

Schedules for Pick-and-Place Robots

Peter Buchholz

`peter.buchholz@tu-dortmund.de`

TU Dortmund University

Alexander Puzicha

TU Dortmund University

Research Article

Keywords: Pick-and-Place Robot, ILP, Scheduling, Traveling Salesman, Decomposition

Posted Date: July 11th, 2025

DOI: <https://doi.org/10.21203/rs.3.rs-6804075/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Schedules for Pick-and-Place Robots

Peter Buchholz^{1*} and Alexander Puzicha^{1†}

^{1*}Informatik IV, TU Dortmund University, Dortmund, D-44227, Germany.

*Corresponding author(s). E-mail(s): peter.buchholz@tu-dortmund.de;

Contributing authors: alexander.puzicha@tu-dortmund.de;

[†]These authors contributed equally to this work.

Abstract

The efficiency of pick-and-place robots that put parts into boxes relies on efficient schedules which determine the order in which parts are picked. The scheduling problem can be formulated as an integer linear problem (ILP). However, the resulting ILP cannot be solved for realistic scenarios. Therefore, approximate solution techniques based on local search heuristics and simplified ILPs are introduced. It will be shown that ILP based solutions outperform search heuristics in terms of solution times and solution quality.

Finally, we consider an extended version of the pick-and-place robot with two arms. It is shown that the resulting scheduling problem can be handled similarly to the problem for the robot with one arm but the number of variables in the resulting ILP increases from $O(P^3)$ for one arm to $O(P^6)$ for two arms where P is the number of parts to be picked. Thus, decomposition based approaches are introduced that reduce the scheduling problem for two arms to several problems with one arm.

Keywords: Pick-and-Place Robot, ILP, Scheduling, Traveling Salesman, Decomposition

1 Introduction

Pick-and-Place robots play an important role in various production facilities. They have to remove parts from a picking area and put them into boxes according to the type of the selected part. Usually the locations and types of the parts in the picking area and the positions of the boxes are known in advance and a schedule has to be found to empty the area as fast as possible. Since the production process is ongoing, schedules have to be computed under time constraints.

In this paper, we develop methods to compute schedules for a new type of pick-and-place robot with two or four pickers and one or two arms. This robot allows in principle parallel picking and depositing parts and can thus speed up

the operations considerably. However, to benefit from this potential parallelism, accesses have to be scheduled appropriately because geometrical constraints have to be taken into account.

The optimal schedule for the robot can be formulated as an *integer linear program* (ILP) with a number of variables in $O(P^3)$ where P is the number of parts to be picked. Since the number of parts in realistic scenarios can be some hundreds, an exact solution is impossible even without real time constraints. Therefore, we approximate the original ILP by ILPs with only $O(P^2)$ or $O((qP)^2)$ (with $q \in (0, 1)$) variables which are, additionally, easier to solve. For larger values of P and the robot version with two arms (i.e., two robot arms and four pickers) we develop a decomposition approach that decomposes the problem

and solves the resulting sub-problems using small ILPs. It will be shown that these approximation approaches using ILPs are much more efficient and result in much better schedules than they are achieved with local search heuristics or simulated annealing applied to the original scheduling problem.

The structure of the paper is as follows. In the next section we introduce the pick-and-place robot and its functionality. Then, in Section 3, the different optimization approaches are developed for the robot with one arm. Afterwards, the robot with two arms is analyzed. In Section 5, real and randomly generated examples are analyzed. The paper ends with the conclusions.

2 Operation of Pick-and-Place Robots

Fig. 1 shows schematically the structure of the robot with one arm. The robot has two guide rails, the lines shown at bottom and top of the picture. Two pickers are located at a guide bar, denoted as a robot arm or simply arm, that can turn left and right on the guide rails. An arm can turn up to some angle $\alpha_{\max}$ which means up to $\alpha_{\max}/2$ left or right from the vertical axis. The pickers can move up and down at the guide bar. Pickers are shown as red dots in the picture.

Each part has to be put in a specific box. To work efficiently, one picker picks a part from the picking area (the upper part in the picture) when the other picker releases its part in the box. However, there are several restrictions. In the situation shown Fig. 1, picker 2 can pick the parts 3 and 4 when picker 1 puts its part in box 2. If subsequently picker 2 delivers part 3 or 4 to box 2, only the remaining part can be picked by the other picker. If parts have to be put into box 1, the robot arm has to be shifted along the guide rails.

In general, we have a number of parts and a number of boxes. Each part belongs to one box. A schedule consists of a sequence of picking operations where in each step each picker picks a part, puts a part in a box or does nothing. An optimal schedule is a schedule that requires the least time to put all parts in their boxes and observes the constraints due to physical and geometrical requirements.

Fig. 1 shows a robot with one arm, an extended version consists of two arms which work synchronously (see Fig. 2), i.e., the pickers of one guide rail pick parts on the left and the others on the right part of the picking area. This robot will be described and analyzed in Sect. 4.

2.1 Geometrical Constraints

The basic setting is a plane with the Euclidian distances. Each point has unique coordinates x and y . We assume that parts and boxes can be characterized by a single point which is the center of mass of the corresponding structure. Assume that the whole system can be described on a plane of size $X_{\max} \times Y_{\max}$ (i.e., $x \in [0, X_{\max}]$, $y \in [0, Y_{\max}]$). Boxes are located in the area below Y_b ($< Y_{\max}$) and parts in the area above Y_b . This is a typical layout on shopfloors. The two guide rails are in y -positions 0 and $Y_{\max}$.

Initially, the arm is in position $(0, 0)$, $(0, Y_{\max})$ and the two pickers are both in the position $(0, Y_{\max}/2)$. The arm can then move left-right and the pickers up-down. Of course, pickers can only overtake one another if at least one picker is empty. Due to the geometrical constraints, the position of the arm in the upper guide rail is in position $x \pm Y_{\max} \left(\frac{1}{\cos^2(\alpha_{\max}/2)} - 1 \right)$ if the position in the lower guide rail is x . Additionally, the position has to be in $[0, X_{\max}]$.

Parts are numbered from 1 through P and boxes from 1 through B . The set of parts and boxes are $\mathcal{P} = \{1, \dots, P\}$ and $\mathcal{B} = \{1, \dots, B\}$, respectively. $\bar{\mathcal{P}} = \{0, 1, \dots, P\}$ contains all parts and 0 to indicate an empty move. Indices i, j, k are used for parts, indices b for boxes.

For $i \in \mathcal{P}$, b_i is the corresponding box to which the part should be delivered and x_i, y_i are the coordinates of part i . Similarly, for $b \in \mathcal{B}$, x_b, y_b are the coordinates. The distance from part $i \in \mathcal{P}$ to box $b \in \mathcal{B}$ equals

$$d_{i,b} = \sqrt{dx_{i,b}^2 + dy_{i,b}^2} \quad (1)$$

where $dx_{i,b} = |x_i - x_b|$ and $dy_{i,b} = |y_i - y_b|$. In all cases $d_{b,i} = d_{i,b}$, i.e. the relation is symmetric. If one picker delivers a part to box b , then part i can be picked by the other picker, if the angle is less than $\alpha_{\max}$. This means that all parts on the same x coordinate can be picked and the guide

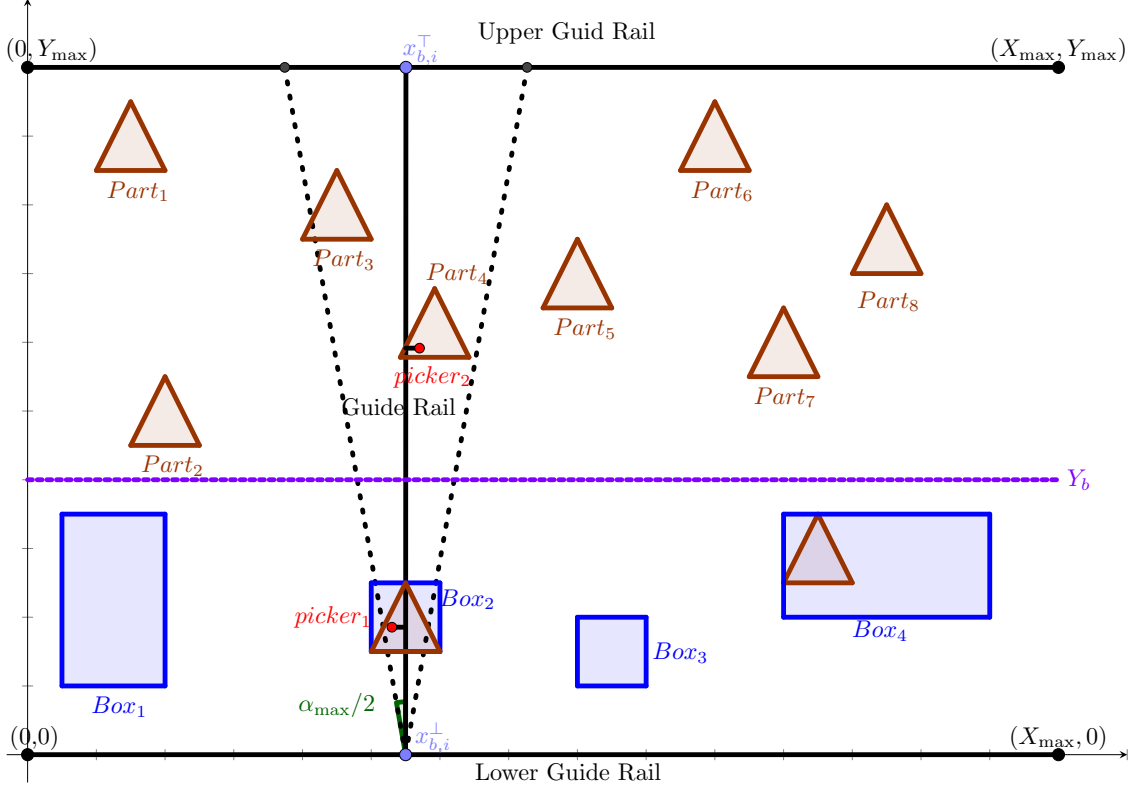


Fig. 1 Picking robot with one arm and two pickers

rail can be turned to the left or right by $\alpha_{\max}/2$ at most. Let φ be the angle of the guide rail that is necessary to access $i \in \mathcal{P}$ if the other picker is at $b \in \mathcal{B}$. The following condition has to hold.

