Computers & Operations Research 138 (2022) 105603

Contents lists available at ScienceDirect

Computers and Operations Research

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

Assembly planning by disjunctive programming and geometrical reasoning

Markó Horváth *, Tamás Kis, András Kovács, Márk Fekula

Institute for Computer Science and Control (SZTAKI), Kende str. 13-17, Budapest, H-1111, Hungary

ARTICLE INFO

Keywords:
Assembly planning
Combinatorial optimization
Disjunctive programming
Geometrical reasoning

ABSTRACT

The challenge in modeling and solving assembly planning problems lies in integrating combinatorial optimization techniques for finding efficient task sequences and resource assignments with geometrical reasoning to ensure the geometrical and technological feasibility of the assembly plans. This paper proposes a Benders decomposition approach that separates the macro-level planning problem, responsible for task sequencing and resource assignment, from micro-level validation on detailed geometrical and technological models. Feedback from the micro to the macro level is provided in the form of disjunctive constraints generated during search, which precludes the repeated occurrence of the collisions encountered in earlier iterations. A disjunctive programming approach is proposed to solve the macro-level planning problem with the added constraints. The efficiency of the approach is demonstrated both in industrial case studies and in computational experiments on generated problem instances.

1. Introduction

This paper proposes an efficient solution method for the assembly planning problem that can be briefly stated as follows. Given a set of parts (components), and a set of candidate assembly tasks to join these parts, find a subset of the tasks, along with a sequence of the selected tasks and an assignment of resources to the selected tasks, such that (i) the selected tasks construct the final product from the individual parts, (ii) the sequence of tasks with the assigned resources has a technologically and geometrically feasible physical realization, and (iii) a cost function is minimized. Checking whether a sequence of assembly tasks has a feasible realization may be tricky, just think about how the 27 parts of the Rubik cube (in case of the original Rubik brand) can be assembled together.

Assembly planning must match often conflicting aspects of product design, production technology, production system configuration, and production economics. To achieve this, it must combine combinatorial optimization techniques with geometrical reasoning over models of the product and the applicable equipment, knowledge-based techniques for technological feasibility, etc. This complexity results that assembly planning is the least automated field in production informatics. Nevertheless, this paper proposes a Benders decomposition based approach that combines discrete optimization in the master problem with geometrical reasoning in the subproblem, where the subproblem returns disjunctive constraints if the solution of the master problem has no feasible realization.

Compared to classical approaches to assembly planning, which compose the complete combinatorial problem model from the geometry of the product and the equipment in a preprocessing phase (Wilson and Latombe, 1994), the proposed approach generates constraints during search. The novel approach enables lifting some of the limiting assumptions of the earlier contributions: e.g., it allows arbitrary assembly paths instead of the former linear movements in a few predefined directions. The latter assumption was necessary in order to generate all constraints on the assembly task sequence by performing a finite number collision detections during preprocessing.

This work is partly based on the Benders solution approach presented in Kardos et al. (2020), nevertheless, with more sophisticated and efficient solution techniques. The underlying models are similar apart from minor differences (e.g., the set of parts that can be grasped by a fixture are characterized in different ways), yet, the solution techniques differ substantially: mixed-integer linear programming is used to solve the Benders master problem instead of constraint programming (CP); sophisticated modeling techniques are proposed for encoding the connectivity status of the parts in different stages of the assembly process; the constraints fed back from the subproblem to the master problem are significantly stronger, which results in less iterations in the Benders approach. In our approach, the subproblem provides violated disjunctive constraints that can be modeled by additional binary variables and linear constraints in the master problem. To the best of our knowledge, the combination of Benders decomposition with disjunctive

* Corresponding author.
E-mail addresses: marko.horvath@sztaki.hu (M. Horváth), tamas.kis@sztaki.hu (T. Kis), andras.kovacs@sztaki.hu (A. Kovács), mark.fekula@sztaki.hu (M. Fekula).

https://doi.org/10.1016/j.cor.2021.105603
Received 7 September 2020; Received in revised form 6 August 2021; Accepted 12 October 2021
Available online 30 October 2021

programming is new. Finally, various techniques, including caching, have been added to speed up the solution of the subproblems as well. Moreover, the computational efficiency of the proposed techniques are investigated in a large set of generated problem instances, which has not been performed for the earlier approach. The generated problem instances are freely available, and they can be used by other researchers for benchmark purposes.

We describe the problem in Section 2, and briefly summarize the related results in Section 3. In Section 4, we provide a mixed-integer linear programming formulation of the problem. As the mathematical program may contain an exponential number of *disjunctive constraints*, we turn to logic-based Benders decomposition to solve it, which is the topic of Section 5. The disjunctive constraints are generated by analyzing the assembly sequence found by solving the master problem, and the analysis is based on geometric reasoning as explained in Section 6. We present computational results on some industrial test cases and also on generated instances in Section 7. We compare our results to previous work in Section 8, and conclude the paper in Section 9.

Terminology An *undirected graph* is an ordered pair $G = (V(G), E(G))$, where $V(G)$ is a set of elements, called nodes, and $E(G) \subseteq V(G) \times V(G)$ is a set of unordered pairs of nodes, called edges. All graphs in this paper are *simple*, that is, undirected and contains no loop edges, i.e., edges $e = \{v, v\}$ for some node v.

A graph G' is a *subgraph of* G, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For a subset $S \subseteq V(G)$ of nodes, the *subgraph induced by* S is the graph $G[S] := (S, \sigma(S))$ where $\sigma(S) := \{\{u, v\} \in E(G) : u \in S,\ v \in S\}$.

For a subset $S \subseteq V(G)$ of nodes let $[S, \bar{S}]_G := \{\{u, v\} \in E(G) : |S \cap \{u, v\}| = 1\}$ be the set of edges between S and $\bar{S} := V(G) \setminus S$, also called a *cut*. A cut $[S, \bar{S}]_G$ is an $r - s$ cut, if $r \in S$ and $s \in \bar{S}$. Moreover, we use notation $d_G(S) := |[S, \bar{S}]_G|$ and $d_G(v) = d_G(\{v\})$. The graph G is *connected* if $[S, \bar{S}]_G \neq \emptyset$ for all $\emptyset \neq S \subsetneq V(G)$. A *tree* is a minimally connected graph, i.e., if any one edge was removed, the graph does not remain connected. A subgraph T of G is a *spanning tree of* G if T is a tree and $V(T) = V(G)$. A *path* P is a tree with $d(v) \leq 2$ for each $v \in V(P)$. In other words, a path is a finite sequence of edges which joins a sequence of nodes which are all distinct, and a graph is connected if there is a path between every pair of its nodes.

A connected graph is *biconnected* if any one node was removed, the graph remains connected. A *biconnected component* is an inclusion-wise maximal biconnected induced subgraph.

2. Problem statement

In the assembly planning problem, there is a set of *parts* V, and a set of candidate *assembly tasks* T for joining them. Each task $t \in T$ comprises an unordered pair of parts, $\langle a, b \rangle$, to be joined. The additional technological data associated to the task, such as the required motion of the parts, will be discussed in detail in Section 6. These candidate tasks define the so-called *liaison graph* of the product, i.e., a graph $G = (V, T)$ with set of nodes V, and set of edges T. All the possible subassemblies can be compactly represented as connected subgraphs of G.

There is a set of conjunctive constraints given in the input (while additional constraints will be generated during search), each of which prescribes that a subassembly $W \subset V$ must be joined before executing task t.

Each assembly task requires two types of resources: a *fixture* and a *tool*, which must be selected from the predefined sets F and Z, respectively. Each fixture $f \in F$ can grasp one specific part v_f directly. Moreover, there is a maximal subassembly V_f with $v_f \in V_f$ defined for f that it can hold *indirectly* via the grasped part. Then, for an arbitrary subassembly W, fixture f can hold W if and only if W is connected, $v_f \in W$, and $W \subseteq V_f$. While the two parts joined by an assembly task are not differentiated in the input, the selection of the fixture in the assembly plan distinguishes the *base part* held (potentially indirectly, via another grasped part) in the fixture from the *moved part*. In contrast,

Fig. 1. Illustrative example: ball valve product.

the set of tools applicable for task $t \in T$ is given in the input, and it is denoted by Z_t.

The processing time of task t is denoted by ρ_t, whereas d_f and d_z stand for the changeover times related to fixture $f \in F$ and tool $z \in Z$, respectively. It is noted that in reality, the processing and changeover times depend on the layout of the assembly cell (e.g., the location of the parts storage and the fixture), the applied resources (e.g., the robot), and the assembly path. Since assembly planning precedes cell configuration in the planning workflow considered, it is reasonable to assume estimated processing times given in the input. Many papers in the literature overcome the difficulty caused by the mutual interrelation of assembly planning and cell configuration by simply minimizing the *number of changeovers*; the proposed approach can be used in this way by using unit processing and changeover times.

Monotonous binary (i.e., two-handed) assembly is assumed, i.e., each assembly task fixes the relative position of the two involved parts as required in the assembled final product, and this connection cannot be untied later. Hence, the initially separated parts are gradually built into larger subassemblies, until they are finally joined in one final product. Consequently, beyond assembling the directly involved two parts, an assembly task also connects all parts in the two corresponding subassemblies.

An *assembly plan* consists of a subset of tasks that constitute a spanning tree of G, and a total ordering of the selected tasks. The assembly plan is *feasible* if there is a technologically and geometrically feasible (explained below) way of building the product using the given sequence of tasks so that no collision occurs during the execution of any of the assembly tasks. The *assembly planning problem* consists in finding a feasible assembly plan which minimizes the total assembly time, which contains the fixture and tool changing times plus the total processing time of the selected tasks.

The computed assembly plan must be *geometrically and technologically feasible*, i.e., it has to be ensured that each selected task can be realized by a motion of the involved objects, including parts and tools, that respect the technological specification of the given task and does not incur any collisions. The characterization of the assembly motions is presented in Section 6, with further details of the applied technological and geometrical models in the Appendix.

