Branch and bound for the fixed-shape unequal area facility layout problem

Downloaded from: https://research.chalmers.se, 2026-09-05 05:16 UTC

Citation for the original published paper (version of record):
Ekstedt, F., Salman, R., Damaschke, P. (2025). Branch and bound for the fixed-shape unequal area facility layout problem. Computers and Industrial Engineering, 203.
http://dx.doi.org/10.1016/j.cie.2025.110987

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Computers & Industrial Engineering 203 (2025) 110987

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Branch and bound for the fixed-shape unequal area facility layout problem

Fredrik Ekstedt [a,*], Raad Salman [a], Peter Damaschke [a,b]

[a] Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Chalmers Science Park, SE-41288, Gothenburg, Sweden
[b] Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, SE-41296, Gothenburg, Sweden

ARTICLE INFO

Keywords:
Unequal area facility layout planning
Branch and bound
Combinatorial optimization
Mixed integer linear programming
Quadratic assignment problem

ABSTRACT

Models of the Facility Layout Problem (FLP) can be useful for guiding the placement of resources in a factory building or similar. In real-world situations, the placement of the resources is often subject to a set of complex geometrical constraints, consisting of safety distances and work areas that cannot be encroached. This can result in disjoint regions or irregular shapes that must be placed so that a set of overlapping rules are fulfilled. In this paper, we formulate this problem as placing a fixed set of arbitrary polygon unions in a plane such that the overlapping constraints are not violated and the sum of weighted distances between them is minimized. A grid-based approximation and a branch and bound algorithm to solve this variation of the problem are developed. We compare the performance with a linearized QAP formulation solved with state-of-the art MILP solvers. The algorithm shows favorable results, solving problem instances with up to 8 resources to optimality within 48 h.

1. Introduction

Planning the placement of machines and resources in a factory environment is often an iterative process where many different requirements and constraints need to be balanced such as efficient logistics, area utilization, production rate, work environment, and safety. In order to formalize and aid this process, different optimization problems have been formulated under the name Facility Layout Problem (FLP) and a variety of solution methods have been proposed. A good overview of problem formulations and solution methods is given in Pérez-Gosende et al. (2021).

We are focusing on a variant of the FLP with the following characteristics (we will use the term resource rather than facility throughout the paper when referring to our own work):

- fixed but general shapes of resources.
- free placements of resources (that is, no pre-defined structure like fixed rows or a U-shape).
- weighted distance goal function.

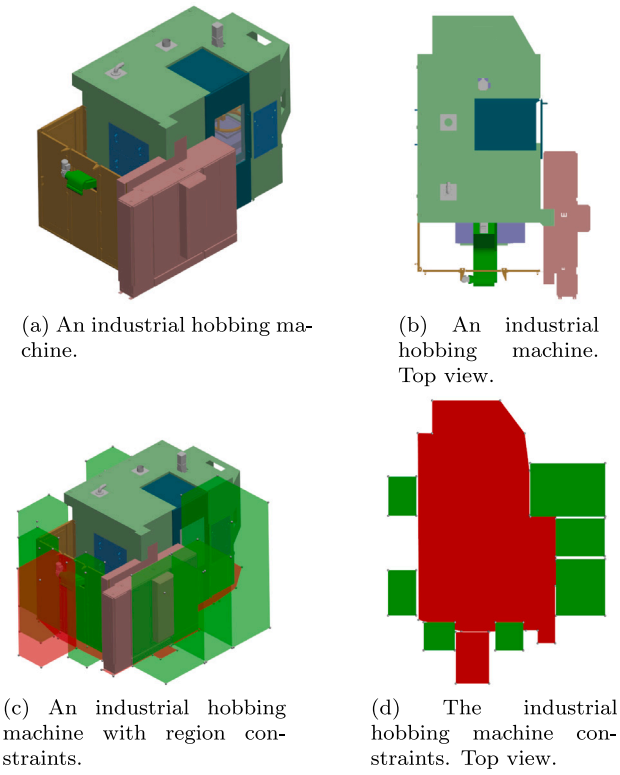
Using arbitrary shapes (rather than rectangles as usually assumed) is motivated by industrial applications where machines can have arbitrary geometric shapes physically, as well as various work and safety areas of arbitrary shapes around them. While the entire region around the machine could be coarsely approximated by bounding rectangles, there is a risk of excluding solutions that are more efficient in terms of area usage and transport lengths that may be feasible in real world scenarios. The free placement of resources allows the generation of solutions without the presupposition that a predetermined shape will be the most efficient one. The weighted distance goal function may capture various types of flow in the facility such as materials being transported between resources, process sequences, or just general movement of people around a factory.

Fig. 1 illustrates an example of an industrial hobbing machine being modeled as an irregular shape. The footprint of the hobbing machine itself may be approximated by a polygon in order to accurately represent the area where there are physical restrictions. An additional region where there are physical restrictions is where the excess material is emptied out of the machine. These regions are represented by red polygons and may never overlap with any other region that belongs to a different resource. Beside the physical restrictions, there are areas around the machine where different maintenance and work activities take place that must remain free from physical restriction in order to ensure a decent and safe work environment. However, these regions may overlap with other work areas since the activities seldom take place at the same time. These areas are represented by green polygons. For examples of feasible and infeasible placements see Fig. 2.

The proposed solution for this version of the FLP is a branch and bound search for the placement of resources on a rectangular grid. To the authors' knowledge, no exact algorithm dealing with arbitrary fixed shapes for resources has previously been published. The proposed method works well for instances of moderate sizes, and while larger

* Corresponding author.
E-mail addresses: fredrik.ekstedt@fcc.chalmers.se (F. Ekstedt), raad.salman@fcc.chalmers.se (R. Salman), ptr@chalmers.se (P. Damaschke).

https://doi.org/10.1016/j.cie.2025.110987
Received 2 May 2024; Received in revised form 8 November 2024; Accepted 16 February 2025
Available online 27 February 2025
0360-8352/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).



(a) An industrial hobbing machine.

(b) An industrial hobbing machine. Top view.

(c) An industrial hobbing machine with region constraints.

(d) The industrial hobbing machine constraints. Top view.

Fig. 1. An example of an industrial hobbing machine and its various spatial constraints. The red polygons constitute regions where no overlap may occur with any other regions belonging to a different resource because of physical restrictions. The green polygons constitute safety and work regions that may overlap with other such areas belonging to other resources.

instances are not possible to solve to optimality within reasonable times, the algorithm could still be useful in approximation schemes.

We consider the following to be the major contributions of our paper:

- Tackling the FLP with unequal, irregular shapes with an exact algorithm. Previously, exact methods have generally assumed rectangular shapes which have been incorporated into a MILP framework.
- Branching on placement only while keeping rotations undecided until a leaf is reached.
- Pruning strategies based on overlap tests and forbidden rotations.

The first point is obviously the most important one, since it extends the set of problems that may be solved. Even though exact solutions are practically limited to smaller problem instances, the techniques may be adjusted to more general search algorithms. The value of the two latter points is in that they significantly reduce the number of evaluated nodes in the search tree, and thereby facilitating the first point.

The rest of the paper is organized as follows. In Section 2 we review the previous work in this area and related ones. In Section 3 we list recurring variables. In Section 4 we describe our problem formulation in detail; first its continuous version, and subsequently the grid approximation. In Section 5, we describe the branch and bound method in detail, in particular the computation of lower bounds. In Section 6 we run some numerical tests on a selected number of test cases and discuss the results. Finally, in Section 7 we discuss possible extensions and improvements of the proposed method.

2. Literature review

The problem was first identified and described in Koopmans and Beckmann (1957) where it was formulated as a number of plants to be placed at an equal number of locations such that the cost of placement and transportation between them is minimized. This formulation is known as the Quadratic Assignment Problem (QAP) and has been used to solve problems in a range of different applications such as different varieties of facility design (Cubukcuoglu et al., 2021), manufacturing (Kaku & Rachamadugu, 1992), and optimization of integrated electronics (Emanuel et al., 2012; Tanaka et al., 2001). QAP is a very difficult but well-studied problem where the solution methods range from approximations for large-scale problems using heuristics or relaxed formulations to exact combinatorial optimization methods (Loiola et al., 2007).

