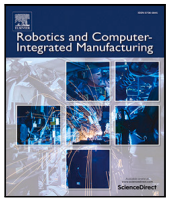
Robotics and Computer–Integrated Manufacturing 78 (2022) 102387

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

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



Full length article

ProSeqqo: A generic solver for process planning and sequencing in industrial robotics

László Zahorán, András Kovács *

EPIC Center of Excellence in Production Informatics and Control, Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH), Budapest, Hungary

ARTICLE INFO

Keywords:
Industrial robotics
Process planning
Task sequencing
Generalized traveling salesman
Combinatorial optimization

ABSTRACT

Task sequencing problems arise in many different forms in various robotic applications. The task sequence can be subject to different constraints, and various cost functions can be used to assess the quality of a solution. Moreover, task sequencing is often closely coupled with other sub-problems of process planning, such as the selection of a robot joint configurations for each task originally defined in the task space, or the selection of cut directions in a cutting problem. This implies that these problems must be solved jointly, in an integrated way. This complexity results that almost all previous approaches to robotic task sequencing aim at solving the sequencing problem arising in a specific application, using a dedicated – typically, meta-heuristic – solution method. Despite this, such custom methods rely on similar mathematical models, namely, different extension of the well-known traveling salesman problem (TSP). In order to avoid such redundancies, this paper proposes a generic problem definition language for robotic task sequencing and process planning problems, as well as a solver called ProSeqqo to compute close-to-optimal solutions for those problems. ProSeqqo relies on the vehicle routing problem (VRP) library of Google OR-Tools, extended with custom algorithms to tackle conditional precedence constraints. It is demonstrated that the proposed language can capture the overwhelming majority of the robotic task sequencing problems investigated in the scientific literature, and the application of the modeling language and the solver is illustrated on five, seemingly very different use cases, including both real industrial applications and lab demonstrations. Results of thorough computational experiments are also presented. The ProSeqqo solver has been made available open source for the scientific community.

1. Introduction

Effective task sequencing has a decisive role in the performance of robotic production and manipulation processes. The task sequencing problem can be posed in different ways depending on the particular application, and it is often interrelated with other sub-problems of process planning, such as the selection of robot configurations for executing tasks originally defined in the Cartesian task space, or the direction of the individual cuts in a robotic cutting application. Accordingly, a plethora of task sequencing approaches have been proposed in the scientific literature, almost always dedicated to a specific application, almost always with a custom (meta-)heuristic solution method.

In this paper, it is highlighted that most of these approaches stand on the same grounds with respect to the underlying mathematical model, namely, the traveling salesman problem (TSP) or one of its many extensions, such as the generalized TSP (GTSP) that can capture discrete choices on the execution modes of tasks. This raises the opportunity to define a generic model that captures the overwhelming majority of task sequencing and related process planning problems arising in different applications.

The paper introduces such a generic model, and a modeling language that allows stating the robotic task sequencing problem in a user-friendly way for process engineers, even without a deep background in operations research. The model is translated into a GTSP with precedence constraints, and solved using the open-source vehicle routing problem (VRP) solver of Google OR-Tools, extended with custom algorithms to boost search performance where necessary.

The solver has been made available open source, together with various examples and appropriate documentation. We expect that, by offering efficient means to modeling and solving typical robotic process planning and task sequencing problems, this software tool brings major benefits to fellow researchers in industrial robotics. Redundancies due to time-consuming reproduction and adaptation of known techniques for a new, but still familiar application can be avoided, and research efforts can focus on developing dedicated solution approaches where those are indeed required for the success of the application.

* Corresponding author.
E-mail addresses: zahoran.laszlo@sztaki.hu (L. Zahorán), andras.kovacs@sztaki.hu (A. Kovács).

https://doi.org/10.1016/j.rcim.2022.102387
Received 1 July 2021; Received in revised form 26 April 2022; Accepted 22 May 2022
Available online 3 June 2022

This paper is structured as follows. After reviewing the related literature in Section 2, a generic model is defined for robotic process planning and task sequencing problems in Section 3. The model is illustrated on a camera-based pick-and-place application. Then, Section 4 introduces the proposed problem definition language. The applied solution techniques are presented in Section 5, and they are evaluated in experiments on five different applications in Section 6. Finally, conclusions are drawn in Section 7.