2.1. Illustrative example

Throughout the paper, the proposed approach will be illustrated on the assembly planning problem of the standard ball valve product shown in Fig. 1. An optimal assembly plan for this product is displayed in Fig. 2. The product is built from 13 individual parts. After merging the four identical screws that join the cover to the house into a composite part, the model contains 10 parts and 11 tasks. Hence, the liaison graph of the product contains 11 edges (11 tasks), out of which

Fig. 2. Optimal assembly plan for the ball valve product. Fixtures (F) and tools (T) are shown only below the figures, without geometries.

9 edges (9 tasks) that constitute a spanning tree must be selected for inclusion in the assembly plan. An example of alternative tasks is that the O-ring that ensures sealing between the house and the cover can be placed either on the cover (see Tasks 3 in Fig. 2) or on the house, before the house and the cover themselves are joined. A part of the precedence relations between the tasks must be inferred by geometrical reasoning: for example, the two inlets must be placed on the ball (Tasks 1 and 2) before inserting the ball into the house (Task 6).

However, finding a feasible plan is just one part of the challenge; beyond that, a plan that minimizes the total assembly time is looked for. This requires considering the resources and the corresponding changeover times as well. In this example, four different fixtures can be applied: two fixtures can grasp the cover or the house (two physical fixturing devices with given geometry), or the human operator can hold one of the plastic inlets while putting the ball and the other inlet on it (two virtual fixtures without a fixed geometry, see Tasks 1 and 2 in Fig. 2). The two available tools are the human hand (without geometry) for placing and insertion tasks, and a screwdriver (with geometry) for screwing tasks. One interesting question is how the number of changeovers can be reduced while respecting the precedence relations stemming from geometry.

3. Literature review

3.1. Assembly planning

The main challenge of assembly planning lies in performing efficient optimization to maximize assembly system performance while ensuring that the complex technological and geometrical constraints are satisfied. Classical approaches to assembly planning aim to explore geometrical and technological constraints before search, and then look for a solution that respects the given constraints. An approach that generates all geometrically feasible task sequences for a product is presented in De Fazio and Whitney (1987). In Wilson and Latombe (1994) and Romney et al. (1995), the concept of non-directional blocking graphs (NDBG) is introduced to characterize the blocking relations between pairs of parts. Given the NDBG, polynomial-time graph algorithms can be applied to extract geometrically feasible assembly sequences. These sequences can be evaluated later using various performance measures related to the efficiency and ease of the assembly process. In Thomas et al. (2003), the stereographical projections of parts are used to generate constraints on the assembly sequence, and then to build the AND/OR graph representation that contains all feasible assembly sequences. A more sophisticated approach to generating precedence constraints from geometrical models is proposed in Morato et al. (2013). However, product geometry can be translated efficiently into precedence constraints between assembly tasks only in the presence of strict assumptions. In most of the above papers, assembly motions correspond to a linear translation of the part, typically, along one of the few given direction vectors. With this assumption, it is sufficient to perform pairwise collision detection between parts along a few candidate linear motions, instead of solving a complex path planning problem involving different subsets of the parts. Moreover, none of the above approaches capture resource requirements.

When geometrical and technological feasibility cannot be fully ensured by constraints generated in a preprocessing phase, the applicable approach is solving the planning problem iteratively, with constraints added during search based on the evaluation of earlier plan candidates. For this purpose, it is typical to separate the so-called *macro-level* planning problem, which is the combinatorial optimization problem responsible for task sequencing and resource assignment, from the *micro-level* problem related to evaluating the plans considering detailed geometrical and technological modes.

A Benders decomposition approach to integrating macro- and micro-level planning was presented in Kardos et al. (2016, 2017). The proposed method departs from a so-called *feature-based* representation of the assembly process to define technologically correct execution of the assembly task, and uses mixed-integer linear programming (MILP) to minimize the total assembly time, considering fixture and tool changeovers as well. In case of any collision identified on the micro level, a disjunctive precedence constraints is fed back to the macro level. The significant limitation that the liaison graph had to be a tree was lifted in a subsequent paper Kardos et al. (2020), where CP was used to solve the macro-level planning problem. Again, the subproblem provided conjunctive constraints for the macro level, but this could lead to a high number of iterations for intricate cases.

In order to speed up the search for geometrically feasible assembly sequences, Rodríguez et al. (2019) proposed iteratively refined simulations, starting with free-flying parts and ending with a detailed simulation of the complete robotic assembly system. The solution space

of sequence planning is discovered using a simple depth-first search, and any collision detected during simulation is propagated to assembly sequence planning to prune the search tree. In Hui et al. (2009), a meta-heuristic that combines a genetic algorithm with ant colony optimization is presented for solving the assembly sequencing problem, considering also the required resources. The validity of the computed sequences is checked by interactive path planning, with the support of swept volumes generated by a B-rep filling algorithm. If the path found for an assembly task is not satisfactory, either due to collisions or its difficulty, a feedback is provided to the macro level to modify the sequence. In Le et al. (2009), the authors look at the assembly planning problem from the path planning point of view, and iterate an ML-RRT algorithm (Cortés et al., 2008), which is a variant of the classical RRT path planner (Lavalle and Kuffner, 2000), until it removes all parts one-by-one from the assembly. Yet, the approach does not capture resource requirements, and settles for finding an arbitrary feasible, collision-free assembly sequence. Focusing on assembly/disassembly for maintenance, the paper Hassan and Yoon (2010) proposes an integrated approach to assembly sequence and path planning. On the macro level, a genetic algorithm aims to minimize a criterion composed of the number of gripper changes, number of orientation changes, and path length. On the micro level, a potential field path planner is used to compute collision-free (dis)assembly paths. At the end, the computed sequences and paths are validated in a virtual environment with haptic control.

Typical optimization criteria are minimizing the number of changeovers (including the changes of orientations, tools, or assembly task types), the assembly time, the tool travel distance, or the assembly cost (Bahubalendruni and Biswal, 2016). Apart from early models that solve assembly planning as a graph search problem, all formulations are NP-hard, and hence, (meta-)heuristics, soft computing techniques, and mathematical programming approaches are frequently applied for solving them (Bahubalendruni and Biswal, 2016; Rashid et al., 2012). Recent surveys on solution methods include Jiménez (2013), Hu et al. (2011), Bahubalendruni and Biswal (2016) and Neb (2019).

3.2. Logic-based benders decomposition

This paper applies a Logic-based Benders decomposition (LBBD) approach to solving the assembly planning problem. LBDD, introduced by Hooker and Ottosson (2003), generalizes Benders decomposition (Benders, 1962; Geoffrion, 1972) by replacing the linear programming dual used in the classical method with an "inference dual". By solving the "dual problem", new inequalities, so-called *Benders cuts*, are computed that exclude superfluous solutions. LBBD has been successfully applied to a host of applications, see Hooker (2011, 2019). The most typical ones are planning and scheduling (Hooker, 2007; Roshanaei et al., 2017), facility location (Fazel-Zarandi and Beck, 2012; Wheatley et al., 2015), route planning (Kloimüllner and Raidl, 2017; Fachini and Armentano, 2020), to name but a few examples. Unlike in most of the applications, in our case the constraints derived from the inference dual of the assembly planning problem will take the form of *disjunctive constraints*, which cannot be modeled by a single linear inequality in general. Such a constraint takes the form

$$\bigvee_{i\in\Gamma}(x \in C_i),$$

where Γ is a finite index set, the C_i are sets of possible values, and the constraint prescribes that x must be a member of one of these sets. Balas (1975, 1979, 1998) pioneered mathematical programming with disjunctive constraints (MP-DC). MP-DC has gained a tremendous interest over the last decades, for an overview we refer to the textbook Balas (2018) (when the C_i are polyhedra, and besides disjunctive constraints, there are only linear inequalities in the program), and to the review paper Grossmann (2002) in case of mixed-integer nonlinear programs. In this paper we consider a special case, where each term $x \in C_i$ of a disjunction can be described by a single linear inequality, and we will simply aggregate these constraints into a single inequality, see Section 4.8.

Table 1
Notation.

V	Set of parts (nodes of the liaison graph G)
T	Set of tasks (edges of the liaison graph G)
POS	Set of positions
Z	Set of tools
F	Set of fixtures
x_{tp}	Assignment variables ($T \to POS$)
q_{prs}	Connection variables (if $q_{prs}=1$ then $G_p(x)$ contains an $r-s$ path)
τ_{pz}	Tool assignment variables ($Z \to POS$)
ϕ_{pf}	Fixture assignment variables ($F \to POS$)
c_p^{tool}	Tooling cost variables ($POS \to \mathbb{R}$)
c_p^{fixt}	Fixturing cost variables ($POS \to \mathbb{R}$)

4. Formulation by a mixed-integer linear program

In this section, we present a mixed-integer linear program for modeling the assembly planning problem. Recall that the liaison graph is $G=(V,T)$, where node set V corresponds to the parts, and edge set T represents the tasks, thus, in order to ease our notation, we use terms *part* and *node*, as well as *task* and *edge* interchangeably.

Recall that an assembly plan (aside from tools and fixtures) can be represented by a selection of $n := |V|-1$ tasks along with a bijective mapping of the selected tasks to positions from the set $POS := \{1,\dots,n\}$.

To describe a solution, we introduce the *assignment variables* x_{tp} that specify for each task $t \in T$ and position $p \in POS$, whether task t is assigned to position p ($x_{tp}=1$) or not ($x_{tp}=0$). An assignment is *feasible* if and only if exactly one task is assigned to each position, and each task is assigned to at most one position. Note that if there are more tasks than positions, then some tasks are necessarily unassigned in any feasible assignment. Clearly, any feasible assignment selects a spanning tree in the liaison graph G. Let $G_p(x)$ denote the subgraph of G containing those edges that correspond to the tasks assigned to the first p positions by a feasible assignment x. We also need the *connection variables* q_{prs} expressing whether parts r and s belong to the same subassembly after performing the first p tasks in the determined sequence. Basically, we will only require q_{prs} to be 0, if there is no $r-s$ path in $G_p(x)$. Finally, to express the objective function, we will use the variables c_p^{tool} and c_p^{fixt} for expressing the tooling and fixturing costs for each position $p \in POS$ (see Table 1).

