

Solving the Three-Dimensional Beacon Placement Problem Using Constraint-Based Methods, Large Neighborhood Search, and Evolutionary Algorithms

Sven Löffler, Viktoria Abbenhaus, George Assaf and Petra Hofstedt

*Department of Mathematics and Computer Science, MINT, Programming Languages and Compiler Construction Group,
Brandenburg University of Technology Cottbus-Senftenberg, Konrad-Wachsmann-Allee 5, Cottbus, Germany*
fl

Keywords: Beacon Placement, Constraint Programming, CSP, COP, Large Neighborhood Search, LNS, Evolutionary Algorithms, Indoor Positioning, Optimization Algorithms, Decision Support Systems.

Abstract: With the increasing prevalence of large building complexes, indoor localization is becoming an area of growing significance. In critical situations, such as emergencies in factories or care facilities, the ability to locate a person quickly can be a matter of life and death. One possibility for localization are Bluetooth beacons, which are either attached to the person or in rooms. We pursue the latter approach, whereby the beacon signals are used to determine the position of a receiving device, e.g. a mobile phone. At this, the use of a sufficient number of beacons in the building must be ensured in order to guarantee adequate coverage. However, to minimize costs, it is equally important to avoid placing unnecessary beacons. This creates a challenging optimization problem that this paper addresses through three distinct approaches: constraint programming, large neighborhood search, and evolutionary algorithms. Using simulated three-dimensional buildings, we test and evaluate these methods, ultimately providing a practical and efficient approach applicable to real-world building environments.

1 INTRODUCTION

Both in industrial settings, such as large factories, and in private residential areas, including dormitories and care facilities, increasingly large building complexes are being constructed where people work, live, and may also encounter emergencies. In such situations, it is critical to locate and assist individuals as quickly as possible. A viable solution for indoor localization, which also respects individuals' privacy, is the use of Bluetooth beacons. However, to minimize costs, it is essential to deploy the fewest number of beacons necessary. This creates a significant optimization challenge, heavily influenced by the architecture of the building. Factors such as wall and window types and thickness, as well as room layouts and sizes, affect the range of each beacon. For reliable position detection using triangulation, every point in the building must be covered by signals from at least three beacons. This paper examines the challenges of this problem and provides solutions tailored to diverse building configurations.

Indoor localization has a wide range of applications, including indoor navigation, asset tracking, per-

sonnel monitoring, and more. Common techniques in indoor positioning algorithms involve Bluetooth Low Energy (BLE) beacons and signal strength measurements, which are often used for triangulation, trilateration, or proximity-based approaches (Sakpere et al., 2017; Bembenik and Falcman, 2020). This article focuses on trilateration, a method for determining the position of a point based on its distances to three reference points.

The optimal placement of beacons for trilateration aims to minimize the number of beacons required (reducing costs) while ensuring seamless coverage of the space (triple signal coverage of all points). Currently, beacon placement is often performed manually, which is not only time-consuming but also prone to errors. Despite extensive testing, manual methods provide no guarantees of complete coverage across the entire building nor certainty regarding the minimum or an acceptably small number of beacons required.

Our novel approaches, based on constraint programming (CP), large neighborhood search (LNS), and evolutionary algorithms (EA), address this challenge by guaranteeing full coverage of all positions with a minimal number of beacons in three-

dimensional buildings. By adopting a conservatively chosen set of parameters, our methods ensure reliable and efficient beacon placement while significantly reducing the risks associated with manual approaches.

The remainder of this paper is organized as follows. Section 2 discusses various existing approaches for beacon placement. Section 3 provides the foundational concepts of constraint programming, large neighborhood search, and evolutionary algorithms. Section 4 describes the proposed methods for determining an optimized beacon placement. In Section 5, the different approaches are evaluated in terms of their applicability and quality across various generated test buildings. Finally, Section 6 concludes the paper with a summary of findings and an outlook on future work.

2 RELATED WORK

This section first discusses related work on the topic of automatic beacon placement, followed by an examination of our previous research on 2D beacon placement.

2.1 Automatic Beacon Placement

Numerous recent studies have focused on optimizing beacon placement for indoor positioning, employing a variety of approaches.

A key aspect of beacon placement is selecting appropriate metrics to evaluate positioning accuracy. The *Geometric Dilution of Precision* (GDOP), originally developed for satellite navigation systems, has been successfully adapted for 5G mmWave networks to optimize base station selection for positioning. In (Rajagopal et al., 2016), GDOP was adapted for indoor environments, leading to a significant reduction in the number of beacons required compared to standard trilateration methods.

Another approach utilizes *Time-of-Flight* (ToF) signals between beacons and target devices. (Wang et al., 2019) investigates beacon position optimization and proposes a greedy algorithm that initially places $O(OPT \ln(m))$ beacons. Additionally, a random sampling algorithm is introduced, reducing the required beacons to $O(OPT \ln(OPT))$, resulting in fewer beacons compared to earlier approaches.

The study by (McGuire et al., 2021) focuses on the self-localization of autonomous vehicles using *Angle-of-Arrival* (AoA) for position calculation, incorporating course angles. It presents the determinant of the Fisher Information Matrix for an arbitrary number of beacons and derives the optimal angular spacing for three beacons through numerical simulations.

Another approach, proposed by (Sharma and Badarla, 2018), treats the beacon placement area as a grid of candidate positions on ceilings and walls. It minimizes the total number of beacons while adhering to GDOP constraints using Mixed Integer Linear Programming (MILP). This method improves the minimum GDOP without increasing the number of beacons.

To our knowledge, no other work has implemented a comparable constraint-programming-based approach for beacon placement exclusively using Boolean variables. Furthermore, known methods are designed for two-dimensional spaces, whereas we aim to extend these approaches to three-dimensional environments. Transitioning to three-dimensional space introduces a critical challenge: the lack of scalability of certain algorithms. The addition of a third dimension significantly increases the search space, often preventing many algorithms from finding sufficiently good solutions within acceptable timeframes.

2.2 Prior Work by the Authors

In this subsection, we present our previous contributions to the field of optimal beacon placement (Löffler et al., 2022). Our prior work has primarily focused on constraint-based two-dimensional beacon placement strategies, laying the foundation for the advancements discussed in this paper.

Figure 1 provides an overview of the approach used to compute an optimal beacon placement as described in (Löffler et al., 2022). The process begins by importing an image of a floor plan, which is then preprocessed into two 2D arrays: an environmental resistance array A_E and a reachability array A_B .

