Optimization of Operating Room Usage

Bruna Ribeiro
Master in Network and Information Systems Engineering
Department of Computer Science
Faculty of Sciences of the University of Porto
2024

Optimization of Operating Room Usage

Bruna Ribeiro

Dissertation carried out as part of the Master in Network and Information Systems Engineering

Department of Computer Science

2024

Supervisor
Prof. Dr. João Pedro Pedroso,
Associate Professor,
Faculty of Science of the University of Porto

Abstract

Global population growth has resulted in a notable increase in health expenditures pressing hospitals to reduce costs, highlighting the importance of more effective resource and time management.

Operating rooms are one of the most important units, representing almost a third of the hospital's budget. The efficient use of these rooms not only reduces costs, but also improves the quality of care and overall patient satisfaction. However, scheduling of the operating rooms is a complex problem, due to limited availability of time and space and uncertainty in the duration of operations. Currently, only simple rules are being used to solve this problem, which can be viewed as a variant of the classic Bin Packing problem.

In this work, we implemented more complex heuristics to solve this problem. The three main methods chosen to optimize operating rooms scheduling were: First Fit (FF), Local Search (LS), and Tabu Search (TS). These algorithms were modified to deal with the non-deterministic nature of the duration of operations. They were tested in a set of synthetic data instances, based on Falkenauer's instances for Bin Packing.

The results show that more complex algorithms such as Multi-start Local Search (MLS), Destroy and Repair Local Search (DLS), Destroy and Repair Tabu Seach (DTS) offer better results than simpler methods like Fist Fit. The MLS algorithm had the best performance when taking into account the solution quality and run time.

Keywords: Operation room usage, Bin Packing, Heuristics, Non-deterministic optimization, Healthcare, PERT method, First Fit, Local Search, Tabu Search, Falkenauer instances

Resumo

O aumento da população mundial resultou num aumento significativo nos encargos em saúde, levando hospitais a reduzir custos, o que evidencia a importância de uma gestão eficaz de recursos e de tempo.

Blocos operatórios são uma das unidades mais importantes, representando quase um terço do orçamento de um hospital. O uso eficiente destes blocos não só reduz os custos, mas também aumenta a qualidade do cuidado hospitalar e satisfação em geral do paciente. No entanto, o agendamento de blocos operatórios é um problema complexo, devido às limitações de tempo e espaço e da incerteza na duração das operações. De momento, apenas regras simples estão a ser usadas para resolver este problema, que pode ser visto como uma variante do clássico problema de Empacotamento em Caixas.

Neste trabalho, implementámos heurísticas mais complexas para resolver este problema. Os três métodos principais escolhidos para otimizar o agendamento de blocos operatórios foram: Primeiro Encaixe, Pesquisa Local, e Pesquisa Tabu. Estes algoritmos foram modificados para lidar com a natureza não determinística da duração de operações. Eles foram testados em instâncias artificiais, baseadas nas instâncias do Falkenauer para Empacotamento em Caixas.

Os resultados mostram que algoritmos mais complexos como a Pesquisa Local Multi-inicial, Pesquisa Local Destruir e Reparar, Pesquisa Tabu Destruir e Reparar fornecem melhores resultados que métodos mais simples como o Primeiro Encaixe. O algoritmo MLS teve o melhor desempenho quando considerados tanto a qualidade da solução e o tempo de execução.

Palavras Chave: Uso de blocos operatórios, Empacotamento em Caixas, Heurísticas, Otimização não-determinística, Cuidados de saúde, Método PERT, Primeiro Encaixe, Pesquisa Local, Pesquisa Tabu, Instâncias de Falkenauer

Acknowledgements

I would like to express my gratitude to everyone who has supported me throughout this journey.

Firstly, I would like to thank my supervisor Prof. João Pedro Pedroso and João Pedro Dionísio who were indispensable.

I would like to thank my university colleagues, those who are and those who were. Special thanks to my friends for always supporting me and always being willing to help.

To my parents Oscar and Sofia, my brother Filipe, Brisa, my grandmothers Gracinda and Amélia, who have always been my biggest supporters and who have always believed in me.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

FFD First Fit Decreasing

FF First Fit

BF Best Fit

PERT Program Evaluation and Review Technique

OR Operating Room

ILP Integer Linear Programming

LS Local Search

TS Tabu Search

CDF Cumulative Distribution Function

MFF Multi-stat First Fit

MLS Multi-stat Local Search

MTS Multi-stat Tabu Search

DLS Destroy and Repair Local Search

DTS Destroy and Repair Tabu Search

ICDF Initial CDF

Chapter 1

Introduction

Several approaches have been explored to deal with the complex challenge of scheduling surgeries and optimizing the use of operating rooms. The problem of scheduling surgeries and optimizing the use of operating rooms can be viewed as a Bin Packing problem.

Each surgical procedure can be considered an "item" with a size determined by its duration, while operating rooms are analogous to "bins" with a limited capacity defined by the available time (typically a day or work shift).

In Bin Packing, the goal is to use the fewest number of bins possible or to optimize the space available within the bins. In surgery scheduling, the goal is to optimize the use of operating rooms to maximize efficiency, minimize downtime, and avoid waste.

The Bin Packing problem is NP-hard, meaning that it is as computationally challenging as the hardest problems in the class NP, for which there is no known algorithm capable of solving them in polynomial time [11]. Unless P=NP, such algorithm does not exist. With the current scientific knowledge, NP-hard problems have a tendency to be intractable, motivating a need to employ heuristic approaches to get good solutions, and relax optimality requirements.

1.1 Deterministic Problem

In the context of operation scheduling, this can be seen as a deterministic problem as long as the duration of each operation is known without ambiguity [16].

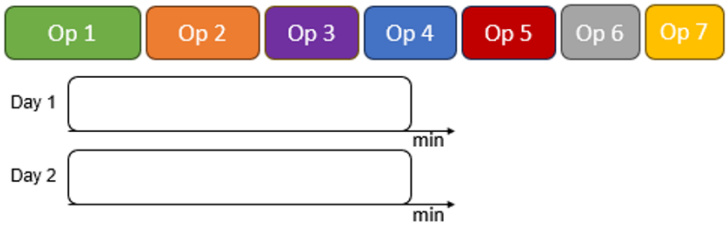

Figure 1.1: Slots without operations

1.1.1 First Fit

First Fit (FF) algorithm is a classic heuristic for solving the Bin Packing problem, where the goal is to minimize the number of bins used to accommodate all given items. The FF approach tries to place each item in the first available bin that has enough space. If there is not enough space in any of the existing bins, a new bin is created [15].

Figure 1.2: FF solution example

The First Fit Decreasing (FFD) approach is similar to FF, but it starts by sorting the items in decreasing order of size. In some instances, this permutation can lead to better solutions, as can be seen in Figure 1.3.

Figure 1.3: FFD solution example

1.1.2 Best Fit

Best Fit (BF) is a simple algorithm that tries to place the current item in the fullest possible bin where it still fits. If there is no such bin, a new bin will be opened.

Although several algorithms fail to significantly improve when the initial order of the items is random, Best Fit proves to be a strong candidate, with significant improvements over previously established performance thresholds [2]. Additionally, it introduces new techniques for analyzing Bin Packing algorithms in random order contexts.

In Figure 1.4, even though Op 5 could fit in the first bin, it was placed in the second one because it becomes fuller.

Figure 1.4: Example of operation placement in BF

These algorithms would provide suitable solutions if we were dealing with a deterministic problem, but our goal is to find a good solution for the problem involving non-deterministic coefficients. Consequently, FF, FFD and BF, were not used in this dissertation.

1.2 Non-Deterministic Problem

In real life, operations can end before or after the expected time, as the duration of each operation is not fixed and can vary considerably [6]. Consequently, the duration of surgeries is considered a random variable, which makes our problem non-deterministic. The non-deterministic nature of surgery means that even with an initial time estimate, the exact duration remains uncertain and can fluctuate, significantly impacting hospital planning.

Figure 1.5: Real case of the duration of operations

1.2.1 PERT

PERT (Program Evaluation and Review Technique) is a method used to manage uncertainty in activity durations. It helps estimate the mean and variance of these durations, providing a more accurate assessment of the time required to complete an activity.

PERT uses a beta distribution for representing activity duration and estimates the mean and variance of the activity's duration using the following parameters [18]:

- o - Optimistic completion time: estimate of the activity's duration under the most favorable conditions;
- m - Most likely completion time: most likely value for the activity's duration;
- p - Pessimistic completion time: estimate of the activity's duration under the least favorable conditions.

Using these three values, the mean and variance are calculated for each item (i.e., for each surgery). This data helps create an accurate profile of the expected duration of surgeries. With these parameters, the mean and variance of a random variable with beta distribution are:

$$\mu = \frac{o + 4m + p}{6} \tag{1.1}$$

$$\sigma^2 = \frac{(p - o)^2}{36} \tag{1.2}$$

The mean represents the expected value of the execution time of an operation. It is useful for predicting how long, on average, an operation will take. Variance measures the spread of execution times around the average. A high variance indicates that execution times can vary significantly from the average, introducing uncertainty and risks of exceeding the estimated time.

Using PERT allows for a more robust and efficient management of operating room schedules, taking into account the uncertainty of operating times. This is crucial to optimize the use of hospital resources reliably.

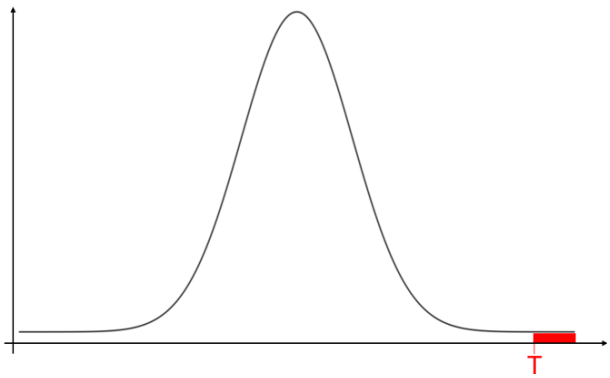

Figure 1.6: Real time duration of operations

As mentioned previously, we assume that the duration of operations is not deterministic. Therefore the total amount of time used in an operation room in a given day can be better

described by a probability distribution that captures both the mean and variance of the execution time. The usual assumption in PERT is that the total duration of a series of activities can be approximated by a normal distribution where the mean is the sum of the means of all activities, and the variance is the sum of their variances.

Considering the mean and variance of all operations, it is necessary to ensure that the total operation time does not exceed the time limit with a certain confidence. This can be done by ensuring that the random variable representing total duration on a day/total length on a bin is less than that limit with a probability at least as large as the desired confidence.

1.3 Operation Room Scheduling

Each surgical procedure can be conceptualized as an item with size determined by its duration, while operating rooms represent bins with limited capacity in terms of available time. In the absence of real data - difficult to obtain in healthcare context - we will use randomly generated, fictitious data with a known optimal solution, where all bins are fully occupied. This synthetic data serves as a standard for evaluating the performance of the algorithms, prioritizing those that generate results closest to the optimal solution, measured by the number of bins required for a complete schedule.

In the deterministic context, so-called "triplet instances" generated by Falkenauer [10] are considered very challenging due to the specific characteristics of their construction: they are generated in such a way that one item is larger than $\frac{1}{3}$ of the bin size and the two other items completely fill the bin; each item's size is drawn from the range (0.25, 0.50). "Triplet instances" generated this way, tend to have a distribution of item sizes that makes it difficult to fill bins efficiently, as it is difficult to combine items without wasting space in the bins. Normally, there is only one possible solutions.

1.4 Thesis Outline

This thesis is organized as follows. Chapter 1 introduces the problem of optimizing operating room utilization, emphasizing its significance in hospital management and the challenges inherent in surgery scheduling. Chapter 2 reviews existing state-of-the-art approaches and algorithms commonly employed to address this problem, encompassing both deterministic and non-deterministic scenarios. Chapter 3 details the conception and development of the algorithms utilized in this research, while Chapter 4 explains the experimental setup, data generation, and testing methodology. Chapter 5 presents the results and analysis of the conducted tests, comparing the efficiency of different algorithms. Finally, Chapter 6 offers conclusions and recommendations for future work.