Now we are ready to describe the mathematical programming formulation of the problem. In the objective function, we aim to minimize the *total assembly time*, that is, the sum of processing times of the selected tasks, and the total changeover time incurred by the tool and fixture changes at each step:

$$\text{Minimize} \sum_{p\in POS} c_p^{tool} + \sum_{p\in POS} c_p^{fixt} + \sum_{t\in T}\sum_{p\in POS} \rho_t x_{pt}. \tag{1}$$

Subject to the constraints (to be declared in subsequent sections):

Assignment: (2a)–(2c),
Connectivity: (3a)–(3i),
Tooling and fixturing: (4a)–(4k),
Maximal subassemblies: (5),
Component limits: (6),
Graspable constraint: (7),
Conjunctive: (8),
Disjunctive: (9).

4.1. Assignment

We formulate the assignment constraints as follows:

$$\sum_{p\in POS} x_{tp} \le 1, \text{ for all } t \in T \tag{2a}$$

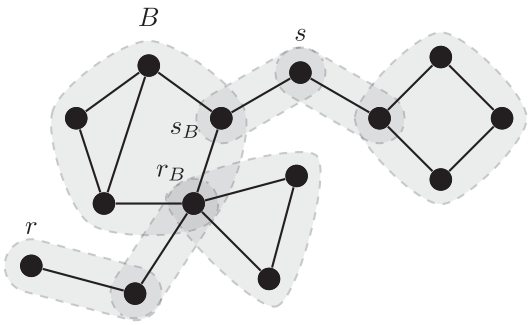

Fig. 3. Example for a graph and its biconnected components. Biconnected components are indicated with dashed boundary. Nodes r_B and s_B are the first and last nodes, respectively, that belong to biconnected component B when traversing along a path from node r to node s.

$$\sum_{t\in T} x_{tp} = 1, \text{ for all } p \in POS \tag{2b}$$

$$x_{tp} \in \{0,1\}, \text{ for all } t \in T,\ p \in POS. \tag{2c}$$

Each task can be assigned to at most one position by (2a), to each position exactly one task must be assigned by (2b), and the x_{tp} variables are binary by (2c).

4.2. Connectivity constraints

The purpose of the connectivity constraints is to ensure that $q_{prs} = 0$ for some $p \in POS$, and $r, s \in V$ if there is no $r-s$ path in $G_p(x)$. This is easy to guarantee by linear constraints if the liaison graph is a tree, see Kardos et al. (2017), but it becomes a challenging modeling problem if G contains one or more cycles. The connectivity constraints (for general G), will be declared in several steps. We start with the following set:

$$q_{prs} = q_{psr}, \text{ for all } p \in POS,\ r, s \in V \tag{3a}$$

$$q_{nrs} = 1, \text{ for all } r, s \in V \tag{3b}$$

$$q_{0rs} = 0, \text{ for all } r, s \in V : r \neq s \tag{3c}$$

$$q_{prr} = 1, \text{ for all } p \in POS \cup \{0\},\ r \in V \tag{3d}$$

$$q_{prs} \in \{0,1\}, \text{ for all } p \in POS \cup \{0\},\ r, s \in V. \tag{3e}$$

Constraint (3a) expresses the symmetry of the variables. Constraint (3b) requires that all of the parts must be connected after the final, nth step. In the beginning, distinct parts are unconnected due to constraint (3c), however, by constraint (3d), any single part is always connected with itself.

Consider the liaison graph G. Note that $G_p(x)$ contains an $r-s$ path if and only if each $r-s$ cut $[S, \bar{S}]$ of G contains at least one edge of $G_p(x)$. Therefore, we need q_{prs} to be zero, if there is an $r-s$ cut containing no edge of $G_p(x)$. If G is a tree, then $G_p(x)$ is a forest, and the latter condition is easy to verify since each $r-s$ cut corresponds to an edge of the unique $r-s$ path in G. However, if G has biconnected components, then the number of $r-s$ cuts can be large. Let $\mathcal{BC}_G$ be the set of biconnected components of G, and for an edge $e \in E(G)$, let $B_G(e) \in \mathcal{BC}_G$ be the unique biconnected component that contains e, and for a node $v \in V(G)$, let $B_G(v) \subseteq \mathcal{BC}_G$ be the set of biconnected components that contain v. The cardinality $|B|$ of a biconnected component $B \in \mathcal{BC}_G$ is the number of its nodes. It is well-known that one can determine the set $\mathcal{BC}_G$ in linear time using a depth-first-search procedure (Hopcroft and Tarjan, 1973). In Fig. 3 we depict a graph and its biconnected components.

Consider a biconnected component $B \in \mathcal{BC}_G$ in the form of $B = G[S] = (S, \sigma(S))$ for a certain subset $S \subseteq V(G)$. Let $\mathcal{K}$ be the set of all inclusion-wise minimal cuts of $G[S]$, and $\mathcal{K}_{rs} \subseteq \mathcal{K}$ be the set of all inclusion-wise minimal $r-s$ cuts of $G[S]$. For each cut $C \in \mathcal{K}$ and position $p \in POS$ we introduce a binary variable $y_{p,C}$ with the following meaning: if $y_{p,C} = 1$ then C contains at least one edge of $G_p(x)$. By this, for each biconnected component $B = (S, \sigma(S))$ we add the following constraints to the model:

$$y_{p,C} \leq \sum_{i=1}^{p} \sum_{t\in[X,\bar{X}]} x_{ti}, \text{ for all } p \in POS,\ C = [X, \bar{X}] \in \mathcal{K} \tag{3f}$$

$$q_{prs} \leq y_{p,C}, \text{ for all } p \in POS,\ r, s \in S : r \neq s,\ C \in \mathcal{K}_{rs} \tag{3g}$$

Constraint (3f) ensures that $y_{p,C} = 0$ holds if $C \cap E(G_p(x)) = \emptyset$, and thus $q_{prs} = 0$ holds for all nodes r and s such that C is an $r-s$ cut due to constraint (3g).

The set of inclusion-wise minimal cuts of a graph can be enumerated in linear time per cut (Tsukiyama et al., 1980). We also remark that as long as we only have lower bounds on variables q_{prs} in the model, it suffices to require q_{prs} to be 0 if there is no $r-s$ path in $G_p(x)$.

Now we turn to paths in the liaison graph. Assume that parts r and s belong to different biconnected components of the liaison graph G, that is, $B_G(r) \cap B_G(s) = \emptyset$. Consider an arbitrary $r-s$ path P in G. Let $\tilde{E}_{rs} := \{e \in E(P) : |B_G(e)| = 2\}$ be the set of edges of the path that belongs to a biconnected component which consists of a single edge, and let $\tilde{B}_{rs} := \{B_G(e) \subseteq \mathcal{BC}_G : e \in E(P),\ 3 \leq |B_G(e)|\}$ be the set of biconnected components with at least three nodes that the path traverses (i.e., uses at least one edge from it). Note that the definitions of $\tilde{E}_{rs}$ and $\tilde{B}_{rs}$ do not depend on path P but only on r and s. Traversing along path P from node r to node s, for each $B \in \mathcal{BC}_G$ let r_B be the first and s_B be the last node that belongs to the component, respectively. Note that $r_B = s_B$ might hold. For the designated nodes r and s in Fig. 3, the corresponding sets $\tilde{E}_{rs}$ and $\tilde{B}_{rs}$ contain three edges, and one component, respectively, moreover, for component B nodes r_B and s_B are also depicted. For a later use we also define the set $\tilde{\mathcal{K}}_{rs} := \bigcup_{B\in\tilde{B}_{rs}} \mathcal{K}_{r_B,s_B}$ of cuts.

For each position $p \in POS$ and for each pair of parts $r, s \in V$ such that $B_G(r) \neq B_G(s)$ we have the following constraints:

$$q_{prs} \leq \sum_{i=1}^{p} x_{ti}, \text{ for all } t \in \tilde{E}_{rs} \tag{3h}$$

$$q_{prs} \leq q_{p,r_B,s_B}, \text{ for all } B \in \tilde{B}_{rs} \tag{3i}$$

Constraints (3h) and (3i) ensure that $q_{prs} = 0$ if there is no $r-s$ path in $G_p(x)$.

Note that we do not require q_{prs} to be 1 if there exists an $r-s$ path in $G_p(x)$ since it would demand a large number of constraints, and due to our preliminary experiments it makes the problem hard to solve. Thus, upper bounds on variables q_{prs} cannot be applied directly, but the biconnected decomposition described above can be used. That is, for example, instead of constraint $q_{prs} \leq 0$, the following inequality can be used:

$$\sum_{t\in\tilde{E}_{rs}} \sum_{i=1}^{p} x_{ti} + \sum_{C\in\tilde{\mathcal{K}}_{rs}} y_{p,C} \leq |\tilde{E}_{rs}| + |\tilde{\mathcal{K}}_{rs}| - 1.$$

4.3. Tooling and fixturing

In order to express the constraints regarding tools and fixtures, we add the following constraints to the model:

$$\sum_{z\in Z} \tau_{pz} = 1, \text{ for all } p \in POS \tag{4a}$$

$$\tau_{pz} \in \{0,1\}, \text{ for all } p \in POS \cup \{0\},\ z \in Z \tag{4b}$$

$$\tau_{0z} = 0, \text{ for all } z \in Z \tag{4c}$$

$$\sum_{f\in F} \phi_{pf} = 1, \text{ for all } p \in POS \tag{4d}$$

$$\phi_{pf} \in \{0,1\}, \text{ for all } p \in POS \cup \{0\},\ f \in F \tag{4e}$$

$$\phi_{0f} = 0, \text{ for all } f \in F \tag{4f}$$

$$x_{tp} + \tau_{pz} \leq 1, \text{ for all } p \in POS,\ t \in T,\ z \in Z \setminus Z_t \tag{4g}$$

$$x_{tp} + \phi_{pf} - q_{p-1,v_f,a} - q_{p-1,v_f,b} \leq 1, \quad \begin{array}{l}\text{for all } p \in POS,\ f \in F \\ t = \langle a, b\rangle \in T.\end{array} \tag{4h}$$

Constraints (4a) and (4d) ensure that for each position exactly one tool and exactly one fixture must be assigned. Inadmissible task-tool assignments are prohibited by constraint (4g). With constraint (4h) we ensure that a task can be used in a fixture only if the grasped part of the fixture is in one of the two subassemblies which are connected by the task.