The environmental resistance array A_E encodes environmental factors E corresponding to materials in the map. These factors were determined experimentally as follows: $E(\text{open spaces}) = 2$, $E(\text{drywall}) = 2.5$, $E(\text{solid walls}) = 4.5$, and $E(\text{glass}) = 10$.

The reachability array A_B is a Boolean array where the entry at position (x, y) indicates whether the corresponding location (x, y) must be triple-covered by the beacons (True) or not (False). The latter case applies, for instance, to areas outside the building or regions blocked by thick walls.

In the next step, both arrays are scaled, aggregating the pixels of the original map into larger pixel blocks. This process is performed conservatively, meaning the aggregated pixel blocks adopt the maximum environmental resistance E of their constituent pixels, and any block remains reachable if even one of its original pixels was reachable.

Using an RSSI-based distance calculation (see

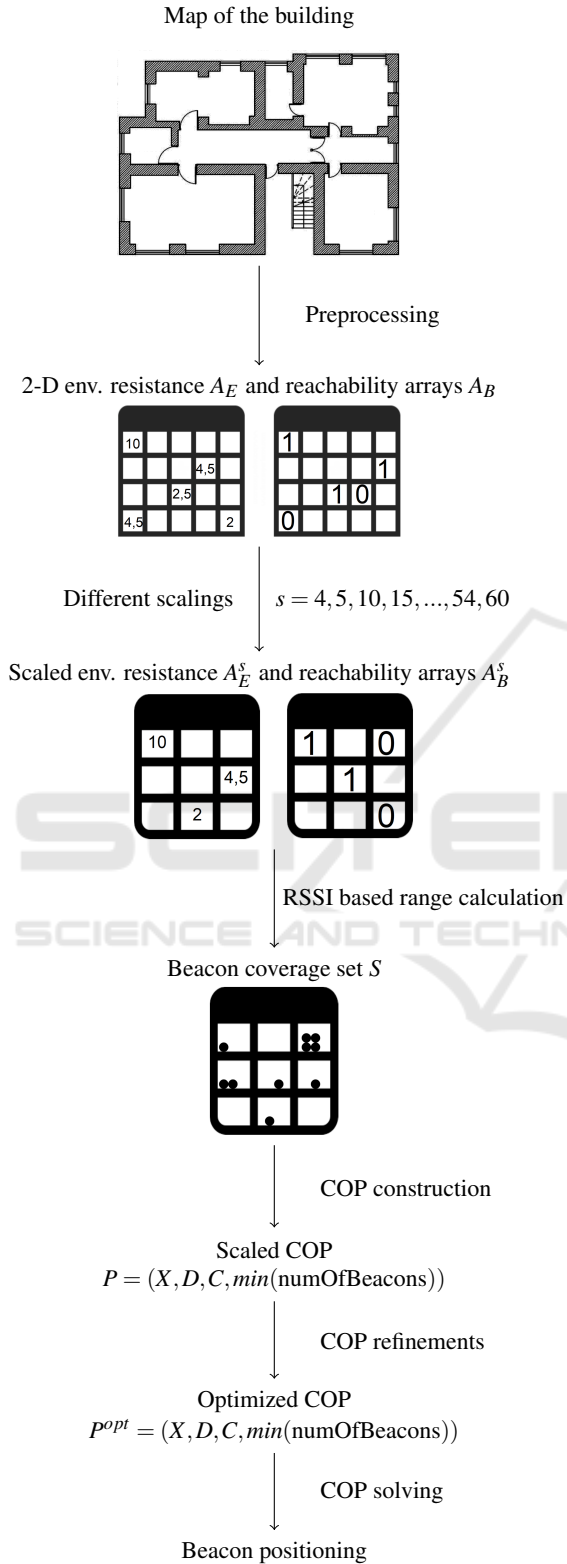


Figure 1: Overview of the beacon positioning process from (Löffler et al., 2022).

Equation 1, (Li et al., 2018)), the potential coverage of a beacon placed at position (x, y) is determined. Specifically, for each position (x, y) , the set of all positions covered by a beacon at that location is calculated, forming a coverage set S .

$$distance = 10^{M_1 - \frac{RSSI}{10 \cdot E}} \quad (1)$$

Subsequently, a constraint optimization problem (COP) P is formulated based on these coverage sets. The COP P is then further optimized through various refinements to P^{opt} for runtime efficiency and solved globally to determine an optimal beacon placement. Among these refinements, parallelization methods (parallel portfolio (Régin and Malapert, 2018)) and the addition of extra constraints were considered.

Figure 2 depicts an example section of a floor plan with scaled pixels overlaid. The different wall types are represented by green (drywall with $E=2.5$), blue (solid wall with $E=4.5$), and red (glass with $E=10$), with varying wall thicknesses. These characteristics result in different signal ranges for individual beacons. Using the previously introduced method, the original pixels are scaled into the defined scaled pixels. This means that any scaled pixel containing walls is assigned the signal resistance value of the highest resistance present within it. For example, the scaled pixel in row 2 and column 3 contains free space ($E = 2$), a drywall ($E = 2.5$), and a glass wall ($E = 10$), resulting in a scaled glass pixel with $E = 10$ (see Figure 3). Additionally, the accessibility of the scaled pixels must be defined. In Figure 2, it is assumed that areas containing walls are not required to be triple-covered by beacon signals. However, in the scaled version shown in Figure 3, this means that scaled pixels with at least one accessible original pixel must remain accessible (Non-accessible areas are marked with hatched patterns in the figure). An example of such a pixel that must remain accessible, even if parts of the original pixels are walls, is the pixel located in row 2, column 3.

The scaled pixels, represented in the algorithm process (see Figure 1) by the arrays A_E^s and A_B^s , serve as the foundation for the subsequent constraint problem, which calculates the optimal beacon placement. In Figure 3, multiple beacons (B) are shown, forming part of a solution that ensures triple coverage of the entire accessible building while minimizing the number of beacons used.

