100.00%	63.45%	50.07%	49.93%
Colley	52.23%	31.94%	12.37%	12.23%
Baron	26.19%	9.06%	0.00%	0.00%
Elhedhli solution	100.00%	51.08%	17.12%	15.86%
Elhedhli cost	100.00%	51.37%	9.21%	8.63%
Elhedhli solution Rr	61.29%	34.24%	11.22%	13.53%

Table 6
Relative error statistics of the best heuristic solution by problem instance group

Problem instance group	Minimum	Mean	Maximum
Various, nonlinear	0.0000%	0.0000%	0.0000%
Various, linear	0.0000%	0.0072%	0.3131%
Small, varying	0.0000%	1.4135%	13.1333%
Holmberg	0.0000%	0.0255%	0.1555%
Sizing	0.0000%	1.9561%	8.1301%
Saw tooth	0.0000%	0.0000%	0.0000%

"optimal" solutions to 357 (51.37%) of the problem instances. The Colley and Baron algorithms found "optimal" solutions to 222 (31.94%) and 63 (9.06%) of the problem instances, respectively. As indicated in the "Top performer" columns, Phase 1 was the top performer on 348 (50.07%) of the problem instances when the three runs were made concurrently, and on 347 (49.93%) of the problem instances when the runs were made sequentially. The next best performer was the Elhedhli solution algorithm, which was a top performer on 119 (17.12%) and 109 (15.68%) of the problem instances when the runs were made concurrently and sequentially, respectively.

For a particular combination of algorithm and problem instance, we define the *relative error* as $\frac{Z-Z^*}{Z^*}$, where Z is the cost of the best solution for that instance found by the algorithm and Z^* is the cost of the "optimal" solution as described above. Table 6 gives summary statistics of the relative errors of the best solution found by Phase 1 of the heuristic for each of the six problem instance groups described in Section 8. Overall, the minimum, mean, and maximum relative errors for the Phase 1 heuristic were 0%, 1.5159%, and 13.1333%, respectively, for the 695 problem instances in our experiment. Allowing the heuristic to continue to Phase 2 found improved solutions to 243 (34.96%) of the problem instances. The improved solution was "optimal" in 31 of those 243 cases. Considering all 695 problem instances, the minimum, mean, and maximum relative errors for the Phase 2 solutions were 0%, 1.1931%, and 6.9212%, respectively.

Figure 1 shows the average solution time for the algorithms included in this analysis method. Note that the figure uses the abbreviated algorithm names given in Table 4. 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 algorithms are iterating, they can exceed the 300-second time limit which was