Recall that d_f and d_z denote the sequence independent changeover time for fixture f and tool z, respectively. We add the following constraints to the model:

$$c_p^{fixt} \geq d_f(\phi_{pf} - \phi_{p-1,f}), \text{ for all } p \in POS,\ f \in F \tag{4i}$$

$$c_p^{tool} \geq d_z(\tau_{pz} - \tau_{p-1,z}), \text{ for all } p \in POS,\ z \in Z \tag{4j}$$

$$c_p^{fixt} \geq 0,\ c_p^{tool} \geq 0, \text{ for all } p \in POS. \tag{4k}$$

Clearly, the fixture changeover time variable c_p^{fixt} is forced to take a value of at least d_f if and only if $\phi_{pf} = 1$, while $\phi_{p-1,f} = 0$, and in an optimal solution equality holds. An analogous statement holds for the tool changeover time variables.

4.4. Maximal subassemblies

For a maximal subassembly V_f that can be held in fixture f we add the following constraints to the model:

$$\phi_{pf} + \sum_{t \in \tilde{E}_{s,v_f}} \sum_{i=1}^{p} x_{ti} + \sum_{C \in \tilde{\mathcal{K}}_{s,v_f}} y_{p,C} \leq |\tilde{E}_{s,v_f}| + |\tilde{\mathcal{K}}_{s,v_f}|$$
$$\text{for all } p \in POS,\ s \in V \setminus V_f. \tag{5}$$

These constraints are the transcription of constraints $\phi_{pf} + q_{p,v_f,s} \leq 1$ applying the technique described at the end of Section 4.2. That is, if fixture f is used in position p ($\phi_{pf} = 1$), then no part s out of V_f can be assembled with the grasped part v_f in the first p positions ($q_{p,v_f,s} = 0$ for all $s \in V \setminus V_f$). Note that this constraint does not forbid to join prohibited parts (i.e., one part from V_f and one from $V \setminus V_f$) in fixture f in the pth position.

4.5. Component limits

Clearly, from any connected component $(S, \sigma(S))$ at most $|S| - 1$ edges can be selected. We add the following constraints to the model:

$$\sum_{t \in \sigma(S)} \sum_{p \in POS} x_{tp} \leq |S| - 1 \quad \text{for all} \quad B = (S, \sigma(S)) \in \mathcal{BC}_G. \tag{6}$$

Note that it is not necessary to add these constraints to the model (i.e., the model is also valid without these constraints), however, due to our preliminary experiments they strengthen the model.

4.6. Graspable constraint

Clearly, a task $t = \langle a, b\rangle \in T$ can be performed only if either part a or b is already connected to a graspable part (that is, a part which can be grasped by a fixture). Let $V^{grasp} := \{v \in V : v = v_f \text{ for some } f \in F\}$ be the set of graspable parts. We add the following constraints to the model:

$$x_{tp} \leq \sum_{u \in V^{grasp}} \left(q_{p-1,u,a} + q_{p-1,u,b}\right) \quad \text{for all } t \in T,\ p \in POS. \tag{7}$$

These constraints are satisfied by all feasible solutions, and they are added to the model to strengthen the LP relaxation.

4.7. Conjunctive constraints

A *conjunctive constraint* is formalized as $\langle t \oplus W\rangle$ where $t \in T$ and $W \subset V$, and it prescribes that task t can be performed in some position of the assembly sequence only if the parts in W are already joined by some assembly tasks in positions $1, \ldots, p-1$. To express this constraint, we choose and fix a part $r \in W$, and add the following inequalities to the model:

$$x_{pt} \leq q_{p-1,r,s}, \text{ for all } p \in POS,\ s \in W \setminus \{r\}. \tag{8}$$

We expect to generate all such constraints in an offline preprocessing phase (using geometric reasoning), and not during search. Note that these constraints can be aggregated into a single one, however, our preliminary experiments showed that (8) is more efficient.

4.8. Disjunctive constraints

A *disjunctive constraint* is used to express that the assembly sequence must satisfy at least one from a couple of logical conditions. It is formalized as $\langle t \ominus W_1, W_2, z, f\rangle$ with $t \in T$, $W_1, W_2 \subseteq V$, $z \in Z$, $f \in F$, where task $t = \langle a, b\rangle$ joins parts a and b. The constraint states that either task t is not executed at all, or:

- part a is *not* connected to at least one of the parts in W_1 before the execution of task t, or
- part b is *not* connected to at least one of the parts in W_2 before the execution of task t, or
- tool z is *not* used for the execution of task t, or
- fixture f is *not* used for the execution of task t.

Some fields of the constraint can be left empty (but task t is always specified), which will be denoted by $W_1 = \emptyset$ or $z = \emptyset$, etc. Hence, $\langle t \ominus W_1, \emptyset, \emptyset, f\rangle$ and $\langle t \ominus \emptyset, \emptyset, z, f\rangle$ are all valid constraints. In such cases, the corresponding terms of the disjunction are ignored. The number of disjunctive constraints is usually exponential in the number of parts, fixtures, and tools. Therefore, we only generate violated constraints during the search, see Section 5.

For a disjunctive constraint $\langle t \ominus W_1, W_2, z, f\rangle$ with $t = \langle a, b\rangle$, we add the following linear constraint to the model for all $p \in POS$:

$$x_{tp} + \tau_{pz} + \phi_{pf} + \sum_{w \in W_1} \left(\sum_{u \in \tilde{E}_{aw}} \sum_{i=1}^{p-1} x_{ui} + \sum_{C \in \tilde{\mathcal{K}}_{aw}} y_{p-1,C} \right) + \tag{9}$$
$$+ \sum_{w \in W_2} \left(\sum_{u \in \tilde{E}_{bw}} \sum_{i=1}^{p-1} x_{ui} + \sum_{C \in \tilde{\mathcal{K}}_{bw}} y_{p-1,C} \right) \leq 2 + |\tilde{E}_{aw}| + |\tilde{\mathcal{K}}_{aw}| + |\tilde{E}_{bw}| + |\tilde{\mathcal{K}}_{bw}|.$$

If task t is not executed at all, then $x_{tp} = 0$ for all $p \in POS$, and thus constraint (9) holds. Otherwise, if task t is executed, say in position p, i.e., $x_{tp} = 1$, then the rest of the left-hand side can be at most $1 + |\tilde{E}_{aw}| + |\tilde{\mathcal{K}}_{aw}| + |\tilde{E}_{bw}| + |\tilde{\mathcal{K}}_{bw}|$, that is, either $\tau_{pz} = 0$ (i.e., tool z is not used during execution of t), or $\phi_{pf} = 0$ (i.e., fixture f is not used during execution of t), or $\sum_{w \in W_1}(\sum_{u \in \tilde{E}_{aw}} \sum_{i=1}^{p-1} x_{ui} + \sum_{C \in \tilde{\mathcal{K}}_{aw}} y_{p-1,C}) \leq |\tilde{E}_{aw}| + |\tilde{\mathcal{K}}_{aw}| - 1$ (i.e., part a is not connected to at least one of the parts in W_1 before the execution of t), or $\sum_{w \in W_2}(\sum_{u \in \tilde{E}_{bw}} \sum_{i=1}^{p-1} x_{ui} + \sum_{C \in \tilde{\mathcal{K}}_{bw}} y_{p-1,C}) \leq |\tilde{E}_{bw}| + |\tilde{\mathcal{K}}_{bw}| - 1$ (i.e., part b is not connected to at least one of the parts in W_2 before the execution of t).

Note that if some field of the constraint is left empty, then the corresponding expressions are omitted and the right-hand side is modified. For example, the constraints corresponding to a disjunctive constraint of the form $\langle t \ominus \emptyset, \emptyset, z, f\rangle$ are $x_{tp} + \tau_{pz} + \phi_{pf} \leq 2$ for all $p \in POS$. Also note that if at least one of the fields W_1 or W_2 are given, then constraint (9) is omitted for position 1, since it is undefined and the disjunction is always satisfied in that case.

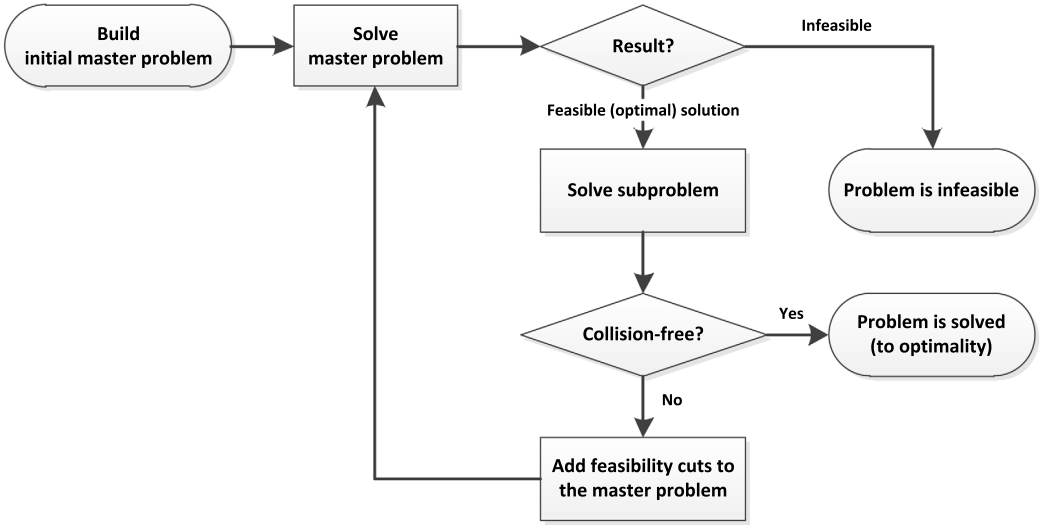

Fig. 4. Sketch of the LBBD solution approach.