In (Löffler et al., 2023), we introduced an alternative approach based on set variables to model and solve the constraint problem. Compared to the approach presented in (Löffler et al., 2022), this method achieved solutions slightly faster but at the cost of reduced solution quality. Since the quality of the solu-

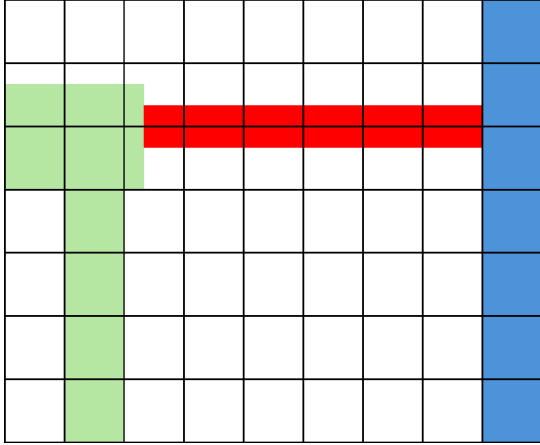


Figure 2: A section of a building floor plan for a single floor, shown in top-down view, with a scaled pixel grid.

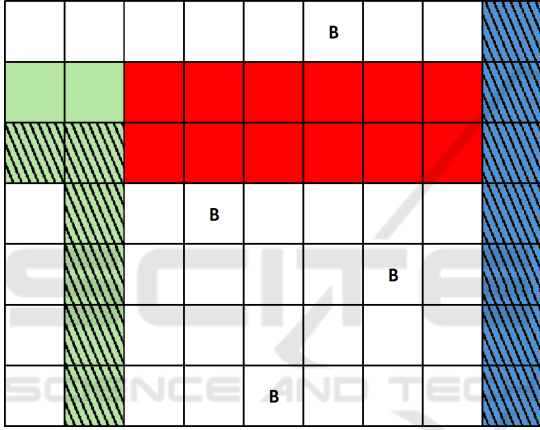


Figure 3: A scaled section of a building floor plan for a single floor with a random distribution of beacons.

tion is of primary importance for our objectives, we have decided to proceed with the finite-domain approach from (Löffler et al., 2022) in the present work. However, given the modular design of methodology presented below, it could also be adapted to use the set-variable-based approach if desired.

However, both methods are tailored to two-dimensional spaces. Extending the approach to three-dimensional spaces introduces significant challenges, as the constraint model becomes excessively large and computationally expensive. This paper addresses this limitation by proposing novel methodologies that efficiently handle the increased complexity and scalability requirements of three-dimensional spaces.

3 PRELIMINARIES

In this section, we provide an overview of the fundamental principles underlying Constraint Program-

ming, Large Neighborhood Search, and Evolutionary Algorithms.

3.1 Constraint Programming

Constraint Programming (CP) is a powerful declarative approach for modeling and solving NP-hard problems. Common research domains in CP include rostering, graph coloring, optimization, and satisfiability problems (SAT) (Marriott, 1998). Using constraint programming, a global optimum can be found and validated, which typically makes this approach highly time-consuming. The general CP workflow can be divided into two main components: 1. the declarative representation as a constraint model, 2. solving the constraint model using an independent solver.

A *Constraint Satisfaction Problem (CSP)* is formally represented as a triple $P = (X, D, C)$, where: $X = \{x_1, x_2, \dots, x_n\}$ is a set of variables, $D = \{D_1, D_2, \dots, D_n\}$ is a set of finite domains, where D_i is the domain of variable x_i , and $C = \{c_1, c_2, \dots, c_m\}$ is a set of constraints that may involve one or more variables from X (Apt, 2003a). A *constraint* is a tuple (X, R) , where X is an ordered set of variables and R is a relation defined over X (Dechter, 2003a).

A *solution* of a CSP is an instantiation of all variables x_i with values $d_j \in D_i$ such that all constraints are satisfied. A *Constraint Optimization Problem (COP)* extends a CSP by introducing an objective variable x_{opt} , which must be either minimized or maximized.

For this work, a critical constraint is the *count* constraint. The $count(X, occ, v)$ constraint restricts the variables in set X such that the value v occurs exactly occ times (Demasse and Beldiceanu, 2024; van Hoeve and Katriel, 2006). An example of the *count* constraint is $count(\{x_1, x_2, x_3\}, \{1, 3\}, 2)$ with $d_1 = d_2 = d_3 = \{1, 2, 3\}$. In this case, the value 2 must appear exactly one or three times in the variables x_1, x_2, x_3 . Possible solutions include, for instance, $x_1 = 1, x_2 = 2, x_3 = 3$ or $x_1 = 2, x_2 = 2, x_3 = 2$.

CSPs and COPs are typically solved using a backtracking search algorithm combined with constraint propagation. Popular solvers for finite domain problems include Google OR-Tools (Perron and Furnon, 2023), Gecode (Christian Schulte, 2019), and Choco-Solver (Prud'homme and Fages, 2022), with the latter being utilized in this work. For more detailed information on solvers and their mechanisms, see (Apt, 2003b; Dechter, 2003b; Rossi et al., 2006).

3.2 Large Neighborhood Search

Large Neighborhood Search (LNS) was first introduced in (Shaw, 1998) as a method for solving big constraint problems. This approach involves *destroying* a significant portion of the variable assignments in a given solution, followed by an optimization step to *repair* the solution and potentially find a better one. This approach can typically only determine locally optimal solutions. The original work focuses on solving the *Vehicle Routing Problem* (VRP). In this paper, we adapt the method for the optimal placement of beacons.

The core idea behind the destruction phase is to modify or remove parts of the current solution based on a heuristic. In the context of a constraint problem, this means reversing variable assignments, effectively freeing those variables to take on new values from their respective domains.

During the repair phase, the resulting partial structure is refined using an optimization technique, such as constraint programming or local search methods. The rationale is that the original CP model is often too large and complex to be solved globally in a reasonable amount of time. By partially destroying the complete solution, a smaller subproblem is created that is computationally more tractable. This subproblem can then be solved to global optimality. However, solving the subproblem globally does not imply a global solution to the original problem.

Through iterative cycles of destruction and repair, the algorithm aims to escape local optima and progressively approach a global solution (without guarantee that this will be achieved). This strategy leverages the balance between exploration and exploitation, allowing the method to effectively navigate the solution space.

3.3 Evolutionary Algorithms

Inspired by Darwin's understanding of evolution (Darwin, 1859), algorithms have been developed to mimic this process computationally. The fundamental idea is that an environment can sustain only a limited number of individuals. However, each individual possesses an inherent drive for reproduction, necessitating a selection process based on the principle of "survival of the fittest."

Each individual represents a unique combination of phenotypic traits, which are evaluated by the environment. If this combination is favorable, the individual has a higher probability of producing offsprings. Darwin's key insight was that small, random variations (mutations in phenotypic traits) occur naturally

during reproduction, passing from one generation to the next. Even though this approach allows for the simultaneous consideration of multiple solutions, it typically still results in only locally optimal solutions.