Chapter 2

State-of-the-Art

This chapter focuses on methodologies used to improve Operating Room (OR) scheduling. It explores how the simple scheduling practices currently applied in hospitals often limit resource efficiency. We look at uncertainty as well as how advanced optimization techniques provide more effective management solutions.

2.1 Current Practice in Hospital Management

Modern hospitals face a constant challenge: balancing cost reduction with maintaining high-quality care and team satisfaction.

"Operating rooms are the most important source of income and expense for hospitals. Therefore, the hospital management focuses on the effectiveness of schedules and plans" [12].

The current OR scheduling practices are generally based on "simple rules" that hinder efficiency and optimization of available resources [12], like:

- Block Scheduling: An approach in which certain periods of time are reserved for specific surgeries or certain medical specialties. This helps organize room usage, but may not maximize efficiency, as the time blocks might not be fully utilized;
- Open Scheduling: In this strategy, there is no allocation of fixed blocks of time for surgeries or specialties. Instead, surgeries are scheduled as they arise, based on availability. While flexible, this may result in longer wait times for patients or underutilization of resources;
- First-Come, First-Served: This method allocates surgeries on a first-come, first-served basis, which can be helpful in avoiding conflicts and simplifying the process. However, it may not be the best way to optimize OR usage, especially in high-demand hospitals.

Integer Linear Programming (ILP) is a mathematical optimization technique that solves optimization problems where the decision variables are integers. ILP was used to solve the

deterministic problem of surgery scheduling [1], where the operation durations are known. The objective is to minimize patient waiting time and the number of delayed surgeries, allocating them efficiently to available surgical centers. The model considered the limited resources of each OR (such as daily time in the OR) and optimized patient allocation over several weeks.

This approach proved effective because, in a scenario with deterministic surgical times, the ILP model generated stable and efficient solutions. This ensured that there were practically no changes to the planning. However, when the uncertainty of surgery duration was incorporated into the non-deterministic problem, the ILP model became inadequate. In the ILP, the sum of durations of operations cannot exceed the capacity. When the actual times of the surgeries are different from those predicted, capacity is exceeded, resulting in intensible models. The ILP has no mechanisms to dynamically adjust these constraints as the actual surgeries exceed or are below the expected time. This resulted in cancellations and delays, as the initial plan based on fixed durations was unable to predict the actual durations of operations.

Operations research offers valuable tools for optimizing operation room scheduling, paving the way for more efficient management. An existing study explores goal scheduling [3], which consists of ideally planning OR for one year and trying to reduce the maximum waiting time to six months. Subsequently, the authors attempt to reduce this to four months and determine the impact that this reduction would have on the need for extra resources in the hospital. The study concluded that reducing the waiting time from six to four months had a large impact on hospital costs.

After surgery in the OR, patients remain in a subsequent ward for a period of time, which causes load interdependence between the previous OR and the nursing unit in the ward. Some studies have used multi-objective optimization [7], where several departments are scheduled simultaneously, in order to increase efficiency. In the context of this dissertation, only one objective will be considered, though it could be extended to a multi-objective setting later.

2.1.1 New Frontiers in Scheduling Optimization

In addition to traditional optimization models, other approaches are being explored that incorporate complex and interrelated factors into the scheduling process. Constraint programming and goal programming have been implemented to incorporate surgical team availability into scheduling, considering the skills and availability of the professionals involved [13]. This approach contributes to optimizing surgery time and minimizing delays, ensuring that human resources are used efficiently.

Optimizing OR management is a dynamic and constantly evolving field. The combination of traditional operations research approaches with innovative techniques and emerging technological tools, such as artificial intelligence and machine learning [4], offers enormous potential to improve efficiency and quality in healthcare delivery in the future. By overcoming the challenges of uncertainty and the complex dynamics of an hospital environment, we can build a future where

OR optimization contributes to improved medical care and patient and professional satisfaction.

2.1.2 Motivation

Simple rules-based OR scheduling practices fail to adequately optimize resource use, resulting in inefficiencies and underutilization of rooms. Although traditional techniques such as ILP are used to solve the scheduling problem in deterministic scenarios, these approaches cannot deal with common uncertainties in real practice, such as the uncertain duration of operations.

Most traditional approaches do not consider uncertainty in surgery times, such as delays or unpredictable events. This has a negative impact on resource allocation and overall efficiency, leading to delays and wasted OR time. This problem makes deterministic models unsuitable for more complex and realistic hospital scenarios.

This work aimss to solve these problems by implementing suitable advanced algorithms and iterative methods. These methods are designed to optimize resource usage even in non-deterministic contexts, taking into account probabilistic distributions for the duration of the operation. This will allow for more contained dynamic adjustments in real time, resulting in a more optimized and reliable scheduling.

2.2 Underlying Techniques and Algorithms

2.2.1 Deterministic Optimization of Operating Room Usage

In mathematical terms, it can be defined as a problem in which all the data are deterministic, that is, constants have fixed and known values. As a result, the problem has a unique and repeatable solution for a given set of initial conditions, without the presence of random variables or stochastic components that could introduce uncertainty into the final result [16].

Figure 2.1 bellow illustrates that the probability of the operations being completed in less than the estimated duration is zero, while the probability is one for any time equal or larger than the estimated duration.

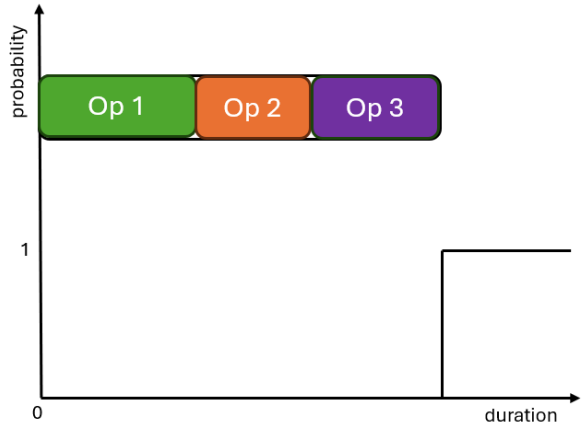

Figure 2.1: Probability over time in a deterministic problem

In the context of ORU, the objective is to maximize the use of available resources, minimize wasted time, and maximize the number of surgeries performed. If the total time used in an OR would simply be the sum of known durations of the scheduled operations, it would be a deterministic problem for which traditional Bin Packing approaches would work well.

2.2.2 Uncertainty

OR optimization is rarely a deterministic problem in practice, since uncertainty from multiple sources has a significant impact on feasibility, with several variables that can affect the time of surgeries, like [14]:

- Workplace-related events: random operation arrival, non-deterministic processing time;
- Operation-related events: operation delays.

Uncertainty in the duration of operations can be both internal to the process (human errors) and external (high number of operations).

Uncertainty in the duration of operations can affect:

- Planning: makes scheduling more challenging, as it is necessary to consider optimistic and pessimistic scenarios to estimate the deadlines that need to be met;
- Costs: delays and reworks caused by uncertainty can generate additional costs, such as overtime, carry additional.

Considering uncertainty allows for more flexible scheduling, enabling adjustments in real time to accommodate unexpected events, thereby minimizing disruptions and optimizing resource utilization.

2.2.3 Non-Deterministic Optimization of Operating Room Usage

A non-deterministic problem is one in which some or all data are random variables, meaning their values are not fixed and can fluctuate based on different circumstances.

In mathematical terms, it can be defined as a problem in which the constants of the problem can be random or uncertain, that is, instead of having fixed and known values, they are modeled as variables X_i with associated probability distributions. For each $i = 1, 2, \ldots, n$, the variable X_i does not have a constant value, but rather a distribution function that describes the probability of X_i assuming a given value. As a result, the solution to the problem is not unique and repeatable, but characterized by a distribution of possible outcomes, reflecting the uncertainty and variability inherent to the variables involved [16].

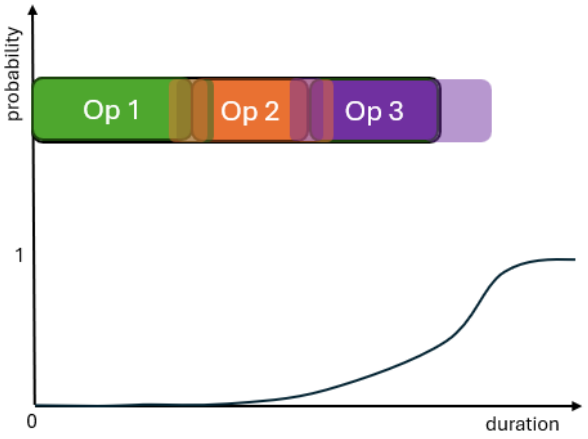

Figure 2.2: Probability over time in a non-deterministic problem

Figure 2.2 shows three operations that unfold over time and illustrates how the probability of the operations being completed gradually approaches one as their allotted duration increases.

Traditional optimization methods such as integer programming can become inefficient or impractical due to their computational complexity. To solve non-deterministic problems, it is common to use alternative approaches that prioritize finding satisfactory solutions in a reasonable time, even if they may not be guaranteed to be optimal. These methods can be categorized into two main groups: [9]:

- Heuristics: these are practical and fast methods for solving optimization or search problems, useful when exact methods are not an option due to time constraints or complexity. Heuristics provide approximate solutions by applying simple rules to efficiently explore the solution space;
- Metaheuristics: these, on the other hand, are high-level search strategies that guide heuristics to find solutions closer to the global optimum. They explore the search space more efficiently, avoiding local optima and increasing the likelihood of finding a globally optimal solution.

Metaheuristics, by combining global and local exploration, can find robust solutions within acceptable timeframes, even under rapidly changing conditions. Consequently, metaheuristics are more effective for optimizing OR schedules.

2.2.3.1 Local Search

Local Search (LS) is a heuristic optimization technique that aims to find satisfactory solutions to combinatorial optimization problems, exploring the space of possible solutions through small iterative modifications to an initial solution.

LS is described as a technique that uses a neighborhood function to explore various potential solutions near the current solution, aiming to enhance its current solution until no further improvements can be achieved, resulting in a local optimum, as presented in Vaessens et al. [19].

LS has been successfully applied in various contexts, as demonstrated by Vanneschi et al. in the second chapter of their book [20].

2.2.3.2 Tabu Search

Tabu Search (TS) is an optimization algorithm that iteratively explores the solution space, beginning with an initial solution, which may or may not be locally optimal. The algorithm moves through the solution space, selecting the best neighboring solution that is not tabu. A solution is tabu if it is in the "tabu list", which stores attributes of recently visited solutions or moves, preventing these solutions from being revisited to avoid cycles and promote exploration of new areas of the search space [5]. While employing a tabu list might temporarily lead to a worse solution, it enables the algorithm to overcome local optima, with the possibility of finding a globally optimal solution in the long term.

TS can be applied to many optimization problems, such as e.g. to the job-shop scheduling problem, as shown by Dell'Amico et al. in [8]. This problem involves allocating a set of jobs, each consisting of several tasks, on a set of machines, with the objective of minimizing the total time required to complete the jobs, while adhering to sequence and capacity constraints. This situation is somewhat comparable to our problem.

2.3 Data Creation

Due to the difficulty of obtaining real data in the hospital context, randomly generated fictitious data was used.

2.3.1 Falkenauer

The Bin Packing problem is a classic combinatorial optimization challenge that seeks to pack items of varying sizes into the minimum number of slots. A perfect solution is achieved when all items are packed without any unused space within the bins.

Emanuel Falkenauer introduced in his article [10] a set of particularly demanding Bin Packing instances. These instances are meticulously designed to possess an optimal solution where all items fit perfectly into the slots, without any empty space. This characteristic renders them invaluable for assessing the efficacy of Bin Packing algorithms, as the benchmark of a perfect solution is explicitly defined. In Algorithm 6 will describe an adaptation of this instance generation method to our non-deterministic context.

Chapter 3

Algorithm Conception and Development

This chapter is dedicated to the methodologies used to develop algorithms for a better operating room scheduling.

3.1 Used Data

To represent operations in a structured and efficient manner within the algorithms, we defined class Operation in Python. This class stores the key attributes of each operation, facilitating data manipulation. The Operation class has three attributes:

- id: unique integer identifier for each operation;
- avg: average of estimated values, calculated using the Equation 1.1;
- var: variance of estimated values, calculated using the Equation 1.2.