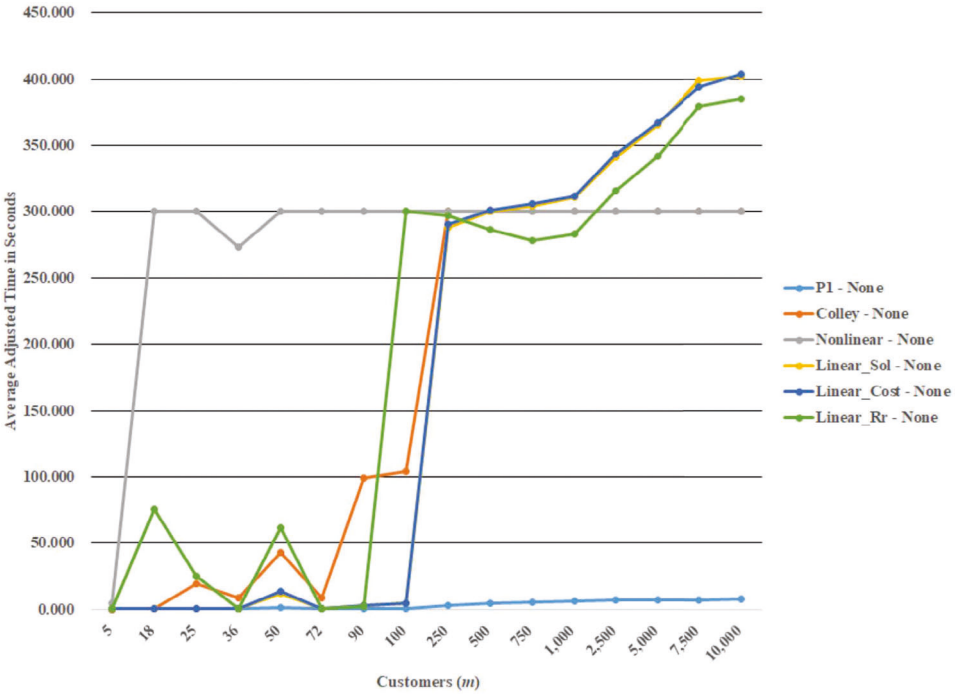

Fig. 1. Independent average solution time.

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 solving time before Gurobi began solving the next iteration. If Gurobi then hit the 300-second time limit, the total solve time was 400 seconds and the algorithm stopped before starting the next iteration.

9.2. *Interactive analysis*

Table 7 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 algorithm found "optimal" solutions to 332 (47.77%) instances when given the Phase 1 incumbent as opposed to only 185 (26.62%) instances when solved without an incumbent. The Elhedhli solution and Cost algorithms found "optimal" solutions to 498 (71.65%) and 503 (72.37%) of the problem instances when given the Phase 1 incumbent as opposed to 247 (35.54%) and 249 (35.83%) 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 63 (9.06%) of the problem instances; with the incumbent, that number increased to 150 (21.58%). As expected, the exact methods were more likely than the heuristic to be top performers in this experiment. Overall, these results show that if one is going to solve the ISP with an exact

Table 7
Interactive top performer table

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
Phase 1	100.00%	25.47%	12.09%	11.94%
Colley	52.23%	26.62%	6.33%	6.62%
Colley w/Phase 1 incumbent	100.00%	47.77%	18.99%	18.85%
Baron	26.19%	9.06%	0.00%	0.00%
Baron w/Phase 1 incumbent	64.89%	21.58%	0.00%	0.00%
Elhedhli solution	100.00%	35.54%	5.61%	4.46%
Elhedhli solution w/Phase 1 incumbent	100.00%	71.65%	22.88%	20.86%
Elhedhli cost	100.00%	35.83%	1.15%	1.29%
Elhedhli cost w/Phase 1 incumbent	100.00%	72.37%	14.10%	15.25%
Elhedhli Rr	61.29%	27.19%	5.18%	6.19%
Elhedhli Rr w/Phase 1 incumbent	76.69%	51.08%	13.67%	14.53%

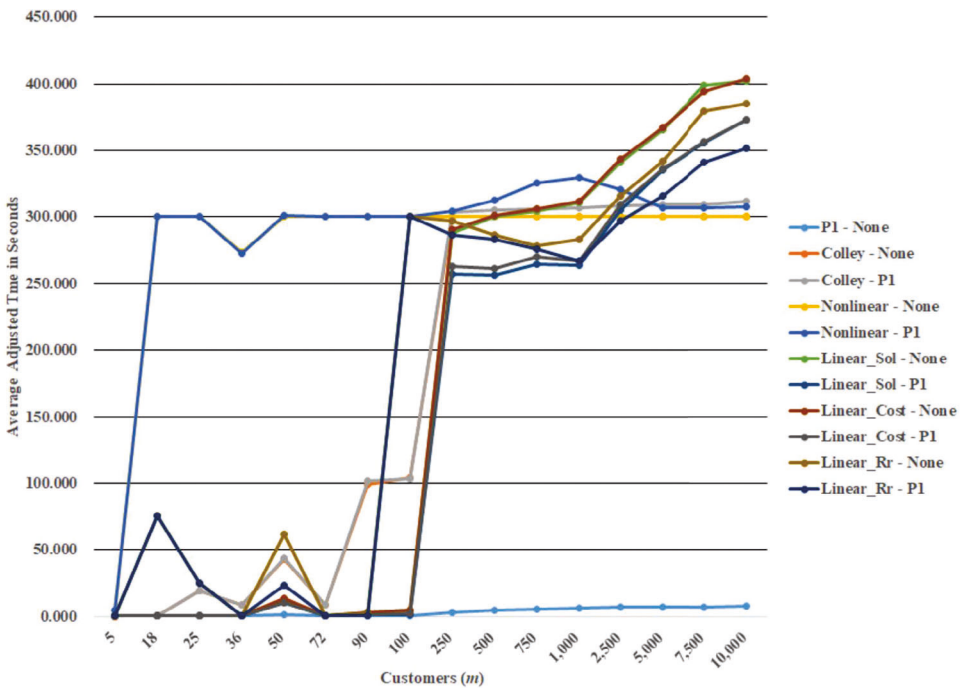

Fig. 2. Interactive average solution time.