$$\sin(\varphi) = \frac{dx_{i,b}}{d_{i,b}} \leq \sin\left(\frac{\alpha_{\max}}{2}\right) \quad (2)$$

2.2 Functionality of the Robot

A schedule is sequence of elements from $\bar{\mathcal{P}}$ which contains each element of $\mathcal{P}$ exactly ones. In other words it contains a sequence of all parts plus possibly several zeros. We do not allow two zeros in a row because this would imply that both pickers do nothing which is obviously a waste of capacity. Let $\sigma = \langle i_1, i_2, \dots, i_n \rangle$ be a schedule, i.e., $i_1, i_n \in \mathcal{P}$, $i_j \in \bar{\mathcal{P}}$ for $1 < j < n$ and $(i_{j-1}, i_j) \in \{\mathcal{P}, \bar{\mathcal{P}}\} \cup \{\bar{\mathcal{P}}, \mathcal{P}\}$. A schedule is valid if for each pair i_{j-1}, i_j ($1 < j \leq n$) i_{j-1} or i_j is 0 or

$$\frac{dx_{b_{i_{j-1}}, i_j}}{d_{b_{i_{j-1}}, i_j}} \leq \sin\left(\frac{\alpha_{\max}}{2}\right)$$

according to (2). Let

$$\mathcal{S}_b = \left\{ j \in \mathcal{P} \mid \frac{dx_{b,j}}{d_{b,j}} \leq \sin\left(\frac{\alpha_{\max}}{2}\right) \right\} \cup \{0\} \quad (3)$$

for $b \in \mathcal{B}$. Consequently, for $i \in \mathcal{P}$, $b \in \mathcal{B}$

$$i \in \mathcal{S}_b \Leftrightarrow |x_b - x_i| \leq \frac{\sin\left(\frac{\alpha_{\max}}{2}\right)}{\cos\left(\frac{\alpha_{\max}}{2}\right)} |y_b - y_i|. \quad (4)$$

Set $\mathcal{S}_i = \mathcal{S}_{b_i}$ contains all parts which can be picked by one picker when the other picker puts part i in its box. For a valid schedule that includes (i_{j-1}, i_j) obviously $i_j \in \mathcal{S}_{i_{j-1}}$ has to hold.

If the pickers are at box b and part i , then this also determines the x -positions of the arm in the lower and upper guide rail. $x_{b,i}^\perp$ and $x_{b,i}^\top$ are the corresponding positions which are given by

$$\begin{aligned} x_{b,i}^\perp &= x_b + \frac{x_b - x_i}{y_i - y_b} y_b & \text{and} \\ x_{b,i}^\top &= x_i + \frac{x_i - x_b}{y_i - y_b} (Y_{\max} - y_i). \end{aligned} \quad (5)$$

Additionally, $i \in \mathcal{S}_b$, $x_{b,i}^\perp, x_{b,i}^\top \in [0, X_{\max}]$ has to hold. If this is not the case, parts that violated

the condition have to be excluded from $\mathcal{S}_b$. The equation only handles the case $b \neq 0$ and $i \neq 0$. If one of both is zero, then we assume that the arm is turned vertically such that $x_{0,i}^\perp = x_{0,i}^\top = x_i$, $x_{b,0}^\perp = x_{b,0}^\top = x_b$ and $x_{0,0}^\perp = x_{0,0}^\top = 0$.

The position of a picker on the arm can be defined as the distance from the origin of the arm to the picker (i.e., the distance from $x^\perp$). If the arm is in position $x_{b,i}^\perp$, then the position of the picker for box b is $p_{b,(b,i)}$ and for part i it is $p_{i,(b,i)}$ which are computed as follows.

$$\begin{aligned} p_{b,(b,i)} &= \sqrt{(x_{b,i}^\top - x_b)^2 + y_b^2} \quad \text{and} \\ p_{i,(b,i)} &= \sqrt{(x_{b,i}^\top - x_i)^2 + y_i^2} \end{aligned} \quad (6)$$

For an empty move it is assumed that the picker is positioned in the middle of the guide rail. Thus, $p_{0,(0,i)} = p_{0,(b,0)} = p_{(0,(0,0))} = Y_{\max}/2$.

For some schedule $\sigma = \langle i_1, \dots, i_n \rangle$ unnecessary 0's can be removed. I.e., if $\langle \dots, i_{j-1}, i_j = 0, i_{j+1}, \dots \rangle$ is part of the schedule and $b_{i_{j-1}} \in \mathcal{S}_{i_{j+1}}$. In this case, the empty move between putting part i_{j-1} in its box and picking part i_{j+1} is redundant and can be removed. For a schedule σ without redundant moves, we define a representation $\hat{\sigma}$ by adding a zero at the beginning and removing all 0's in between. From $\hat{\sigma}$, the original schedule σ can be easily recreated by removing the leading and closing 0 and adding a 0 between element i_j and i_{j+1} whenever $b_{i_j} \notin \mathcal{S}_{i_{j+1}}$. Representation $\hat{\sigma}$ will be used to formulate the ILP for computing an optimal schedule.

2.3 Duration of a Schedule

The duration of a schedule results from the sum of its moves. A detailed analysis requires a detailed simulation of the kinematics which is beyond the scope of this paper. It turns out that for the computation of quasi optimal schedules it is sufficient to choose a simple model where we assume that pickers go up and down with maximal velocity v_p and acceleration a_p and the arm moves left or right with maximal velocity v_g and acceleration a_g . Furthermore, each operation of picking a part or putting it in a box requires a time t_o before the arm and picker move on.

We first consider a situation of three consecutive steps $i, j, k \in \mathcal{P}$ as a part of a schedule which

implies $j \in \mathcal{S}_i$ and $k \in \mathcal{S}_j$. Now assume that we begin with the situation that picker one just released part i to box b_i and picker two just picked part j . In the next step picker one has to move from (x_{b_i}, y_{b_i}) to (x_k, y_k) in time t_{p_1} , picker two from (x_j, y_j) to (x_{b_j}, y_{b_j}) in time t_{p_2} and the guide rail has to move from $(x_{b_i,j}^\perp, x_{b_i,j}^\top)$ to $(x_{b_j,k}^\perp, x_{b_j,k}^\top)$ in time $t_{g^\perp}$ for the move of the arm in the upper guide rail and $t_{g^\top}$ for the arm in the lower guide rail. All these steps can be made in parallel. Thus, we obtain for the duration of the whole step

$$\begin{aligned} t_{i,j,k} &= \max(t_{p_1}(i, j, k), t_{p_2}(i, j, k), t_{g^\perp}(i, j, k), t_{g^\top}(i, j, k)) \\ t_{p_1}(i, j, k) &= f_p(|p_{b_i,(b_i,j)} - p_{k,(b_j,k)}|) \\ t_{p_2}(i, j, k) &= f_p(|p_{j,(b_i,j)} - p_{b_j,(b_j,k)}|) \\ t_{g^\perp}(i, j, k) &= f_b\left(\left|x_{b_i,j}^\perp - x_{b_j,k}^\perp\right|\right) \\ t_{g^\top}(i, j, k) &= f_b\left(\left|x_{b_i,j}^\top - x_{b_j,k}^\top\right|\right) \end{aligned} \quad (7)$$

where

$$\begin{aligned} f_p(x) &= \begin{cases} \sqrt{\frac{2x}{a_p}} & \text{if } x \leq \frac{v_p^2}{2a_p}, \\ \frac{v_p}{a_p} + \frac{x - \frac{v_p^2}{2a_p}}{v_p} & \text{else,} \end{cases} \\ f_b(x) &= \begin{cases} \sqrt{\frac{2x}{a_g}} & \text{if } x \leq \frac{v_g^2}{2a_g}, \\ \frac{v_g}{a_g} + \frac{x - \frac{v_g^2}{2a_g}}{v_g} & \text{else,} \end{cases} \end{aligned} \quad (8)$$

equal the duration to move the picker and arm, respectively, by some distance. Since we have already considered indices equal to 0 in (5) and (6), the computed times are also valid if indices are 0. Let $\sigma = \langle i_1, \dots, i_n \rangle$ be some schedule then

$$T_\sigma = t_{0,0,i_1} + t_{0,i_1,i_2} + \sum_{j=1}^{n-2} t_{i_j,i_{j+1},i_{j+2}} + t_{i_{n-1},i_n,0} + nt_o \quad (9)$$

is the duration of the schedule.

3 Optimal Schedules for the One-armed Variant

We begin with the formulation of the optimization problem for one-armed robots by first defining an ILP that minimizes (9). This ILP can be analyzed exactly or approximately using some heuristic approach. Then two variants of the ILP are presented that contain less variables but introduce an

approximation. Finally, we present a decomposition approach that decomposes the problem and defines several smaller ILPs.

3.1 Exact ILP Solution

As common for ILP formulations of scheduling problems, a schedule is coded by variables $\delta_{i,j} \in \mathbb{B}$, $i, j \in \bar{\mathcal{P}}$ and $i \neq j$. A valid schedule has to observe the following constraints.

$$\begin{aligned} \sum_{j \in \bar{\mathcal{P}} \setminus \{i\}} \delta_{i,j} &= 1 \quad \text{for all } i \in \bar{\mathcal{P}} \\ \sum_{i \in \bar{\mathcal{P}} \setminus \{j\}} \delta_{i,j} &= 1 \quad \text{for all } j \in \bar{\mathcal{P}} \end{aligned} \quad (10)$$

The constraints imply that each part is predecessor and successor of some part or an empty move in a schedule. This formulation of the constraints assures that each element from $\bar{\mathcal{P}}$ occurs exactly once in a schedule. I.e., a representation $\hat{\sigma}$ without 0's between two moves can be represented. The single zero element indicates the start and end of the schedule.