The instances are converted to a data structure that algorithms can more easily handle, simplifying the subsequent calculation of mean and variance values and add the data to the Operation class.

Addicionally for completely defining an instance, the size of the bin/slot must be specified (denoted as capacity below, as well as the desired confidence level for not exceeding it - denoted as min_conf).

3.1.1 PERT

Providing non-deterministic data is typically difficult in practice. In order to facilitate this process, we use the PERT method. Hence, for the complete definition of an instance, the user

must provide the size of the bin (i.e., the duration of a period of operating room planning); and the minimum, most likely and maximum size of each item (i.e., of the duration of each operation to schedule).

3.1.2 Minimum Confidence Level

The minimum confidence (min_conf) is a user-defined threshold value to determine whether a solution is acceptable or not. This value is essential to ensure that the algorithm does not accept solutions that fall below a specified probability of successful completion on time.

The Cumulative Distribution Function (CDF) of a query variable X is a function that maps each value x to the probability that X is less than or equal to x. Mathematically, a CDF is defined as:

$$F_X(x) = P(X \leq x) \tag{3.1}$$

where:

- $F_X(x)$ represents the CDF of the random variable X.
- $P(X \leq x)$ is the probability that X takes a value less than or equal to x.

It can be used to assess the probability of the scheduling being completed on time.

When determining whether to modify a given solution, in a deterministic scenario, we heuristically select the item that results in a more filled bin without exceeding its capacity. The same logic can be applied to this non-deterministic configuration and leads us to the following conditions, which give us equivalent guarantees:

- A condition that guarantees that the newly discovered solution has a CDF that meets a specified minimum confidence level (min_conf);
- A condition that ensures that the newly found solution has a better CDF than the current solution. A lower CDF value indicates a fuller bin.

This combination of criteria enables algorithms to replace inferior solutions and consistently strive for a more advantageous solution. This will be used in all algorithms.

3.1.3 Choosing The Best Solution

The main objective is to minimize the number of bins, i.e., number of slots required to schedule all operations in operating rooms. However, in addition to simply minimizing the number of

slots, there is another beneficial criterion: selecting a solution in which, when sorting the slots in ascending order of CDF, the last slot is as empty as possible.

In real-world scenarios, it may be necessary to modify the schedule, for example, by relocating operations to other days. An empty last slot facilitates this relocation process, as there is more space to accommodate additional operations or adjust existing ones on other days without causing significant disruptions. The metric by which we compare two different solutions is than: a better solution has either a lower number of bins, or the same number and a higher minimum CDF than a worse solution.

In summary, this approach not only minimizes resource utilization but also increases flexibility for future adjustments, which is highly valuable in dynamic environments such as OR.

3.2 First Fit

First Fit (FF) algorithm is a simple heuristic for assigning items to bins, i.e., operations to slots. This algorithm iterates over a list of items, and for each item it tries to fit it into the first bin that has enough space. If it does not fit in any existing bin, a new one will be created. The algorithm continues until all items are assigned to bins [15]. To see if an item can fit in a bin, one needs to calculate its CDF.

Equation 3.2 calculates the standard score or z_value associated to items in a bin. The z value is a statistical metric that quantifies how many standard deviations a specific value (capacity) is from the average. It is useful for understanding the relationship between available capacity and expected value (average), taking into account data variability (variance).

$$z = \frac{capacity - average}{\sqrt{variance}} \tag{3.2}$$

In our setting, capacity is the size of each bin and average and variance are calculated taking into account the item placed there, assuming that the aggregate completion is well represented by a normal distribution. In such a situation, we have:

$$average = \sum_{i \in B} x_i \tag{3.3}$$

$$variance = \sum_{i \in B} \sigma_i^2 \tag{3.4}$$

where B is the set of items in the bin, x_i is the average for the random variable corresponding to the size of item i and σ_i is its standard derivation.

Our Fist Fit implementation 1 begins by initializing an empty set B, which will keep the solution. It iterates over each item (i) that is in remain,i.e., that has not yet been allocated.

Than, for each item, it calculates the mean (using Equation 3.3) and variance (using Equation 3.4) that would be obtained if this item were added to B. Based on these values, calculate the z_value using Equation 3.2 and the cdf using Equation 3.1 and the z_value. If the calculated cdf satisfies the minimum confidence level (min_conf), the item is added to B. That process is repeated for all items in remain. At the end, the algorithm returns B, which contains indices in data for all items to place in the bin.

Algorithm 1: First Fit

Data: data, capacity, remain, min_conf

```
B ← ∅
foreach i ∈ remain do
    mean ← new mean if the i is added to B, Eq. 3.3
    variance ← new variance if the i is added to B, Eq. 3.4
    Calculate z_value with Eq. 3.2
    cdf ← norm.cdf(z_value) Eq. 3.1
    if cdf satisfy the min_conf then
        B ← B ∪ {i}
return B
```

3.3 Local Search

Algorithm 2 is a Local Search implementation aimed at improving the initial solution represented by B through exchanges between items in B and items in remain.

It starts by checking for exchanges between B and remain that can improve the solution. For each item i in B and for each j in remain, it verifies if swapping i for j does not exceed the maximum size of B, with a confidence level of min_conf. If the swap occurs and the confidence level of B decreases: i is removed from B and added to remain and j is added to B and removed from remain. This iterative process continues until no further beneficial exchanges can be identified. In the end, it returns B (indices of the items in the bin) and update value ofremain.

Notice that Local Search operates in a individual bin (slot). Having a very well packed bin is usually good for the overall solution, but that may not always be the case. Actually, packing too tighthy the initial bins (when many items remain are available for local improvements) may even be detrimental to the overall solution, as we will see in Chapter 4.

Algorithm 2: Local Search

```
   Data: data, capacity, B, remain, min_conf
 1 while there are improving exchanges between B and remain do
 2 |   foreach i ∈ B do
 3 |   |   foreach j ∈ remain do
 4 |   |   |   if exchanging i with j does not exceed the capacity of B, with confidence
   |   |   |    min_conf then
 5 |   |   |   |   if confidence for B is lower after the exchange then
 6 |   |   |   |   |   B ← B ∪ {j} \ {i}
 7 |   |   |   |   |   remain ← remain ∪ {i} \ {j}
 8 return B, remain;
```

3.4 Tabu Search

Tabu Search is an optimization method that combines local search with a memory structure to prevent revisiting previously explored solutions. The algorithm begins with an initial solution (B) and iteratively attempts to improve it by swapping elements between B and the remain set of unallocated operations. Tabu memory is employed to prevent the algorithm from becoming trapped in cycles of repeating the same solutions.

We represent a solution to our problem as a list $s = [s_1, s_2, \ldots, s_n]$ where each s_i is a list of indices os items in data that will be put in bin i. Hence, the number of bins in s is given by the number n such lists.

Tabu Serch provided in Algorithm 3 iterates N times, where N is a parameter. Within each iteration, it checks if there are improving exchanges between B and remain. For each item i in B and for each j in remain, it verifies if j is not tabu and if swapping i for j does not exceed the capacity of B, with a confidence level of min_conf. If the swap occurs and the confidence level of B decreases: i is removed from B and added to remain and j is added to B and removed from remain. After the exchange, it checks if B is better than any previously found solution. If so, best will be B. When no more improving exchanges can be found between b and remain, a non-improveming move is made: i is removed from B and added to remain and j is added to B and removed from remain, where i an j correspond to the best (thought not improving) exchanges. This exchange might not lead to an immediate improvement but could enable better solutions in the future. Item j is considered tabu, and hence will not be used in exchanges, for tabu_len iterations. Finally, the algorithm returns the best solution found (best).

Algorithm 3: Tabu Search

Data: data, capacity, B, remain, min_conf, N, tabu_len

```
1  for i ← 1 to N do
2      while there are improving exchanges between B and remain do
3          foreach i ∈ B do
4              foreach j ∈ remain do
5                  if j is not tabu and exchanging i with j does not excced the capacity of B,
                    with confidence min_conf then
6                      if confidence for B is lower after the exchange then
7                          B ← (B ∪ {j}) \ {i}
8                          remain ← (remain ∪ {i}) \ {j}
9                          if best is not initialized or B is better than best then
10                             best ← B
11     Select i and j as the best non-improving move
12     B ← (B ∪ {j}) \ {i}
13     remain ← (remain ∪ {i}) \ {j}
14     Make j tabu for tabu_len iterations
15 return best
```

3.5 Multi-start of the Algorithms

Heuristics may be improved by simple repetition, as long as there is some randomness in the process and the best solution observed is retained.

Algorithm 4 iteratively applies an algorithm multiple times to find a better solution. In each iteration, the sequence of items as available in the instance data is randomly permutated, and a solution is constructed using the FF algorithm (Algorithm 1). The initial order of items and consequently the solution generated by FF can lead to a better or worse final result.

The algorithm makes N attempts to build a solution and improve it. For each iteration, it initializes an empty variable solution and randomly shuffles the items in remain (unallocated items). While remain is not empty, FF attemps to allocate items from remain into B and potentially using LS (Algorithm 2) or TS (Algorithm 3) to improve B. Afterward, B is added to the solution. It checks if solution is better than any previously solution found. If so, solution replaces best, which is returned at the end.

Algorithm 4: Multi-start

```
Data: Data, N
for i ← 1 to N do
    solution ← [ ]
    remain ← randomized set of indices in data
    while remain not empty do
        B ← FF(remain)
        B ← Improvement(bin, remain)        // It can be nothing, LS or TS
        solution ← solution + [B]
    if best is not initialized or solution is better than best then
        best ← solution
return best
```

Multi-start Local Search is a technique that aims to find a refined solution through multiple executions of LS (Algorithm 2), with different random initial order of data. The items are allocated using FF (Algorithm 1) and the solution is improved using LS.

Multi-start Tabu Search is a similar technique where TS (Algorithm 3), is used instead of LS.

Multi-start algorithms are particularly significant because, through multiple iterations with random permutations of the data, they explore diverse possible sequences of operations. This allows the algorithm to avoid getting stuck on suboptimal solutions, increasing the likelihood of finding a better solution, as we can see in Figure 3.1.

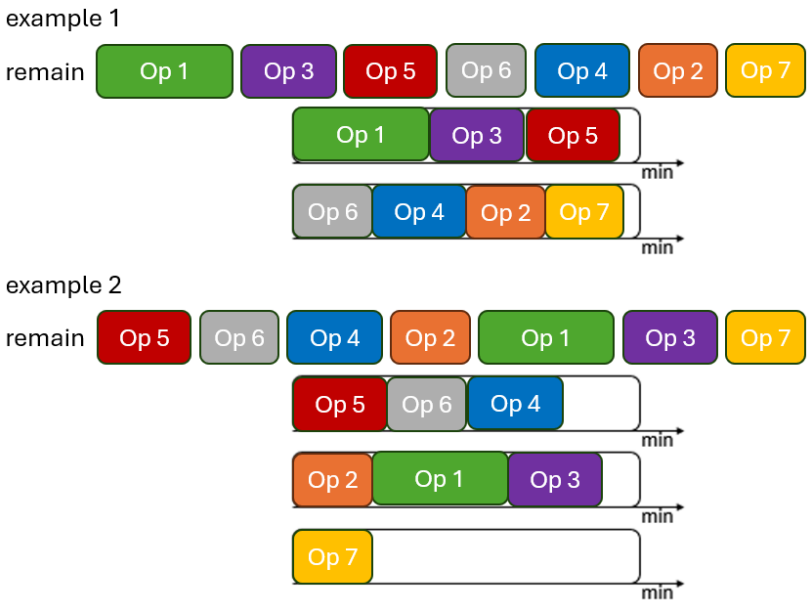

Figure 3.1: Multi-start With Different Initial Data

3.6 Destroy-and-repair

Algorithm 5 proposes a method for Destroy-and-repair. It employs a solution improvement strategy using an algorithm as the primary mechanism for exploring diverse operation allocation configurations within bins/slots.

The implementation proposed in Algorithm 5 performs N iterations, each aiming to construct and refine a solution. In every iteration, the last bin in solution is removed and its elements are added to remain. Then, for each bin B in the solution, a random fraction frac of the items is removed and added to remain. Next, for each remaining bin B, FF (Algorithm 1) tries to reallocate some items in remain. After reallocation, it attempts to improve B using LS (Algorithm 2) or TS (Algorithm 3). Finally it checks if the solution is better than any previously found solution. If so, the solution replaces best, which is returned at the end.