method then it is worth the additional time to first find an incumbent solution with Phase 1 of the heuristic.

Figure 2 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 8
Complete top performer table

Algorithm	Found	"Optimal"	Top Performer	
			Concurrent	Sequential
Phase 1	100.00%	25.47%	12.09%	11.94%
Phase 1 + Phase 2	100.00%	29.93%	0.86%	0.86%
Colley	52.23%	26.04%	5.76%	5.90%
Colley w/Phase 1 incumbent	100.00%	37.99%	9.06%	9.06%
Colley w/Phase 2 incumbent	100.00%	45.04%	8.20%	8.20%
Baron	26.19%	9.06%	0.00%	0.00%
Baron w/Phase 1 incumbent	64.89%	21.58%	0.00%	0.00%
Baron w/Phase 2 incumbent	64.75%	24.32%	0.00%	0.00%
Elhedhli solution	100.00%	35.11%	5.47%	4.17%
Elhedhli solution w/Phase 1 incumbent	100.00%	68.35%	21.44%	19.42%
Elhedhli solution w/Phase 2 incumbent	100.00%	70.79%	2.73%	1.73%
Elhedhli cost	100.00%	35.40%	0.72%	1.01%
Elhedhli cost w/Phase 1 incumbent	100.00%	69.21%	12.95%	14.10%
Elhedhli cost w/Phase 2 incumbent	100.00%	71.94%	2.45%	3.02%
Elhedhli Rr	61.29%	27.19%	5.04%	6.19%
Elhedhli Rr w/Phase 1 incumbent	76.69%	49.78%	12.66%	13.38%
Elhedhli Rr w/Phase 2 incumbent	76.69%	51.37%	0.58%	1.01%

9.3. Complete analysis

Table 8 summarizes the complete analysis. Continuing the heuristic to Phase 2 improved the quality of the solutions; it found "optimal" solutions to 177 (25.47%) instances using only Phase 1 and 208 (29.93%) 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 484 of the 695 problem instances. When stopping after Phase 1, the heuristic never timed out.

Using the Phase 2 solution as an incumbent also improved the quality of the solutions found by the exact methods. For example, the Colley algorithm found "optimal" solutions to 264 (37.99%) problem instances in this experiment when it was given the Phase 1 incumbent; that increased to 313 (45.04%) with the Phase 2 incumbent. The improvements for the other exact methods were more modest. For example, the Elhedhli cost algorithm found "optimal" solutions to 481 (69.21%) problem instances in this experiment when it was given the Phase 1 incumbent; that increased to 500 (71.94%) 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 Elhedhli solution algorithm with the Phase 1 incumbent was a top performer 21.44% of the time and a top performer with the Phase 2 incumbent 2.73% 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 solution time.