An FLP may be categorized as single-floor (2D) or multi-floor (3D). A good overview of multi-floor FLP is provided by Ahmadi et al. (May 2017). We will only consider single-floor FLP in the rest of the paper.

There are special cases of the FLP that restrict the placement of objects to predetermined geometric configurations, e.g. Single Row FLP (SRFLP) and Multiple Row FLP (MRFLP) (Keller & Buscher, 2015; Tubaileh & Siam, 2017). These assumptions are too restrictive for the applications we are looking at, but they do lend some efficient algorithms to tackle large problems.

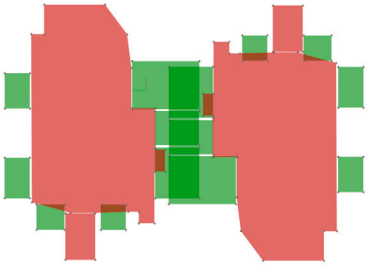
Another variant of the problem is the Unequal Area FLP (UA-FLP) where rectangular objects of unequal area are to be placed in an open space. In many cases the objects are not of fixed size but instead constraints are imposed on the objects' aspect ratios and areas. Several solutions have been offered for this problem, see for instance (Anjos & Vieira, 2016; Jankovits et al., 2011). Even though this version of the FLP focuses more on the division of spaces instead of the placement of fixed size objects, the algorithms in the cited papers have some interesting features. In a two-stage process, a general global outline is first found with a simplified model where resources are approximated by discs, and overlap conditions are replaced by penalty terms simulating repellent forces. This results in a set of center positions for the objects. In the second stage, these center positions are used as seeds for finding final positions and shapes of rectangles. This has some similarities with our approach; in particular the initial search for a general outline. However, we have much less flexibility since our shapes are fixed, and it would be much less likely to find a feasible solution in the second step.

When the shapes of the resources are fixed, the problem is called fixed-shape UA-FLP. In the literature it is almost always assumed that shapes are rectangular. See for instance (Solimanpur & Jafari, 2008) where a non-linear MILP model is formulated. The absolute value non-linearities stemming from Manhattan distances and the rectangular overlap conditions are linearized by introducing auxiliary variables. The resulting model is then solved utilizing branch and bound for a linear MILP model. Even in the case of rectangular shapes, methods involving MILP models are quite involved, and with fixed general shapes, this seems to be extremely difficult.

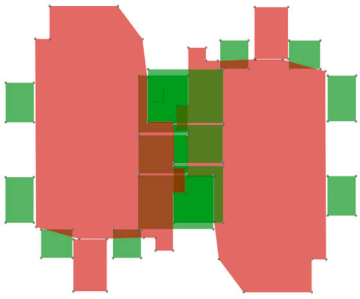
In a few cases, irregular shapes have also been considered. For instance, in Bock and Hoberg (2007) a very detailed grid-based mathematical model is proposed, taking various aspects into account, e.g. different entry and exit point to machines for material flow. In the end, the optimization problem is tackled by construction and improvement heuristics. In Huang and Wong (2017), a similar cell-based model is used, but in this case, the shapes of facilities are not fixed except for the area and the condition that the region should be connected. A large MILP is defined and solved by a standard solver.

3. List of variables

Below, we list all recurring variables of the paper. We omit variables that are only used in a single context.



(a) A feasible placement of two industrial machines. Work areas (green) overlap with each other but physically restricted areas (red) do not overlap with anything.



(b) An infeasible placement of two industrial machines. Work areas (green) overlap with physically restricted areas (red).

Fig. 2. An example of two industrial machines and different feasible and infeasible placements.

3.1. Input data

- R, R_k: Main factory region and resource regions, subsets of $\mathbb{R}^2$,
- $P_{k,l}$: polygons that combine into regions R_k,
- K: number of resources,
- $i, j \in \mathbb{Z}_+$: abstract vertices representing resources,
- V: set of vertices,
- $e = (i, j)$: edge (undirected) between vertex i and j,
- E: set of edges,
- $w_e, w_{i,j}$: edge weights (symmetric),
- W: set of edge weights,
- d: a distance metric on $\mathbb{R}^2$.

3.2. Decision variables

- p_k: placement of resource k,
- φ_k: rotation of resource k.

3.3. Method parameters

- h: grid cell size,
- Λ: rectangular grid,
- N: number of discrete rotation angles,
- $\Phi(N)$: set of discrete rotation angles.

3.4. Auxiliary variables

- $\mathrm{d}_{\min}$: distance between two resources below which overlap is guaranteed,
- $\mathrm{d}_{\max}$: distance between two resources above which no overlap is guaranteed,
- f_k: boolean vector for resource k indicating rotations for which infeasibility is guaranteed,
- LB: lower bound,
- a, A: already placed resources and set thereof,
- u, U: unplaces resources, and set thereof.

4. Problem formulation

In our version of the FLP, we assume a set of resources with fixed shapes and distance weights between pairs of resources that captures the total cost per distance unit of having these resources apart. More precisely, we are given

- a main layout region $R \subset \mathbb{R}^2$ where resources are to be placed,
- a number of regions $R_k \subset \mathbb{R}^2, k = 1, \dots, K$ describing the shapes of the resources,

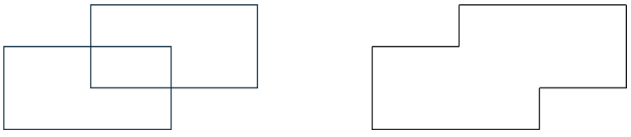


Fig. 3. Two simple polygons and its union.

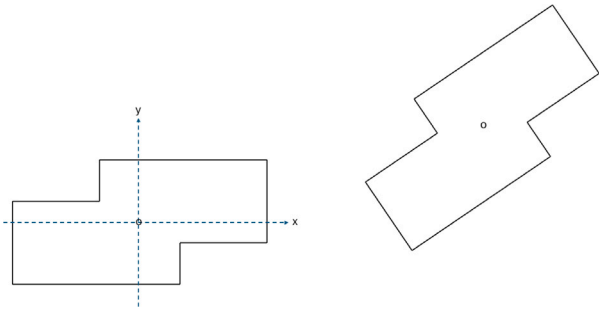


Fig. 4. A region centered at the origin (left), and rotated around its reference point (o) and then translated (right).

- an undirected weighted graph (V, E, W) with $V = \{1, \dots, K\}$,
- a distance measure $\mathrm{d} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$.

The regions are assumed to be of the form

$$R_k = \bigcup_{l=1}^{L_k} P_{k,l},$$

where each $P_{k,l}$ is (the interior of) a simple polygon (closed with no intersecting edges). The main layout region is defined in the same way. This enables great flexibility and should include all shapes that could be considered in practice, while also enabling fast intersection tests between region pairs. In Fig. 3, we show a very simple example of a region consisting of two rectangles. Note that it could be represented by a single simple polygon, but it might be worthwhile to keep the representation as a union of polygons. For instance, overlap tests for rectangles are significantly faster than for general.

The regions are defined in local coordinate systems whose origins are assumed to be the reference points for computing distances between resources. The metric is assumed to be the Euclidean or the Manhattan distance, the latter being natural in many layouts where objects as well as walking paths are mostly axis-parallel. The weights $W = \{w_{ij} \geq 0 : (i, j) \in E\}$ specify costs per distance unit between reference points in the chosen metric for having resources apart.