4.9. Solving the MILP

The MILP formulation presented above can be solved by arbitrary MILP solver. Yet, in order to improve the default branch-and-cut procedure of the commercial solver, several problem-specific branching strategies were experimented. Surprisingly, the best results were yielded by a simple strategy that assigns fixtures to given positions. That is, in each node of the search tree, the first position p without an assigned fixture is determined (i.e., there is no fixture f such that ϕ_{pf} is fixed to 1). Then, $|F|$ branches are created by assigning the ith fixture to position p on the ith branch, i.e., fixing ϕ_{pf_i} to 1. When for each position there is an assigned fixture, the default branching rule of the solver is used. The evaluation of this branching strategy is presented in the computational experiments.

5. Logic-based Bender's decomposition

Since the number of disjunctive constraints that may have to be added to the mathematical program (1)–(8) can be exponential, and identifying all of them may be an arduous task, we apply logic-based Bender's decomposition to solve the above problem. Initially, the master problem consist of the constraints (1)–(8), and the disjunctive constraints (9) are added in gradually. This means that initially, geometric and technological feasibility are neglected.

The sketch of the solution procedure is depicted in Fig. 4. Once an assembly plan is built by solving the master problem, the subproblem checks whether this assembly plan admits a feasible, collision-free realization. If not, then one or more constraints violated by the current assembly plan are generated and added to the master problem, and the procedure is repeated until no violated constraint is found. Since the master problem is solved to optimality in each iteration, the final, collision-free assembly plan is also optimal for the entire assembly problem.

To see this, we call the subproblem solver *complete* if for any assembly plan $(x^*, \phi^*, \tau^*, q^*)$ satisfying (1)–(8) and those disjunctive constraints (9) that are already added to the master problem, if and only if the plan is geometrically or technologically infeasible, it returns a disjunctive constraint $\langle t \ominus W_1, W_2, z, f \rangle$ that is violated by the current plan, but should be satisfied by all feasible plans.

Proposition 1. *If the subproblem solver is complete, then the above procedure terminates in a finite number of steps with an optimal solution for the assembly planning problem.*

Proof. Since the constraints returned by the subproblem solver must be respected by all assembly sequences (this follows from the completeness of the subproblem solver), the procedure terminates with an optimal solution provided that in the last iteration, an optimal solution is found for the master problem. The finite number of steps is ensured by the fact that the number of distinct disjunctive constraints is finite. □

The subproblem solver does work on a geometric representation of the components, and that of the assembly plan. Hence, it solves the *inference dual*, a method initiated by Hooker and Ottosson (2003). While Hooker and Ottosson bound the objective function value of the master problem, in our application the objective function value is influenced only indirectly by the disjunctive cuts, i.e., they can exclude integer solutions of the master problem, which represent technologically infeasible assembly sequences.

6. Geometrical reasoning for solving the inference dual

6.1. Overview

In the proposed decomposition approach, the inference dual investigates the geometrical feasibility of the assembly plans by checking if the plans can be realized without any collisions. Validation is performed separately for each task in the fully specified assembly plan. For an arbitrary static configuration of the objects in the 3D space, potential collisions or distances between pairs of objects can be queried using open source collision detection libraries, such as the Proximity Query Package (PQP) (Larsen et al., 2000) or the Flexible Collision Library (FCL) (Pan et al., 2012). Collision detection for static configurations offered directly by these libraries has been extended to checking continuous motions using a *conservative advancement* (CA) approach (Schwarzer et al., 2004).

The feature-based assembly model defines the following important relative positions of the base and the moved subassemblies during their motion (see Fig. 5):

- The *final position* P_F is the relative position of the two parts or subassemblies in their assembled state.
- The *near position* P_N is a relative position near to the final position without touching or colliding geometries. During the execution of the corresponding task, the subassembly is moved from P_N to P_F by a linear translation defined by the technological parameters of the corresponding task.

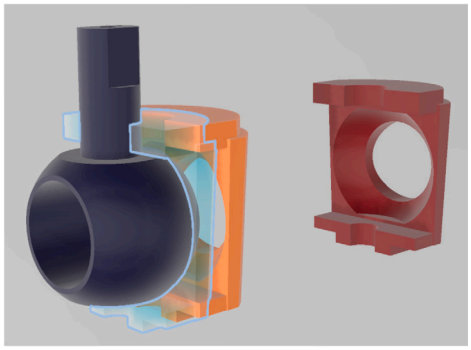

Fig. 5. Noteworthy positions of the moved part during the execution of a task: the remote (red), near (yellow), and final (transparent with wireframe) position of the inlet relative to the ball in the ball valve assembly.

- The *remote position* P_R is a virtual position of the moved subassembly far from the base subassembly and the fixture in the open space where the collision-free motion of the moved subassembly can be guaranteed. Since assembly planning precedes workstation layout design in the proposed workflow, the true starting position of the moved subassembly, such as a storage location, is not known. Hence, the virtual remote position will be used instead.

From now on, the fully defined linear motion from P_N to P_F is called the *feature motion*, whereas the motion from P_R to P_N, which is not determined in the assembly task, is named *approach motion*.

Geometrical validation takes place in the *configuration space* of the relative positions of the moved subassembly to the base subassembly, denoted by C. Let C_{free} denote the set of collision-free configurations. A fully defined linear motion $\overline{P_1P_2}$ is said to be collision-free, denoted by $\overline{P_1P_2} \in C_{\text{free}}$, if it is fully located in C_{free}, i.e., $P' \in C_{\text{free}}$ for any $P' = (1-t)P_1 + tP_2$ with $t \in [0,1]$.

If there exists no predefined motion between two end points P_1 and P_2, then P_1 and P_2 can be connected without any collisions, denoted by $P_1 \overset{\rightsquigarrow}{P}_2 \in C_{\text{free}}$, if there exists a finite sequence of way points $\{q_1, q_2, \ldots, q_k\}$ with $q_1 = P_1$ and $q_k = P_2$ such that $\overline{q_iq_{i+1}} \in C_{\text{free}}$ for all $i = 1, \ldots, k-1$.

Now, a task is collision-free if and only if for the above defined positions it holds that $\overline{P_NP_F} \in C_{\text{free}}$ and $P_R \overset{\rightsquigarrow}{P}_F \in C_{\text{free}}$. The following two subsections present the algorithms to verify these two conditions.

6.2. Validation of the feature motion

The validation of the fully defined feature motion $\overline{P_NP_F}$ in a given assembly task consists in (1) identifying the base and moved subassemblies in the task from the assembly sequence; (2) positioning the fixture geometry w.r.t. the base, as well as the tool geometry w.r.t. the moved subassembly; and (3) performing collision detection for $\overline{P_NP_F}$ using the above CA method. In case of any collision, all colliding pairs of objects are unambiguously identified. Each colliding pair consists of one base object (a part or the fixture) and one moved object (a part or the tool).

In case of collisions, the algorithm generates a disjunctive constraint $\langle t \ominus W_1, W_2, z, f \rangle$ for each colliding pair of objects as follows (see also Section 4.8 and Eq. (9)):

- Task t is the task that incurs a collision.
- If the colliding base object is part w_1, then $W_1 = \{w_1\}$ and $f = \emptyset$; otherwise, the colliding base object is fixture f_1, which incurs $W_1 = \emptyset$ and $f = f_1$.
- Similarly, if the colliding moved object is part w_2, then $W_2 = \{w_2\}$ and $z = \emptyset$; otherwise, the colliding moved object is tool z_1, which incurs $W_2 = \emptyset$ and $z = z_1$.

The generated constraint expresses that at least one of the colliding objects, w_1, w_2, f_1, or z_1 must not be present during the execution of task t.

Finally, in order to speed up the validation of feature motions, caching is applied: for each task and each pair of objects, successful collision checks (i.e., tests resulting in no collision) are stored in a cache. In case the same pair of objects in the same task is investigated in another plan later during subsequent iterations, this precludes the repeated execution of the collision test. Observe that there is no need to store the results of unsuccessful collision tests, since the generated disjunctive constraint will prevent the repeated occurrence of the same collision.

6.3. Validation of the approach motion

In contrast to the feature motion, there is no predefined motion available for the approach motion $P_R \overset{\rightsquigarrow}{P}_F$. Accordingly, the validation of the approach motion requires solving a path planning problem, where the base subassembly and the fixture define the obstacles for moving the ensemble of the moved subassembly and the tool, represented as a single 3D solid object. For solving this path planning problem, an implementation of the *Rapidly-exploring Random Trees* (RRT) (Lavalle and Kuffner, 2000) algorithm is applied in the configuration space of the relative positions of the base and the moved subassemblies. RRT is known to be probabilistically complete, i.e., given sufficient computation time, it finds a solution with a probability converging to one. On the other hand, RRT is unable to provide a proof of infeasibility. Hence, search is stopped if a given iteration limit is reached, in which case the approach motion is regarded as colliding, and a single disjunctive constraint $\langle t \ominus W_1, W_2, z, f \rangle$ is generated as follows (see also Section 4.8 and Eq. (9)):

- Task t is the task incurring a collision.
- Part sets W_1 and W_2 contain the complete base and moved subassemblies.
- Fixture f and tool z are the resources assigned to task t in the plan.

The generated constraint states that task t cannot be executed with this configuration of the base and moved subassemblies (or with a superset of these subassemblies), by using the given fixture and tool.

It is highlighted that in almost all cases, the moved subassembly can be translated directly between the remote and the near positions. This direct translation is tested before performing actual RRT search. For typical assemblies, this greatly reduces the time required for the validation of the approach motion. Since path planning is a computationally demanding problem, approach motions are validated only if all feature motions in the plan are feasible. Also, validation is interrupted upon the first failure.

The generated constraint could be strengthened by executing path planning problems with reduced subassembly configurations. One straightforward approach can be trying to eliminate parts one-by-one from the two subassemblies using direct translation. Nevertheless, collisions during the approach motion were atypical for all the investigated industrial use cases (see Section 7). Therefore, the strengthening of the constraints shall be the focus of future research after identifying applications where this requirement is relevant.