Figure 3 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

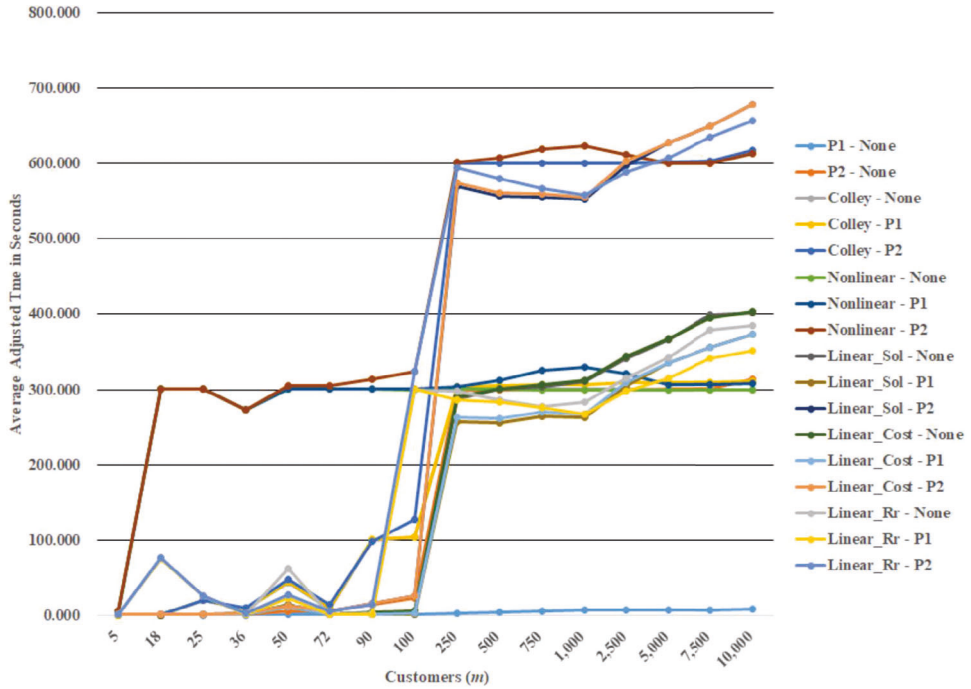

Fig. 3. Complete average solution time.

Table 9
Complete analysis: relative error statistics by algorithm

Algorithm	Minimum	Gap median	Maximum
Phase 1	0.00%	1.09%	13.13%
Phase 1 + Phase 2	0.00%	0.89%	6.92%
Colley w/ Phase 1	0.00%	1.09%	13.13%
Colley w/ Phase 2	0.00%	0.89%	6.92%
Elhedhli solution	0.00%	0.61%	2,531.00%
Elhedhli solution w/Phase 1	0.00%	0.00%	1,901.14%
Elhedhli solution w/Phase 2	0.00%	0.00%	1,740.30%
Elhedhli cost	0.00%	0.61%	2,531.00%
Elhedhli cost w/Phase 1	0.00%	0.00%	1,901.14%
Elhedhli cost w/Phase 2	0.00%	0.00%	1,740.30%

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

Table 9 summarizes relative error statistics for the best solutions found by each of the algorithms listed in Table 8 that found solutions to all problem instances. Phase 1 + Phase 2 had the narrowest range with a maximum relative error of 6.92%. Phase 1 had a larger range with a maximum of 13.13% and a median of 1.09% (compared to 0.89% for Phase 1 + Phase 2). It is interesting to note that the relative error statistics for Colley w/ Phase 1 and Phase 2, are the same as Phase 1

Table 10
Cost comparison for problem instances s1B4c and s1D3d

	s1B4c		s1D3d	
Algorithm	Cost	Gap	Cost	Gap
Phase 1	1,029,189.63	3.62%	5,727,331.51	1.26%
Phase 1 + Phase 2	1,022,454.08	2.99%	5,725,584.50	1.23%
Colley	998,328.30	0.64%		
Colley w/Phase 1	1,001,279.54	0.93%	5,679,137.37	0.40%
Colley w/Phase 2	991,918.32	0.00%	5,656,265.33	0.00%
Baron				
Baron w/Phase 1	1,006,953.00	1.49%	5,727,331.51	1.26%
Baron w/Phase 2	1,006,953.00	1.49%	5,725,584.50	1.23%
Elhedhli solution	1,130,889.50	12.29%	28,618,786.80	405.97%
Elhedhli solution w/Phase 1	1,166,682.07	14.98%	72,478,314.01	1,181.38%
Elhedhli solution w/Phase 2	1,011,588.38	1.94%	76,700,022.54	1,256.02%
Elhedhli cost	1,130,889.50	12.29%	31,308,434.39	453.52%
Elhedhli cost w/Phase 1	1,166,682.07	14.98%	72,478,314.01	1,181.38%
Elhedhli cost w/Phase 2	1,011,588.38	1.94%	76,700,022.54	1,256.02%
Elhedhli Rr				
Elhedhli Rr w/Phase 1	1,166,682.07	14.98%	72,478,314.01	1,181.38%
Elhedhli Rr w/Phase 2	1,011,588.38	1.94%	76,700,022.54	1,256.02%

and Phase 1 + Phase 2, respectively. The maximum gaps for the other algorithms based on exact solution methodologies were all more than 1,900%.