For the objective function we define weights $w_{i,j,k}$ that describe the time of the last step, i.e., the putting of part j in its box and the picking of part k starting from the position defined by (i, j) .

$$w_{i,j,k} = \begin{cases} t_{0,0,k} + t_o & \text{if } j = 0 \\ t_{i,j,k} + t_o & \text{if } j \neq 0 \text{ and } j \in \mathcal{S}_i, k \in \mathcal{S}_j, \\ t_{0,j,k} + t_o & \text{if } j \neq 0 \text{ and } j \notin \mathcal{S}_i, k \in \mathcal{S}_j, \\ t_{i,j,0} + t_{j,0,k} + 2t_o & \text{if } j \neq 0 \text{ and } j \in \mathcal{S}_i, k \notin \mathcal{S}_j, \\ t_{0,j,0} + t_{j,0,k} + 2t_o & \text{if } j \neq 0 \text{ and } j \notin \mathcal{S}_i, k \notin \mathcal{S}_j, \end{cases} \quad (11)$$

where the values for $t_{i,j,k}$ are given in (7). The first case considers the situation where k is the first element in the schedule, the second and third case situations where the operations of the last step can be done in parallel and the last two cases situations where the operations of the last step have to be done sequentially. With these weights the objective function of the ILP is given by

$$\min \left(\sum_{i \in \bar{\mathcal{P}}} \sum_{j \in \bar{\mathcal{P}} \setminus \{i\}} \sum_{k \in \bar{\mathcal{P}} \setminus \{i,j\}} w_{i,j,k} \delta_{i,j} \delta_{j,k} \right). \quad (12)$$

It is straightforward to show that (12) exactly mimics (9) for some schedule $\hat{\sigma}$ where 0s are introduced if $k \notin \mathcal{S}_{b_j}$. Unfortunately, the function

is not linear since it contains products $\delta_{i,j} \delta_{j,k}$. However, it is well known that this can be linearized (Asghari et al., 2022; Vielma, 2015) by introduction of new variables $\gamma_{i,j,k} \in \mathbb{B}$ and new constraints

$$\gamma_{i,j,k} \leq x_{i,j}, \gamma_{i,j,k} \leq \delta_{j,k} \text{ and } \gamma_{i,j,k} \geq \delta_{i,j} + \delta_{j,k} - 1. \quad (13)$$

This is a standard approach which, however, adds new integer variables and the corresponding constraints to the ILP. The number of variables for the resulting ILP is in $O(P^3)$ which precludes the solution of the ILP for larger values of P .

The constraints defined in (10) are not sufficient to find only valid schedule because they not exclude it sub-tours. E.g., $\delta_{1,2} = \delta_{2,1} = \delta_{0,3} = \delta_{3,4} = \delta_{4,0} = 1$ would be a valid solution with $P = 4$ but defines $\langle 1, 2, 1 \rangle$ and $\langle 0, 3, 4, 0 \rangle$ which is not a valid schedule. This problem is known basically from the *traveling salesman problem* (TSP) (Applegate et al., 2007; Gutin, 2024; Laporte, 1992; Pataki, 2003). There are different possibilities to add constraints that avoid subtours. One possibility are the *Miller, Tucker, Zemlin* (MTZ) constraints (Pataki, 2003)

$$\begin{aligned} u_i &\in \mathbb{R} & 2 \leq u_i \leq P & \quad i = 2, \dots, P \\ u_i - u_j + 1 &\leq (P-1)\delta_{i,j} & \forall i \in \{2, \dots, P\} & \\ & & \forall j \in \{2, \dots, P\} & i \neq j \end{aligned} \quad (14)$$

The formulation adds P variables and $(P-2)^2$ constraints. It results in a mixed integer linear program (MILP) that allows only valid schedules as feasible solutions. It is known as a weak formulation of the ILP which means that it is usually not recommended for larger instances of the TSP (Pataki, 2003). However, the scheduling problem for the robot differs from a TSP since it contains additional constraints that reduce the solution space such that the MTZ constraints often work well.

An alternative approach results in the sub-tour constraints. The following constraints are added.

$$\forall \mathcal{Q} \subset \bar{\mathcal{P}} \text{ with } |\mathcal{Q}| \leq P/2 : \sum_{(i,j) \in \mathcal{Q}} \delta_{i,j} < |\mathcal{Q}| \quad (15)$$

This formulation adds an exponential number of constraints to the problem. Thus, it is usually not be applied from scratch and constraints are added on the fly during the solution.

The different versions of the ILP can be analyzed with a solver like Gurobi (Anand et al., 2017). The robot schedule has to be computed under time constraints. Thus, the main point is that the solution is robust which means that always an acceptable and most times an almost optimal solution is computed with the given time budget. If the solution is required after $\hat{t}$ seconds, then the ILP solver runs for $\hat{t} - \Delta$ seconds where Δ covers the setup and post processing time and some time buffer. The output of the ILP solver is a schedule σ , if the solver is able to find a feasible solution. Furthermore, the solver computes a lower bound from the relaxed problem. Ideally, the *optimality gap* between the computed lower bound and the best solution is small which assures that the computed solution is almost optimal. To avoid situations where the ILP generates no feasible solution, a generic schedule may be predefined and used as a fallback solution. As example, the always feasible solution σ_f is defined:

$$\begin{aligned}\sigma_f &= \langle 1, 0, 2, 0, \dots, P \rangle \\ &= \langle 0, \cup_{i=1}^P (i, 0) \rangle \\ &= \forall i \in \mathcal{P} : \delta_{0,i} = \delta_{i,0} = 1\end{aligned}\quad (16)$$

This schedule can be improved by removing zeros if two consecutive non-zero elements can be picked together.

3.2 Heuristic Solutions

Since the exact ILP solution is not feasible for larger and realistic values of P (see the results in Sect. 5), one may apply some heuristics to minimize (12). A large number of heuristic methods exist in the literature to solve scheduling problems or related problems (Baker and Trietsch, 2018; Branke et al., 2015; Funke et al., 2005; Li and McMahon, 2007). We use two heuristic approaches which are available in the *python-tsp* package (Goulart, 2024) and are adopted to the more complex objective function (12) that considers triplets rather than pairs of consecutive indices.

The first heuristics is a *local search* approach that starts with some valid schedule, i.e., some permutation of the number 1 through P starting and ending with 0. For a given schedule $\hat{\sigma}$ a neighborhood $\mathcal{N}(\hat{\sigma})$ is computed using the 2-opt approach (Croes, 1958) or one of the schemes proposed in Goulart et al. (2018). Schedules from

$\mathcal{N}(\hat{\sigma})$ are analyzed (i.e., (12) is evaluated) until a schedule with a smaller duration is found. This schedule becomes then $\hat{\sigma}$ and the approach continues until no better schedule is found in $\mathcal{N}(\hat{\sigma})$ or the available time budget is exhausted. The local search approach always finds a valid schedule but may trap in the next local minimum.

An alternative is simulated annealing (Van Laarhoven and Aarts, 1987) where we use the same neighborhood definition but a different acceptance function. In contrast to local search, simulated annealing may escape from local minima but in the end also stops based on local optimality conditions. For details of the acceptance function see Goulart (2024).

Heuristic approaches have the advantage of always finding a feasible schedule but this schedule might be not much better than the initial solution where the algorithm has been started. The evaluation of the objective function becomes time consuming, if the set $\mathcal{N}(\hat{\sigma})$ contains schedules which differ in several positions from $\hat{\sigma}$. On the other hand such schedules have to be part of the set to obtain better solutions.

3.3 Approximate ILP Solutions

The major problem of the *exact* ILP solution is the objective function (12) that requires the introduction of variables $y_{i,j,k}$ such that the number of decision variables is increased from $O(P^2)$ to $O(P^3)$. To avoid this blow-up, the weights have to be approximated. Thus define,

$$\hat{w}_{i,j} = \begin{cases} \max(f_p(d_{i,b_i}), f_p(d_{j,b_j})) & \text{if } b_i \in \mathcal{S}_j, \\ f_p(d_{i,b_i}) + \max(f_p(d_{j,b_j}), f_b(d_{b_i,b_j})) & \text{if } b_i \notin \mathcal{S}_j. \end{cases} \quad (17)$$

If $b_i \in \mathcal{S}_j$, the time is determined by the parallel movement of the two pickers. If $b_i \notin \mathcal{S}_j$, the first picker has to move to its box, then the second picker and the arm have to move. Thus, the weights approximate the exact duration without using the exact positions of the arm and the pickers.

With these weights an objective function for the ILP can be defined as

$$\min \left(\sum_{i \in \mathcal{P}} \sum_{j \in \mathcal{P} \setminus \{i\}} \hat{w}_{i,j} \delta_{i,j} \right). \quad (18)$$

Together with the constraints (10) and some constraints to avoid sub-tours, the objective function defines an ILP with $O(P^2)$ variables. The value of the objective function does not exactly correspond to duration of the schedule but a good schedule according to (18) will also be good according to (12) because it contains a lot of parallelism. This will also be justified by the examples below.

Up to now geometric constraints have not been explicitly considered in the optimization methods. They are implicitly described in the coefficients of the objective function. We now consider an ILP where we explicitly integrate the geometrical constraints in the ILP constraints and compute schedule σ with zeros rather than $\hat{\sigma}$.