Algorithm 1: Evolutionary Algorithm.

Data: Population size n , crossover rate r_c , mutation rate r_m , max generations g
Result: Best individual found during the evolution process

```

1  $P = \text{generatePopulation}(n)$ 
2 for  $i = 1$  to  $g$  do
3    $P' = \text{crossover}(P, r_c)$ 
4    $P' = \text{mutate}(P', r_m)$ 
5    $P = \text{survivorSelection}(P, P', n)$ 
6 end
7 return  $\text{best}(P)$ 

```

Algorithm 1 outlines the procedure of an evolutionary algorithm (Popyack, 2016). Initially, a population of n random individuals is generated (line 1). As long as the maximum number of generations g has not been reached (line 2), a crossover operation is performed between two or more individuals, depending on the crossover rate r_c (line 3). Typically, individuals with higher fitness values are preferentially selected for crossover.

Following the crossover, the resulting offspring undergo mutation based on the mutation rate r_m , which involves introducing random changes to certain values of an individual (line 4). This enhances genetic diversity within the population.

Subsequently, the new population P is formed by selecting the n fittest individuals from the combined set of the current population P and the newly generated offspring P' (line 5). After g iterations, the algorithm returns the best individual found (line 7).

Typical application areas of evolutionary algorithms include optimization problems (Slowik and Kwasnicka, 2020), such as finding optimal model parameters in machine learning (Shanthi and Chethan, 2023).

4 MODELING THE PROBLEM

In this section, we first describe our representation of the three-dimensional space of buildings using pixels, before discussing our various solution approaches.

4.1 Representation of the 3-D Space

In (Löffler et al., 2022; Löffler et al., 2023), we used two-dimensional $n \times n$ pixels to represent the floor plan of a single level. Each pixel was assigned one of the values $E = 2, 2.5, 4.5$ or 10 to reflect the characteristics of the floor plan. Larger pixel sizes led to more pessimistic abstractions, as the worst value within a pixel's area was always assumed for the entire pixel.

Extending this representation into a three-dimensional space by using $n \times n \times n$ voxels is impractical. For instance, with a ceiling-plus-floor height of $h = 3$ meters and a pixel side length of 10 cm, this would result in 30 stacked voxels per floor. Consequently, the computation of coverage sets S , i.e., the beacon ranges, would become significantly more complex for even a single floor.

Unlike horizontal layouts, where walls can appear at various positions, floors and ceilings are generally uniform in vertical structure. This implies the presence of a solid ceiling (or floor) of thickness h^c with $E = 4.5$ (solid wall), which does not need to be reachable, and beneath it, a space of room height h^r . This space could be open ($E = 2$), contain a drywall ($E = 2.5$), a solid wall ($E = 4.5$), or glass ($E = 10$), among other possibilities.

To simplify vertical abstraction and considering that the general beacon range significantly exceeds the floor height h (room height h^r plus ceiling thickness h^c), each pixel is assigned dimensions of $n \times n \times h$ with $h = h^c + h^r$. This allows us to compute the beacon coverage sets (S) for a single floor using the same method as in the two-dimensional case. Coverage for positions above and below the current floor is handled by applying an offset to the RSSI calculation such that $E = 4.5 + h^c * 0.01$, and the distance is reduced by h . This dynamic factor (0.01 per cm) accounts for ceiling thickness h^c , and was determined experimentally.

For floors farther above or below, both the dynamic attenuation factor ($0.01 * h^c$) and the base distance (e.g., 3 meters in this example) are incrementally increased for each level. Based on our findings, beacon signals can generally reach one floor above and below, with limited coverage extending two floors in both directions.

In conclusion, the space considered for coverage calculations increases approximately fivefold, encompassing the current floor as well as the two above and two below.

4.2 A Boolean Constraint-Based Approach

The constraint-based approach essentially follows the method outlined in (Löffler et al., 2022) and Figure 1. The corresponding COP is illustrated in Figure 4.

For each potential position (i, j) where a beacon can be placed, a Boolean variable $x_{i,j}$ is created, indicating whether a beacon is placed at that position (1) or not (0). The set $X^{S_{i,j}} \subseteq X$ contains all variables corresponding to positions where a beacon can cover the position (i, j) . To enable trilateration, at least three beacons must cover this position, as enforced by the first *count* constraints c_1 .

To ensure minimal interference in trilateration, two beacons must be placed at least 3 meters apart. This requirement is captured by the second *count* constraints c_2 , which ensure that for all variables in $X^{S_{i,j}}$, representing a $3m \times 3m$ neighborhood around (i, j) , at most one variable can take the value 1 , meaning only one beacon can be placed within this area.

The total number of beacons, denoted by the variable x_{count} , is determined using the constraint c_3 . This is achieved by counting all occurrences of the value 1 in X (i.e., all positions where a beacon is required). Finally, the counted beacons are minimized ($minimize(x_{count})$). The difference from the 2D model lies in the beacon coverage sets $S_{i,j}$, which now include not only positions on the same level but also those up to two levels above and below.

Although this model is theoretically correct, it reveals significant memory limitations during the later evaluation. This is due to the increase in size of the count constraints c_2 , as the enlarged coverage sets $S_{i,j}$ span multiple levels, making the constraints too large to handle within memory. Additionally, the overall solution speed is substantially reduced. Consequently, this approach appears feasible only for very large pixel sizes. However, using such large pixels likely results in poor solution quality due to overly pessimistic abstractions. For this reason, alternative solution approaches were explored in the subsequent sections.

4.3 A Large Neighborhood Search Approach

As an initial attempt, we propose a randomized LNS approach. This method begins by randomly positioning $n = \sqrt{l * w}$ beacons on each floor of the building, where l and w represent the length and width (in meter) of the building, respectively.