2. Literature review

2.1. Mathematical models for task sequencing

The archetype of mathematical models for task sequencing is the well-known TSP: a minimum-cost Hamiltonian cycle is looked for in a graph whose vertices correspond to the tasks, whereas edge weights capture the cost of executing one task after the other. When sequencing is coupled with some choice of execution mode, various extensions of TSP become of interest. Discrete choices, if choices for different tasks are independent, can be captured by a GTSP [1], which involves a partitioning of the graph vertices into disjoint classes, and a minimum-cost tour that visits each class exactly once is sought. A special case of GTSP in robotics is the robotic task sequencing problem (RTSP) [2], where each class contains the inverse kinematic (IK) solutions for a task executed in a given task-space point. If the visited point must be selected from a geometrical region, rather than from a discrete set, then the problem becomes a TSP with neighborhoods (TSPN) [3]. The particular challenge of TSPN is that its solution requires coupling combinatorial optimization approaches with geometrical reasoning. An exact mixed-integer non-linear programming (MINLP) solution approach for TSPN with polyhedral or ellipsoid regions is proposed in [4].

The above models are often extended with various side constraints to reflect the requirements of practical applications. The extension of the classical TSP with precedence constraints is called the sequential ordering problem (SOP) [5], whereas a similar extension of GTSP with precedence constraints between the classes resulted in the precedence-constrained GTSP (PCGTSP). Recent solution methods for PCGTSP include a large neighborhood search (LNS) meta-heuristic [6] and an exact branch-and-bound approach [7]. A related, generic family of models is the vehicle routing problem (VRP), which can capture multiple vehicles (corresponding to robots or other resources executing the tasks), time windows, as well as precedence and capacity constraints. Efficient solution methods for this rich model often involve a combination of constraint programming and meta-heuristics, such as tabu search or guided local search [8]. A detailed review of common mathematical models and solution methods for task sequencing is provided in [9].

2.2. Survey of robotic task sequencing problems

This section reviews process planning and task sequencing approaches in various robotic applications with the goal of identifying the common requirements towards mathematical models in such applications. The findings are summarized in Table 1, which also indicates if a given problem fits into the model proposed in this paper. For problems that cannot be captured, the challenging features are highlighted. It is noted that in some of the referred papers, the mathematical model is not named explicitly, but the problem can be captured using the indicated model.

In applications where task sequencing is performed independently of any other decision on the execution modes, the natural model is a TSP, and the duty of the modeler is to characterize the cost or time of executing the tasks in a given order. Such an approach for robotic operations in general is presented in [10], which also captures tasks that comprise visiting multiple positions and task clusters that must be executed continuously after each other in arbitrary order. In [11], an application to fruit picking is introduced. The so-called traversing tuft problem, a task sequencing problem in carpet production is mapped to a TSP in [12], where edge weights encode tufting, positioning, and needle switching times. In [13], the problem of sequencing the pallet operations in machining is modeled as a SOP, considering tool changeovers, table rotations, idle movements, as well as precedence constraints.

Various applications couple task sequencing with different process planning decisions, such as the IK solutions for the visited task space points, the entry and exit points along contours, or the directions of cuts or strokes, which leads to a GTSP model. Such an approach, where the classes of vertices correspond to IK solutions for a task, is presented in [14], and solved using a genetic algorithm. A similar approach, illustrated on a drilling application motivated by the Airbus Shop Floor Challenge, is discussed in [2]. Coverage path planning for the inspection of free-form surfaces is formulated as a GTSP in [15], where each class corresponds to a surface primitive that must be inspected, and the vertices in it to collision-free IK solutions for candidate viewpoints. Similarly, path planing for a 5-axis on-machine inspection system is addressed in [16], with different possible probe orientations for each feature to inspect.

The minimization of non-productive times in material extrusion additive manufacturing is investigated in [17,18], where vertices in a class encode the entry and exit points for closed contours and processing directions for open filling rasters within each layer. For solving the problem, heuristics and mixed-integer linear programming are combined in [17], whereas the performance of different heuristics is compared in [18].

A task-oriented programming system is proposed for remote laser welding in [19]. The task sequencing problem is encoded in a TSP over the mid-points of the welding seams, and weld directions are determined afterwards, using a greedy improvement heuristic. The ensemble of the two problems can be cast as a GTSP. In a similar application, [20] focuses on the robot configurations applicable to weld a seam, rather than on the position of the welding seam itself. The configuration space of the redundant robot is sampled, which leads to a GTSP model.

In [21], a PCGTSP model is applied to task sequencing in milling, where the vertex classes describe candidate entry and exit points for each contour, and classes corresponding to embedded contours are connected by precedence constraints. Task sequencing in laser cutting is coupled with the selection of a direction for the individual cuts in [22]. A rich set of technological constraints, such as precedence relations between inner and outer contours, piercing, pre-cuts, and sharp angles is captured and encoded into a PCGTSP representation. While this model can be mapped into the formalism proposed in this paper, the conversion is non-trivial and needs a slight abuse of notation. A thorough review of models and algorithms specifically for laser cutting applications can be found in [28].