Algorithm 5: Destroy-and-repair

```
Data: data, solution, N, frac
for i ← 1 to N do
    remain ← final bin from solution
    frac ← Uniform(0, 1)
    for B ← all other bins in solution do
        Randomly remove frac% items from B
        Add these items to remain
    for B ← all other bins in solution do
        Fill B with FF(remain)
        B ← Destroy-and-repair(B, remain)        // It can be LS or TS
    if best is not initialized or solution is better than best then
        best ← solution
return best
```

Note that filling a bin to its maximum capacity may initially seem like a good strategy, as it maximizes space utilization in that bin. However, this approach may not lead to the best overall solution. Packing too many items into a bin very efficiently can prevent subsequent bins from being filled optimally. This is problematic when an item is placed in a bin early in the allocation process but could later fit more suitably in another bin. Therefore, balancing the desire to fill a bin well with the need to maintain flexibility for future allocations is crucial for ensuring a better final solution. This is why Destroy-and-repair (Algorithm 5) is useful, as it removes an already allocated item and attempts to allocate it to a different bin where it might fit better.

Chapter 4

Results and Analysis

This chapter presents and analyzes the results of tests carried out to evaluate the efficiency of different operation allocation algorithms.

4.1 Data Creation

As mentioned in section 2, for creating data for the algorithm, we developed a non-deterministic version of the "triplet instances" proposed by Falkenauer in [10].

Algorithm 6 generates instances using Falkenauer triplets, where three operations must fill a bin to its capacity with the specified confidence level. The value of bin_size indicating the size of the bins, is written in the instance file. The random number generator is initialized with seed, ensuring reproducibility of the experiments as subsequent runs with the same seed produce the same results. For each item triplet, mean is randomly chosen of a uniform distribution between a (1/3 bin_size) and b (1/2 bin_size); our aim is that, with high probability, three items can be packed into a slot without wasted space.

Figure 4.1: Slot without wasting space given a certain confidence level

Recall that, as in done PERT, the beta distribution has three parameters: the most likely, optimistic and pessimistic durations. These will be referred to as mi, oi and pi.

Next, m1 will be the rounded value of the mean. Then o1 and p1 are determined using the PERT_stdev function which calculates a and b such that with the desired confidence level, 3 similar items (in case of parameter a) or 2 items (in case of parameter b) can be can be placed with the target_cdf then we fit the parameters of the beta have a standard derivation of mean/10.

The same process is performed to calculate o2, m2, and p2, but now considering the remaining space in the bin. The PERT_mean function computes o3, m3 and p3 as the largest values of the mean, so that the standard derivation is approximately m3/10 and the bin size is not exceeded, for the given confidence level.

The parameters (o, m and p) of the three items are written in the instance file.

Algorithm 6: Generate Triplet Instances

Data: seed, bin_size, n_items, target_cdf

1 Write bin_size in the instance file;
2 Inicialize random generate with seed;
3 **for** $i \leftarrow 1$ **to** *n_items*/3 **do**
4 mean ← Uniform(a, b); *// Falkenauer uses a=1/3 bin_size, b=1/2 bin_size*;
5 m1 ← round(mean); *// o1, m1, p1 are parameters for the beta distribution*;
6 o1, p1 ← PERT_stddev(mean/10, targer_cdf) *// find o1 and p1 such that the standard length equals mean/10*;
7 Calculate o2, m2, p2 in the same way, using the remaining space left in the bin;
8 o3, m3, p3 ← PERT_mean(bin_size, target_cdf, o1, m1, p1, o2, m2, p2) *// completely fill bin for desired CDF*;
9 Write beta parameters for each of the items in the instance file
10 **return**;

Data is created for various minimum confidence levels. Each file starts with a capacity of the bins and contains rows of three values: most likely, optimistic and pessimistic parameters of each item, for parameterizing the beta distribution.

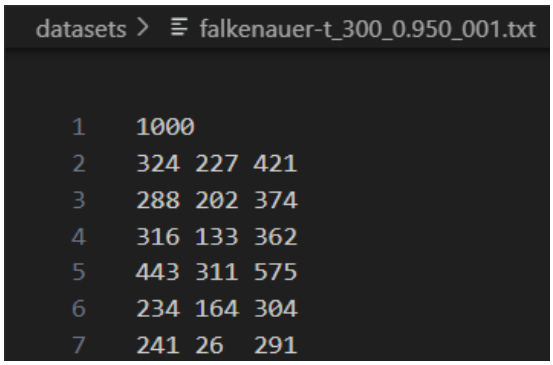

Figure 4.2: Example of triplet instance

4.2 Results Obtained

The main objective in this section is to evaluate the effectiveness of the developed algorithms in packing all items in as few bins as possible, as mentioned in Chapter 1. This can be interpreted as allocating operations in operating rooms, with the focus on minimizing the number of slots

used, while also taking into account the required CPU time. For each combination of size and confidence level, 100 instances were created. The instance sizes are:

- 60 operations;
- 300 operations.

The confidence levels analyzed are:

- 0.950;
- 0.990;
- 0.999.

Each of these instances was tested with the following four algorithms:

- 100000 executions in Multi-start First Fit (MFF);
- 1000 executions in Multi-start Local Search (MLS);
- 1 execution of Destroy and Repair Local Search (DLS);
- 1 execution of Destroy and Repair Tabu Search (DTS);

For an instance generated with a confidence level of 0.950, the algorithms can be configured to operate at three different confidence levels: 0.950, 0.990, and 0.999. This way, we have a series of tests where the instance, created with a certain level of confidence, is evaluated by algorithms with three different levels of required confidence.

In summary, for each combination of size, instance confidence level and solution confidence level, the solutions were grouped and the average the average of our 100 corresponding instances was taken.

These experiments were conducted on a computer running MacOS v13.6.7 6-core Intel Core i7-8700B processors with 64 GB of RAM running at 3.20 GHz from 2018. All programs were implemented in Python (version 3.10) with Python implementation v7.3.15. The code is available in this github page.

4.2.1 Instance Confidence Level Equals Solution Confidence Level

For Falkenauer instances, we know in advance that the optimal number of slots for instances of size 60 is 20 (60/3), and for instances of size 300, the optimal number of slots is 100 (300/3). Hence, when the confidence level of the instance coincides with the confidence level of the algorithmic solution (for example, an instance generated with confidence 0.950 for a solution

with confidence level 0.950), we can evaluate how close the obtained solution is to the optimal value. In such cases, we can count how many times each algorithm achieves the optimal value.

In the following tables, Size refers to instance size, ICDF represents the initial CDF value of the instance, the CDF refers to the CDF value used during the solution run. The tested algorithms were MFF, MLS, DLS, and DTS.

Table 4.1: Number of times reaching the optimal value

Size	**ICDF**	**CDF**	**MFF**	**MLS**	**DLS**	**DTS**
60	0.950	0.950	0	0	0	0
60	0.990	0.990	0	0	0	0
60	0.999	0.999	0	3	1	4
300	0.950	0.950	0	0	0	0
300	0.990	0.990	0	0	0	0
300	0.999	0.999	0	0	0	0

In Table 4.1, we can see how many times each algorithm got the optimal value. Analyzing the table, we can see that the MFF never got the optimum value. However, the other algorithms reached the optimal value in instances of size 60 with confidence level 0.999. MLS obtained the optimal value 3 times, DLS obtained it 1 time, and DTS obtained it 4 times. So DTS was the algorithm that got better results.

Next, for each combination of instance size, instance confidence level, and solution confidence level, the following data was collected:

- Average number of slots;
- Average CPU time

This data is plotted on the graphs, where:

- The X Axis represents the CPU time required to execute the algorithms, in seconds. CPU time reflects the speed of the algorithms, that is, how quickly they converge to a solution.
- The Y Axis represents the number of slots used to allocate all operations. Lower values indicate better solutions.

These chart are particularly useful for identifying which algorithms converge most quickly to a high-quality solution and which are most effective in terms of the quality of the solution obtained.

The choice of the logarithmic scale in the X axis was motivated by the need to capture the behavior of different algorithms over time for a particular instance. Given that their execution

time can vary (some may take milliseconds and some may take seconds or more), a linear scale would not be able to represent that variation well. The various lines for each algorithm represent the different intermediate solutions found over time. Each fine line is an execution. The reason why there is no single line per algorithm is that iterative algorithms tend to explore and improve solutions differently. A thick line represents the average solutions, for the 100 instances of each type.

The figures for the size 60 instances show the execution time differences and the quality of the solution of the four algorithms. They can be found below.

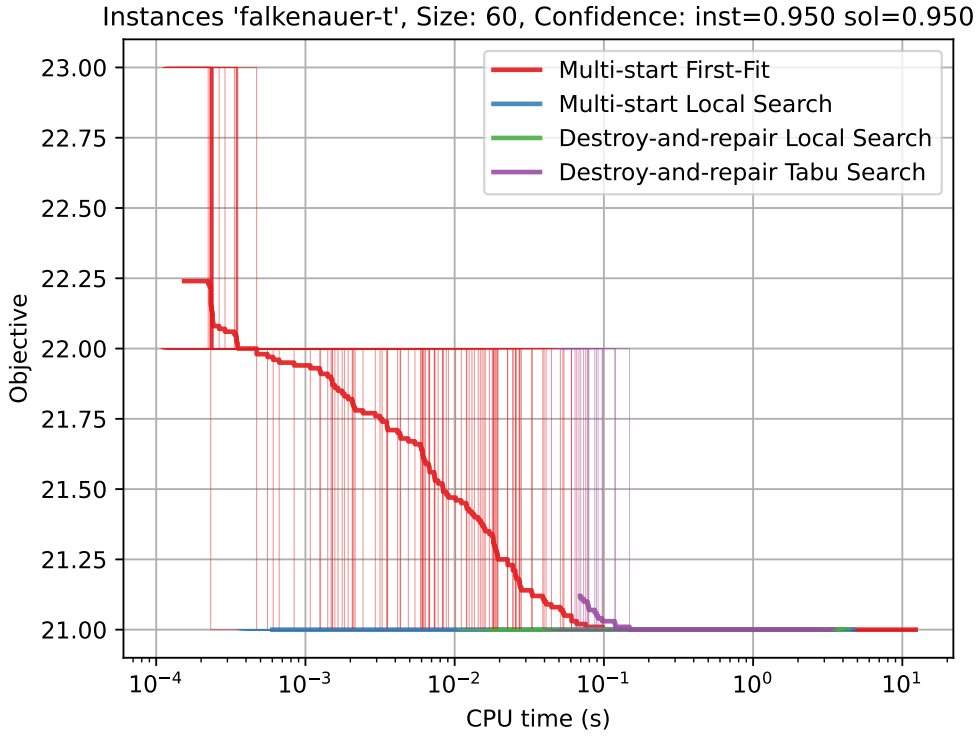

Figure 4.3: Performance at size 60, instance confidence 0.950 and solution confidence 0.950