Next, we use the *repair* algorithm presented in Algorithm 2 to extend this initial placement B into

$P = (X, D, C, f)$ with:

$$\begin{aligned}
 X &= \{x_{i,j} \mid \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\} \cup \{x_{count}\}, & \text{(one variable for each position } (i, j) \text{ in the } n \times m \text{ map)} \\
 D &= \{D_{i,j} = \{0, 1\} \mid \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\} \cup & \text{(at position } (i, j) \text{ is a beacon (1) or not (0))} \\
 &\quad \{D_{count} = \{3, \dots, n * m\}\} & \text{(maximal number of beacons is between 3 and } n * m) \\
 C &= c_1 = \{count(X^{S_{i,j}}, [3, 4, \dots, |X^{S_{i,j}}|], 1) \mid \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\} \cup & \text{(every position } (i, j) \text{ is covered by at least 3 beacons)} \\
 &\quad c_2 = \{count(X^{3_{i,j}}, [0, 1], 1) \mid \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\} & \text{(every two beacons have at least a distance of 3 meter)} \\
 &\quad c_3 = \{count(X, x_{count}, 1) \mid \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\} & \text{(count the used beacons)}
 \end{aligned}$$

$minimize(x_{count})!$

Figure 4: The COP which represents the beacon positioning problem.

one that ensures the entire building is covered threefold. For this purpose, the coverage of all points in the space is calculated based on the placed beacons, and the set of points S^{not} is determined, representing all points that are not yet covered by at least three beacons (line 1). Additional beacons are randomly placed at the positions corresponding to some of the points in S^{not} until all points are covered by at least three beacons (lines 2 to 5). Once this condition is met, the necessity of each beacon is evaluated. For every beacon, the points it covers are analyzed to determine the minimum number of beacons covering all these points (line 6). If this minimum value exceeds three, the beacon is considered unnecessary, as its removal would still ensure that all positions remain covered threefold. Subsequently, unnecessary beacons with highest such value are randomly removed one at a time until no such beacons remain (lines 7 to 10). This process results in an initial feasible solution.

Algorithm 2: Repair

Data: An incomplete beacon placement B ,
The scaled environment array A_E^s .
Result: A possible beacon placement B

```

1  $S^{not} = calculateNotCovered(B, A_E^s)$ 
2 while ( $|S^{not}| > 0$ ) do
3    $B = placeRandomBeacon(A_E^s, S^{not})$ 
4    $S^{not} = calculateNotCovered(B, A_E^s)$ 
5 end
6  $S^{ToMany} = calculateUnnecessary(B, A_E^s)$ 
7 while ( $S^{ToMany} > 0$ ) do
8    $B = removeRandomly(B, S^{ToMany})$ 
9    $S^{ToMany} = calculateUnnecessary(B, A_E^s)$ 
10 end
11 return  $B$ 

```

After generating this initial solution, the LNS process begins. The method iteratively *destroys* and *repairs* the existing solution until a predefined time limit is reached. Once the time limit expires, the best solution found during the process is returned. To *destroy* a solution, one-fifth of the beacons in the current beacon placement B are randomly removed. The repair step is then performed using the previously introduced *repair* Algorithm 2.

This approach combines random decisions (such as determining which beacons to place or specifically remove) with a greedy strategy, where unnecessary beacons are removed in order of least necessity. Compared to the constraint-based approach described in the previous Section 4.2, it is not necessary to compute all possible beacon coverages for every point, but only those that have been placed during the process. This results in significant savings in both computation time and memory usage, but it also means that the obtained solution is not guaranteed to be globally optimal. However, given the problem size, it can be assumed that the constraint model would also fail to find a globally optimal solution within an acceptable time frame.

During the development of the approach, the idea arose that it might be more advantageous to start not with a random placement of beacons, but with a beacon positioning uniformly distributed across the space. Due to the varying wall types and thicknesses, such an even distribution does not guarantee that the entire space is triple-covered. Consequently, a *repair* process is initially invoked from this starting configuration, which first randomly completes the solution and then removes redundant beacons randomly. Subsequently, the procedure continues with the same *destroy* and *repair* behavior as the purely random LNS approach. In Section 5, this method will be compared to the completely random approach.

4.4 A Constraint-Based LNS Approach

Once both a constraint-based and an LNS approach for finding an optimal beacon placement were established, the next logical step was to investigate whether these two methods could be combined. Since the combination, due to LNS, no longer retains a global search characteristic, solving the problem as a single COP appears unnecessary. Instead, it was decided to generate a separate COP for each floor and solve them sequentially from the bottom to the top, whereby passing on partial solutions (see Algorithm 3).

Algorithm 3: An initial level-based beacon placement.

Data: The scaled environment array A_E^s ,
number of floors h

Result: A possible beacon placement B

```

1  $B = \text{empty}()$ 
2 for ( $\text{int } i = 1$  to  $h$ ) do
3    $B = \text{solveCOP}(B, i, A_E^s)$ 
4    $S^{\text{ToMany}} = \text{calculateUnnecessary}(B, A_E^s)$ 
5   while ( $S^{\text{ToMany}} > 0$ ) do
6      $B = \text{removeRandomly}(B, S^{\text{ToMany}})$ 
7      $S^{\text{ToMany}} =$ 
8        $\text{calculateUnnecessary}(B, A_E^s)$ 
9   end
10 return  $B$ 

```

The placements of beacons calculated for the lower floors are considered given when solving for the higher floors (line 3). This means that a position on an upper floor may already be singly, doubly, or even multiply covered by one or more beacons placed on lower floors. Consequently, the requirement for further coverage is reduced to 2, 1, or even 0. Once the placements for the current floor are added to B (line 3), any now superfluous beacons are removed. This could also include beacons on lower floors that have become obsolete due to the placement of beacons on higher floors (lines 4–8). This approach mirrors the one used within the *repair* method outlined in Algorithm 2.

After generating an initial solution, this solution is iteratively *destroyed* and *repaired* until a time limit is reached. In this *destruction* phase, a single level (floor) is selected, and all beacon placements on that level are removed. The *repair* function then places new beacons on the selected level using the constraint-level-based method from line 3 of Algorithm 3. Subsequently, any newly unnecessary bea-

cons are removed (lines 4–8). When the time limit is reached, the best solution found so far is returned.

This method is not global but incorporates the optimal planning of a single floor, which is then utilized in a greedy manner. It is hypothesized that this approach will yield better solutions than the entirely random LNS approach discussed in the previous section.

Another way to combine the two approaches is to determine the initial placement using the constraint-level-based method, and then refine this placement using the randomized LNS approach presented in Section 4.3. This ensures that the initial solution has a certain level of quality and is not entirely random, while also allowing subsequent placements to be computed faster based on the simpler calculation as described in Section 4.3.

4.5 Evolutionary Algorithms

Finally, we consider evolutionary algorithms, as presented in Algorithm 1. For the generation of the initial population, $\frac{3}{4}$ of the individuals are created completely randomly (as described in the very beginning of Section 4.3), while the remaining $\frac{1}{4}$ are evenly distributed across a single level with random completion, as outlined at the end of Section 4.3. Each individual represents a solution to the beacon placement problem. This means that all constraints, particularly the coverage constraints, are satisfied; however, the number of beacons may still be significantly above the minimum. After generating the initial population, it is iteratively updated until the time limit is reached.