4.1. Continuous version

With the inputs specified above, the continuous optimization problem may now be formulated as

Definition 1. Find K translation-rotation pairs $(p_k, \varphi_k) \in \mathbb{R}^2 \times [0, 2\pi)$ such that the following objective function is minimized

$$\sum_{(k,l)\in E} w_{k,l}\, \mathrm{d}(p_k, p_l),$$

subject to the following constraints:

- $R_k(p_k, \varphi_k) \subset R$ for $k = 1, \dots, K$,
- $R_k(p_k, \varphi_k) \cap R_l(p_l, \varphi_l) = \emptyset$ for all $1 \le k < l \le K$,

where $R_k(p_k, \varphi_k)$ denotes the region R_k rotated clockwise an angle φ_k and then translated by p_k (see Fig. 4).

This formulation is implicitly equivalent to translations followed by rotations around the same points that are used for calculating distances between objects. This means that the rotations do not affect the objective function, something that we will exploit. Since the reference point for computing distances often may be closer to the boundary of the region, using e.g. the center of mass for the region to rotate around could seem more natural. However, it does not alter the set of possible solutions as long as the translations variables are continuous.

The overlap test $R_k(p_k, \varphi_k) \cap R_l(p_l, \varphi_l) = \emptyset$ may be generalized in various ways. In particular, when the regions are defined as unions of polygons, different polygons may have different overlap requirements. One particular case that we have been working with is that some polygons have *hard* constraints, typically representing machines themselves. These may not overlap with any other polygons. There are also polygons with *soft* constraints, typically representing areas around a machine that may not be blocked by other machines. These polygons may then overlap within themselves, but not with polygons with hard constraints.

Due to the overlap restrictions, the feasible search space is non-convex, in many cases even disconnected, and the number of local optima grows exponentially with the number of resources. This makes the problem intractable for classical gradient-based methods, and some kind of combinatorial search must be used. For problem instances of moderate size, discretizing the resource placement to a rectangular grid may be a viable option. After finding the optimal grid solution, a gradient-based search may be employed to find a local optimum, which should be the global optimum for a fine enough grid size.

4.2. Discrete version

In light of the discussion in the previous section, we define a rectangular grid

$$\Lambda(p_0, h) = \{p \in \mathbb{R}^2 : p = p_0 + h * v; v \in \mathbb{Z}^2\},$$

where $p_0 \in \mathbb{Z}^2$ is the *grid center*, and $h \in \mathbb{R}_+$ is the *grid size*. As we will see later, the grid size is quite a crucial parameter, since it balances requirements for accuracy versus computational speed. We will mostly just write Λ below for brevity.

The discrete version of the problem is achieved by simply restricting the placement variables in the continuous version in Definition 1 to Λ. We further restrict the rotations to a finite set of values $\Phi(N) = \{\varphi = n2\pi/N; n = 0, \dots, N-1.\}$. The number of rotations could be quite moderate, 4 or 8 should be sufficient in most cases.

4.3. Discretization error

Obviously, an exact solution of the discrete version will be an approximate solution of the continuous version, and one may ask how large the deviation is. Generally, there is no simple and complete answer to this question, but some relevant points can be made. To do that, we will assume that the continuous problem has a unique global optimum.

In a tight scenario, too low spatial and/or rotational resolution (large h and small N) may lead to there being no feasible discrete solution in the catchment basin of the global optimum (the set of points for which the continuous gradient path leads to the optimum), or even at all. It is not possible to say something more precise and general about this, neither in terms of how much the discrete optimum will differ from the global continuous one, nor how fine the resolution must be to avoid missing it altogether.

If we assume that the resolution is fine enough, the catchment basin will for each resource contain the four grid points that surrounds the position of the resource of the global optimum. We further assume that the angular resolution is fine enough so that there are sets of feasible discrete rotations for all these grid points. Then the deviation in position for each resource will be at most $\sqrt{2}h$ in the Euclidian metric and $2h$ in the Manhattan metric. Again, from such a position a simple gradient search may be employed if desired.

4.4. A MILP model

The discrete formulation may be seen as a variant of the Quadratic Assignment Problem (QAP) which is notoriously difficult to solve. Even for small instances, finding the exact optimal solution takes a very long time and there are no polynomial approximation algorithms (Sahni & Gonzalez, July 1976). The main differences from the standard QAP is that the number of resources is typically significantly lower than the number of locations while they are equal in the standard QAP, and there are constraints that forbid the occurrence of certain pairs of resource placements due to overlap between their regions whereas in the standard QAP the only restriction is usually that no two resources may be placed at the same location.

For some fixed p_0, h and N, let $L = \{l = (x, \varphi) : x \in \Lambda(p_0, h), \varphi \in \Phi(N)\}$ be the set of possible locations for the resources, and assume an arbitrary order for these locations $j = 1, \dots, |\Phi||\Lambda|$. Since the edge weights w_{ik} are assumed to be symmetric, it is sufficient to include the resource pairs (i, k) when $k > i$ in the problem. Furthermore, no two resources can be placed in the exact same location. We introduce the sets $V_2 = \{(i, k) \in V \times V : k > i\}$ and $L_2 = \{(j, l) \in L \times L : j \ne l\}$ as the sets of resource pairs and location pairs that need to be included in the problem. While the objective function normally is quadratic in models of the QAP, we present a linearized version.

$$\min_{y} \quad \sum_{(i,k)\in V_2} \sum_{(j,l)\in L_2} y_{ijkl} d_{jl} w_{ik} \tag{1a}$$

s.t.

$$\sum_{j=1}^{|L|} x_{ij} = 1, \qquad \forall i \in V \tag{1b}$$

$$y_{ijkl} \le f_{ijkl}, \qquad \forall (i,k) \in V_2, \forall (j,l) \in L_2 \tag{1c}$$

$$y_{ijkl} \le x_{ij}, \qquad \forall (i,k) \in V_2, \forall (j,l) \in L_2 \tag{1d}$$

$$y_{ijkl} \le x_{kl}, \qquad \forall (i,k) \in V_2, \forall (j,l) \in L_2 \tag{1e}$$

$$y_{ijkl} \ge x_{ij} + x_{kl} - 1, \qquad \forall (i,k) \in V_2, \forall (j,l) \in L_2 \tag{1f}$$

$$0 \le y_{ijkl} \le 1, \qquad \forall (i,k) \in V_2, \forall (j,l) \in L_2 \tag{1g}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in V, \forall j \in L \tag{1h}$$

The binary variables x_{ij} determine if resource i is placed at location j. In order to linearize the objective function, we have substituted the variables y_{ijkl} for $x_{ij}x_{kl}$. The parameters d_{jl} represent the distance between locations j and l, while the parameters w_{ik} represent the weight associated with resources i and k. The constraints (1b) ensure that only one resource is placed at each location. The constraints (1c) ensure that any two resources do not overlap. The binary parameters

f_{ijkl} tell us if resource i can be placed at location j at the same time as resource k is at location l without overlap. These parameters can be precomputed and the variables that correspond to $f_{ijkl} = 0$ may be removed from the problem altogether in order to reduce it. The constraints (1d)–(1f) ensure that $y_{ijkl} = x_{ij}x_{kl}$.

However, since the number of linear variables becomes $\mathcal{O}(|V_2||L_2|)$ and binary variables $\mathcal{O}(|V||L|)$, the problem becomes difficult to solve for generic MILP solvers even when providing small problem instances.

5. Branch and bound

We propose a simple branch and bound search based on the following basic principles:

- using best-first search,
- branching on the grid placement variables p_k,
- using a fixed ordering of the resources to be placed,
- leaving rotations undecided during branching,
- when reaching a leaf, deciding if a feasible set of rotations exist,
- searching for infeasible sub-solutions and pruning when finding those.

Using best-first search is a compromise between quickly reaching valid solutions which could provide upper bounds, and that found solutions have low total scores. It is also less memory consuming than e.g. breadth-first search. All of the remaining points will be discussed in more detail below.