For these schedules define the following constraints.

$$\begin{aligned} \sum_{j \in \mathcal{P} \setminus \{i\}} \delta_{i,j} &= 1 \quad \text{for all } i \in \mathcal{P}, \\ \sum_{i \in \mathcal{P} \setminus \{j\}} \delta_{i,j} &= 1 \quad \text{for all } j \in \mathcal{P}, \\ \delta_{i,j} &= 0 \quad \text{for } j = i \text{ or } j \notin \mathcal{S}_i. \end{aligned} \quad (19)$$

The constraints assure that for each $i \in \mathcal{P}$ exactly one $\delta_{j,i} = 1$ and $\delta_{i,j} = 1$. Thus, the number of non-zero decision variables is in $O((qP)^2)$ where q is the percentage of parts that can be picked by the second picker, if the first picker is above some box b . If we assume that parts are randomly distributed in the rectangle with the corner points $(0, Y_b)$, $(0, Y_{\max})$, $(X_{\max}, Y_b)$ and $(X_{\max}, Y_{\max})$, q can be computed as

$$q = \frac{\sin\left(\frac{\alpha_{\max}}{2}\right)(Y_{\max} + Y_b)}{2 \cos\left(\frac{\alpha_{\max}}{2}\right) X_{\max}} \quad (20)$$

for boxes b where the arm can move the full angle $\alpha_{\max}$ (i.e., boxes that are not too near to one of the boundaries). For realistic values $q < 0.1$ holds.

A simple objective function is then

$$\min \left(\sum_{i \in \mathcal{P}} \sum_{j \in \mathcal{P} \setminus \{i\}} \delta_{i,j} \right) \quad (21)$$

This objective function minimizes the length of schedule σ and thus the number of zeros in a schedule which means that the number of empty moves is minimized. The objective function does not take into account that parallel movements of

the pickers are most efficient if the distances are similar. To consider this aspect we define weights

$$\check{w}_{i,j} = \begin{cases} f_p(d_{i,b_i}) & \text{if } j = 0, \\ f_p(d_{j,b_j}) & \text{if } i = 0, \\ \max(f_p(d_{i,b_i}), f_p(d_{j,b_j})) & \text{else.} \end{cases} \quad (22)$$

With these weights the following objective function is defined.

$$\min \left(\sum_{i \in \mathcal{P}} \sum_{j \in \mathcal{P} \setminus \{i\}} \check{w}_{i,j} \delta_{i,j} \right). \quad (23)$$

Objective function (21) or (23) together with the constraints (19) and some constraints to avoid sub-tours defines an ILP with $O((qP)^2)$ decision variables which results in a schedule σ .

If the resulting schedule contains a single 0, then it is uniquely determined because the schedule starts with the part following 0. However, if it contains more than one 0, then the starting point of the schedule and the ordering of the sub-schedules between the zeros is not fixed.

To clarify the latter point, we consider the always feasible schedule σ_f where $\delta_{0,1} = \delta_{1,0} = \delta_{0,2} = \delta_{2,0} = \delta_{0,3} = \delta_{3,0} = 1$ (see Eq. 16). This implies that the schedule does not allow any parallel movements and parts have to be picked one after the other in arbitrary order. The objective functions (21) and (23) do for example not distinguish between $\sigma = \langle 1, 0, 2, 0, 3, \rangle$ and $\sigma = \langle 1, 0, 3, 0, 2, \rangle$ or any other ordering. Consequently, the schedule only determines the ordering of sequences of parts that contain no zeros.

To find an optimal ordering we set up a second ILP. Let Z be the number of zero elements that follow a non-zero element or are predecessor of a non-zero element. Observe that the number of non-zero predecessors and successors of zero element are identical. Let $\mathcal{Z}_{\mathcal{P}}$ and $\mathcal{Z}_{\mathcal{S}}$ be the sets of non-zero indices that precede or follow a zero. Define decision variables $\delta_{i,j}^Z \in \mathbb{B}$ for $i \in \mathcal{Z}_{\mathcal{P}}$ and $j \in \mathcal{Z}_{\mathcal{S}}$. Constraints

$$\begin{aligned} \sum_{j \in \mathcal{Z}_{\mathcal{S}}} \delta_{i,j}^Z &= 1 \quad \text{for all } i \in \mathcal{Z}_{\mathcal{P}}, \\ \sum_{i \in \mathcal{Z}_{\mathcal{P}}} \delta_{i,j}^Z &= 1 \quad \text{for all } j \in \mathcal{Z}_{\mathcal{S}} \end{aligned} \quad (24)$$

assure that each element preceding or following a zero element occurs exactly one. Then define

weights $\hat{c}_{i,j} = t_{i,0,j}$ which is the time to move from the box of part i (i.e., b_i) to part j . The objective function

$$\min \left(\sum_{i \in \mathcal{Z}_P} \sum_{j \in \mathcal{Z}_S} \hat{w}_{i,j} \delta_{i,j}^{\mathcal{Z}} \right). \quad (25)$$

minimizes this time. Thus, the ILP with objective function (25), constraints (24) and additional constraints to avoid sub-tours results in a unique order of parts,

If in the above example $\delta_{1,3}^{\mathcal{Z}} = \delta_{3,2}^{\mathcal{Z}} = \delta_{2,1}^{\mathcal{Z}} = 1$ then part 3 should be picked immediately after 1 and not after 2. This does not define the first part to be picked. Thus, schedules $\langle 1, 0, 3, 0, \rangle$, $\langle 3, 0, 2, 0, 1 \rangle$ and $\langle 2, 0, 1, 0, 3 \rangle$ fulfill the conditions.

To determine the part to start with, (9) is evaluated for all P parts and the order defined by the schedule. This can be done efficiently because only the first two values change and the value of the sum in (9) can be reused during the computation of the time for a schedule. The starting part is then determined by choosing the part which results in the smallest relation.

Consequently, if the geometric constraints are used as constraints in the ILP two problems have to be solved, one with $O((qP)^2)$ decision variables and one with $O(Z^2)$ decision variables. Usually this is more efficient and results are as good as the results for the variant without geometric constraints. This will be shown by the examples in Sect. 5.

3.4 A Decomposition Approach

It will be shown that the ILP based approaches presented above work well up to about 300 parts and a time window of about 5 minutes. This is sufficient for most practical scenarios. However, if the number of parts is larger than 400, the probability that the ILP finds no or only relatively weak schedules increases. For these cases we develop a decomposition approach which will also be used as basic approach for computing schedules for the robot with two arms.

The decomposition approach is inspired by known decomposition techniques for ILPs (Galati and Ralphs, 2005; Mannon, 2020) that are adopted to the specific structure of the scheduling

problem for pick-and-place robots. The idea is to decompose the set of parts according to their location in the plane and the location of their boxes. Set $\mathcal{P}$ is decomposed into $R^2 \in \mathbb{N}$ sets $\mathcal{P}_{r^\uparrow, r^\downarrow}$ ($r^\uparrow, r^\downarrow \in \{1, \dots, R\}$) such that part i belongs to $\mathcal{P}_{r^\uparrow, r^\downarrow}$ if

$$\frac{r^\uparrow - 1}{R} X_{\max} < x_i \leq \frac{r^\uparrow}{R} X_{\max} \quad \text{and} \quad \frac{r^\downarrow - 1}{R} X_{\max} < x_{b_i} \leq \frac{r^\downarrow}{R} X_{\max}.$$

This decomposition decomposes the region into sections according to the x -index. Parts in $\mathcal{P}_{r^\uparrow, r^\uparrow}$ have boxes in the same section and usually will be picked together. Thus, a locally optimal schedule for $\mathcal{P}_{r^\uparrow, r^\uparrow}$ will likely be a good sub-schedule for a schedule for the whole set $\mathcal{P}$. Parts in $\mathcal{P}_{r^\uparrow, r^\downarrow}$ have to be moved from region $r^\uparrow$ to $r^\downarrow$ which usually requires an empty move of the other picker. On the way back from $r^\downarrow$ to $r^\uparrow$ a part from $\mathcal{P}_{r^\uparrow, r^\downarrow}$ might be picked and moved. Thus, parts from $\mathcal{P}_{r^\uparrow, r^\downarrow} \cup \mathcal{P}_{r^\downarrow, r^\uparrow}$ are likely to be scheduled together. E.g., for $R = 2$ ILPs for $\mathcal{P}_{1,1}$, $\mathcal{P}_{2,2}$ and $\mathcal{P}_{1,2} \cup \mathcal{P}_{2,1}$ have to be solved to compute local sub-schedules $\sigma_{1,1}$, $\sigma_{2,2}$ and $\sigma_{1,2}$. For the computation of the local schedules one of the proposed approaches can be applied. Results presented in Sect. 5 indicate that objective function (23) together with the constraints (19) and some constraints to avoid sub-tours is a good choice for a first step. Afterwards, the ILP consisting of objective function (25) with constraints (24) and constraints to avoid sub-tours is used to find the final schedule.

After the local schedules are available, they have to be combined to form a schedule for the whole problem. Usually it is sufficient to concatenate the local sub-schedules. If the number of zero moves in the sub-schedule is not too large, one may also set up an ILP to reorder the different parts in the complete schedule. However, examples show that this step is usually not necessary.

4 Optimal Schedules for Robots with Two Arms

The pick-and-place robot with two arms is shown in Fig. 2. It allows simultaneous operations of 4 pickers. In the following, we first introduce the functionality and geometrical constraints before the optimization problem is formulated and

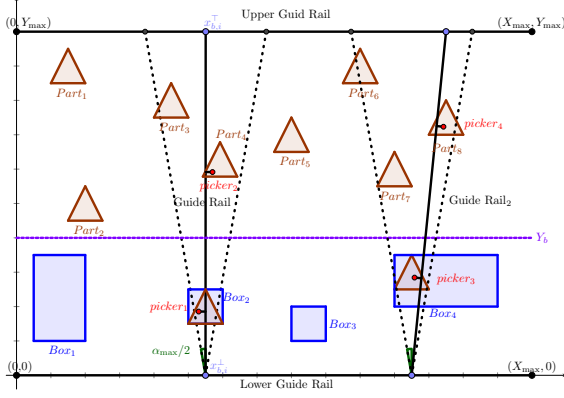


Fig. 2 Picking robot with four pickers

decomposition based optimization methods are derived.