The *crossover* operation is performed as follows: First, all individuals are weighted based on the inverse of the number of beacons they use. Two individuals, A and B , are then randomly selected with probabilities proportional to their weights (i.e., the fewer beacons they use, the higher their probability of selection). For each level, it is determined, again based on the number of beacons used on that level, whether the level is inherited from individual A or B (the fewer beacons on a level, the higher the likelihood it is selected). It is ensured that at least one level from each individual A and B is included. The resulting beacon placement is then randomly completed and reduced with the *repair* method as described in Section 4.3.

Following the crossover, there is a 50% chance of a *mutation*. The *mutation* process mimics the *destroy-and-repair* cycle used in the LNS variant (see Section 4.3): 20% of the beacons are randomly removed, and new beacons are added until all positions are covered threefold. The solution is then reduced to the minimum necessary beacons.

The newly formed population is reduced using a

fitness function (selecting individuals with the lowest number of beacons), ensuring that only the fittest individuals survive. This new population is then used for further crossover operations. The entire process is repeated until the time limit is reached, at which point the best solution found so far is returned.

5 EVALUATION

This section begins by detailing the experimental setup, followed by an evaluation and discussion of the results obtained from the different solving approaches.

5.1 The Experimental Setup

All experiments were conducted on an LG Gram laptop equipped with an Intel(R) Core(TM) i7-1165G7 11th-generation quad-core processor clocked at 2.80 GHz and 16 GB DDR3 RAM operating at 2803 MHz. The system ran Microsoft Windows 10 Enterprise. The programming language used was Java with JDK Version 17.0.7, alongside the ChocoSolver Version 4.10.7 as constraint solver (Prud'homme and Fages, 2022).

For the evaluation, we generated random building layouts. The number of floors $h \in \{3, 5, 7\}$ as well as the width and length $w, l \in \{30\text{ m}, 40\text{ m}, 50\text{ m}\}$ were randomly selected. On each floor, 25 walls were randomly placed with random thicknesses between 25 cm and 60 cm, and with random types (drywall, massive wall, glass). While this approach does not necessarily produce realistic building layouts, it generates diverse resistance profiles for the Bluetooth beacons, similar to those encountered in real buildings, which is the key aspect of the problem being addressed.

We applied the various methods introduced in Section 4 to determine optimal beacon placements for the randomly generated buildings. In addition to the different methods, we also tested various resolutions (pixel sizes). Larger pixel sizes simplify the computations due to the reduced number of pixels. However, this may lead to lower solution quality, as the pixel values are calculated pessimistically (always taking the highest resistance value within the pixel, even if only part of the pixel exhibits this resistance). Not all methods could be applied to all scaling levels for every building, due to hardware limitations.

We used the following naming convention for our approaches: A^s represents the approach $A \in \{\text{COP}, \text{LNS-R}, \text{LNS-U}, \text{LNS-COP}, \text{LNS-COP-R}, \text{EA}\}$ and pixel size $s \in \{5, 10, 20, 25, 40, 50, 75, 100\}$. COP refers to the pure COP approach from Section 4.2.

LNS-R is the randomized LNS approach discussed in Section 4.3. LNS-U denotes the LNS variant starting with a uniform distribution of beacons (also Section 4.3). LNS-COP describes the level-based LNS approach using constraint programming, as introduced in Section 4.4. LNS-COP-R refers to the creation of an initial solution using the level-based constraint programming approach from Section 4.4, followed by refinement using the randomized LNS approach from Section 4.3 for further solution processing. EA describes the application of an evolutionary algorithm, as detailed in Section 4.5. A time limit of 10 minutes was applied to all approaches.

5.2 Results and Evaluation

Table 1 summarizes the results of our test series. It includes the six different solution approaches (COP, LNS-R, LNS-U, LNS-COP, LNS-COP-R, EA) with different scalings ($s \in \{5, 10, 20, 25, 40, 50, 75, 100\}$) and presents, in various columns, the percentages of instances for which at least one solution was found (Solvable), the number of instances where the obtained solution was optimal compared to the other approaches (#Best), and the average number of beacons needed for the best solution (#Beacons). The sum of the "Best" entries (31) exceeds the total number of test instances (29). This is because if two methods achieve equally optimal solutions (with the same number of beacons), both are credited with an increment in the "Best" value.

Figure 5 focuses on the new approaches and visualizes the success probability (black marker for each approach) and the number of required beacons (bar for each approach) from Table 1.

Initially, the pixel sizes $s \in \{5, 10, 20, 25, 50, 75, 100\}$ were planned. However, it quickly became apparent that the COP, LNS-COP, and LNS-COP-R instances could not find solutions for $s = 25$ or smaller. Consequently, these instances were excluded from the table. Instead, a value of $s < 50$ was sought where the instances were at least partially solvable, leading to the inclusion of $s = 40$ for these cases. For EA, the scalings $s = 10$ and $s = 5$ were also omitted. This is because such fine-grained scalings result in too few individuals being generated to justify labeling the method as an EA algorithm. Therefore, $s = 20$ was chosen as the smallest pixel size for EA.

It is evident that the original COP approach from (Löffler et al., 2022) performs reliably with large pixel sizes ($100\text{ cm} \times 100\text{ cm}$), achieving a 100% solvability rate. However, it performs poorly due to the imposed time limit, requiring an average of 825 bea-

Table 1: A comparison of the results between the different solution approaches for 29 different buildings.