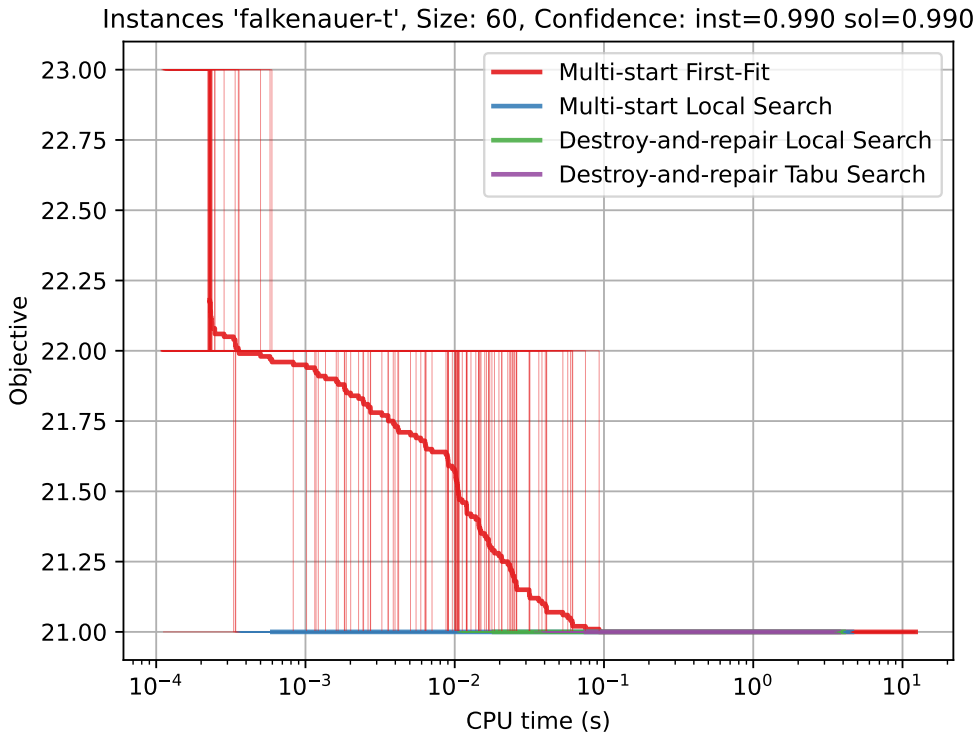

Figure 4.4: Performance at size 60, instance confidence 0.990 and solution confidence 0.990

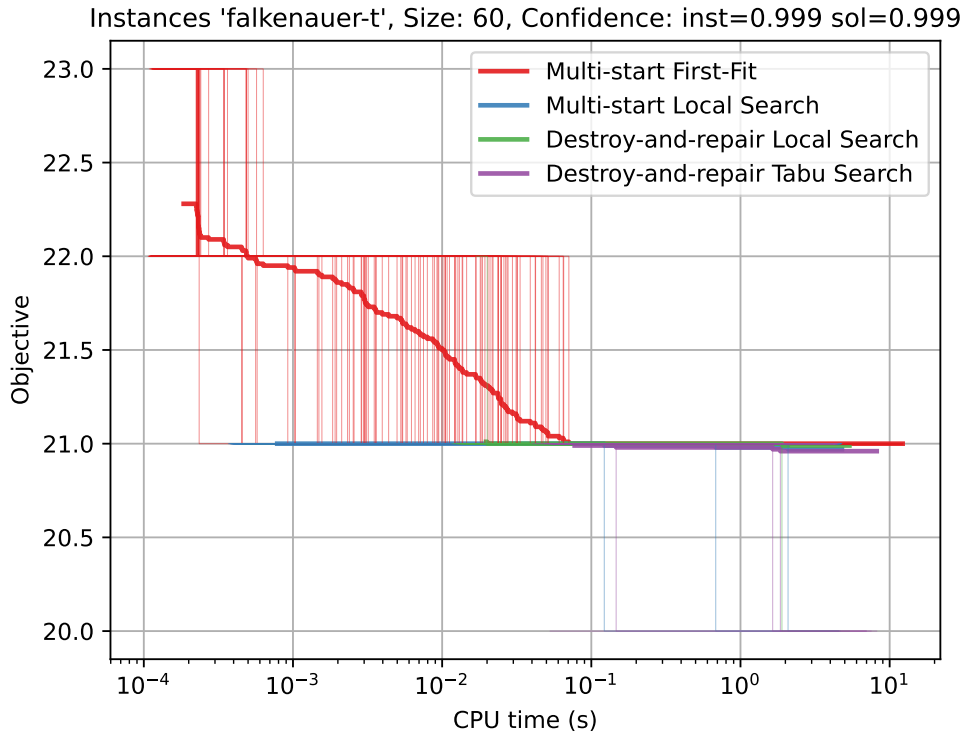

Figure 4.5: Performance at size 60, instance confidence 0.999 and solution confidence 0.999

Looking at the figures above, we can see that MFF has a shorter runtime but clearly produces worse results. MLS, DLS, and DTS have quite similar results, but among the three, MLS stands out for having the shortest runtime.

When we specifically analyze Figure 4.5 with the results for the 0.999 confidence level, it is possible to note that MLS, DLS, and DTS have reached the optimal value for some instances. Of the three, DTS was the algorithm that reached the optimal value more times, a total of 4 times, as we can see in the Table 4.1. This is a very positive result because the "triplet instances" are quite difficult to solve and usually have only a single optimal solution.

The figures for instances with size 300, also show the differences between the execution time and the quality of the solution of the four algorithms.

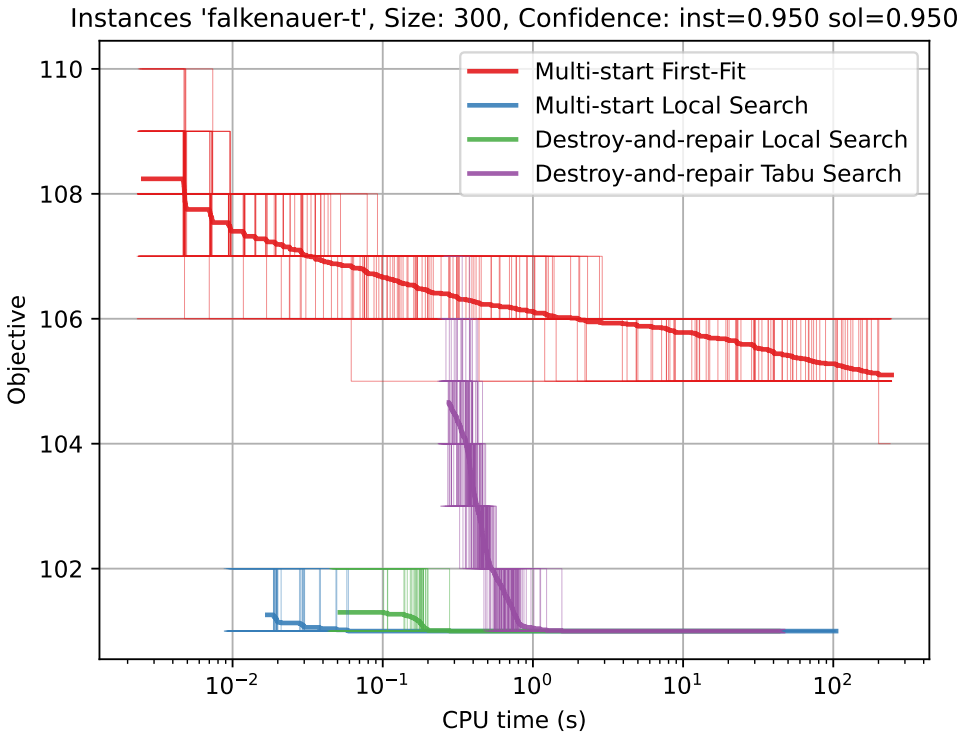

Figure 4.6: Performance at size 300, instance confidence 0.950 and solution confidence 0.950

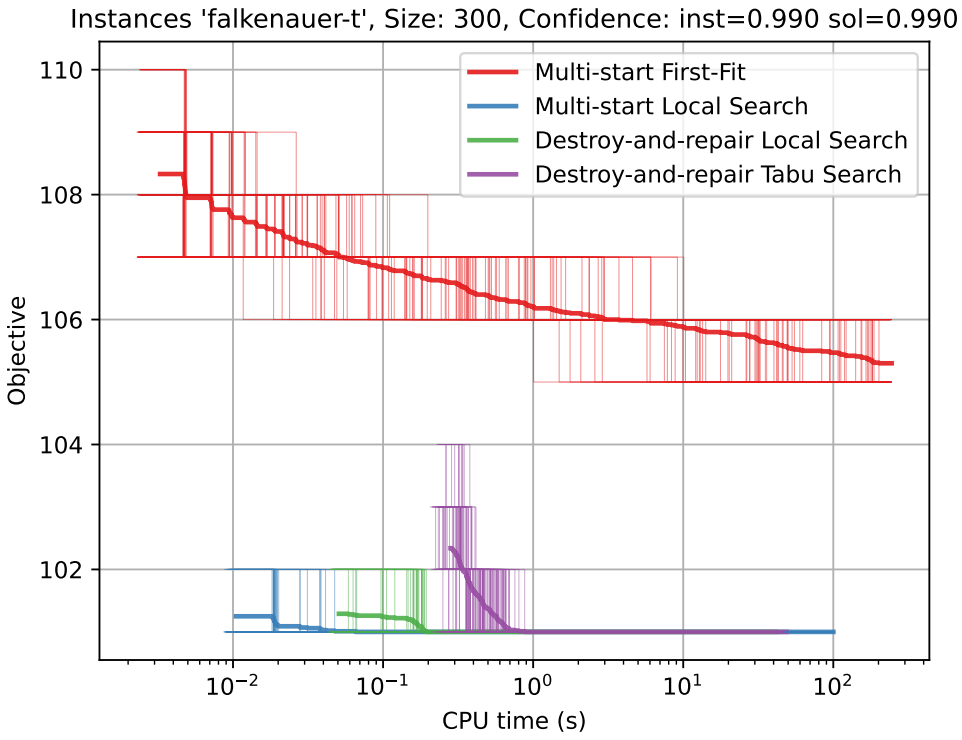

Figure 4.7: Performance at size 300, instance confidence 0.990 and solution confidence 0.990

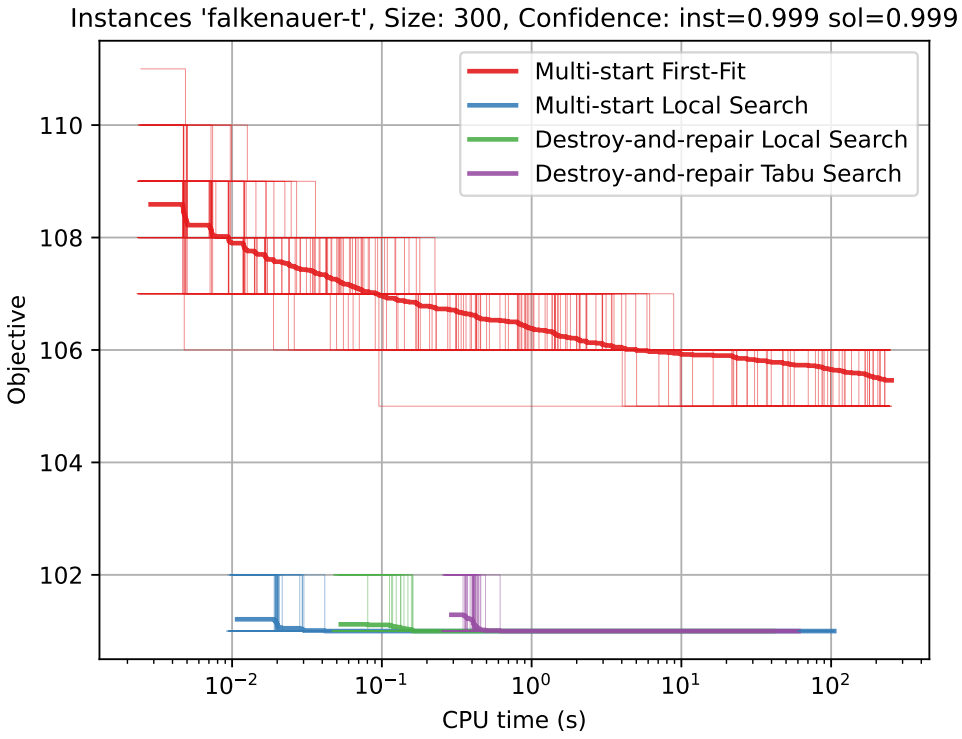

Figure 4.8: Performance at size 300, instance confidence 0.999 and solution confidence 0.999

Looking at the figures, we can clearly see that MFF had bad results, not even coming close to the minimum values obtained by the other algorithms. In all the figures, MLS and DLS had quite similar results, but MLS' runtime is shorter. DTS had a lot of variation between the three instance types: with confidence level 0.950 (Figure 4.6), it initially had pretty large values; with confidence level 0.990 (Figure 4.7), it initially had large values but performed better than with a confidence level of 0.950; with confidence level 0.999 (Figure 4.8), it performed much better initially and was comparable to MLS and DLS. But in the end, in all the figures, it arrived to good results. In these figures we can also clearly see that among the three algorithms with the best results, MLS has stood out for the least runtime.