4.1 Function of the Two-armed Robot

The two arms work synchronously, i.e., pickers pick and release parts. The next step starts after all pickers did their job. Thus, in a single step, up to two pickers pick parts and up to two pickers release their parts to the boxes. If this is not possible due to the constraints, up to three pickers may be idle in a step. Consequently, in each step up to two parts are picked and, simultaneously, up to two parts are released. A schedule $\sigma^{(2)} = \langle (i_1, j_1), \dots, (i_n, j_n) \rangle$ consists of pairs $(i_k, j_k) \in \mathcal{P}^2$ which contains each part from $\mathcal{P}$ exactly once. We assume that for two consecutive pairs $(i_k, j_k), (i_{k+1}, j_{k+1})$ at least one element is from $\mathcal{P}$, otherwise all pickers would be idle. $\sigma^{(2)}$ consists of a schedule for the first arm by using only the entries i_k and a schedule for the second arm by using only the entries j_k . Usually, the number of parts that are picked by one arm differs from the number of parts picked by the other arm.

For a valid schedule, the geometrical constraints have to be met. This implies $x_{i_1} < x_{j_1}$ and for all $k \in \{2, \dots, n\}$:

$$\begin{aligned} b_{i_{k-1}} &\in \mathcal{S}_{i_k}, & b_{j_{k-1}} &\in \mathcal{S}_{j_k}, \\ [(i_{k-1}, j_{k-1}), (i_k, j_k)] &\in \mathcal{C} & \Leftrightarrow & \\ x_{b_{i_{k-1}}, i_k}^\perp &< x_{b_{j_{k-1}}, j_k}^\perp, & x_{b_{i_{k-1}}, i_k}^\top &< x_{b_{j_{k-1}}, j_k}^\top \end{aligned} \quad (26)$$

The constraints include the constraints for a single arm and additional constraints assuring that the first arm is left of the second arm.

The computation of valid schedules for two arms is much more complex than for one arm. The problem is that deadlocks may occur because the arms cannot overtake one another. Thus, if the left arm picks a part belonging to a box far right and the right arm picks a part for a box far left, the conflict has to be resolved. To preclude deadlocks or other problems, the following assumptions are made:

- Whenever a part is picked, it will be released to its box in the next step. If this is not possible, the part is not picked.
- If only one of the two arms can pick a part, the first (left) arm will have priority.
- If the left arm is idle for at least two steps, it is shifted to position $[0, 0], [0, Y_{\max}]$. Similarly, if the right arm is idle for at least two steps, it is shifted to position $[X_{\max}, 0], [X_{\max}, Y_{\max}]$.

Geometrically it might be possible to avoid for example the shift of an arm to the far left or far right position but the clear rules used here avoid errors and accidents of the robot.

To analyze the duration of a schedule we have again to consider three consecutive elements from $\sigma^{(2)}$, $\langle (i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, j_k) \rangle$ with the interpretation that the first picker of the first arm is above box $b_{i_{k-2}}$, the second picker is above i_{k-1} , the first picker of the second arm is above box $b_{j_{k-2}}$, the second picker above part j_{k-1} and the next move puts the pickers of the first arm to box $b_{i_{k-1}}$ and part i_k , whereas the pickers of the second arm moves to box $b_{j_{k-1}}$ and part j_k . Furthermore, $x_{b_{i_k}} < x_{b_{j_k}}$ has to hold because picked parts have to be released to their boxes after the next move. Again zero indices can be part of schedule to consider the constraints (26). The duration of the last move in the sequence equals

$$t_{(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, j_k)} = \max(t_{i_{k-2}, i_{k-1}, i_k}, t_{j_{k-2}, j_{k-1}, j_k}) \quad (27)$$

where the times for the moves of the two arms are computed via (7). Due to the movement of idle arms we have $t_{0,0,i} = t_{i,0,0} = f_b(x_i)$ for the first (i.e., left) arm and $f_b(X_{\max} - x_i)$ for the second (i.e., right) arm and $t_{0,0,0} = 0$ which can happen if one arm is idle while the other is still working.

The overall duration of a schedule is then given by

$$T_{\sigma^2} = t_{(0,0),(0,0),(i_1,j_1)} + t_{(0,0),(i_1,j_1),(i_2,j_2)} + \sum_{k=1}^{n-2} t_{(i_k,j_k),(i_{k+1},j_{k+1}),(i_{k+2},j_{k+2})} + t_{(i_{n-1},j_{n-1}),(i_n,j_n),(0,0)} + nt_o. \quad (28)$$

4.2 Optimization of Schedules

The problem with two arms is related to the multiple traveling salesman problem (MTSP) (Cheikhrouhou and Khoufi, 2021) which is often solved by heuristics (Zheng et al., 2022). Like in the case with one arm, the problem differs from the MTSP because the geometric constraints restrict the solution space drastically but also make it much more complicated to find valid schedules and avoid potential deadlocks.

We first outline how the problem can be formulated as an (M)ILP. A schedule $\sigma^{(2)}$ contains elements (i, j) and can be described by decision variables $\delta_{(i,j),(k,l)} \in \mathbb{B}$ with the interpretation that the first picker at the first arm puts a part in box b_i , the second picker picks part k , the first picker of the second arm puts a part in box b_j and the second picker picks part l . The constraints (26) can then be used to set decision variables to zero and coefficients of the objective function can be computed similar to the case with one arm. However, this implies that the number of decision variables is in $O(P^4)$ rather than $O(P^2)$ and the ILP for the exact solution has $O(P^6)$ rather than $O(P^3)$ decision variables. Since we have already remarked that the problem with $O(P^3)$ variables cannot be solved for realistic scenarios, it is hopeless to strive for an exact solution or even for the approximate ILP solutions that have been proposed for the system with one arm. Consequently, we do not consider exact or approximate ILP solutions of a complete schedule. Instead we describe a possible coding to use heuristics like *local search* or *simulated annealing*. In the following paragraph we consider schedules computed by decomposition.

For the heuristic algorithms, a schedule is described by a string $i_1, i_2, \dots, i_{2P}$ which contains every element of $\mathcal{P}$ exactly once, the remaining elements are 0. This representation allows one to describe a set of possible schedules including those where only one arm is active. This set contains an

optimal schedule. The interpretation is that in the j -th of P steps, one picker of the first arm picks part i_{2j-1} and the other picker is above $b_{i_{2j-3}}$ and one picker of the second arm picks part i_{2j} and the second picker is above $b_{i_{2j-2}}$. We assume $i_j = 0$ for $j \leq 0$ and use the semantics explained for one arm if some indices are 0. E.g., $i_{2j} = 0$ implies that the second arm is in a vertical position above $b_{i_{2j-2}}$ and if $i_{2j-2} = 0$ additionally holds, then the second arm is in a vertical position at $X_{\max}$. Furthermore $T_{(0,0),(0,0),(0,0)} = T_{0,0,0} = 0$.

We do not assume that the schedule observes the geometrical constraints. These constraint will be implicitly integrated adding zero moves. To do so we add necessary zeros by running through the schedule considering in each step a triple of indices, i.e., for $\sigma^{(2)} = \langle (i_1, j_1), (i_2, j_2), \dots, (i_{2P}, j_{2P}) \rangle$ triplets are generated in the order $[(0, 0), (0, 0), (i_1, j_1)], [(0, 0), (i_1, j_1), (i_2, j_2)], \dots, [(i_{n-2}, j_{n-2}), (i_{n-1}, j_{n-1}), (i_n, j_n)], [(i_{n-1}, j_{n-1}), (i_n, j_n), (0, 0)]$. Let $[(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, j_k)]$ be the current triple. It can be assumed that for $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1})$ (26) holds and $x_{b_{i_{k-1}}} < x_{b_{j_{k-1}}}$. The next tuple (i_k, j_k) can be added as it is if $b_{i_{k-1}} \in \mathcal{S}_{i_k}, b_{j_{k-1}} \in \mathcal{S}_{j_k}, [(i_{k-1}, j_{k-1}), (i_k, j_k)] \in \mathcal{C}$ and $x_{b_{i_k}} < x_{b_{j_k}}$. If this is not the case, zero moves have to be added. To do this in a well defined form, the following cases are considered in the order a) through i) and the first case for which

the conditions are met is chosen.

- a) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, j_k)$
if $b_{i_{k-1}} \in \mathcal{S}_{i_k} \wedge b_{j_{k-1}} \in \mathcal{S}_{j_k} \wedge [(b_{i_{k-1}}, i_k), (b_{j_{k-1}}, j_k)] \in \mathcal{C} \wedge x_{b_{i_k}} < x_{b_{j_k}},$
- b) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, 0), (0, j_k)$
if $b_{i_{k-1}} \in \mathcal{S}_{i_k} \wedge [(b_{i_{k-1}}, i_k), (b_{j_{k-1}}, 0)] \in \mathcal{C} \wedge [(b_{i_k}, 0), (j_k, 0)] \in \mathcal{C}$
- c) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, j_k), (i_k, 0)$
if $b_{j_{k-1}} \in \mathcal{S}_{j_k} \wedge [(b_{i_{k-1}}, 0), (b_{j_{k-1}}, j_k)] \in \mathcal{C} \wedge [(b_{j_k}, 0), (i_k, 0)] \in \mathcal{C}$
- d) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, 0), (i_k, j_k)$
if $x_{i_k} < x_{j_k} \wedge x_{b_{i_k}} > x_{b_{j_k}}$
- e) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, 0), (0, 0), (0, j_k)$
if $b_{i_{k-1}} \in \mathcal{S}_{i_k} \wedge x_{i_k} < x_{b_{j_{k-1}}}$
- f) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, 0), (i_k, 0)$
if $b_{j_{k-1}} \in \mathcal{S}_{j_k} \wedge x_{j_k} > x_{b_{i_{k-1}}}$
- g) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, 0), (i_k, 0), (0, j_k)$
if $x_{b_{i_k}} < x_{j_k}$
- h) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, 0), (0, j_k), (i_k, 0)$
if $x_{b_{j_k}} > x_{i_k}$
- i) $(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, 0), (i_k, 0), (0, 0), (0, j_k)$
else

(29)

From these expansion the time t_k until parts (i_k, j_k) have been picked can be computed. E.g., in case a) we have $t_k = t_{(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (i_k, j_k)} + t_o$ according to (27), for case i) t_k becomes $t_{(i_{k-2}, j_{k-2}), (i_{k-1}, j_{k-1}), (0, 0)} + t_{(i_{k-1}, j_{k-1}), (0, 0), (i_k, 0)} + t_{(0, 0), (i_k, 0), (0, 0)} + t_{(i_k, 0), (0, 0), (0, j_k)} + 4t_o$. In this case, it is necessary to add moves to pick both parts i_k and j_k because of the geometric constraints. I.e., i_k and j_k cannot be picked together and none of these parts can be picked when i_{k-1} and j_{k-1} are released.