Algorithm 1 Branch and bound

```
S := ∅                                          ▷ Set of complete solutions
Q := ∅          ▷ Priority queue based on lower bound of members
globalLB := computeLowerBound(emptySolution)
globalUB := ∞
Initialize(Q)          ▷ Add first resource at all possible grid points
while Q ≠ ∅ do
    q := Q.pop()
    if q.lowerBound > globalUB then
        B := branch(q)                              ▷ see Sections 5.1–5.3
        for b ∈ B do
            if isLeaf(b) and b.upperBound < globalUB and hasFeasibleRotations(b) then
                S.push(b)
                B.delete(b)
                globalUB := b.upperBound
            else
                b.lowerBound := computeLowerBound(b)        ▷ see Section 5.4
                Q.push(b)
            end if
        end for
    end if
end while
```

5.1. Branching

We are branching on the grid placement variables p_k, using a fixed ordering of resource placements, i.e. a permutation σ of $\{1, \dots, K\}$. That means that, at a node in the branching tree with depth n, we have decided placement variables $p_{\sigma(1)}, \dots, p_{\sigma(n)}$. We then branch on $p_{\sigma(n+1)}$, the placement of the next resource in the given ordering.

Some of these placements may directly be discarded since they lead to overlap with an already placed resource, or the newly placed resource is not inside the overall region R. This can be done even though we have not decided yet on resource rotations, using the analysis in Section 5.2. We further do a pruning based on a deeper analysis of forbidden rotations, as we will describe in Section 5.3. Finally, if the position has not been discarded yet, a lower bound is computed according to Section 5.4. If the lower bound is higher than the overall upper bound (best solution found so far), this branch is also discarded. Otherwise, it is placed in the queue of branches to be further processed.

5.2. Exploring resource overlaps

Since the grid size must be chosen smaller than the resource regions, there will seldom be actual solutions with regions placed at neighboring grid points. This needs to be taken into account somehow to achieve tighter bounds. One way to incorporate this information is to precompute, for each pair of resources k and l, minimum and maximum distances $\mathrm{d}_{\min}$ and $\mathrm{d}_{\max}$ such that

- there is overlap irrespective of rotations when $\mathrm{d}(p_k, p_l) < \mathrm{d}_{\min}(k, l)$,
- there is no overlap irrespective of rotations when $\mathrm{d}(p_k, p_l) > \mathrm{d}_{\max}(k, l)$.

Even though only the minimum distances will be used for bounds as described below, both may be calculated with the same effort, and may be used to avoid overlap testing when $\mathrm{d}(p_k, p_l) < \mathrm{d}_{\min}(k, l)$ or $\mathrm{d}(p_k, p_l) > \mathrm{d}_{\max}(k, l)$.

5.3. Rotations

As pointed out above, the rotations do not affect the objective function, but only the feasibility of solutions. Therefore, we postpone the decision of the rotation variables until we reach a leaf in the branching tree. Given all region placements at a leaf, we do a depth-first search of the corresponding rotation decision tree to see if there is any valid solution. If so, this solution is saved as the currently best, and the upper bound is updated. When the optimal feasible placement of regions is finally found, an enumeration of all solutions w.r.t. rotations may be computed if desired.

Even though we do not explore all combinations of rotations during the search, it might still be worthwhile to do a partial search for impossible combinations. It is fairly cheap to do pairwise tests for two placed regions, and each such test will give an $N \times N$ Boolean matrix.

To elaborate, for each placed resource k we define a Boolean vector $f_k \in \mathbb{B}^N$, which keeps record of known forbidden (infeasible) rotations. Note that $f_k(n) = 1$ means that there is no feasible solution using rotation n for resource k, while $f_k(n) = 0$ means that we cannot (yet) rule out that there might be a feasible solution.

When a new resource l is placed in a partial solution, its Boolean infeasibility vector f_l is initiated by checking for which rotations the placement region is inside R. Then pairwise tests are made with already placed resources. In these pairwise tests, a Boolean matrix $O_{k,l}$ is computed for placed resources k, l, where $O_{k,l}(m, n)$ is set to 1 when rotation m of resource k is overlapping with rotation n of resource l. Any zero entry in f_k is updated by checking if the corresponding rotation has any corresponding (potentially) feasible rotation of resource l with no overlap:

$$f_k(m) = \forall_{n=1}^{N} \left(\neg f_l(n) \vee O_{k,l}(m, n) \right).$$

The infeasibility vector f_l is updated correspondingly. As soon as a resource has an infeasibility vector with all ones, we may prune this part of the search tree.

We have identified three strategies for making pairwise tests when a new resource is placed:

1. Just test once against previously placed regions.
2. If any previously placed region's feasibility vector was updated during (1), redo pairwise tests between all previously placed resources.
3. Keep doing pairwise tests iteratively until no feasibility vectors are changed anymore.

We chose to use (2) since it gave a moderate increase in the number of pruned branches without being significantly more costly.

Besides pruning of the search tree, there is another benefit from the above analysis. When a leaf is reached in the branch and bound search tree, we have a complete placement of all resources. However, we do not know for sure that is possible to choose rotations so that we get a feasible solution. It remains to search a corresponding decision tree for rotations, and the feasibility vectors may be used to prune this search tree.

5.4. Computing lower bounds

The easiest bound is to assume that all distances between regions are h, the smallest distance on the grid. In that case the objective function is simply h multiplied with the sum of all edge weights. This will be referred to as the *trivial* lower bound. Obviously we need to do better. To do that, we start with a simple but central definition and observation.

Definition 2. A partitioning Π of an edge-weighted graph $G = (V, E)$ is a family of edge-weighted graphs (V_k, E_k), $V_k \subseteq V$, such that the weight of every edge $e \in E$ equals the sum of the weights that e has in all graphs in Π.

For any partitioning Π of G, the sum of the costs of optimal grid layouts of all graphs (V_k, E_k) in Π is a lower bound on the cost of an optimal grid layout of G. We will refer to this as the *additive property* of graph partitionings.

The term grid layout simply refers to the assignment of graph vertices to grid points. An optimal grid layout is a grid layout that minimizes the sum in Definition 1. It is easy to verify that the additive property still holds with the addition of the overlap constraints and requirements of being inside the main region in Definition 1.

To utilize the above ideas, we start by partitioning the vertex set V set into placed (assigned) vertices A and unplaced vertices U. We then make a first edge partitioning into edges between already placed vertices, edges between placed and unplaced vertices (inter edges), and edges between unplaced vertices (intra edges). According to the additive property, lower bounds may be computed for these three sets of edges separately, and then added to form a total bound. Note that the intra bounds only depends on the resources that are left to be placed, and therefore will be identical for all nodes of the same depth in the branching tree. Thus, we may spend more computational efforts on these bounds.

For edges between already placed vertices, we can simply use the actual distances to get an exact bound. For the other two sets, it gets more involved, and we will treat them separately below. In general, it is all about finding further subgraphs to force some edges to have longer distances than h. To begin with, the trivial bound is easily improved by using minimum distances instead of h as the edge lengths. For a subset of edges $E' \subseteq E$, we define the modified trivial bound by

$$\mathrm{LB}_{\mathrm{modtriv}}(E') = \sum_{(k,l)\in E'} w_{k,l}\, \mathrm{d}_{\min}(k,l).$$

The goal below is to try to beat this bound for various subgraphs.

5.4.1. Inter edges

In this case, we try to place each single unplaced vertex independently. This corresponds to a partitioning of inter edges into subsets $E_{A,u} = \{(a,u) : a \in A\}$, one for each $u \in U$.

We are thus faced with the problem of finding the optimal placement of a single vertex given a set of already placed vertices. If we relax the requirement of placements on the grid and feasibility constraints, this problem has actually been studied to some extent. For the Euclidean distance, this is known as the Weber problem (Tellier, 1972), which is very difficult to solve for $|A| > 3$. For the Manhattan distance, the problem may be separated into two independent 1D problems which easily can be solved. The latter bound may be turned into a Euclidean bound by a division by $\sqrt{2}$. This bound might not be particularly sharp though. If the vertices in A are tightly placed, even the Manhattan bound could be untight, since it tends to place u on the same grid position as an already placed vertex. But the above techniques do not generalize well to take minimum distances into account anyway, so we will not be employing them.