6.4. Discussion on the generated disjunctive constraints

The above disjunctive constraints aim at avoiding a collision by formulating the cause of the collision in terms of part connectivity and resource assignments. For a feature motion, a collision can be characterized by the triplet of the involved task, the base object, and the moving object. It can be observed that the disjunctive constraint $\langle t \ominus W_1, W_2, z, f \rangle$, as defined in Section 6.2, provides a *necessary* and

sufficient condition for avoiding the particular collision, as it expresses that upon the execution of the given task, either the colliding base object or the colliding moving object must not be present. In contrast, the disjunctive constraints proposed in Kardos et al. (2020) focus on the precedence relations between the tasks (edges) that connect the colliding parts (vertices) in the liaison graph in the particular master solution. This is a necessary, but not sufficient condition for avoiding the given collision: if the liaison graph contains cycles, then the two colliding objects can be connected by multiple paths, and the disjunctive constraint of Kardos et al. (2020) precludes only one of those paths. Consequently, Kardos et al. (2020) can require as many iterations to prevent a collision as the number of paths between the two involved vertices, whereas the proposed approach generates a disjunctive constraint that eliminates the collision in a single iteration. If the liaison graph is a tree, then the two approaches are equivalent. Moreover, both types of feasibility cuts are definitely stronger than a classical no-good cut that excludes only a specific task sequence with a specific resource assignment.

For collisions during the incompletely defined approach motion, the disjunctive constraint proposed in Section 6.3 is a *necessary*, but *not sufficient* condition for avoiding future collisions during the execution of a specific task, since specific parts or resources whose presence results in geometrical infeasibility cannot be identified. Yet, the proposed constraints on part connectivity and resource assignment are definitely stronger than the earlier cuts of Kardos et al. (2020) that preclude only a subset of the task sequences that realize the given connectivity status.

7. Experimental evaluation

Experimental evaluation investigated two questions: the applicability and effectiveness of the overall planning approach in industrial case studies, and the computational efficiency of the disjunctive programming approach to solve the Benders master problem on a large set of randomly generated benchmark instances of different sizes. For the sake of these experiments, the proposed models and algorithms were implemented in a prototype planning system. The master problem solver was implemented in the C++ programming language using LEMON[1] for graph algorithms, and the callable library of FICO Xpress.[2] The subproblem solver was implemented in C++ using the Proximity Query Package (PQP) of Larsen et al. (2000) for collision detection and OpenGL visualization. Part geometries were available in triangle mesh models, which were parsed using the Assimp model loader.[3]

The performance of the proposed solver was also compared to the CP approach adapted from Kardos et al. (2020). The adaptation of the problem model involved taking the sum of the tool and fixture changeover times instead of their maximum in the objective (1); replacing the original weight limit for the fixtures with the maximal subassemblies constraint (5); and using the stronger disjunctive cuts on part connectivity proposed in this paper (9) instead of the precedence constraints of Kardos et al. (2020). The CP approach was implemented in MiniZinc constraint modeling language[4] using Google OR-Tools CP solver.[5] Since the performance with the most recent version 9.0.9048 (9.0, shortly) of OR-Tools is remarkably worse than what could be expected based on Kardos et al. (2020), we repeated the experiments with version 7.1.6720 (7.1, shortly) used in that paper, and present the results with both versions wherever relevant.

[1] Library for Efficient Modeling and Optimization in Networks, version 1.3.1, https://lemon.cs.elte.hu/trac/lemon.

[2] FICO Xpress Optimization, version 8.8.0, https://www.fico.com/en/products/fico-xpress-optimization.

[3] Open Asset Import Library, version 5.0.0, https://www.assimp.org/.

[4] MiniZinc open-source constraint modeling language, version 2.5.5, https://www.minizinc.org.

[5] OR-Tools optimization suite, versions 9.0.9048 and 7.1.6720, https://developers.google.com/optimization.

Fig. 6. Industrial case study: automotive supercharger.

7.1. Industrial case studies

The applicability and effectiveness of the approach was investigated on two real industrial sample products of medium complexity: the standard ball valve introduced in the illustrative example in Section 2.1, and an automotive supercharger assembly, see Fig. 6.

The supercharger assembly consists of 29 individual parts. After merging multiple elementary tasks and parts into so-called composites whenever, by engineering considerations, those tasks and parts must be handled together (e.g., merging four parallel-axis screws that join the same parts into a single composite part, as well as the four screwing tasks into a single composite screwing task), the processed model contains 18 parts joined by 17 assembly tasks arranged in a tree-structured liaison graph (see the number of raw/processed parts in Table 2).

Five different fixtures can be applied for the assembly process, including an option that the human operator holds a given base part in his hand, and uses his other hand to join the moved part. The geometrical model of these fixtures was not available in the industrial data set, hence, they were used in the master problem but ignored during collision detection in the subproblem (see the total number of fixtures and fixtures with geometry in Table 2). Two tools were assumed: a screwdriver for the screwing tasks, modeled as a rigid body with a given geometry; and a human hand for placing and insertion tasks, whose geometry could not be specified as a rigid body, and therefore, was excluded from collision detection (see the total number of tools and tools with geometry).

Over 1.3 million triangles were required to capture the complex geometry of this product with a suitable resolution. A particular difficulty with this product is that some assembly tasks exploit the flexibility of the parts (e.g., insulation and cable connectors) to join them. For this reason, the appropriate pairs of parts had to be excluded from collision detection when checking the corresponding task.

The reported experiments were run on a laptop computer with Intel i7 2.70 GHz CPU and 16 GB RAM under Windows 10 operating system. Experimental results are presented in Table 2. The upper part of the table displays problem sizes, the middle part presents the number and the type of disjunctive constraints generated by the subproblem, whereas the lower part shows the computation times. The total solution time is divided into three parts: *subproblem time* contains the total time required for collision detection, path planning, and constraint generation in all the iterations. Similarly, *master time* reflects the total computation time required for solving the series of master problems in every iteration. Model *loading time* is by far dominated by the time of parsing the large STL geometries. Since loading time is independent

Table 2
Characteristics of the industrial case studies and computational results.

	Supercharger		Ball valve	
Num. tasks	17		11	
Num. parts (raw/processed)	29/18		13/10	
Num. fixtures (total/with geom.)	5/0		4/2	
Num. tools (total/with geom.)	2/1		2/1	
Num. triangles	1 327 738		148 546	
Num. iterations	4		2	
Num. disjunctive constraints	33		3	
Disj. from part–part collision	26		3	
Disj. from part-tool collision	7		-	
Loading time [s]	12.8		1.2	
Subproblem time (total) [s]	6.6		0.9	
Subproblem time (per iteration) [s]	1.7		0.5	
Master solver	MILP	CP v7.1	MILP	CP v7.1
Master time (total) [s]	25.0	7183.0	2.0	2.5
Master time (per iteration) [s]	6.3	1795.8	1.0	1.3
Total computation time [s]	31.6	7189.6	2.9	3.4

of the proposed algorithms, it is excluded from the *total computation time* reported in the table. Moreover, *master times* are reported for two versions of the master problem solver: the MILP solution approach proposed in this paper, and the adapted version of the CP approach from Kardos et al. (2020) as a reference solver.

The results demonstrate that the proposed decomposition approach with the MILP master solver could solve both case studies efficiently: the supercharger assembly required 4 iterations and a total computation time of 31.6 s to find the optimal solution, which was dominated by the solution time of the Benders master problem. During these iterations, 33 disjunctive constraints were generated by the subproblem, mostly due to part-to-part collisions, and partly due to tool-to-part collisions along the feature motion. For the ball valve, the constraints generated by the subproblem in the first iteration successfully eliminated all collisions, resulting in a feasible and optimal plan in the second iteration, with a very short total computation time of 2.9 s (see the computed plan in Fig. 2). It is noted that other types of collisions, such as collisions between a tool and a fixture or collisions during the approach motion could also be observed in some preliminary experiments, but these were atypical, since product designs and equipment designs are formed to preclude accessibility issues if the parts are assembled in a feasible order.

It is also stimulating to compare these results to those achieved using a CP master solver, adapted from Kardos et al. (2020). The two solution approaches found equivalent master solutions in each iteration, and computation times were also comparable on the smaller ball valve problem. However, on the more challenging supercharger problem, the CP approach required considerably, nearly 300 times higher computation times than the proposed MILP. Yet, these results do not contradict the findings of Kardos et al. (2020): there, the master solver was run with a time limit of 120 s per iteration, sometimes terminating with a suboptimal solution. A potential cause of some further increase of computation time can be using the *sum* instead of the *maximum* of the tool and fixture changeover times in the objective, since sum-type objectives are usually more challenging for CP solvers. The experiments were repeated with the most recent version 9.0 of OR-Tools, but that version terminated with suboptimal solutions or no feasible solution at all even with very high time limits.

7.2. Computational experiments on generated instances

Computational experiments on randomly generated instances were performed in order to find the limits of the proposed MILP solution approach, that is, to find the maximum instance size that can be solved in a reasonable time, and to compare this solution approach to the CP approach of Kardos et al. (2020).

Since the computational challenge lies in solving the master problem, and because the randomized generation of product geometries could hardly result in realistic products, the experiments focused solely on the master problem. This way, one problem instance corresponds to one iteration in the complete Benders procedure. Yet, to cover all aspects that can arise during the Benders iterations, artificially generated disjunctive constraints were also added to the instances, corresponding to cuts computed in previous iterations by the subproblem solver. All experiments were performed on a workstation with an i9-7960X 2.80 GHz CPU with 16 cores, under Debian 9 operating system using 4 threads.

Due to space limitations, a Supplementary Material is associated to the paper at hand, which contains

- the description of the procedure for generating the instances,
- the properties of the generated five instance families,
- the evaluation of the proposed MILP approach on these instances,
- an analysis of how certain characteristics of the instances, e.g., the number of conjunctive and disjunctive constraints affect the efficiency of the solver,
- and the evaluation of a problem-specific branching strategy to improve the MILP solution approach.

The main conclusions from those experiments are briefly summarized below, whereas a comparison to the CP approach adapted from Kardos et al. (2020) is presented in detail on instances with cycles in the liaison graph.