In general, across all cases, we observed that the MFF algorithm consistently failed to produce values near the optimum. When comparing execution time and results, the MLS algorithm emerges as a viable solution.

4.2.2 Instance Confidence Level Different Solution Confidence Level

The other figures, with the instance's confidence level different from the solution's confidence level, can be found in Appendix A.

Notice that, when ICDF is different from CDF, the optimal value is unknown. Table 4.2 shows the number of times each algorithm reached the known minimum value (found as the best of the solution of the algorithms). For example, if we find the value 22 and it is the lowest known value, we count this executions. This allows us to compare the relative performances of the algorithms and see which method approaches a more optimized solution to these scenarios.

Table 4.2: Number of times reaching the minimum known value

Size	**ICDF**	**CDF**	**MFF**	**MLS**	**DLS**	**DTS**
60	0.950	0.950	100	100	100	100
60	0.950	0.990	10	100	95	91
60	0.950	0.999	52	100	92	91
60	0.990	0.950	100	100	100	100
60	0.990	0.990	100	100	100	100
60	0.990	0.999	3	100	99	96
60	0.999	0.950	10	100	100	100
60	0.999	0.990	100	100	100	100
60	0.999	0.999	94	97	95	98
300	0.950	0.950	0	100	100	100
300	0.950	0.990	0	100	99	85
300	0.950	0.999	0	87	90	88
300	0.990	0.950	0	100	100	100
300	0.990	0.990	0	100	100	100
300	0.990	0.999	0	100	100	100
300	0.999	0.950	0	100	100	100
300	0.999	0.990	0	100	100	100
300	0.999	0.999	0	100	100	100

Analyzing Table 4.2, we can see that MLS was the algorithm that reached more minimum values. The DLS and DTS performed quite similarly, but the DTS was a little bit better. MFF arrived rather few times at minimum values.

Chapter 5

Conclusion

This work has presented efficient approaches to optimize the use of the operating rooms in hospitals. This problem is similar to a non-deterministic version of the bin packing problem, which we solve by means of the application of heuristics (First Fit) and metaheuristics (Local Search and Tabu Search).

The results showed that while simpler algorithms, as is the case of MFF, offer fast solutions, they do not always provide good results. On the other hand, more complex methods, such as MLS, DLS and DTS, sometimes provide better solutions in terms of objective value despite requiring a longer runtime. However, if we balance the runtime and solution quality, the algorithm that had better results was MLS, showing that in many cases there is no need for the use of improved versions like DLS or DTS. Indeed, the MLS already offers excellent results without excessively compromising the runtime.

Optimization of the number of slots required to schedule all operations in a hospital directly influences resource management as well as operating costs. In addition, it was highlighted that the use of robust algorithms was shown to be efficient even in scenarios where there is uncertainty in the duration times of operations, which reinforces the potential of the application of these methods in real contexts.

Despite the limitations faced, such as the absence of real data and the need to work with artificially generated data, the results obtained suggest that the proposed algorithms are promising and can be applied in real situations, bringing significant benefits for hospital management.

There are several directions for future work. One is the integration with other hospital units. Operation room scheduling optimization should not be considered in isolation. Effective integration with other hospital units, such as the Intensive Care Unit (ICU), is crucial to ensure efficient patient flow. The use of neural networks to predict ICU occupancy, for example, allows for a more comprehensive and coordinated scheduling approach between the different areas of the hospital [17]. This integration ensures that scheduling takes into account the capacity and availability of each unit, optimizing the overall flow of patients and resources.

Another possible way for future work would be the use of Machine Learning for the prediction of the duration of operations. Predictive models, based on actual historical data, could provide more accurate estimates of operations durations, considering factors such as, i.e., the type of operation, patient profile, or medical staff. Hence, the use of Machine Learning could facilitate to generate more realistic synthetic data, allowing optimization algorithms to be tested and refined on a more real set of instances.

Bibliography

[1] Bernardetta Addis, Giuliana Carello, Andrea Grosso, and Elena Tànfani. Operating room scheduling and rescheduling: a rolling horizon approach. *Flexible Services and Manufacturing Journal*, 28(1):206–232, 2016.

[2] Susanne Albers, Arindam Khan, and Leon Ladewig. Best fit bin packing with random order revisited. *Algorithmica*, 83:2833–2858, 2021.

[3] M Arenas, A Bilbao, Rafael Caballero, Trinidad Gómez, Maria Victoria Rodríguez, and Francisco Ruiz. Analysis via goal programming of the minimum achievable stay in surgical waiting lists. *Journal of the Operational Research society*, 53(4):387–396, 2002.

[4] Valentina Bellini, Marco Guzzon, Barbara Bigliardi, Monica Mordonini, Serena Filippelli, and Elena Bignami. Artificial intelligence: a new tool in operating room management. role of machine learning models in operating room optimization. *Journal of medical systems*, 44 (1):20, 2020.

[5] Mirjana M Cangalovic, Vera V Kovacevic-Vujcic, Lav Ivanovic, Milan Drazic, and Miroslav D Asic. Tabu search: A brief survey and some real-life applications. *Yugoslav journal of operations research*, 6(1):5–18, 1996.

[6] M Charlesworth and JJ Pandit. Rational performance metrics for operating theatres, principles of efficiency, and how to achieve it. *Journal of British Surgery*, 107(2):e63–e69, 2020.

[7] An Jen Chiang, Angus Jeang, Po Cheng Chiang, Po Sheng Chiang, and Chien-Ping Chung. Multi-objective optimization for simultaneous operating room and nursing unit scheduling. *International Journal of Engineering Business Management*, 11:1847979019891022, 2019.

[8] Mauro Dell'Amico and Marco Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations research*, 41(3):231–252, 1993.

[9] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. *Int. J. Comput. Eng. Res. Trends*, 351(5):2349–7084, 2015.

[10] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2:5–30, 1996.

[11] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[12] Şeyda Gür and Tamer Eren. Application of operational research techniques in operating room scheduling problems: literature overview. *Journal of Healthcare Engineering*, 2018(1): 5341394, 2018.

[13] Şeyda Gür, Mehmet Pınarbaşı, Hacı Mehmet Alakaş, and Tamer Eren. Operating room scheduling with surgical team: a new approach with constraint programming and goal programming. *Central European Journal of Operations Research*, 31(4):1061–1085, 2023.

[14] Máté Hegyháti, Krisztián Attila Bakon, and Tibor Holczinger. Optimization with uncertainties: a scheduling example. *Central European Journal of Operations Research*, 31 (4):1239–1263, 2023.

[15] David S. Johnson. *Near-optimal bin packing algorithms*. Phd thesis, Massachusetts Institute of Technology, 1973.

[16] AM Mathai and HJ Haubold. *Fractional and multivariable calculus*, volume 122. Springer, 2017.

[17] Julian Schiele, Thomas Koperna, and Jens O Brunner. Predicting intensive care unit bed occupancy for integrated operating room scheduling via neural networks. *Naval Research Logistics (NRL)*, 68(1):65–88, 2021.

[18] Bureau of Naval Weapons Special Projects Office. Program evaluation research task, summary report, phase 1. Technical report, United States Department of the Navy, Washington, D.C., 1958. Later renamed to Program Evaluation and Review Technique (PERT).

[19] Rob JM Vaessens, Emile HL Aarts, and Jan Karel Lenstra. A local search template. *Computers & Operations Research*, 25(11):969–979, 1998.

[20] Leonardo Vanneschi and Sara Silva. Optimization problems and local search. In *Lectures on Intelligent Systems*, pages 13–44. Springer, 2023.

Appendix A

Figures with the instance's confidence level different from the solution's confidence level

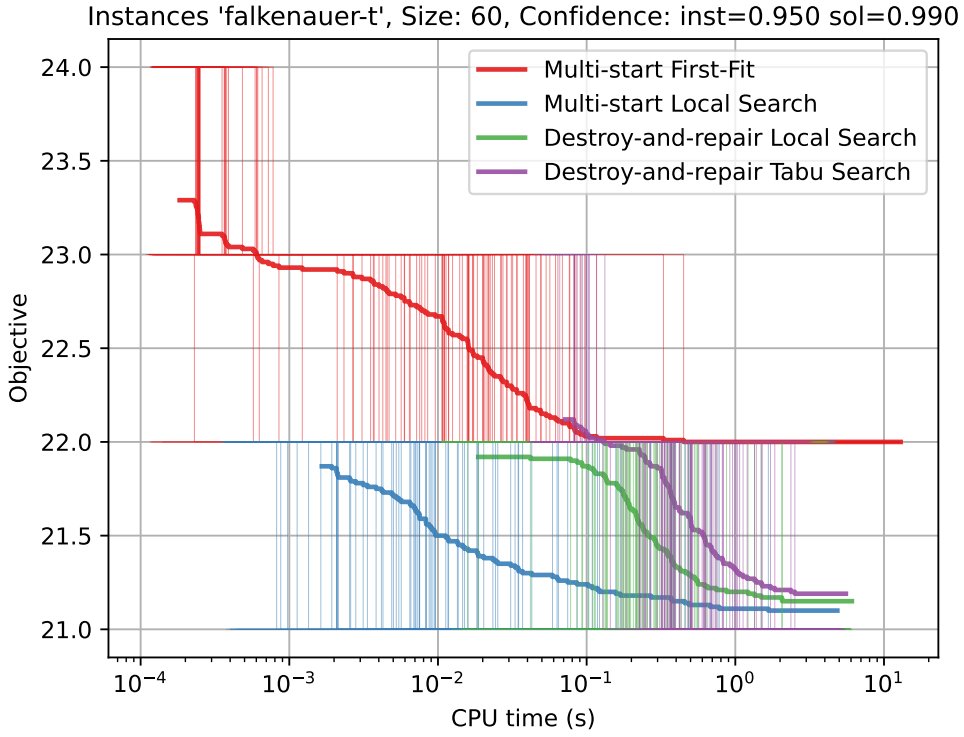

Figure A.1: Performance at size 60, instance confidence 0.950 and solution confidence 0.990

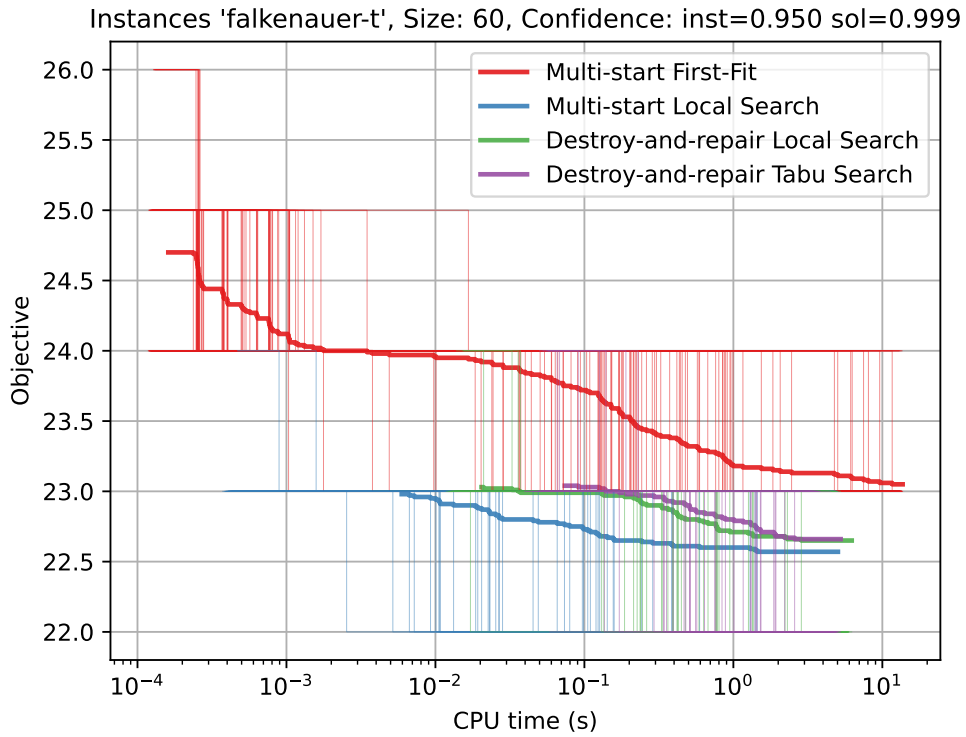

Figure A.2: Performance at size 60, instance confidence 0.950 and solution confidence 0.999