If we denote by the length of a schedule the index of the last non-zero element (i.e., $n = |\sigma^{(2)}| = \arg \max_k (i_k > 0 \vee j_k > 0)$), then $T_{\sigma^{(2)}} = \sum_{i=1}^{|\sigma^{(2)}|+1} t_i$. Implicitly, the expansion computes also a valid schedule by introducing zero moves, which can be implemented for the robot from an ordering of the parts including an assignment to robot arms to parts.

With (29) the duration of a schedule described by a string as introduced above can be evaluated. This representation can then be used in *local search* and *simulated annealing*. However, in contrast to the case with one arm, the representation

contains many zeros and makes the local modifications less efficient. Therefore, it is important to start with a good schedule which is then improved.

One possibility to generate an initial schedule $\sigma = \langle i_1, \dots, i_P \rangle$ for one arm. Then elements are reordered resulting in $\sigma^{(2)} = \langle i_1, i_{P/2}, i_2, i_{P/2+1}, \dots, i_{P/2-1}, i_P \rangle$, where we assume in this example that n is an even number. If σ removes parts from left to right, then the reordering assures that the left arm removes parts in the left side and the right arm removes parts from the right side. More appropriate initial schedules will be computed by decomposition as shown below.

4.3 Decomposition based Optimization

The decomposition approach that has been presented for the robot with one arm computes sub-schedules for parts that are located in specific sections of the picking area. If for example two sections are defined, then parts in $\mathcal{P}_{1,1}$ and $\mathcal{P}_{2,2}$ can usually be picked in parallel by the two arms. There is only a small area at the boundary where the pickers may interfere and zero moves have to be added. Similarly, if the left arm moves a part from section i to section j and the right arm moves a part from section k to l , then these moves do not interfere if $i < k$ and $j < l$. Consequently, sub-schedules can be computed and then assigned to arms. Possibly sub-schedules have to be synchronized. I.e., if the left arm moves parts between sections i and j ($i < j$) and the right arm between section k and l ($k < l$) with $i < k$, $j < l$ but $k \leq j$, then a move from j to i of the left arm has to be done simultaneously with a move from l to k by the right arm and a move from i to j has to be synchronously done with a move from k to l .

As an example assume that $\sigma_1 = (i_1, i_2, \dots, i_n)$ is a sub-schedule for $\mathcal{P}_{12} \cup \mathcal{P}_{21}$ and $\sigma_2 = (j_1, j_2, \dots, j_m)$ is a sub-schedule for $\mathcal{P}_{23} \cup \mathcal{P}_{32}$. Then σ_1 is handled by the left arm and σ_2 by the right arm. If i_1 is in the first section with a box in the second section and j_1 is in the second section with a box in the third section, then $((i_1, j_1), (i_2, j_2), \dots)$ is a possible ordering for the joint trace. If instead j_1 is in the third section with a box in the second section, then $((i_1, 0), (i_2, j_1), (i_3, j_2), \dots)$ is a better ordering.

The introduced procedure describes how to build joint schedules based on a decomposition of the picking region in sections. For two sections, the approach is obvious. The left arm works on the schedule for $\mathcal{P}_{1,1}$ whereas the right arm serves the schedule for $\mathcal{P}_{2,2}$. Then one arm, usually the one that finishes first, starts with the sub-schedule for $\mathcal{P}_{1,2} \cup \mathcal{P}_{2,1}$. This results in the valid and easy to compute schedule but about half of the parts are served by one arm. Consequently, the potential parallelism of two robot arms is not fully exploited. However, by using more than two sections, the potential parallelism can be increased. Thus, for 3 sections $\mathcal{P}_{1,1} \cup \mathcal{P}_{1,2} \cup \mathcal{P}_{2,1}$ are handled by the first arm and $\mathcal{P}_{2,3} \cup \mathcal{P}_{3,2} \cup \mathcal{P}_{3,3}$ are handled simultaneously by the second arm. Finally, $\mathcal{P}_{2,2} \cup \mathcal{P}_{1,3} \cup \mathcal{P}_{3,1}$ are handled sequentially by one arm. Assuming a random distribution of parts and boxes over the sections, 2/3 of the parts are handled in parallel. This approach can be applied to more than 3 sections but requires that parts and boxes are approximately randomly distributed.

If the distribution is not random, the distribution of sub-schedules to robot arms is the major point to gain efficiency. Assume that the region is decomposed into R regions as described above. Define for $1 \leq i < j \leq R$ $\#^{\cup} \mathcal{P}_{i,j}$ as the number of parts in the region $\mathcal{P}_{i,j} \cup \mathcal{P}_{j,i}$ and $\rho_{i,j} = \#^{\cup} \mathcal{P}_{i,j} / R$, the percentage of parts in the regions $\mathcal{P}_{i,j}$ and $\mathcal{P}_{j,i}$. For $i = j$ we define $\rho_{i,i} = \#^{\cup} \mathcal{P}_{i,i} / (2R)$.

To assign the sub-schedules to robot arms we set up a *linear program* (LP). Let $\delta_{i,j,k,l}^{\rho} \in \mathbb{R}_{\geq 0}$ for $i \leq j, k \leq l, i < k, j < l$ be a decision variable with the interpretation that $\delta_{i,j,k,l}^{\rho}$ is the percentage of parts that is simultaneously moved between the sections i and j by the left arm and between section k and l by the right arm. We use $\delta_{0,0,k,l}^{\rho}$ to indicate that the left arm is idle while the right arm moves between section k and l and, similarly, $\delta_{i,j,0,0}^{\rho}$ to indicate that the right arm is idle. Then the following constraints are defined.

$$\begin{aligned} & \sum_{k=i+1}^R \sum_{l=\max(j+1,k)}^R \delta_{i,j,k,l}^{\rho} + \delta_{i,j,0,0}^{\rho} + \\ & \sum_{k=1}^{i-1} \sum_{l=k+1}^{j-1} \delta_{k,l,i,j}^{\rho} + \delta_{0,0,i,j}^{\rho} \geq \rho_{i,j} + \rho_{j,i} \end{aligned} \quad (30)$$

$$\forall 1 \leq i \leq j \leq R$$

The objective function is defined as

$$\min \left(\sum_{i=1}^R \sum_{j=i}^R \sum_{k=i+1}^R \sum_{l=\max(j+1,k)}^R \delta_{i,j,k,l}^{\rho} + \sum_{i=1}^R \sum_{j=1}^R \delta_{i,j,0,0}^{\rho} + \sum_{k=1}^R \sum_{l=1}^R \delta_{0,0,k,l}^{\rho} \right) \quad (31)$$

After the values for $x_{i,j,k,l}$ are available, the sub-schedules can be assigned to the arms. E.g., about $\min \left(\delta_{i,j,k,l}^{\rho} / \rho_{i,j}, 1 \right) \#^{\cup} \mathcal{P}_{i,j}$ consecutive elements from a sub-schedule for $\mathcal{P}_{i,j} \cup \mathcal{P}_{j,i}$ are assigned to the left arm and, simultaneously, about $\min \left(\delta_{i,j,k,l}^{\rho} / \rho_{k,l}, 1 \right) \#^{\cup} \mathcal{P}_{k,l}$ elements from a schedule for $\mathcal{P}_{k,l} \cup \mathcal{P}_{l,k}$ are assigned to the right arm.

The sub-schedules are computed to allow maximal parallelism for the pickers of one arm and the assignment to the arms maximizes the parallelism between the arms. The choice of R is a free parameter which depends on the number and location of parts. If R is too small, parallelism between the arms is reduced, if it is too large, the sub-schedules cannot exploit maximal parallelism of the pickers.

5 Examples

We analyze various examples to compare the different approaches. All experiments are performed on a workstation with i9 – 9900X 3.50 Ghz cpu with 10 physical cores, 20 virtual processors and 64 GB main memory running Linux 11. The MILPs are solved using Gurobi 11.03. The local search heuristics are implemented in python using some functions from python-tsp (Goulart, 2024) which are adopted to the specific problem here. In the following we first consider examples for the robot with one arm and then for the two-armed version.

For each of the following examples 12 different configurations varying positions of parts and boxes are analyzed and box-plots for the 12 result values are shown in the figures. The following parameters are fixed in all experiments $X_{\max} = 4000$, $Y_{\max} = 5000$, $Y_b = 2000$ and $\alpha_{\max} = 10^\circ$.

5.1 Examples for one Robot-Arm

We begin of the exact analysis of the ILP using the formulation defined in Sect. 3.1 including the

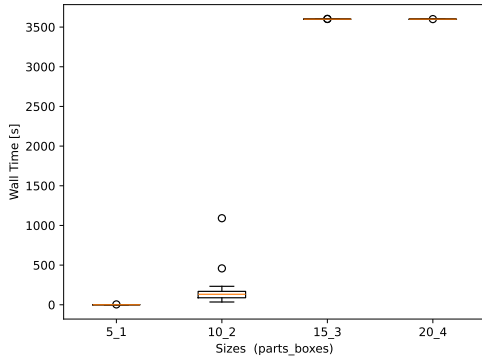


Fig. 3 Wall-times in seconds for an exact solution

MTZ constraints. Examples are analyzed for $P = 5, 10, 15, 20$ and $B = P/5$. The solution stops after 3600 seconds of wall-clock time. Fig. 3 shows the boxplots of the times to solve the different problem sizes and Fig. 4 the required CPU times that are much higher because Gurobi uses up to 20 threads to solve the ILP in parallel.