The first four instance families, containing 10–30 parts, were generated to investigate the effect of conjunctive and disjunctive constraints. The experiments showed that *conjunctive constraints* have a significant impact on computation time: while all instances *without* conjunctive constraints were solved to optimality by the MILP, the solver terminated with a considerable optimality gap on some instances *with* conjunctive constraints, with at least 26 parts, after the 7200 s time limit. Average gaps were between 0.40% and 9.50%, while maximum gaps between 2.10% and 19.00% depending on instance family and problem size. On the other hand, *disjunctive constraints* have only a minor impact on computation times, implying that the solver can be efficient even with a high number of Benders iterations, and accordingly, with many disjunctive constraints fed back from the subproblem.

In the experiments, the problem-specific branching strategy improved significantly the efficiency of the MILP: computation times decreased and final gaps could also be reduced (0.00%–4.10% average gaps and 0.00%–11.5% maximum gaps). Finally, on instances with tree-shaped liaison graphs or very few cycles in the liaison graph, MILP and CP yielded disparate results: the latest version 9.0 of the CP solver hit the time limit on all instanced with more than 15 parts. The performance of the legacy CP version 7.1 was significantly better, even comparable to the MILP on small-to-medium instances, but it was still outperformed by the MILP. Further details are provided in the Supplementary Material.

Below, we present in detail the comparison of the MILP and the CP solvers on instances with cycles in the liaison graph (Family 5). In Fig. 7, the curves show the average computation time for each solver (average over instances with the same number of parts), whereas the bars in the background display the average number of additional edges (tasks) in the liaison graph, i.e., $|T| - (|V| - 1)$. The lowest bar corresponds to 2 extra tasks, while the highest to 7 extra tasks.

The results show that MILP scales well with instance size: instances with at most 16 parts are solved to optimality within a second, instances with at most 23 parts in a minute. Even the largest instances with 30 parts could be solved within 2000 seconds on average. The MILP-based approach with the problem-specific branching strategy performs even better, namely, on instances with at least 25 parts, this branching rule reduced solution times significantly (for the largest instances this improvement is 74%, on average). The latest CP version 9.0 struggles on

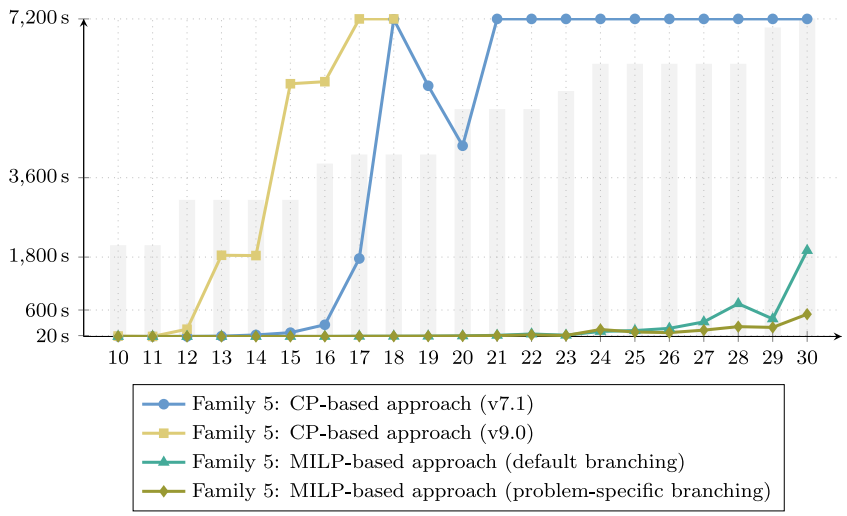

Fig. 7. Results of different solution approaches on instances with cycles in the liaison graph (Family 5).

these instances, thus, it is tested only on instances with at most 18 parts. The legacy CP version 7.1 performed better than that, however, it is still clearly inferior to the proposed MILP, as it could not solve to optimality any of the instances with more than 20 parts. The minimum, average and maximum optimality gaps achieved by CP version 7.1 on these instances were 1.8%, 20.0% and 42.5%, respectively.

8. Summary of main contributions

This section summarizes the main contributions made by the paper to solving the assembly planning problem using LBBD.

- *Efficient handling of cycles in the liaison graph:* Allowing cycles in the liaison graph is a substantial extension compared to the previous model by Kardos et al. (2017), where only tree-structured liaison graphs were considered. This enables handling alternative assembly tasks. Although cycles were already introduced by Kardos et al. (2020), that paper applied a less efficient, straightforward CP-based approach to solving the master problem. The efficiency of the proposed MILP-based approach compared to CP-based has been illustrated in computational experiments both on industrial problems and on generated instances. The novel MILP-based approach solved larger instances (e.g., up to 7 extra edges instead of 3 extra edges), and it computed proven optimal solutions in less than a minute for various problems for which CP could not find any feasible solution at all.
- *Modeling graph connectivity in the MILP:* The handling of cycles in the liaison graph necessitated introducing new, sophisticated modeling techniques for capturing the connectivity status of the graph, see Section 4.2.
- *New types of cuts from the subproblem to the master problem:* In both previous papers Kardos et al. (2017) and Kardos et al. (2020), the disjunctive constraints fed back from the subproblem to the master problem express generalized precedence constraint between tasks. In contrast, the current paper proposes constraints on the connectivity of parts, irrespective of the tasks that realize the connections. This type of constraint is substantially stronger in case of cycles in the liaison graph, and results in less Benders iterations.
- *Subproblem solver:* Finally, this paper is the first to present the subproblem solver in detail, together with various techniques to increase its efficiency, including caching for the validation of the feature motion.

9. Conclusions

This paper proposes a novel combination of Benders decomposition with disjunctive programming for solving the assembly planning problem. The Benders master problem is a MILP augmented with disjunctive constraints, while the subproblem (inference dual) is based on the geometrical modeling of the subassemblies, the fixtures and tools, and uses path planning and collision detection algorithms to identify infeasible assembly tasks in a plan. If a task in a sequence proves infeasible, a disjunctive constraint is generated and fed back to the Benders master problem. The geometric inference methods use adaptations of well-known techniques to generate new, strong disjunctive constraints on the assembly plan. The main benefit of the approach is that it ensures the feasibility of the assembly plans on fine-grained models, which is not tractable using classical approaches that aim to perfectly characterize the set of all feasible task sequences before search.

The efficiency of the proposed approach is demonstrated both on industrial case studies and on generated problem instances, which show that the approach can solve problems of industrially relevant sizes in a reasonable amount of time.

CRediT authorship contribution statement

Markó Horváth: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft, Writing – review & editing. **Tamás Kis:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **András Kovács:** Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing, Supervision. **Márk Fekula:** Methodology, Software.

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 been supported by the NKFIH, Hungary grant no. ED_18-2-2018-0006. A. Kovács acknowledges the support of the János Bolyai scholarship, Hungary.

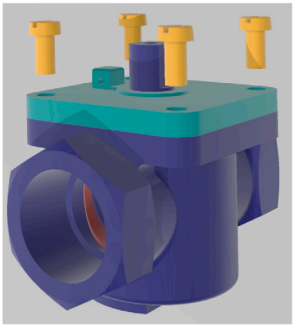

Fig. A.8. Example of composite feature: joining a set of pre-positioned parts using four identical parallel-axis screws.

Appendix A. Feature-based assembly model

The feature-based model applied in this research is an object-oriented representation of all the geometrical and technological information required to plan the assembly process of the given product. The model consists of the following entities:

- All *parts* of the assembly are modeled as 3D solid objects, see Appendix B for details.
- *Features* characterize the liaisons of parts with all relevant geometrical and technological information. With binary (i.e., two-handed) assembly assumed, each feature connects exactly two sets of parts. When such a part set is not singleton (i.e,. it is a subassembly), then the contained parts be must joined prior to the execution of the current feature. Observe that even in this case, the feature-based model can be represented by a classical liaison graph, rather than a hyper-graph, by selecting an arbitrary part from the subassembly as the end point of the classical binary edge, and recording the additional precedence constraint (not directly captured in the liaison graph).
  The current implementation supports *placing*, *insertion*, and *screwing* features. A detailed definition of a richer set of assembly features is available, e.g., in Wang et al. (2011). The technological parameters of the features used for planning include the *direction vector* (direction of the feature motion $\overrightarrow{P_N P_F}$), as well as the *insertion depth* (for insertion and screwing features) and the *safety distance* (all features) whose sum determines the length of $\overrightarrow{P_N P_F}$. Finally, the estimated duration of the assembly tasks is also part of the feature model.
  The applied model allows so-called *composite features*, i.e., a grouping of individual features that must be performed together by design, using identical resources. A typical example of a composite feature is assembling a set of pre-positioned parts (the base parts in the feature) using a set of identical screws (moved parts), see Fig. A.8. In the macro-level planning model, one task is generated for each assembly feature, either individual or composite.
  It is highlighted that the liaison graph defined by the features may contain cycles, which represent significantly different alternatives in the assembly process. In the ball valve case study, such alternatives are placing the o-ring sealing on house or on the cover before assembling the house and the cover themselves (Fig. A.9). When the liaison graph is a tree, then all features must be executed.
- *Tools* are characterized by their changeover times (for macro-level planning) and their geometry (for micro-level validation). Moreover, the applicability of a tool to perform an assembly feature is given in the input together with the relative position of the tool to the moved part.
- *Fixtures* are defined by the grasped part, the maximal subassembly that can be held by the fixture, and their changeover time (used by macro-level planning), as well as their geometry, together with the transformation matrix that specifies the relative position of the grasped part w.r.t. the fixture (used for micro-level validation). In case a fixture can grasp multiple parts or one part in multiple ways, then the physical fixture is described by multiple fixture objects in the model.

Appendix B. Geometrical models of parts and resources

All physical objects involved in the assembly process, including parts, fixtures, and tools are modeled as free-form 3D solid objects that move during the assembly task according to the laws defined in the corresponding assembly feature. Each solid object is described by a triangle mesh, which has two key benefits: the mesh representation can be generated from all major CAD systems in STL file format; and (2) collision detection can be performed efficiently on this representation of free-form geometries (Larsen et al., 2000; Pan et al., 2012). On the other hand, a particular challenge related to this representation is that mesh geometries are inevitably imprecise, see Fig. B.10. Consequently, parts that touch each other in reality often occur as colliding geometries in the model, which must be handled by the appropriate post-processing of the raw results of the collision queries. The same difficulty arises with deformable parts, such as the sound protection parts in our automotive case study. Algorithms for tackling these challenges have been proposed in Kardos and Váncza (2018). For fixtures and tools that cannot be described by a solid geometry, such as a human hand used for holding or moving a part, geometries are omitted, and these resources are exempt from collision checks.