Task sequencing problems have been investigated in various applications with the consideration of precise geometry, often modeled as a variant of TSPN. The primary difficulty in capturing these problems using a generic solver is the integration of combinatorial optimization with geometrical reasoning. This approach is taken to task sequencing and path planning for remote laser welding in [23], captured as a TSP with neighborhoods and durative visits (TSP-ND). In [24], a TSPN representation and a so-called constricting insertion heuristic solution approach is proposed for robotic cutting and deburring operations. In these applications, sampling the geometrical space can often provide a suitable discrete approximation of the original continuous problem, which enables the encoding of the problem into the generic representation introduced here.

Nevertheless, some applications require side constraints specific for the given technology. In robotic spray painting, performing task sequencing jointly with the selection of paint stroke directions and IK solutions would fit into the above models, but reasoning about

Table 1
Overview of task sequencing problems from the literature. Column *Captured* displays if the problem is captured by the approach proposed in this paper: *Y*: yes; *(Y)*: yes, but with a misuse of the formalism; *D*: after discretizing the geometrical space; *N*: no. In the latter case, the challenging feature is also identified.

	Application	Model	Discrete choice	Objective	Captured	Challenging features
[10]	Generic model	TSP	–	Min. distance	Y	
[11]	Fruit picking	TSP	–	Min. distance	Y	
[12]	Carpet tufting	TSP	–	Min. cycle time, min. dead yarn	Y	
[13]	Machining	SOP	–	Min. cycle time	Y	
[14]	Generic model	GTSP	IK solution	Min. cycle time	Y	
[2]	Drilling	GTSP	IK solution	Min. distance	Y	
[15]	Inspection	GTSP	Viewpoint, IK solution	Min. distance	Y	
[16]	Inspection	GTSP	Probe orientation	Min. distance	Y	
[17]	Additive manufacturing	GTSP	Entry/exit points, directions	Min. cycle time	Y	
[18]	Additive manufacturing	GTSP	Entry/exit points, directions	Min. cycle time	Y	
[19]	Remote laser welding	GTSP	Weld directions	Min. distance	Y	
[20]	Remote laser welding	GTSP	Entry/exit points	Min. cycle time	Y	
[21]	Milling	PCGTSP	Entry/exit points	Min. cycle time	Y	
[22]	Laser cutting	PCGTSP	Cut directions	Min. cycle time	(Y)	
[23]	Remote laser welding	TSP-ND	Task start/end positions	Min. distance	D	
[24]	Cutting and deburring	TSPN	Task start/end positions	Min. distance	D	
[25]	Spray painting	Custom	Stroke directions, IK solution	Min. distance, max quality	N	Violation of min–max time lags (objective)
[26]	Drilling	Custom	Tool sequence for hole	Min. cycle time	N	Sequence-dependent edge costs
[27]	Welding	Custom	–	Min. cycle time, min. distortion	N	Predicting part distortion

the time lags between overlapping paint strokes, in order to ensure the proper drying time for the paint requires custom models [25]. Moreover, trade-offs between productivity and product quality are captured by an objective function that contains a linear combination of the total distance traveled by the robot and the violation of the soft constraints on the minimum and maximum time lags. In [26], the sequencing of drilling tasks is investigated. The challenge is that a large-diameter hole can be drilled in various ways, using different series of tools with increasing diameter. Moreover, the series of subtasks with different tools is not executed continuously, but a single tool is typically applied to many holes before a changeover. In a TSP, this would correspond to sequence-dependent edge costs, which cannot be tackled using classical TSP algorithms efficiently. A thorough survey on drilling path optimization is presented in [29], where it is found that 79% of the reviewed papers model the problem as a simple TSP, 13% as a SOP, and 8% apply more complex models like the one above.

Finally, in robotic welding, the distortion and residual stress of parts is heavily dependent on the welding sequence. In [27], the sequencing of the welding tasks that do not affect distortion is captured by a TSP, whereas sensitive tasks are sequenced using a genetic algorithm considering a temperature field model for estimating distortion, and finally, the two sub-sequences are merged. Obviously, the effective handling of such technology-specific requirements needs customized computational techniques.

2.3. Contributions

The above literature review highlights that previous approaches to robotic task sequencing rely on well-understood mathematical models, notably, on TSP or one of its numerous extensions. In particular, the overwhelming majority of the investigated problems fits into a PCGTSP model. Despite this, all the surveyed papers undertook to develop custom algorithms for very similar problems, leading to considerable redundancy in research efforts. This claim is also supported by the conclusions of the critical survey [29] that a considerable part of the research in task sequencing (especially in drilling) is currently spent on reproducing earlier results, which could be saved by wittingly reusing existing generic solution techniques.

In order to respond to the shortage of generic solvers in robotic task sequencing and to support fellow researchers in the field, this paper proposes an expressive problem definition language for robotic task sequencing, easily editable by domain experts even without a deep background in operations research. Moreover, it introduces an efficient solver, publicly available with open source, to solve the problems encoded in the proposed format. The expressive power of the language and the efficiency of the solver is demonstrated in five rather different applications.