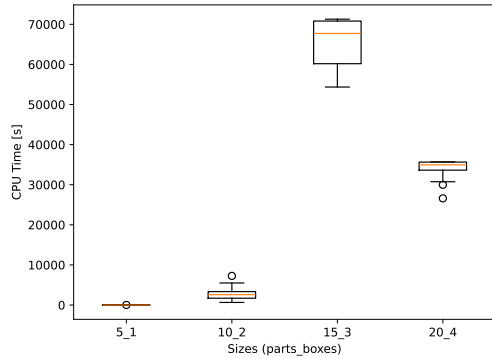


Fig. 4 CPU times in seconds for an exact solution

It can be seen that already the examples with $P = 15$ and $B = 3$ cannot be solved within one hour up to optimality. The optimality gap is about 50% for this configuration. For the configuration with $P = 20$, $B = 4$, the optimality gap grows to about 80%. For the configuration with $P = 20$ parallelism cannot be fully exploited because the pre-processing takes a lot of time. It can be seen that within 5 minutes, which is the time window in practical applications, only the smallest configuration with $P = 5$ can be solved to optimality.

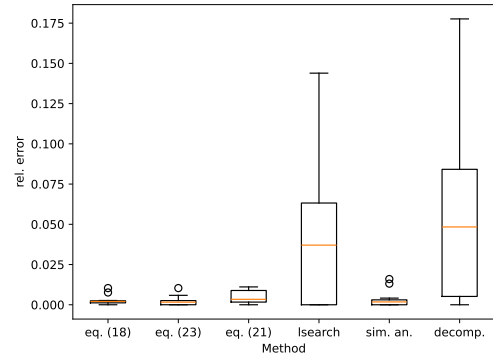


Fig. 5 Relative difference of the duration of the optimal schedule and the schedules computed with approximate approaches

Fig. 5 shows for the examples with $P = 15$ the relative difference between the duration of the schedule computed the exact method and the schedules resulting from different approximation methods. The number of the equation indicates the objective function, the two heuristic approaches start from a randomly generated schedule and the decomposition approach uses objective function (23) and decomposes the region in two sections. Although the optimality gap of the exact solution is not closed, none of the approximate techniques was able to find a better schedule which indicates that the schedule computed with the exact ILP formulation is optimal. Apart from local search and the decomposition approach, the approximate techniques produce schedule with a duration that is less than 2% worse than the best schedule. For local search and decomposition the median of the duration is about 5% larger than the best solution and values go up to 15% or 17.5% for the decomposition approach. However, it should be noted that the decomposition approach is not intended to be used for a small number of parts. All approximate methods require only a few seconds for computation of the solution.

We now consider some realistic configurations under realistic time constraints. The time limit is set to 250 seconds which is clearly below 300 seconds which is the required deadline. In the following figures we show the CPU times rather than the wall-clock times to circumvent the effect of parallel executions which are part of the ILP solvers but not of the implementation of the heuristic solution algorithms. Consequently, for

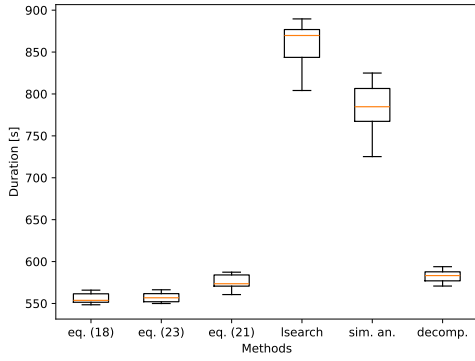


Fig. 6 Duration of the schedules for $P = 200$ and $B = 40$ computed with different methods

ILP solvers wall-clock times are significantly below the CPU times.

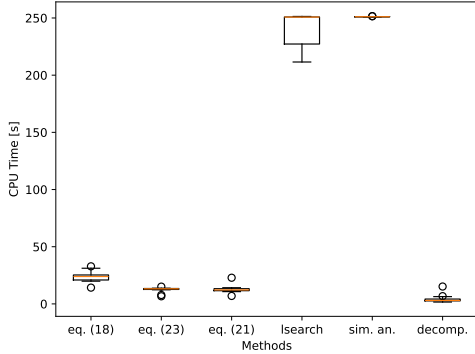


Fig. 7 CPU times for computing the schedules for $P = 200$ and $B = 40$ with different methods

We start with $P = 200$ and $B = 40$. Fig. 6 shows the duration of the schedules computed with the different approaches. It can be noticed that ILPs with objective functions (18) and (23) give the best results and the results for the ILP with objective function (21) and the decomposition approach are only slightly worse. The two heuristic solutions are not competitive, the resulting schedules require about 50% more time than the schedules resulting from the ILP solvers. Fig. 7 shows the CPU times to compute the schedules. It can be noticed that the ILP solutions require less than 50 seconds whereas the heuristic solvers stop prematurely by reaching the time limit. However, even if we let the heuristic solvers run until

convergence, which might take more than an hour of CPU time, the results do not improve significantly. This indicates that the objective function has many local optima due to the geometric constraints.

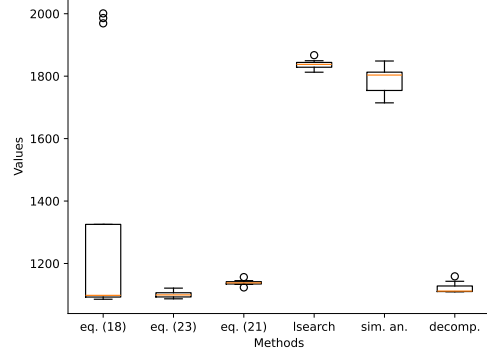


Fig. 8 Duration of the schedules for $P = 400$ and $B = 80$ computed with different methods

We now consider a larger configuration with $P = 400$ and $B = 80$. The duration of the schedules computed with the different approaches are shown in Fig. 8, the corresponding CPU times to compute the schedules are shown in Fig. 9. The ILP solution with objective function (23) and partially with (18) compute the best schedules. However, for objective function (18), the median of the duration is small but there are some outliers with a long duration. The reason can be seen by considering the CPU times. The ILP solver with objective function (18) and sometimes also with objective function (21) are stopped prematurely by reaching the time limit which may result in bad schedules or even in situations where no feasible solution is found and a random schedule has to be used as fallback solution. Again the heuristic solvers are not competitive whereas the decomposition approach results in good schedules that are computed in a very short time.

The largest configuration we consider has the parameters $P = 600$ and $B = 120$. It can be seen that the ILP solvers without decomposition reach their limits. About half of the configurations cannot be solved within the time limit although results, at least for (23) and (21), are still much better than the results of the heuristic solvers.

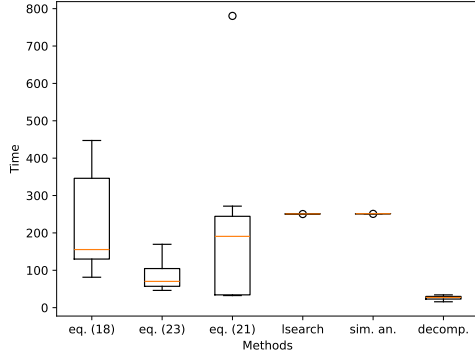


Fig. 9 CPU times for computing the schedules for $P = 400$ and $B = 80$ with different methods

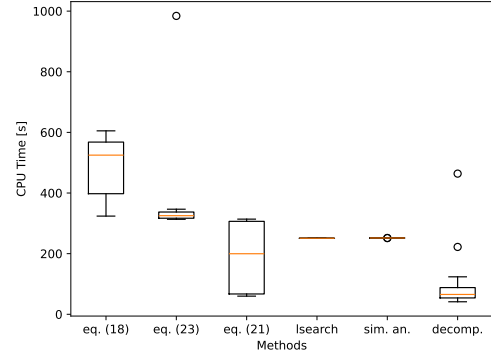


Fig. 11 CPU times for computing the schedules for $P = 600$ and $B = 120$ with different methods

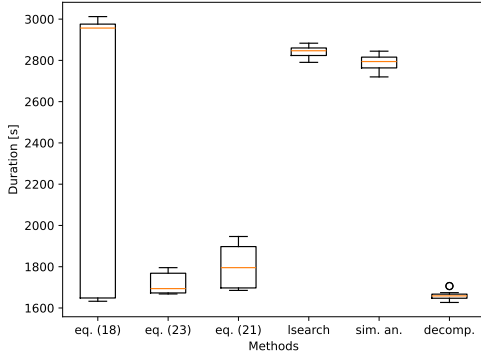


Fig. 10 Duration of the schedules for $P = 600$ and $B = 120$ computed with different methods

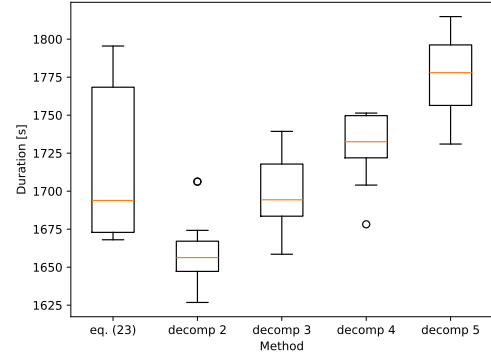


Fig. 12 Duration of the schedules for $P = 600$ and $B = 120$ computed with different decompositions

The best results are computed by the decomposition approach and the required CPU time of this approach is in all cases below 100 seconds and most times below 50 seconds.