Table 10 compares costs and relative errors of the best solutions found by each algorithm applied to two representative instances from the sizing dataset. Problem instance s1B4c is an example in which many of the nonoptimal solutions were relatively close in cost to the "optimal" solution. The cost of the best solution found for s1B4c was 991,918.32. The second and third-best solutions had costs of 998,328.30 and 1,001,279.54, respectively, with corresponding relative errors of 0.64% and 0.93%; and there were several nonoptimal solutions with relative errors between 1% and 4%. Problem instance s1D3d is an example in which most of the nonoptimal solutions were quite far from "optimal." Although the second-best solution found for s1D3d had a relative error of only 0.4%, most of the nonoptimal solutions had large relative errors of more than 400%.

10. Conclusions

This paper develops a fast, effective heuristic for the ISP 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 paper 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 problem instances have been made available to share with other

researchers on the SMU Scholar website (Colley, 2021). 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 content delivery networks. The heuristic was also shown to improve the performance of the exact methods by quickly finding high-quality incumbent solutions.

Our computational study shows that both time-limited ISPs and cases where provably optimal solutions are required can benefit significantly from relatively simple heuristics. Continued exploration in this direction is likely to produce further improvements. Investigating the trade-off between solution running time and solution quality using the metaheuristics from the literature is a potential direction for future research. A statistical analysis of the quality of the Phase 1 solutions produced by each of the customer sorting orders described in Section 7 may provide helpful insights about additional options for Phase 1 such as sorting by the standard deviation. Alternatively, such analysis might identify types of problem instances for which the computational effort and resources can be reduced by only using a subset of the sort orders and/or capacity-allocation rules that consistently find the best solutions. The heuristic might be further improved by incorporating additional local search subroutines in Phase 2. One advantage of the heuristic over the exact methods is that it makes it relatively straightforward to change the objective function (e.g., changing from the $M/M/1$ model to the $M/M/s$ model). Thus, planned future work also includes adapting the heuristic to ISP variants with multiple-server queues.

Another potentially fruitful direction for future work is expanding our computational study to include recent and promising exact methodologies for ISP (e.g., Aboolian et al., 2022 and Ahmadi-Javid and Hoseinpour, 2022) that were not previously available to us. Additionally, the combination of the Colley model with the Phase 1 incumbent shows potential in certain use cases. Like the model proposed by Elhedhli (2006), the Colley model provides a mechanism for the exact solution of a nonlinear MIP with linear-programming-based methodology and it may be appealing to practitioners since it can be implemented with a straightforward application of a commercial solver. However, the empirical efficiency of finding provably optimal solutions to ISPs with the Colley model is an open question; further work is needed to determine if the model can be strengthened by tightening the big-M constant in constraint (23) and/or developing valid inequalities for the model.

Acknowledgments

This material is based upon work supported by the National Science Foundation CMMI division under Award No. 2227548.

References

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

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

Aboolian, R., Elhedhli, S., Karimi, M., 2022. An efficient approach for service system design with immobile servers, stochastic demand, congestion, and consumer choice. *Journal of Supply Chain and Operations Management* 20, 1, 1.

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

Ahmadi-Javid, A., Hoseinpour, P., 2022. Convexification of queueing formulas by mixed-integer second-order cone programming: an application to a discrete location problem with congestion. *INFORMS Journal on Computing* 34, 5, 2621–2633.

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

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

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

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

Amiri, A., Pirkul, H., 1997. New formulation and relaxation to solve a concave-cost network flow problem. *Journal of the Operational Research Society* 48, 278–287.

AMPL, 2009. *AMPL Version 10.6.16*. AMPL Optimization LLC.

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

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

Boffey, B., Galvão, R., Espejo, L., 2007. A review of congestion models in the location of facilities with immobile servers. *European Journal of Operational Research* 178, 643–662.