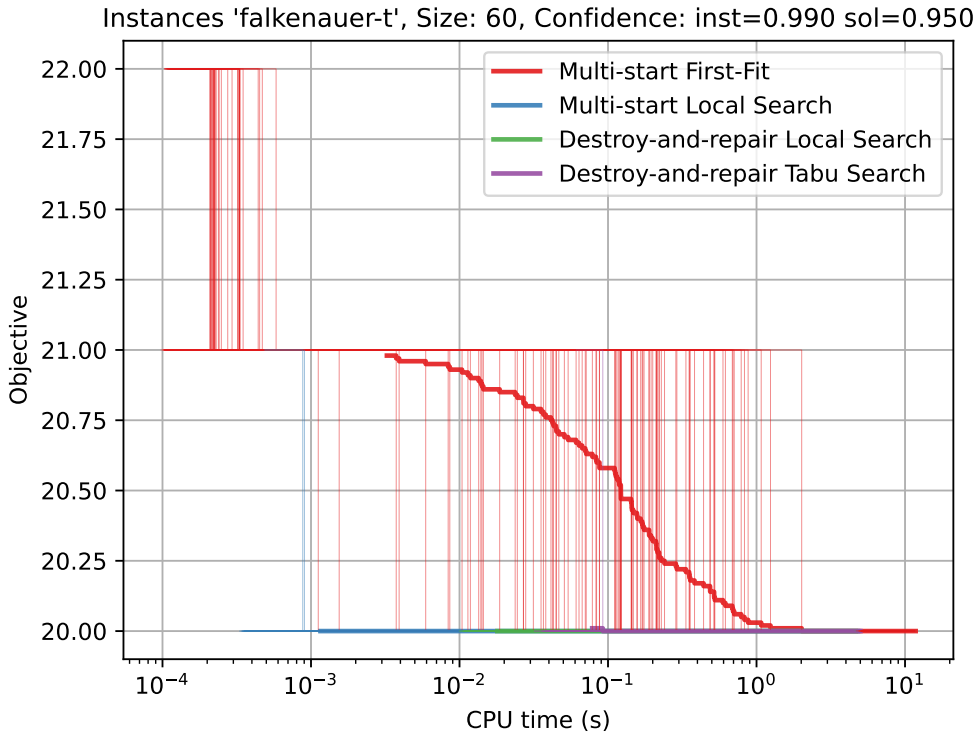

Figure A.3: Performance at size 60, instance confidence 0.990 and solution confidence 0.950

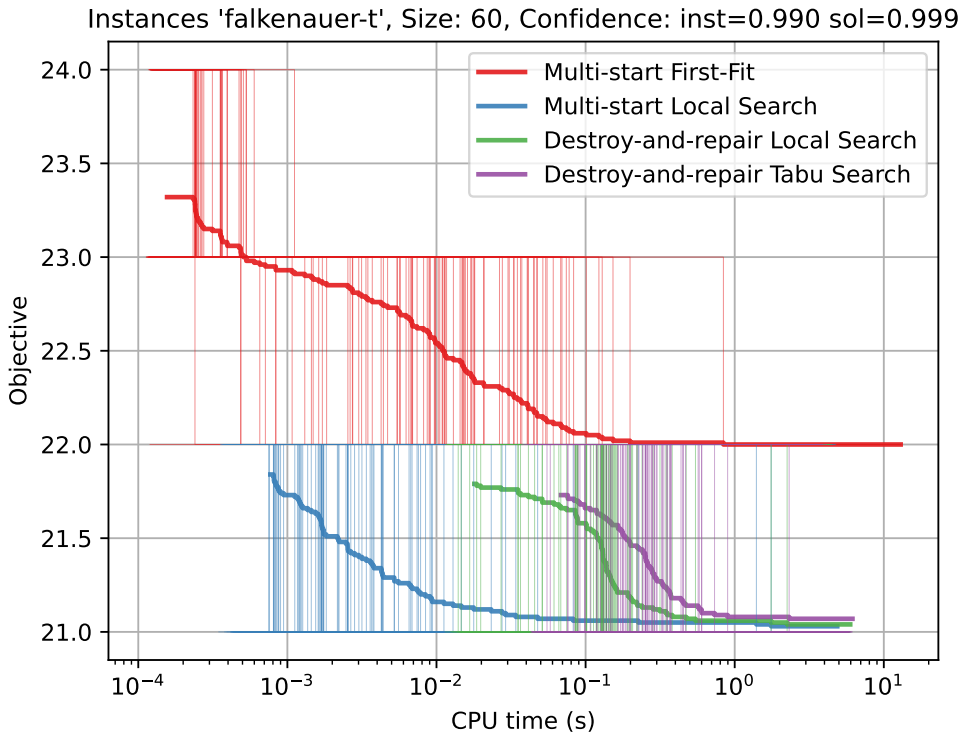

Figure A.4: Performance at size 60, instance confidence 0.990 and solution confidence 0.999

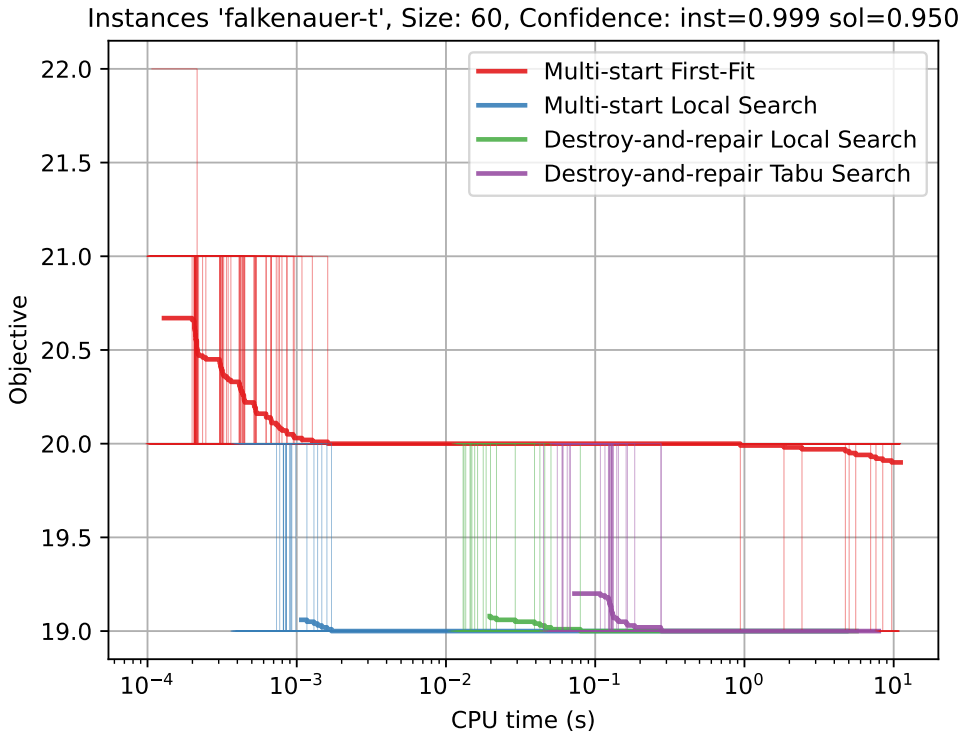

Figure A.5: Performance at size 60, instance confidence 0.999 and solution confidence 0.950

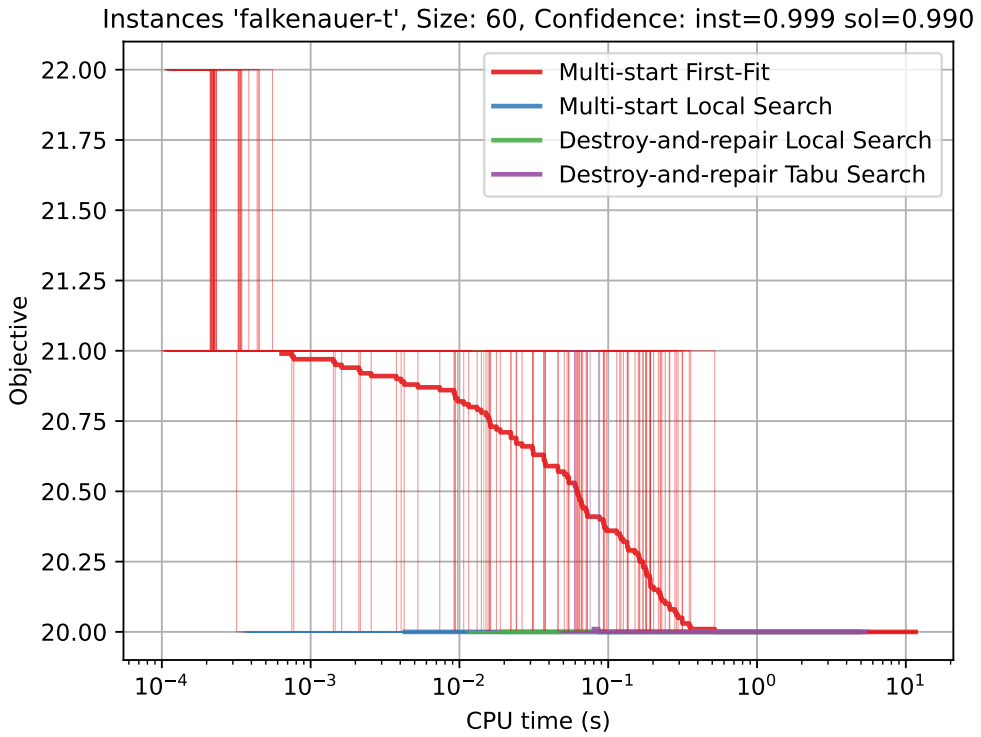

Figure A.6: Performance at size 60, instance confidence 0.999 and solution confidence 0.990

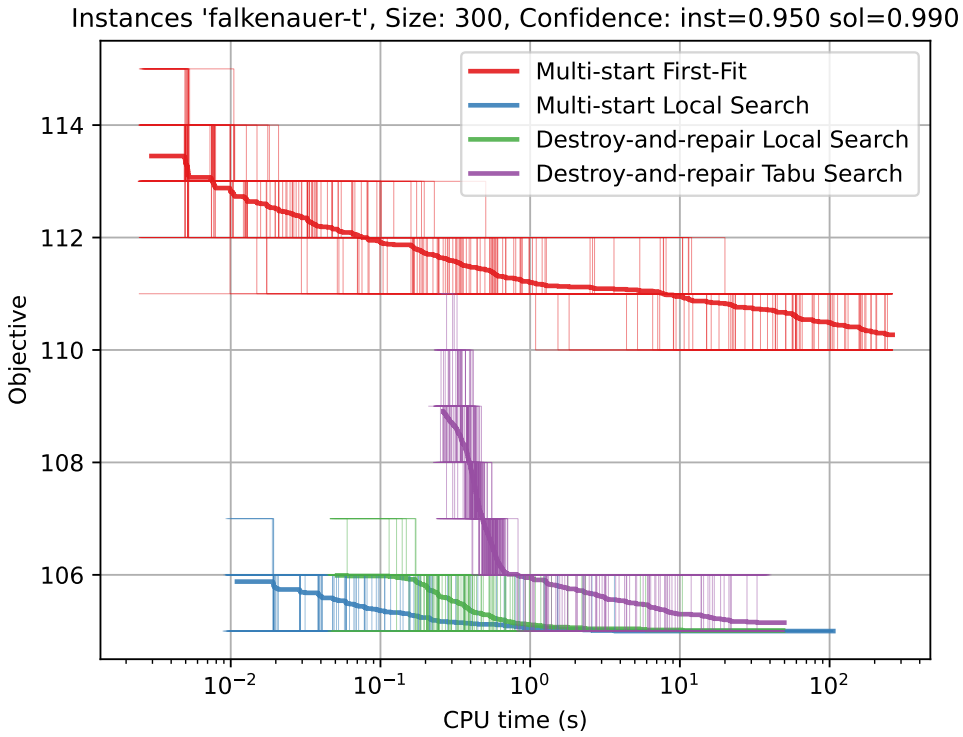

Figure A.7: Performance at size 300, instance confidence 0.950 and solution confidence 0.990