To take minimum distances into account, we opted for a brute force approach and to do an exhaustive search of the optimal grid point to place each unplaced resource independently, subject to the minimum distance constraints. For each $a \in A$ and $u \in U$, the "forbidden region" for placing u with respect to a is given by $B(a,u) = \{v \in \Lambda : \mathrm{d}(v, p_a) < \mathrm{d}_{\min}(u,a)\}$. For a certain $u \in U$, it is sufficient to restrict the search to a certain rectangular "bounding box" subgrid $\Lambda_{\mathrm{BB}}(A,u)$ having the following properties

- $\Lambda_{\mathrm{BB}}(A,u) = \{v_0 + h * v; v \in \{0,\dots,N_x\} \times \{0,\dots,N_y\}\}$ for some $v_0 \in \Lambda$ and $N_x, N_y \in \mathbb{Z}_+$, i.e. it has a rectangular shape,
- $B(u,a) \subset \Lambda_{\mathrm{BB}}(A,u)$ for all $a \in A$, i.e. it contains all forbidden region grid points,
- there are no vertices from any $B(u,a)$ on the boundary of $\Lambda_{\mathrm{BB}}(A,u)$, i.e., points in $\Lambda_{\mathrm{BB}}(A,u)$ with neighbors both in $\Lambda_{\mathrm{BB}}(A,u)$ and $\Lambda \setminus \Lambda_{\mathrm{BB}}(A,u)$.

To see that the optimal placement on Λ for $u \in U$ is always found in $\Lambda_{\mathrm{BB}}(A,u)$, consider any $g \in \Lambda \setminus \Lambda_{\mathrm{BB}}(A,u)$. There is a boundary point g' which is closer to all points in A in one coordinate direction and at least equally close in the other. This point may be found by moving g either along the x-axis or y-axis until we reach a boundary point. If we do not reach a boundary point, we may instead choose the nearest corner point of $\Lambda_{\mathrm{BB}}(A,u)$. Thus for any l^p metric, the distance to all placed vertices will be smaller for g', and hence also the objective function. Since the boundary points are not in any $B(a,u)$, g' is feasible and g cannot be the optimum.

5.4.2. Intra edges

For edges between unplaced vertices, we try to identify dense subgraphs where we can force some edges to have lengths bigger than h. Unfortunately, this is a very difficult problem for $|U| > 3$ when taking minimum distances into account. For 3-cliques we may sometimes improve on the modified trivial bound in the following way. We start by relaxing the placements to $\mathbb{R}^2$. Let d_0 be the largest minimum distance for an edge in a 3-clique. Denote the two other minimum distances by d_1 and d_2 and their corresponding edge weights by w_1 and w_2. It is easy to show that if $d_0 \le d_1 + d_2$, the modified trivial bound cannot be improved. If $d_+ := d_0 - d_1 - d_2 > 0$ however, at least d_+ must be added to the length of one of the other edges, and $d_+ \min(w_1, w_2)$ may be added to the trivial bound giving an improved bound

$$\mathrm{LB}_{\mathrm{mod3clique}} = \mathrm{LB}_{\mathrm{modtriv}} + d_+ \min(w_1, w_2).$$

For larger subgraphs there are no easy ways to deduce significant improvements over the modified trivial bound. For a subgraph with 4 nodes, another alternative is to recursively solve the corresponding subproblem exactly. This may be feasible since we only have do it a limited number of times. For larger subgraphs, this becomes unfeasible though. We instead opt for partitioning them into subgraphs of size 3 and 4, referred to below as 3-subgraphs and 4-subgraphs. This is done in an iterative fashion using the steps below described for a general sub-graph G'.

1. Initiate the bound $\mathrm{LB}_{\mathrm{intra}}(G') := 0$ and set $G_{\mathrm{rem}} := G'$.
2. If $|G_{\mathrm{rem}}| < 3$, go to 6.
3. If $|G_{\mathrm{rem}}| = 3$, add $\mathrm{LB}_{\mathrm{mod3clique}}(G_{\mathrm{rem}})$ to $\mathrm{LB}_{\mathrm{intra}}(G')$ and go to 6.
4. If $|G_{\mathrm{rem}}| = 4$, add $z^*(G_{\mathrm{rem}})$ to $\mathrm{LB}_{\mathrm{intra}}(G')$ by solving the G_{rem} subproblem exactly, and go to 6.
5. If $|G_{\mathrm{rem}}| > 4$

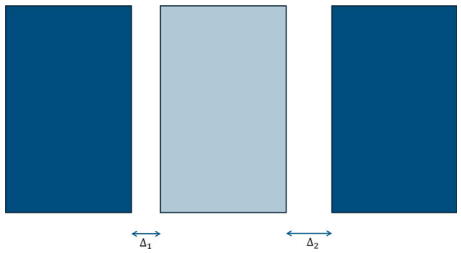


Fig. 5. Spacing when placing a resource between two identically shaped resources in a row.

(a) find the 4-subgraph G_4 by choosing the 4 first resources in the branching order that belong to G_{rem}, and compute $z^*(G_4)$ by solving the G_4 subproblem exactly,
(b) find the 3-subgraph G_3 in the same fashion as G_4 and compute its bound $\text{LB}_{\text{mod3clique}}(G_3)$,
(c) let $G_{3/4}$ be the choice of G_3 or G_4 which has the largest gain of its bound compared to the corresponding trivial bound,
(d) remove $G_{3/4}$ from G_{rem} and add its corresponding bound to $\text{LB}_{\text{intra}}(G')$,
(e) go to 2.

6. add trivial bounds for all remaining untreated edges in G'.

5.5. Branching order

Unfortunately, inter and intra edges bounds come with contradicting requirements for the best branching order. For inter edge bounds, it is better to have resources with many relations early in the branching. For intra edges bound, it is better to have many dense subgraphs left to the end. After some preliminary tests, we opted for prioritizing inter edges bounds, and choose a branching order guided by having the highest edge weight sum.

5.6. Grid size selection

As discussed in Section 4.3, the grid size h is a crucial parameter. Not so much for having the resources placed about h away from their optimal positions; this may be corrected by a gradient search if at all relevant. The main issue is to be able to find the best general "topological" arrangement of resources, or even any arrangement at all. Consider a very simple case as in Fig. 5 below where we try to place a rectangle between two identical rectangles. If $h < \Delta_1 + \Delta_2$ we are guaranteed that placing the middle resource on a grid is possible given the placements of the two first resources.

This implies that h should be chosen as a portion of the smallest dimension of any resource, or even as the smallest dimension of any polygon. We define the smallest dimension of any subset of the plan,e $\text{mindim}(S), S \subset \mathbb{R}^2$, as the minimum dimension of the bounding rectangle of S. We then suggest that the grid size is chosen as

$$h = \alpha \min_{k,l}\{\text{mindim}(P_{k,l})\},$$

for some $\alpha > 0$ which we name the *relative grid size*. For ordinary rectangular regions R_k, $\alpha = 0.4$ seems to be a good general compromise. For more complex resource regions, there was unfortunately no simple general choice, but in general, a larger value was used.

6. Numerical experiments and results

The proposed branch and bound algorithm was tested using six different benchmark problem instances. The instances are inspired by real-world problems of layout planning for manual sub-assembly stations and are comprised of shelves, toolboxes, assembly fixtures, and a TV-screen. For detailed information about these problem instances see Appendix. While these particular problem instances only use rectangular shapes for the resource regions, the algorithm itself does not make any assumptions regarding the regularity of the regions. The only thing that differs when using more complex shapes is that the numerical overlap tests become more computationally demanding and in the end the goal is to reduce the number of these tests much as possible. These benchmark instances serve to compare the efficiency of different branching and bounding methods. To illustrate the performance of the algorithm on irregular shapes and to compare the solution quality compared to using simpler constraint modeling we test the algorithm on cases where a set of industrial hobbing machines are to be placed on a factory floor using either detailed modeling of the constraints or a simple bounding box.