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

Burke, E. K., Kendall, G., 2014. *Search Methodologies*. Springer, New York.

Castillo, I., Ingolfsson, A., Sim, T., 2009. Social optimal location of facilities with fixed servers, stochastic demand, and congestion. *Production and Operations Management* 18, 6, 721–736.

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

Chandra, P., Weisstein, E.W., 2021. Fibonacci number. https://mathworld.wolfram.com/FibonacciNumber.html.

Colley, A.Q., 2021. Customer and capacity assignment for a set of immobile servers using heuristics. https://scholar.smu.edu/engineering_management_research/3/.

Colley, A.Q., 2022. Heuristics for capacity allocation and queue assignment in congested service systems with stochastic customer demand and immobile servers. Ph.D. thesis, Southern Methodist University, Dallas, TX.

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

Derringer, G., Suich, R., 1980. Simultaneous optimization of several response variables. *Journal of Quality Technology* 12, 4, 214–219.

Elhedhli, S., 2006. Service system design with immobile servers, stochastic demand, and congestion. *Manufacturing & Service Operations Management* 8, 1, 92–97.

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

Geem, Z.W., Kim, J.H., Loganathan, G.V., 2001. A new heuristic optimization algorithm: harmony search. *Simulation* 76, 2, 60–68.

Glover, F., Laguna, M., 1997. *Tabu Search*. Springer, New York.

Gurobi Optimization, LLC, 2021. *Gurobi Optimizer Reference Manual*.

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

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

Hoseinpour, P., 2022. Modeling and solving an economies-of-scale service system design problem. *International Transactions in Operational Research* 31, 1975–1993.

Kılınç, M.R., Sahinidis, N.V., 2018. Exploiting integrality in the global optimization of mixed-integer nonlinear programming problems with BARON. *Optimization Methods and Software* 33, 3, 540–562.

Li, B., Golin, M.J., Italiano, G.F., Deng, X., Sohraby, K., 1999. 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*, Vol. 3, pp. 1282–1290.

Marianov, V., Boffey, T.B., Galvão, R.D., 2009. Optimal location of multi-server congestible facilities operating as $M/E_r/m/N$ queues. *Journal of the Operational Research Society* 60, 5, 674–684.

Marianov, V., Serra, D., 2002. Location–allocation of multiple-server service centers with constrained queues or waiting times. *Annals of Operations Research* 111, 1, 35–50.

Mirjalili, S., 2015. The ant lion optimizer. *Advances in Engineering Software* 83, 80–98.

Oracle, 2003. Java 1.6. https://www.oracle.com/java/technologies/javase/webnotes.html.

Pasandideh, S.H.R., Chambari, A., 2010. A new model for location-allocation problem within queuing framework. *Journal of Optimization in Industrial Engineering* 3, 6, 53–61.

Pasandideh, S.H.R., Niaki, S., 2012. Genetic application in a facility location problem with random demand within queuing framework. *Journal of Intelligent Manufacturing* 23, 3, 651–659.

Peng, Y., 2019. Resource Allocation and Task Scheduling Optimization in Cloud-Based Content Delivery Networks with Edge Computing. Ph.D. thesis, Southern Methodist University, Dallas, TX.

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

Rajagopalan, S., Yu, H.L., 2001. Capacity planning with congestion effects. *European Journal of Operational Research* 134, 2, 365–377.

Riccio, L.J., 1984. Management science in New York's Department of Sanitation. *INFORMS Journal on Applied Analytics* 14, 2, 1–13.

Stiermaier, S., 2010. Facility location and allocation problems with stochastic customer demand and immobile servers. Ph.D. thesis, Vienna University of Technology, Vienna, Austria.

Taguchi, G., 1986. Orthogonal arrays and linear graphs. American Supplier Institute, Inc.

Vidyarthi, N., Jayaswal, S., 2014. Efficient solution of a class of location–allocation problems with stochastic demand and congestion. *Computers & Operations Research* 48, 20–30.

Wang, Q., Batta, R., Rump, C.M., 2002a. Algorithms for a facility location problem with stochastic customer demand and immobile servers. *Annals of Operations Research* 111, 17–34.

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