Approach	Solvable	#Best	#Beacons
COP ¹⁰⁰	100%	0	825
COP ⁷⁵	89.7%	0	1320
COP ⁵⁰	93.1%	0	1582
COP ⁴⁰	96.5%	0	1596
LNS-R ¹⁰⁰	100%	0	206
LNS-R ⁷⁵	100%	0	217
LNS-R ⁵⁰	100%	0	221
LNS-R ²⁵	100%	0	221
LNS-R ²⁰	100%	0	219
LNS-R ¹⁰	100%	0	207
LNS-R ⁵	100%	2	175
LNS-U ¹⁰⁰	100%	0	205
LNS-U ⁷⁵	100%	0	217
LNS-U ⁵⁰	100%	0	221
LNS-U ²⁵	100%	0	227
LNS-U ²⁰	100%	0	223
LNS-U ¹⁰	100%	0	213
LNS-U ⁵	100%	0	184
LNS-COP ¹⁰⁰	93.1%	21	167
LNS-COP ⁷⁵	93.1%	5	167
LNS-COP ⁵⁰	27.6%	2	142
LNS-COP ⁴⁰	41.4%	0	175
LNS-COP-R ¹⁰⁰	93.1%	1	173
LNS-COP-R ⁷⁵	93.1%	0	173
LNS-COP-R ⁵⁰	27.6%	0	144
LNS-COP-R ⁴⁰	41.4%	0	176
EA ¹⁰⁰	100%	0	188
EA ⁷⁵	100%	0	201
EA ⁵⁰	100%	0	207
EA ²⁵	100%	0	211
EA ²⁰	100%	0	208

cons. Even though, with unlimited time, a globally optimal solution could theoretically be found. Using smaller pixel sizes results in some instances becoming unsolvable within the 10-minute time limit or due to memory constraints. Specifically, instances with pixel sizes smaller than $40\text{ cm} \times 40\text{ cm}$ could not be solved using this approach.

On the other hand, as the pixel size decreases, the number of required beacons increases significantly (e.g., to 1,596 for $s = 40$). This increase is attributable to the vast size of the search space, of which only a small fraction can be explored. Despite employing a global-based constraint approach, this method fails to yield a satisfactory or globally optimal solution within an acceptable timeframe.

The randomized LNS approach LNS-R introduced in Section 4.3 significantly outperforms the original constraint-based approach, yielding substantially better solutions (averaging between 175 and 221 beacons) while consistently solving all tested pixel sizes (s ranging from 5 to 100) with a 100% solvability rate.

It can be observed that larger pixel sizes ($s = 100$) initially produce better solutions than medium-sized pixels ($s = 50$), while very small pixel sizes ($s = 5$) ultimately yield the best solutions. Notably, this approach achieves the best solution twice across all compared methods (#Best).

This behavior can be explained as follows: with larger pixel sizes, the search space is smaller, enabling a broader exploration of possible solutions. As the pixel size decreases, the search space grows larger. However, the pessimistic approximations inherent to smaller pixels become less pronounced. Consequently, while fewer solutions can be found in larger search spaces, these solutions are less pessimistic and potentially better than those derived from larger pixel sizes.

The analysis indicates that the balance between pixel size and pessimistic approximation works best for small pixel sizes, followed by large pixel sizes, with medium sizes performing the least effectively.

The approach employing an initial uniform distribution of beacons (LNS-U) did not lead to any significant improvement. For large pixel sizes, no notable difference between LNS-U and LNS-R is observed. However, as the pixel size decreases, LNS-U performs increasingly worse compared to LNS-R.

Both level-based LNS approaches using constraint programming (i.e. LNS-COP and LNS-COP-R), as introduced in Section 4.4, do not always find a solution (solvability ranging from 27.6% to 93.1%). However, for the problems that are solved, the number of required beacons is notably low (142 to 176 on average). The averages of 144 and 142 beacons for LNS-COP-R⁵⁰ and LNS-COP⁵⁰, respectively, should be interpreted with caution, as these approaches could only solve approximately one-quarter of the instances. It is likely that these instances represent the easier problems, which require fewer beacons.

In general, approaches with larger pixel sizes are preferable due to their higher solvability rates. Among these, the fully constraint-based LNS approach (LNS-COP) consistently outperforms the hybrid approach (LNS-COP-R), in which the search begins with a constraint-based solution and then transitions to a randomized search (fewer beacons are required across all tested scenarios).

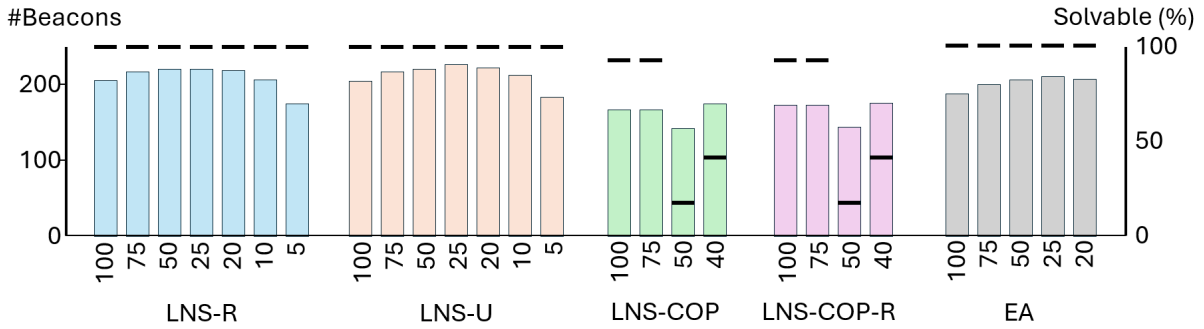


Figure 5: A visual comparison of the success probability of the different approaches and their required number of beacons.

The inability of the LNS-COP approaches to always find a solution, as compared to COP^{100} , is attributed to the division of computational time across different floors. Each level was allocated only 1 minute to find a solution so that ultimately some time remains for the *destroy* and *repair* process. If any floor fails to find a solution within the time limit, the entire problem remains unsolved. In contrast, the COP^{100} approach tackled the entire problem as a single instance, benefiting from the full 10-minute runtime. It is anticipated that the performance of the LNS-COP approaches would significantly improve with increased computational time, enabling them to solve a greater number of problem instances.

The most promising approach overall is LNS- COP^{100} , which delivered the best solution across all compared methods and resolutions in 21 out of 29 cases. On the other hand, it solves only 27 out of 29 instances, indicating that this approach should be used in parallel with a robust method capable of consistently finding a solution.

The evolutionary algorithm consistently finds a solution (solvability of 100%), unlike the constraint-based approaches. However, the solutions produced by this method sometimes require more beacons than the randomized LNS approach (LNS-R) and always require more beacons than the hybrid constraint-randomized approach (LNS-COP-R) and the LNS constraint-based approach (LNS-COP). Interestingly, the algorithm tends to perform best for the largest pixel sizes, yielding the most efficient solutions in those cases.