The algorithms were set to terminate after 48 h and return the best solution. The experiments were run on a machine with a AMD Ryzen 9 3900X 3.79 GHz processor and 32 GB of RAM.

6.1. Benchmark instances

Table 1 shows detailed results from running the benchmark instances. There is a big difference between the optimality gaps at the root and it seems like the bounding methods perform much worse for the cases **C.5**, **C.7b**, and **C.8**. Since none of the resource positions are fixed at the root node, the lower bound is completely determined by the intra edges bound which tries to predict the placement of the unplaced resources relative to each other. As mentioned in Section 5.5, we have experimented with branching orders that benefit the intra edges bound but saw that those improvements were generally offset by reduced performance of the inter edges bound which in the end led to higher computational times in general.

Table 1
Results from numerical experiments using best-first search and the proposed bounding method.

Case	K	|L|	Lower bound	Upper bound	Gap (%)	CPU (s)
C.4	4	576	465.2	497.8	6.6	1
C.5	5	576	881.8	1078.9	18.3	11
C.6	6	576	905.8	1004.2	9.8	14
C.7a	7	576	933.2	1025.8	9.0	58
C.7b	7	576	1836.0	2833.5	35.2	2599
C.8	8	1024	2388.0	3848.7	38.0	79 405

To illustrate how the algorithm scales with the problem instance size, we formulate the performance index $\frac{|L|^K}{t}$ where $|L|^K$ is the number of potential solutions for a problem instance with $|L|$ locations and K resources and t is the computational time measured in seconds. So a higher value of this index means a higher rate of solutions processed per time unit. Since the problem instance size increases exponentially with the number of locations and resources, the index also needs to increase at least exponentially for the algorithm to scale decently. In Fig. 6 this trend seems to be mostly true for the benchmark instances with one exception being **C.7b** where the algorithm performs worse than for **C.7a**. In this case, the graph is denser than in **C.7a** and bounding methods seem to have trouble with capturing the characteristics of the optimal solution. Fig. 7 illustrates the density of the graphs and the optimal solutions of **C.7a** and **C.7b** where the performance of the bounding method and the algorithm differs even though the instance sizes are the same.

6.1.1. Bounding method comparison

The algorithm was also run using two alternative bounding methods, the Gilmore-Lawler bounding (GLB) method (Gilmore, 1962) and the trivial bounding method that is outlined in Section 5.4. While there are more sophisticated methods that can provide very tight bounds for

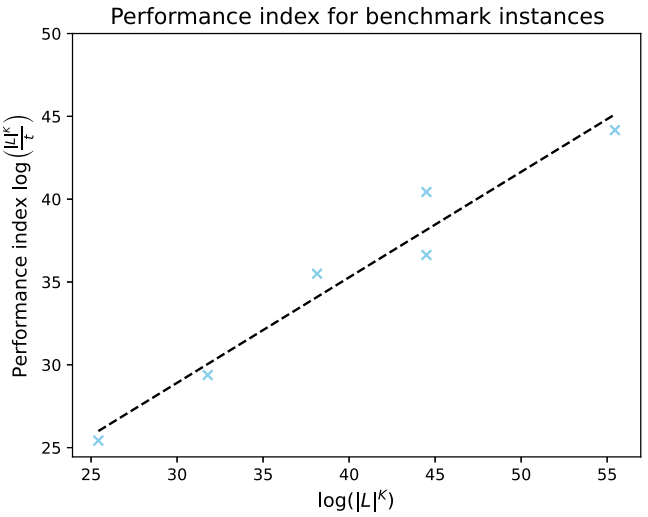
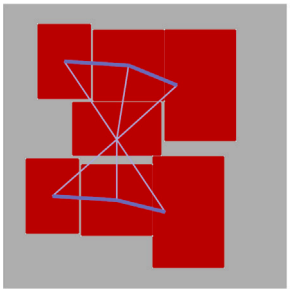
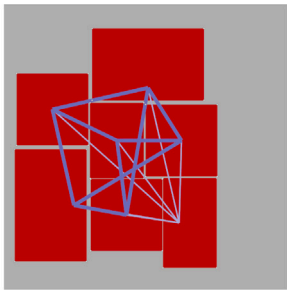


Fig. 6. Algorithm performance index for the benchmark instances. Higher is better. The equation of the line is $\log(y) \approx 0.64 \log(x) + 9.81$.



(a) Optimal solution for problem instance **C.7a**.



(b) Optimal solution for problem instance **C.7b**.

Fig. 7. Optimal solutions for two problem instances with the same size but different graph density. Bounding methods perform better on less dense graphs.

a wide range of QAP instances (Adams et al., 2007; Burer & Vandenbussche, 2006; Hahn et al., 2012), they can be very computationally demanding and in contrast, the GLB method is widely considered the fastest and easiest to implement. We utilize the Hungarian method to solve the linear assignment problems in the GLB. To tighten the bound further, we assign a very high cost to assignments that correspond to placements that do not exceed the minimum distance value for a pair of resources and therefore, would be considered infeasible in the original problem. It is clear that the GLB method was not meant for the problem that we are tackling in this paper but it is the best comparison we could find in the literature.

The results are shown in Fig. 8. The GLB method results in slower solving times and only the three smallest instances were solved to optimality and verified within the prescribed time limit. For the instance **C.7a** the optimal solution was found but not verified within the time limit. Because of the large number of locations, the matrices for the linear assignment problems become quite large and therefore, solving them more computationally demanding than our proposed method. The trivial bounding method could not verify the solution found for the largest problem instance **C.8** and also consistently performed worse that the proposed bounding method. In general, this is due to the limited geometrical information that is incorporated in the trivial bounding method.

The quality of the lower bounds are illustrated by just comparing the optimality gap at the root node in Fig. 9. While this gives limited information (bounds can quickly be strengthened as deeper levels of the

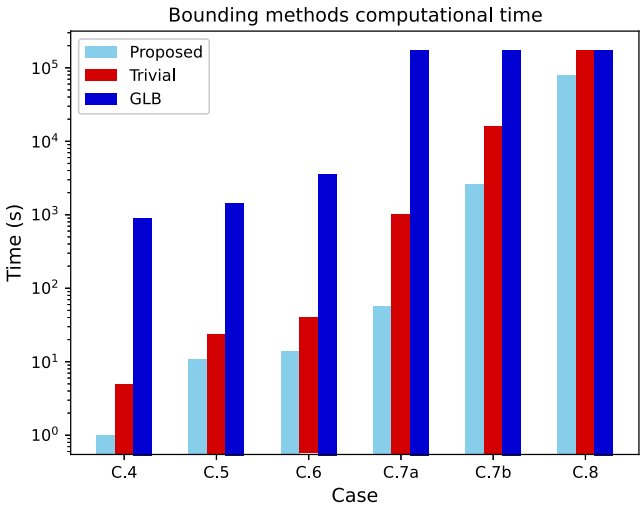


Fig. 8. Comparison of bounding methods computational time. Note that the time limit of 48 h is reached without verifying optimality for "Trivial" in case **C.8** and for "GLB" in cases **C.7a**, **C.7b**, and **C.8**.