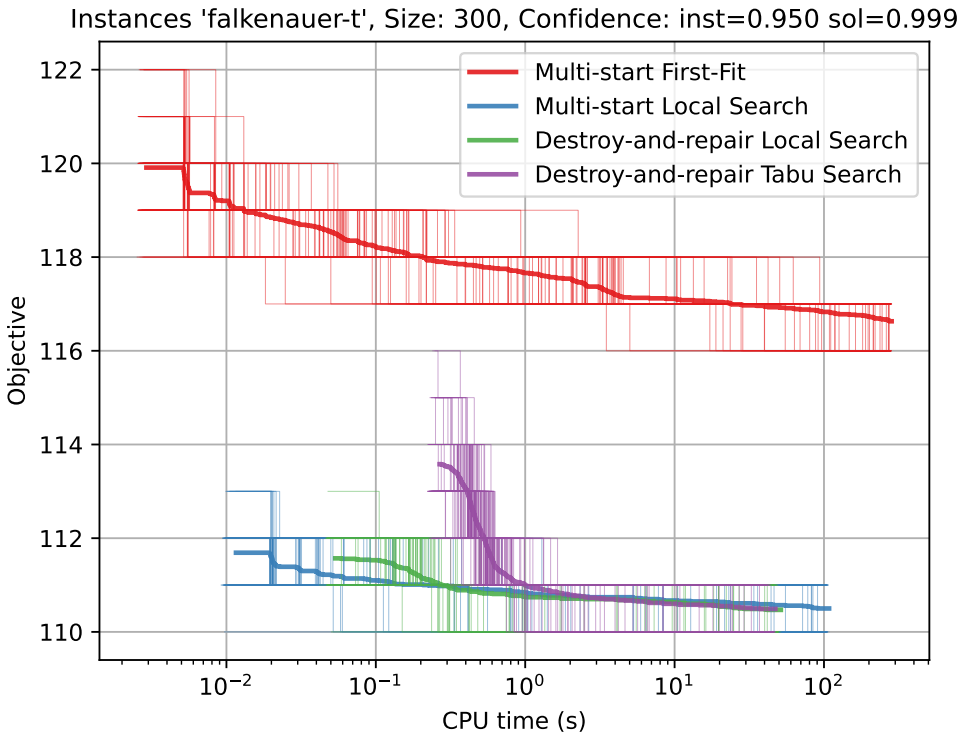

Figure A.8: Performance at size 300, instance confidence 0.950 and solution confidence 0.999

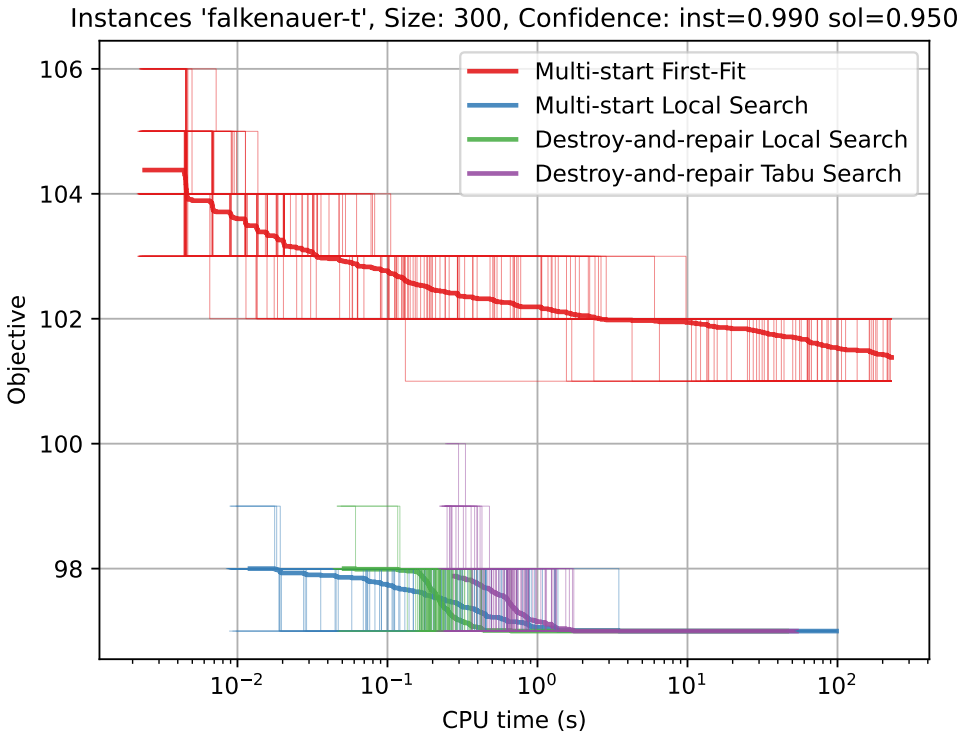

Figure A.9: Performance at size 300, instance confidence 0.990 and solution confidence 0.950

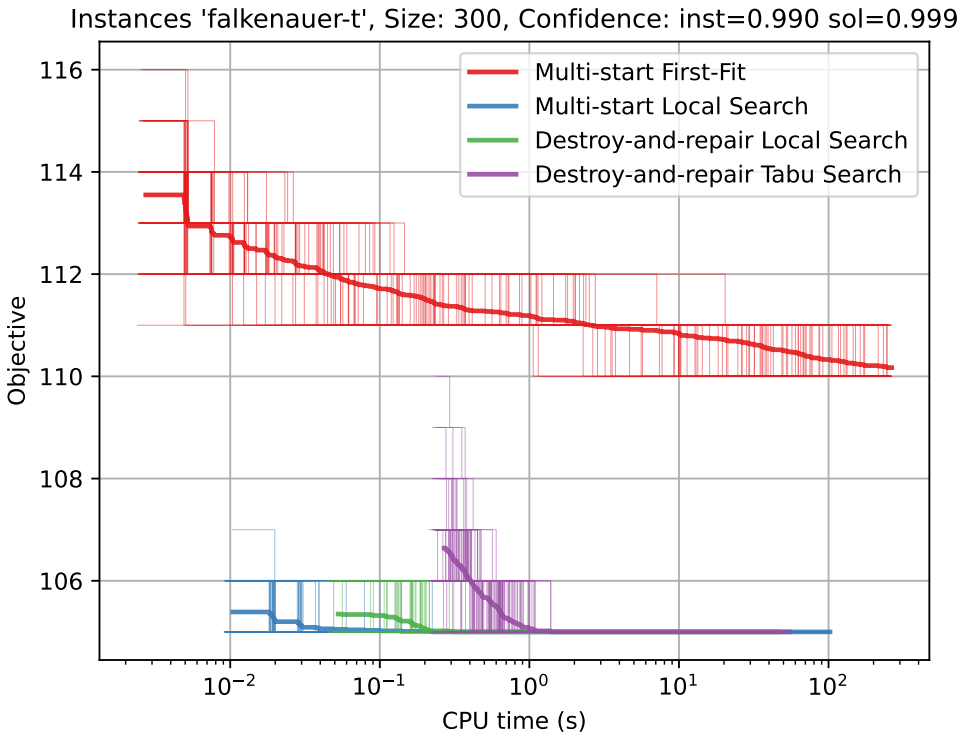

Figure A.10: Performance at size 300, instance confidence 0.990 and solution confidence 0.999

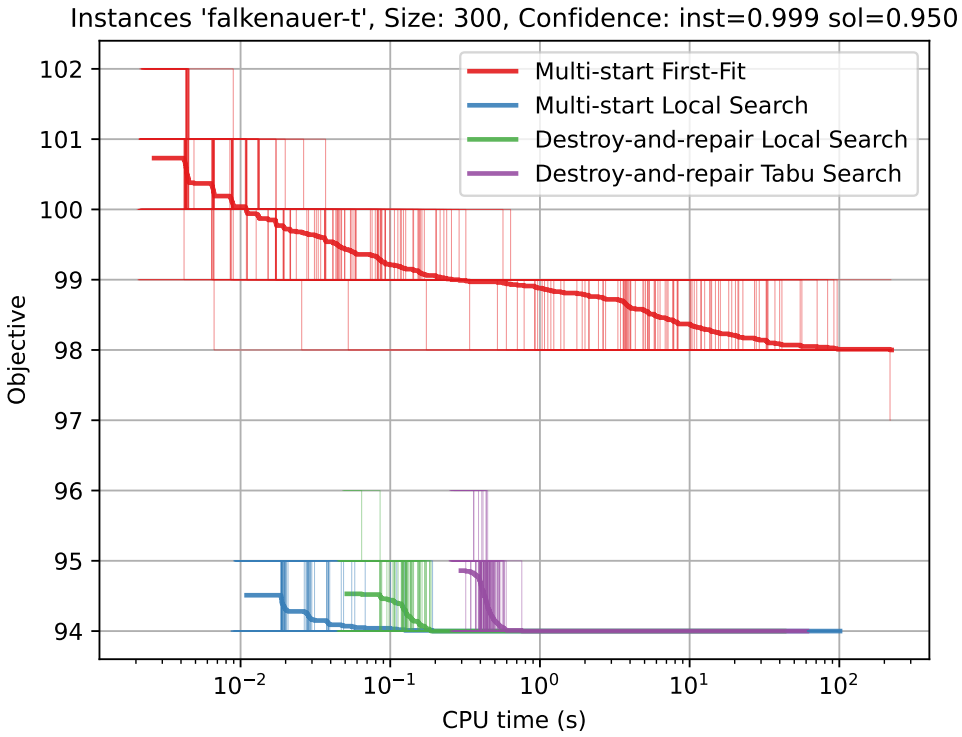

Figure A.11: Performance at size 300, instance confidence 0.999 and solution confidence 0.950

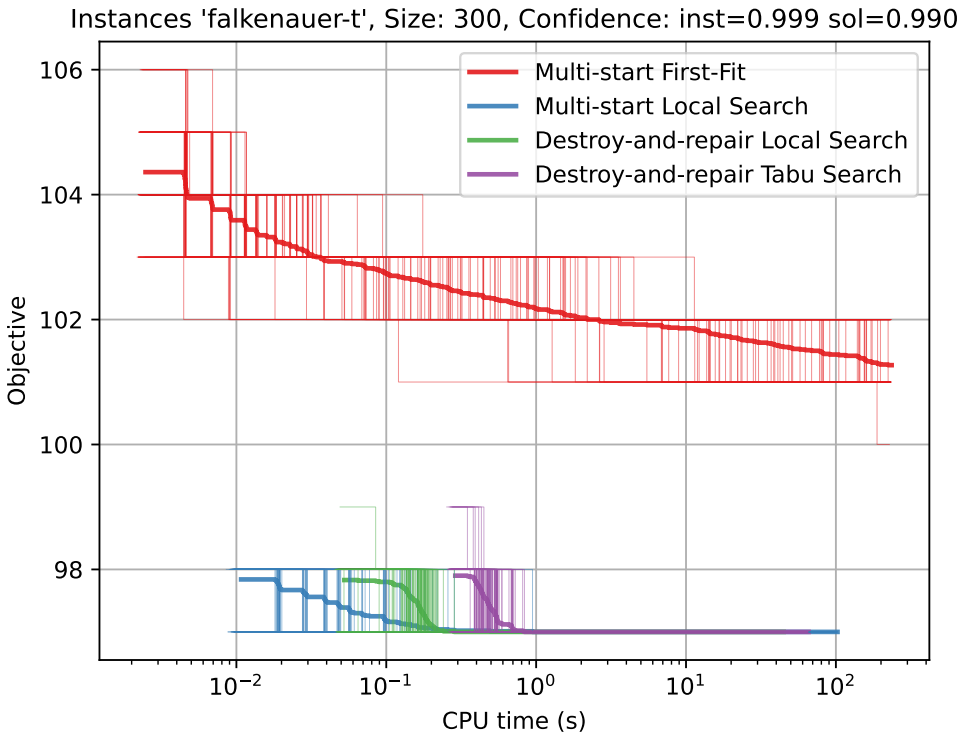

Figure A.12: Performance at size 300, instance confidence 0.999 and solution confidence 0.990