In conclusion, both the LNS-R and EA methods reliably produce high-quality solutions and significantly outperform the original constraint-based approach. However, the LNS- COP^{100} method emerged as the most effective, achieving the best solution in 21 out of 29 cases. For practical applications, a parallel portfolio approach (R  gin and Malapert, 2018) combining these three methods (LNS-R⁵, EA¹⁰⁰, and LNS- COP^{100}) is recommended. This combination

ensures that a high-quality solution can always be found.

6 CONCLUSION AND FUTURE WORK

In this work, we investigated various approaches to solving the beacon placement problem in three-dimensional spaces. These included a pure constraint-based approach (COP), a randomized LNS method with (LNS-U) and without (LNS-R) uniform initial beacon placement, two constraint-based LNS approaches with either random (LNS-COP-R) or constraint-based continuation (LNS-COP) for further solution exploration, and an evolutionary algorithm (EA). All newly developed methods significantly outperformed the original pure constraint-based approach, thereby enabling the practical application of these techniques to real-world, multi-level, large-scale buildings.

Among the proposed methods, the constraint-based LNS approach appears particularly promising, as it often delivers high-quality solutions. Future work will focus on advancing all the methods presented here. Specifically, the development of problem-specific search strategies for the constraint-based LNS approaches seems especially promising. As demonstrated in other problem domains (L  ffler et al., 2024), such strategies can often expedite the discovery of an initial solution. Additionally, integrating the constraint-based LNS with local search techniques, as outlined in (L  ffler and Hofstedt, 2024), holds great potential. Such an integration would allow leveraging the solutions from other methods (e.g., LNS, EA) as starting points for the constraint-based LNS, potentially enhancing both efficiency and solution quality.

REFERENCES

- Apt, K. (2003a). *Constraint satisfaction problems: examples*. In (Apt, 2003b). Chapter 2.
- Apt, K. (2003b). *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA.
- Bembenik, R. and Falcman, K. (2020). BLE indoor positioning system using rssi-based trilateration. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 11(3):50–69.
- Christian Schulte, Mikael Lagerkvist, G. T. (2019). Gecode 6.2.0, 2019, <https://www.gecode.org/>, last visited 2019-11-22.
- Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. Murray, London. or the Preservation of Favored Races in the Struggle for Life.
- Dechter, R. (2003a). Constraint networks. In (Dechter, 2003b), chapter 2, pages 25–49.
- Dechter, R. (2003b). *Constraint processing*. Elsevier Morgan Kaufmann, San Francisco, CA 94104-3205, USA.
- Demasse, S. and Beldiceanu, N. (2024). Global Constraint Catalog. <http://sofдем.github.io/gccat/>. last visited 2025-05-27.
- Li, G., Geng, E., Ye, Z., Xu, Y., Lin, J., and Pang, Y. (2018). Indoor positioning algorithm based on the improved rssi distance model. *Sensors*, 18:2820.
- Löffler, S., Becker, I., Bücker, C., and Hofstedt, P. (2023). Enhanced optimal beacon placement for indoor positioning: A set variable based constraint programming approach. In Gini, G., Nijmeijer, H., and Filev, D. P., editors, *Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2023, Rome, Italy, November 13-15, 2023, Volume 1*, pages 70–79. SCITEPRESS.
- Löffler, S., Becker, I., and Hofstedt, P. (2024). Enhancing constraint optimization problems with greedy search and clustering: A focus on the traveling salesman problem. In Rocha, A. P., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 16th International Conference on Agents and Artificial Intelligence, ICAART 2024, Volume 3, Rome, Italy, February 24-26, 2024*, pages 1170–1178. SCITEPRESS.
- Löffler, S. and Hofstedt, P. (2024). A constraint-based greedy-local-global search for the warehouse location problem. In Maglogiannis, I., Iliadis, L. S., MacIntyre, J., Avlonitis, M., and Papaleonidas, A., editors, *Artificial Intelligence Applications and Innovations - 20th IFIP WG 12.5 International Conference, AIAI 2024, Corfu, Greece, June 27-30, 2024, Proceedings, Part III*, volume 713 of *IFIP Advances in Information and Communication Technology*, pages 291–304. Springer.
- Löffler, S., Kroll, F., Becker, I., and Hofstedt, P. (2022). Optimal beacon placement for indoor positioning using constraint programming. In *19th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2022, December 5-8, 2022*, pages 1–8, Abu Dhabi, United Arab Emirates. IEEE.
- Marriott, K. (1998). *Programming with Constraints - An Introduction*. MIT Press, Cambridge.
- McGuire, J., Law, Y. W., Chahl, J., and Doğançay, K. (2021). Optimal beacon placement for self-localization using three beacon bearings. *Symmetry*, 13(1).
- Perron, L. and Furnon, V. (2023). Google LLC, Google OR-Tools, 2023. <https://developers.google.com/optimization/>, last visited 2025-06-11.
- Popyack, J. L. (2016). Gusz eiben and jim smith (eds): Introduction to evolutionary computing - springer, 2015, 299 pp, ISBN: 978-3-662-44874-8. *Genet. Program. Evolvable Mach.*, 17(2):197–199.
- Prud'homme, C. and Fages, J. (2022). Choco-solver: A java library for constraint programming. *J. Open Source Softw.*, 7(78):4708.
- Rajagopal, N., Chayapathy, S., Sinopoli, B., and Rowe, A. (2016). Beacon placement for range-based indoor localization. In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8.
- Régin, J. and Malapert, A. (2018). Parallel constraint programming. In Hamadi, Y. and Sais, L., editors, *Handbook of Parallel Constraint Reasoning*, pages 337–379. Springer.
- Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier, Amsterdam, First edition.
- Sakpere, W., Oshin, M. A., and Mlitwa, N. B. (2017). A state-of-the-art survey of indoor positioning and navigation systems and technologies. *South Afr. Comput. J.*, 29(3).
- Shanthi, D. L. and Chethan, N. (2023). Genetic algorithm based hyper-parameter tuning to improve the performance of machine learning models. *SN Comput. Sci.*, 4(2):119.
- Sharma, R. and Badarla, V. (2018). Geometrical optimization of a novel beacon placement strategy for 3d indoor localization. In *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6.
- Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Slowik, A. and Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Comput. Appl.*, 32(16):12363–12379.
- van Hoeve, W.-J. and Katriel, I. (2006). *Global Constraints*. In (Rossi et al., 2006), First edition. Chapter 6.
- Wang, H., Rajagopal, N., Rowe, A., Sinopoli, B., and Gao, J. (2019). Efficient beacon placement algorithms for time-of-flight indoor localization. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Chicago, IL, USA, November 5-8, 2019*, pages 119–128.