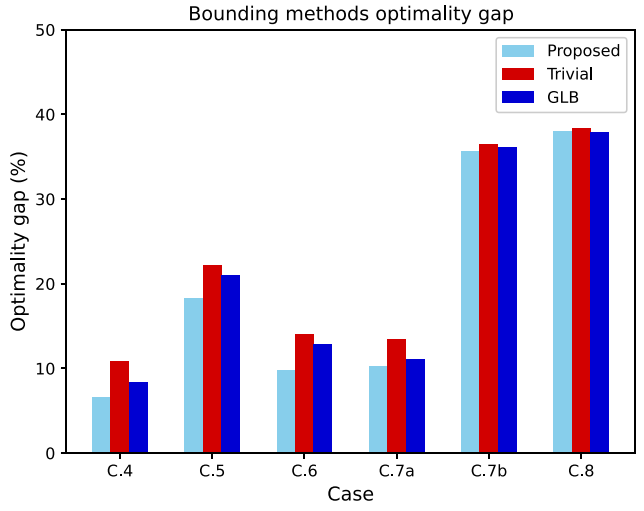


Fig. 9. Comparison of bounding methods optimality gap. Note that the time limit of 48 h is reached without verifying optimality for "Trivial" in case **C.8** and for "GLB" in cases **C.7a**, **C.7b**, and **C.8**.

search tree are explored), it gives some indication of the potential merits of the different bounding methods. The trivial bound is consistently the weakest since it allows all resources to be placed at the smallest possible distance relative to each other. Furthermore, the trivial bound is not able to extract any further geometrical information deeper in the search tree when some resource positions have been fixed. It is however very fast and simple to compute once all minimum distances $\mathrm{d}_{\min}(k, l)$ have been determined. The GLB bound is consistently weaker at the root node compared to the proposed bounding method except for **C.8** where it is marginally stronger. However, as we have reasoned before, the computational time is much worse with not much pay-off.

6.1.2. MILP solver

The MILP model outlined in Section 4.4 was run on the same problem instances with the same time limit using the COIN-OR CBC 2.9.8 solver through the C-interface. The f_{ijkl} parameters were precomputed before running the solver. None of the problem instances were solved within the prescribed time limit as the number of variables becomes very large. As an example, for the smallest problem instance

the number of y_{ijkl} variables was 523512 after excluding unnecessary ones due to symmetries and infeasible pairwise positions.

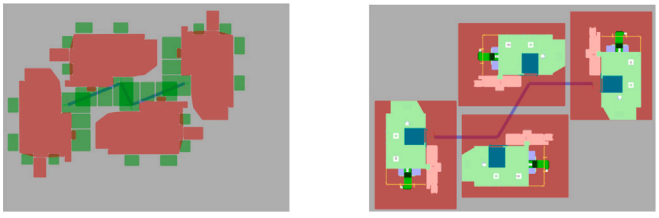
6.2. Instances with irregular shapes

Using the proposed bounding method we test the algorithm on instances where modeling resources with irregular geometries and more detailed spatial constraint modeling could be beneficial. In this scenario, a number of industrial hobbing machines are to placed on a factory floor. Cases named **HM.x** have detailed constraints while **HM.x.BB** use a bounding box with some margins to model the occupied area around the machine.

Table 2
Results from numerical experiments on problem instances with industrial hobbing machines.

Case	K	|L|	Lower bound	Upper bound	Gap (%)	CPU (s)
HM.4	4	1872	339.8	775.2	48.4	1264
HM.4.BB	4	1872	894.0	1062.7	15.9	13
HM.5	5	1872	533.1	1175.0	54.6	28 205
HM.5.BB	5	1872	1192.0	1499.9	20.5	147

In Table 2 we can clearly see that the algorithm performs better when we utilize a bounding box as the machine region and this is due to mainly two factors. One being that the overlap tests become more computationally demanding to verify when modeling the constraints with several regions while overlap becomes very easy to check when having a singular rectangular region per machine. Secondly, the lower bounding method performs worse in the case with the more detailed constraint modeling since now pairwise machines can be placed very close to each other, resulting in small minimum distances. In a real layout however, all machines cannot be placed that close to each other due to the overlapping constraints.



(a) Optimal solution for problem instance **HM.4**.

(b) Optimal solution for problem instance **HM.4.BB**.

Fig. 10. Optimal solutions when using detailed constraint modeling (a) and a simple bounding box (b). The bounding box is defined such that no work area or physically restricted area can overlap with each other.

However, while the algorithm unfortunately cannot extract as much geometrical information in these cases, we can see that the optimal solution is improved by 27% in **HM.4** compared to **HM.4.BB** and by almost 22% in **HM.5** compared to **HM.5.BB**. By utilizing the allowed overlap when having more detailed constraints the distance between machines can be reduced and therefore, the area can also be used more efficiently as can be seen in Fig. 10.

7. Conclusions and further research

In this paper a branch and bound algorithm that can solve instances of the FLP that are under a wide range of geometric constraints is presented. Since the layout planning process is generally not time sensitive the proposed algorithm is able to produce optimal solutions to industrially relevant cases with up to 8 resources within a reasonable time limit. Problem instances of this size could model a station within a factory or a smaller area on the factory floor. For larger cases such as production lines, large factory logistics areas or complete factory floors the performance of the proposed solution method is limited and a different approach should be taken. Nonetheless, we give some suggestions of how the proposed solution techniques could be expanded and utilized in a broader way.

In general, the current lower bounds are not strong enough and the computational time is dominated by overlap tests between resources so naturally it would be beneficial to spend more time on finding stronger bounds in order to prune more branches in the search tree. For the inter edges lower bounds, we could try various methods to find optimal placements of more than one new unplaced resource at a time. For intra edges bounds, one could look into decomposing the problem into subgraphs with more than 4 nodes as discussed previously. One approach could be to utilize a column generation scheme in order to generate promising subgraphs of a maximum size.

There is a possibility to improve the computational efficiency of the proposed algorithm by utilizing a different branching strategy. Some options include depth-first search which could be more efficient at finding upper bounds quickly, the currently chosen best-first strategy which may lead the algorithm to more promising branches, and a mix where depth-first is utilized in the beginning and then switching to best-first. There is also a choice between prioritizing pruning based on the feasibility of the partial solution (i.e. verifying that no overlaps have occurred) and pruning based on the strength of the lower bound which could further optimize the algorithm implementation.

In this proposed algorithm a uniform grid is used in order to discretize the available space. While many of the ideas proposed in this paper are reliant on the uniformity of the grid, it might be of interest to look into other means of discretizing the search space in some adaptive way in order to reduce the number of explored locations.

While improving an exact algorithm for the FLP with irregular shapes is both theoretically and practically interesting, probably only smaller problem instances will be solvable within reasonable times. Therefore, some approximative scheme that utilizes this exact approach in some constructive way in order to generate good starting solutions for subsequent heuristic algorithms could be of interest.

CRediT authorship contribution statement

Fredrik Ekstedt: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Conceptualization. **Raad Salman:** Writing – review & editing, Writing – original draft, Validation, Software, Project administration, Methodology, Funding acquisition. **Peter Damaschke:** Writing – review & editing, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has benefited from funding from the Swedish Innovation Agency Vinnova, Sweden, as part of the AITOC research project within the ITEA3 cluster. Funding was also provided by Chalmers Production Area of Advance, Chalmers University of Technology, Göteborg, Sweden.

Appendix

The problem instances that have been tested are inspired by real world manual assembly stations. They consist of four different objects: a shelf, an assembly fixture, a toolbox, and a TV-screen. The measurements of the objects are outlined in Table A.1 below. All cases have a maximum allowed space of 8 by 8 meters except C.8 which has an allowed space of 10 by 10 m. The cases are described in more detail in the coming sections. Each section contains two tables, one with the amounts of each resource type in the problem instance, and the other describing the non-zero relationship weights between each object.

Table A.1
Measurements of resource objects.

Object	Measurements
Shelf (S)	3 × 2 metres
Assembly fixture (AF)	2 × 2 metres
Toolbox (T)	2 × 1.5 metres
TV-screen (TV)	2.5 × 1.5 metres

Table A.2
Amounts of each object in C.4.

Object	Amount
Shelf (S)	1
Assembly fixture (AF)	1
Toolbox (T)	1
TV-screen (TV)	1

Table A.3
Relationships between objects in C.4.

Relationship	Weight
S-AF	100
T-AF	100
TV-AF	20
TV-S	20
TV-T	20

Table A.4
Amounts of each object in C.5.