Appendix C. Modeling the motion of the parts and resources

For collision detection during assembly, not only the geometry, but also the relative motion of the above objects has to be captured. In

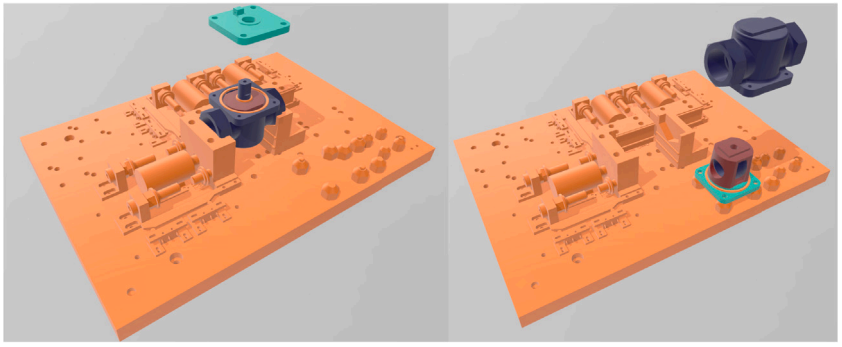

Fig. A.9. Alternative assembly processes corresponding to cycles in the liaison graph: joining the o-ring sealing to the house (left) or to the cover (right) in the ball valve case study.

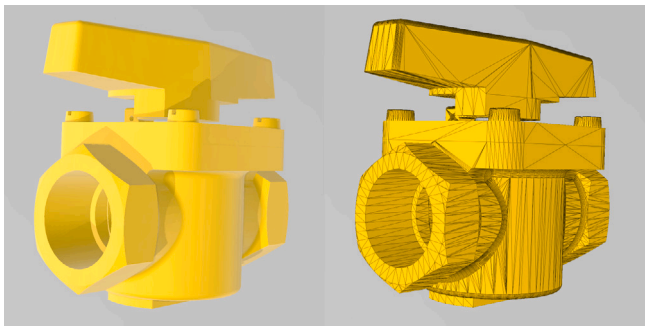

Fig. B.10. The smooth geometry (left) and the triangle mesh model (right) of the ball valve assembly.

two-handed assembly, two groups of objects move relative to each other: (1) the base subassembly together with the fixture, and (2) the moved subassembly together with the tool. Within either group, the relative position of the objects is fixed. This motion can be described in the configuration space of the relative positions of the base and the moved objects. The current implementation assumes 3D translations only, which is sufficient in most industrial use cases, but the proposed techniques can be naturally extended to allow both translations and rotations, according to the special Euclidean group SE(3). All motions considered are either linear (e.g., the feature motion $\overrightarrow{P_N P_F}$) or piecewise linear (the approach motion $\overset{\rightsquigarrow}{P_R P_F}$) in this configuration space.

Appendix D. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cor.2021.105603.

References

Bahubalendruni, M.R., Biswal, B.B., 2016. A review on assembly sequence generation and its automation. Proc. Inst. Mech. Eng. C 230, 824–838.

Balas, E., 1975. Disjunctive programming: cutting planes from logical conditions. In: Nonlinear Programming 2. Academic Press, pp. 279–312.

Balas, E., 1979. Disjunctive programming. In: Annals of Discrete Mathematics, vol. 5. Elsevier, pp. 3–51.

Balas, E., 1998. Disjunctive programming: Properties of the convex hull of feasible points. Discrete Appl. Math. 89, 3–44.

Balas, E., 2018. Disjunctive Programming. Springer.

Benders, J., 1962. Partitioning procedures for solving mixed-variables programming problems. Numer. Math. 4, 238–252.

Cortés, J., Jaillet, L., Siméon, T., 2008. Disassembly path planning for complex articulated objects. IEEE Trans. Robot. 24, 475–481.

De Fazio, T., Whitney, D., 1987. Simplified generation of all mechanical assembly sequences. IEEE J. Robot. Autom. 3, 640–658.

Fachini, R.F., Armentano, V.A., 2020. Logic-based Benders decomposition for the heterogeneous fixed fleet vehicle routing problem with time windows. Comput. Ind. Eng. 106641.

Fazel-Zarandi, M.M., Beck, J.C., 2012. Using logic-based Benders decomposition to solve the capacity-and distance-constrained plant location problem. INFORMS J. Comput. 24, 387–398.

Geoffrion, A.M., 1972. Generalized benders decomposition. J. Optim. Theory Appl. 10, 237–260.

Grossmann, I.E., 2002. Review of nonlinear mixed-integer and disjunctive programming techniques. Opt. Eng. 3, 227–252.

Hassan, S., Yoon, J., 2010. Haptic based optimized path planning approach to virtual maintenance assembly / disassembly (MAD). In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1310–1315.

Hooker, J.N., 2007. Planning and scheduling by logic-based benders decomposition. Oper. Res. 55, 588–602.

Hooker, J.N., 2011. Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction, second ed. John Wiley & Sons.

Hooker, J.N., 2019. Logic-based Benders decomposition for large-scale optimization. In: Large Scale Optimization in Supply Chains and Smart Manufacturing. Springer, pp. 1–26.

Hooker, J.N., Ottosson, G., 2003. Logic-based Benders decomposition. Math. Program. 96, 33–60.

Hopcroft, J., Tarjan, R., 1973. Algorithm 447: efficient algorithms for graph manipulation. Commun. ACM 16, 372–378.

Hu, S.J., Ko, J., Weyand, L., ElMaraghy, H., Lien, T., Koren, Y., Bley, H., Chrysolouris, G., Nasr, N., Shpitalni, M., 2011. Assembly system design and operations for product variety. CIRP Annals–Manufacturing Technology 60, 715–733.

Hui, C., Yuan, L., Kai-fu, Z., 2009. Efficient method of assembly sequence planning based on GAAA and optimizing by assembly path feedback for complex product. Int. J. Adv. Manuf. Technol. 42, 1187–1204.

Jiménez, P., 2013. Survey on assembly sequencing: a combinatorial and geometrical perspective. J. Intell. Manuf. 24, 235–250.

Kardos, C., Kovács, A., Váncza, J., 2016. Towards feature-based human-robot assembly process planning. Procedia CIRP 57, 516–521.

Kardos, C., Kovács, A., Váncza, J., 2017. Decomposition approach to optimal feature-based assembly planning. CIRP Annals–Manufacturing Technology 66, 417–420.

Kardos, C., Kovács, A., Váncza, J., 2020. A constraint model for assembly planning. J. Manuf. Syst. 54, 196–203.

Kardos, C., Váncza, J., 2018. Mixed-initiative assembly planning combining geometric reasoning and constrained optimization. CIRP Annals–Manufacturing Technology 67, 463–466.

Kloimüllner, C., Raidl, G.R., 2017. Full-load route planning for balancing bike sharing systems by logic-based benders decomposition. Networks 69, 270–289.

Larsen, E., Gottschalk, S., Lin, M., Manocha, D., 2000. Fast proximity queries with swept sphere volumes. In: Proc. of ICRA2000, IEEE International Conference on Robotics and Automation, pp. 3719–3726.

Lavalle, S.M., Kuffner, J.J., 2000. Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and Computational Robotics: New Directions. pp. 293–308.

Le, D., Cortés, J., Siméon, T., 2009. A path planning approach to (dis)assembly sequencing. In: 2009 IEEE International Conference on Automation Science and Engineering, pp. 286–291.

Morato, C., Kaipa, K.N., Gupta, S.K., 2013. Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters. Comput. Aided Des. 45, 1349–1364.

Neb, A., 2019. Review on approaches to generate assembly sequences by extraction of assembly features from 3D models. Procedia CIRP 81, 856–861.

Pan, J., Chitta, S., Manocha, D., 2012. FCL: A general purpose library for collision and proximity queries. In: Proc. of ICRA2012, IEEE International Conference on Robotics and Automation, pp. 3859–3866.

Rashid, M.F.F., Hutabarat, W., Tiwari, A., 2012. A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. Int. J. Adv. Manuf. Technol. 59, 335–349.

Rodríguez, I., Nottensteiner, K., Leidner, D., Kassecker, M., Stulp, F., Albu-Schäffer, A., 2019. Iteratively refined feasibility checks in robotic assembly sequence planning. IEEE Robot. Autom. Lett. 4, 1416–1423.

Romney, B., Godard, C., Goldwasser, M., Ramkumar, G., 1995. An efficient system for geometric assembly sequence generation and evaluation. In: Proc. of the 15th ASME International Computers in Engineering Conference, pp. 699–712.

Roshanaei, V., Luong, C., Aleman, D.M., Urbach, D., 2017. Propagating logic-based Benders' decomposition approaches for distributed operating room scheduling. European J. Oper. Res. 257, 439–455.

Schwarzer, F., Saha, M., Latombe, J.-C., 2004. Exact collision checking of robot paths. In: Algorithmic Foundations of Robotics V. Springer, pp. 25–41.

Thomas, U., Barrenscheen, M., Wahl, F., 2003. Efficient assembly sequence planning using stereographical projections of C-space obstacles. In: Proc. of the IEEE International Symposium on Assembly and Task Planning, pp. 96–102.

Tsukiyama, S., Shirakawa, I., Ozaki, H., Ariyoshi, H., 1980. An algorithm to enumerate all cutsets of a graph in linear time per cutset. J. ACM 27, 619–632.

Wang, L., Keshavarzmanesh, S., Feng, H.-Y., 2011. A function block based approach for increasing adaptability of assembly planning and control. Int. J. Prod. Res. 49, 4903–4924.

Wheatley, D., Gzara, F., Jewkes, E., 2015. Logic-based Benders decomposition for an inventory-location problem with service constraints. Omega 55, 10–23.

Wilson, R.H., Latombe, J.-C., 1994. Geometric reasoning about mechanical assembly. Artificial Intelligence 71, 371–396.