The decomposition approach has only been used by decomposing the region into two sections. Fig. 12 and 13 show the results and CPU times for the decomposition approach with more than two sections. It can be noticed that the CPU time to compute the schedules exceeds the CPU time to compute the schedule from (23) for more than two sections and results become worse if the number of sections is increased. Thus, two sections seem to be the best choice for this problem size. If the number of parts is increased, a larger number of sections might become necessary.

5.2 Examples for two Robot-Arms

We now consider a robot with two arms. The same configurations are used that have already been analyzed for the robot with one arm. The first set includes configurations with $P = 200$ and $B = 40$. The duration of the schedules for two arms is compared with the duration of the best schedule that is available for the robot with one arm. For two arms schedules are computed with *local search* starting from a random initial schedule (*ls*), simulated annealing starting from a random initial schedule (*sa*), the optimal schedule for the robot with one arm which is split in the middle such that the first half can be served by one arm and the second half by the second arm (*d1*), the decomposition approach using two sections (*d2*), the decomposition approach using three sections (*d3*), the decomposition approach with two sections

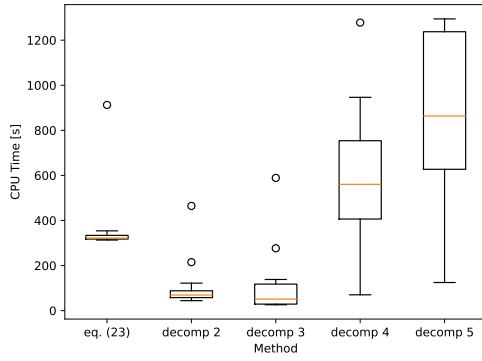


Fig. 13 CPU times for computing the schedules for $P = 600$ and $B = 120$ computed with different decompositions

and a subsequent improvement of the resulting schedule by *local search* using the remaining time budget (*d2ls*), and the decomposition approach with three sections and a subsequent improvement of the resulting schedule by *local search* using the remaining time budget (*d3ls*).

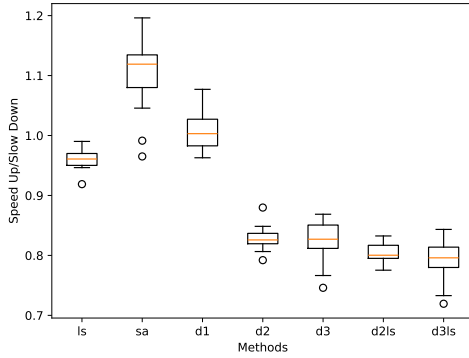


Fig. 14 Speed up or slow down of the schedules for the robot with two arms and $P = 200$, $B = 40$

Fig. 14 shows the duration of the schedules in relation to the duration of the best available schedule for the robot with one arm. It can be noticed that within the time bound for computing the optimal schedule, which is again 250 seconds, *simulated annealing* is not able to find a schedule for the robot with two arms that beats the best schedule for the robot with one arm. The optimal schedule for one arm does not results in an efficient schedule for two arms, durations remain in the average the same (see *d1*). *local*

search results in schedules for the robot with two arms which slightly better than the best schedule for the robot with one arm. The two decomposition approaches result in schedules that always improve the best schedule for the robot with one arm. The duration is in all cases reduced by more than 10% and most times by 15% – 20%. The decomposition in three rather than two sections does not improve the results, the variance becomes higher, the average duration almost remains. A subsequent *local search* to improve the schedules resulting from decomposition has only a minor effect, the duration is reduced by 2% – 3%.

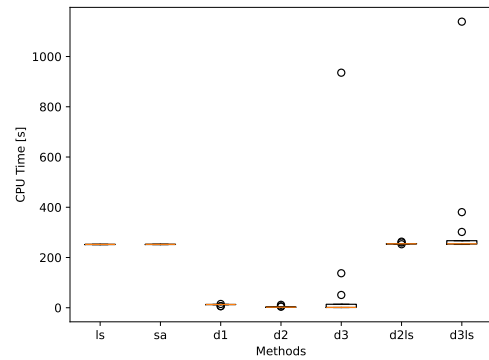


Fig. 15 CPU times for computing the schedules for the robot with two arms and $P = 200$, $B = 40$

Fig. 15 shows the CPU times to compute the schedules. The decomposition approaches are usually very efficient requiring only a few seconds for computing the schedule. However, in three out of twelve examples the decomposition in three sections results in some ILPs that are hard to solve and require long solution times. This cannot be avoided because the problem is NP hard but possibly could be reduced by increasing the maximal optimality gap which is used for stopping the ILP solver. In the examples we use a maximal optimality gap of 1%. The heuristic optimization methods use the available time budget. Consequently, it is questionable whether an improvement of a schedule computed by decomposition using *local search* is really worth the effort in view of the small improvements of the duration.

Finally, we consider the largest configuration with $P = 600$ and $B = 120$. Fig. 16 shows the speed up or slow down of the durations of the

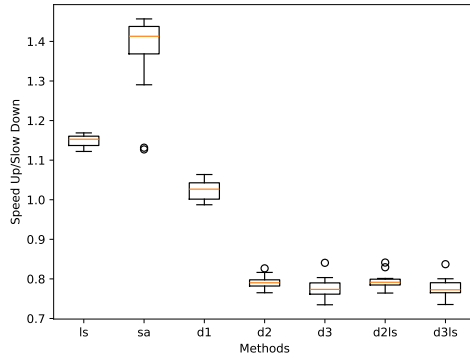


Fig. 16 Speed up or slow down of the schedules for the robot with two arms and $P = 600$, $B = 120$

computed schedules for the robot with two arms compared to the duration of the best schedule for the robot with one arm. Results are in line with the results for the smaller configuration. The improvement by the second arm is now slightly larger, most times above 20% of the duration and the improvement by a subsequent *local search* step is minor. This is partially due to the longer runtimes of the ILP solvers for computing the sub-schedules in the decomposition leaving less time for the *local search* steps. However, even if we give the heuristic methods much more time, results are not much better.

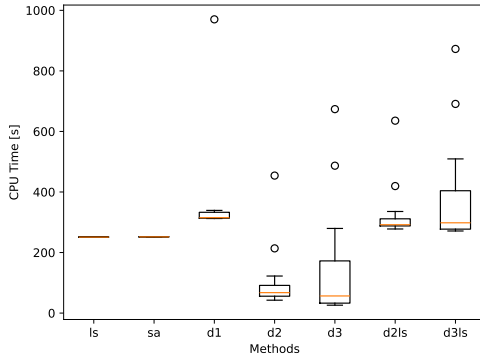


Fig. 17 CPU times for computing the schedules for the robot with two arms and $P = 600$, $B = 120$

6 Conclusions

In this paper we present different approaches to compute schedules for pick-and-place robots with one or two arms and two or four pickers. Schedules have to be computed under real time constraints during the operation of the system. It is shown that the problem can be formulated as an ILP which, however, can only be solved for very small configurations. Therefore, several approximate formulations have been introduced. It is shown that ILP based approaches outperform heuristic optimization methods in terms of solution quality and computational efficiency. For larger configurations, decomposition based approaches seem to be the method of choice, especially for the robot with two arms.

The developed algorithms are usable in practical applications because they compute good schedules in a very short time far below the available threshold. Nevertheless, improvements are possible for example by considering more flexible decomposition schemes.

References

- Anand, R., Aggarwal, D., Kumar, V.: A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems* **20**(4), 623–635 (2017)
- Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: *The Traveling Salesman Problem: A Computational Study*, (2007)
- Asghari, M., Fathollahi-Fard, A.M., Mirzapour Al-E-Hashem, S., Dulebenets, M.A.: Transformation and linearization techniques in optimization: A state-of-the-art survey. *Mathematics* **10**(2), 283 (2022)
- Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation* **20**(1), 110–124 (2015)
- Baker, K.R., Trietsch, D.: *Principles of Sequencing and Scheduling*. John Wiley & Sons, Hoboken, New Jersey (2018)

Cheikhrouhou, O., Khoufi, I.: A comprehensive

- survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Comput. Sci. Rev.* **40**, 100369 (2021) <https://doi.org/10.1016/J.COSREV.2021.100369>
- Croes, G.A.: A method for solving traveling-salesman problems. *Operations Research* **6**(6), 791–812 (1958)
- Funke, B., Grünert, T., Irnich, S.: Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics* **11**, 267–306 (2005)
- Goulart, F., Maravilha, A.L., Carrano, E.G., Campelo, F.: Permutation-based optimization for the load restoration problem with improved time estimation of maneuvers. *International Journal of Electrical Power & Energy Systems* **101**, 339–355 (2018)
- Goulart, F.: Python TSP. <https://pypi.org/project/python-tsp/> (2024)
- Galati, M., Ralphs, T.K.: Decomposition in integer linear programming. Corl technical report 04t-019-r1, Lehigh University (2005)
- Gutin, G.: Traveling salesman problem. In: *Encyclopedia of Optimization*, pp. 1–11. Springer, New York (2024). <https://doi.org/10.1007/978-3-030-54621-2>
- Laporte, G.: The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* **59**(2), 231–247 (1992)
- Li, W., McMahon, C.A.: A simulated annealing-based optimization approach for integrated process planning and scheduling. *International Journal of Computer Integrated Manufacturing* **20**(1), 80–95 (2007)
- Mannon, P.A.: *Partitioning Methods for NP-Hard Routing Problems*. The University of Texas at Austin, Austin, Texas (2020)
- Pataki, G.: Teaching integer programming formulations using the traveling salesman problem. *SIAM Review* **45**(1), 116–123 (2003)
- Vielma, J.P.: Mixed integer linear programming formulation techniques. *SIAM Review* **57**(1), 3–57 (2015)
- Van Laarhoven, P.J., Aarts, E.H.: *Simulated Annealing*. Springer, New York (1987)
- Zheng, J., Hong, Y., Xu, W., Li, W., Chen, Y.: An effective iterated two-stage heuristic algorithm for the multiple traveling salesmen problem. *Comput. Oper. Res.* **143**, 105772 (2022) <https://doi.org/10.1016/J.COR.2022.105772>