Object	Amount
Shelf (S)	1
Assembly fixture (AF)	2
Toolbox (T)	1
TV-screen (TV)	1

Table A.5
Relationships between objects in C.5.

Relationship	Weight
S-AF1	100
S-AF2	100
T-AF1	100
T-AF2	100
TV-AF1	20
TV-AF2	20
TV-S	20
TV-T	20

A.1. C.4

See Tables A.2 and A.3.

A.2. C.5

See Tables A.4 and A.5.

A.3. C.6

See Tables A.6 and A.7.

A.4. C.7a

See Tables A.8 and A.9.

Table A.6
Amounts of each object in C.6.

Object	Amount
Shelf (S)	2
Assembly fixture (AF)	2
Toolbox (T)	1
TV-screen (TV)	1

Table A.7
Relationships between objects in C.6.

Relationship	Weight
S1-AF1	100
S2-AF2	100
T-AF1	100
T-AF2	100
TV-AF1	20
TV-AF2	20
TV-S1	20
TV-S2	20
TV-T	20

Table A.8
Amounts of each object in C.7a.

Object	Amount
Shelf (S)	2
Assembly fixture (AF)	2
Toolbox (T)	2
TV-screen (TV)	1

Table A.9
Relationships between objects in C.7a.

Relationship	Weight
S1-AF1	100
S2-AF2	100
T1-AF1	100
T2-AF2	100
TV-AF1	20
TV-AF2	20
TV-S1	20
TV-S2	20
TV-T1	20
TV-T1	20

Table A.10
Amounts of each object in C.7b.

Object	Amount
Shelf (S)	2
Assembly fixture (AF)	3
Toolbox (T)	1
TV-screen (TV)	1

A.5. C.7b

See Tables A.10 and A.11.

A.6. C.8

See Tables A.12 and A.13.

Data availability

Data will be made available on request.

Table A.11
Relationships between objects in C.7b.

Relationship	Weight
S1-AF1	100
S1-AF2	100
S1-AF3	100
S2-AF1	100
S2-AF2	100
S2-AF3	100
T-AF1	100
T-AF2	100
T-AF3	100
TV-AF1	20
TV-AF2	20
TV-AF3	20
TV-S1	20
TV-S2	20
TV-T	20

Table A.12
Amounts of each object in C.8.

Object	Amount
Shelf (S)	2
Assembly fixture (AF)	3
Toolbox (T)	2
TV-screen (TV)	1

Table A.13
Relationships between objects in C.8.

Relationship	Weight
S1-AF1	100
S1-AF2	100
S1-AF3	100
S2-AF1	100
S2-AF2	100
S2-AF3	100
T1-AF1	100
T1-AF2	100
T1-AF3	100
T2-AF1	100
T2-AF2	100
T2-AF3	100
TV-AF1	20
TV-AF2	20
TV-AF3	20
TV-S1	20
TV-S2	20
TV-T1	20
TV-T2	20

References

Adams, W. P., Guignard, M., Hahn, P. M., & Hightower, W. L. (2007). A level-2 reformulation–linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*, *180*(3), 983–996. http://dx.doi.org/10.1016/j.ejor.2006.03.051.

Ahmadi, A., Pishvaeea, M. S., & Jokar, M. R. A. (May 2017). A survey on multi-floor facility layout problems. *Computers & Industrial Engineering*, *107*, 158–170. http://dx.doi.org/10.1016/j.cie.2017.03.015.

Anjos, M. F., & Vieira, M. V. (2016). An improved two-stage optimization-based framework for unequal-areas facility layout. *Optimization Letters*, *10*, 1379–1392. http://dx.doi.org/10.1007/s11590-016-1008-6.

Bock, S., & Hoberg, K. (2007). Detailed layout planning for irregularly-shaped machines with transportation path design. *European Journal of Operational Research*, [ISSN: 0377-2217] *177*(2), 693–718. http://dx.doi.org/10.1016/j.ejor.2005.11.011.

Burer, S., & Vandenbussche, D. (2006). Solving lift-and-project relaxations of binary integer programs. *SIAM Journal on Optimization*, *16*(3), 726–750. http://dx.doi.org/10.1137/040609574.

Cubukcuoglu, C., Nourian, P., Tasgetiren, M. F., Sariyildiz, I. S., & Azadi, S. (2021). Hospital layout design renovation as a quadratic assignment problem with geodesic distances. *Journal of Building Engineering*, *44*, Article 102952. http://dx.doi.org/10.1016/j.jobe.2021.102952.

Emanuel, B., Wimer, S., & Wolansky, G. (2012). Using well-solvable quadratic assignment problems for VLSI interconnect applications. *Discrete Applied Mathematics*, *160*(4–5), 525–535. http://dx.doi.org/10.1016/j.dam.2011.11.017.

Gilmore, P. C. (1962). Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics*, *10*(2), 305–313.

Hahn, P. M., Zhu, Y. R., Guignard, M., Hightower, W. L., & Saltzman, M. J. (2012). A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS Journal on Computing*, *24*(2), 202–209. http://dx.doi.org/10.1287/ijoc.1110.0450.

Huang, C., & Wong, C. K. (2017). Discretized cell modeling for optimal facility layout plans of unequal and irregular facilities. *Journal of Construction Engineering and Management*, *143*(1), Article 04016082. http://dx.doi.org/10.1061/(ASCE)CO.1943-7862.0001206.

Jankovits, I., Luo, C., Anjos, M. F., & Vannelli, A. (2011). A convex optimisation framework for the unequal-areas facility layout problem. *European Journal of Operational Research*, *214*, 199–215. http://dx.doi.org/10.1016/j.ejor.2011.04.013.

Kaku, B. K., & Rachamadugu, R. (1992). Layout design for flexible manufacturing systems. *European Journal of Operational Research*, *57*(2), 224–230. http://dx.doi.org/10.1016/0377-2217(92)90044-A, Facility Layout.

Keller, B., & Buscher, U. (2015). Single row layout models. *European Journal of Operational Research*, *245*(3), 629–644. http://dx.doi.org/10.1016/j.ejor.2015.03.016.

Koopmans, T. C., & Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*(1), 53–76. http://dx.doi.org/10.2307/1907746.

Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research*, *176*(2), 657–690. http://dx.doi.org/10.1016/j.ejor.2005.09.032.

Pérez-Gosende, P., Mula, J., & Díaz-Madroñero, M. (2021). Facility layout planning. An extended literature review. *Int. J. Production Res.*, *59*, 3777–3816. http://dx.doi.org/10.1080/00207543.2021.1897176.

Sahni, S., & Gonzalez, T. (July 1976). P-complete approximation problems. *Journal of the ACM*, *23*(3), 555–565. http://dx.doi.org/10.1145/321958.321975.

Solimanpur, M., & Jafari, A. (2008). Optimal solution for the two-dimensional facility layout problem using a branch-and-bound algorithm. *Computers & Industrial Engineering*, *55*, 606–619. http://dx.doi.org/10.1016/j.cie.2008.01.018.

Tanaka, K., Horio, Y., & Aihara, K. (2001). A modified algorithm for the quadratic assignment problem using chaotic-neuro-dynamics for VLSI implementation. *1*, In *IJCNN'01. international joint conference on neural networks. proceedings (cat. no.01CH37222)* (pp. 240–245 vol.1). http://dx.doi.org/10.1109/IJCNN.2001.939024.

Tellier, L. N. (1972). The Weber problem: Solution and interpretation. *Geograph. Analysis*, *4*, 215–233. http://dx.doi.org/10.1111/j.1538-4632.1972.tb00472.x.

Tubaileh, A., & Siam, J. (2017). Single and multi-row layout design for flexible manufacturing systems. *International Journal of Computer Integrated Manufacturing*, *30*(12), 1316–1330. http://dx.doi.org/10.1080/0951192X.2017.1314013.