Obviously, any single model, however generic it is meant to be, cannot capture all possible planning problems arising in the field. The literature review identified two such limitations of the proposed approach. Firstly, if task sequencing is coupled with decisions in continuous space (e.g., with viewpoint selection in inspection or in remote laser welding [23]; start/end point selection on continuous curves in cutting [24]), then the problem must be discretized by sampling before encoding it into the proposed formulation. Secondly, it might be impossible to express the objective function as the sum of transition costs on the (PCG)TSP edges, e.g., for characterizing product quality in spray painting [25], machining times in multi-hole drilling [26], or part distortion in welding [27]. These applications still need dedicated solution approaches.

3. Problem definition

3.1. Formal definition

In a robotic process planning and task sequencing problem, there is given a set of n processes, $P_1, P_2, \dots, P_n$, to be executed sequentially. Each process P_i can be executed by using one of the process alternatives $A_{i1}, A_{i2}, \dots, A_{iJ(i)}$. Alternative A_{ij} consists of a sequence of tasks $T_{ij}^1, T_{ij}^2, \dots, T_{ij}^{K(ij)}$. If the alternative is selected for execution, then all the corresponding tasks must be executed consecutively, in the predefined order. Each of these tasks T_{ij}^k can be executed using one of the robot motions $M_{ij}^{k\ell}$, where each motion $M_{ij}^{k\ell}$ takes the robot through a sequence of configurations $(C_{ij}^{k\ell 1}, C_{ij}^{k\ell 2}, \dots, C_{ij}^{k\ell N(ijk\ell)})$.[1] It is noted that many applications involve point-like tasks, in which case $N(ijk\ell) = 1$. The hierarchy of processes, alternatives, tasks, motions and configurations is illustrated in Fig. 1.

Hence, the complete operating cycle of the robot consists of a series of selected effective motions, from $C_{ij}^{k\ell 1}$ to $C_{ij}^{k\ell N(ijk\ell)}$, and idle motions connecting the final configuration of an effective motion, $C_{ij}^{k\ell N(ijk\ell)}$, with the starting configuration of the next selected effective motion. If $k < K(i,j)$, then this is a starting configuration of the next task in the

[1] Configurations can be defined in arbitrary dimensions, e.g., in the 2D or 3D task space, in the 6D robot joint configuration space, or in higher dimensions for fully characterizing the state of the robotic system, including the configuration of grippers, fixtures, or other equipment.

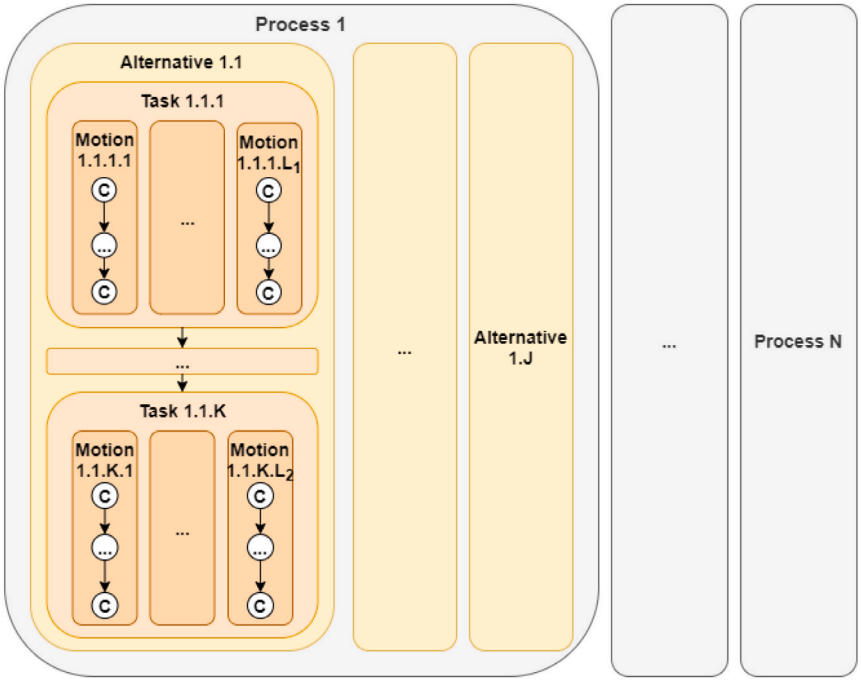


Fig. 1. Hierarchy of processes, alternatives, tasks, motions and configurations in the model.

given alternative, $C_{ij}^{(k+1)\ell'1}$ for some ℓ'. Otherwise, if $k = K(i,j)$, then it is a starting configuration of the first task of the next process, $C_{i'j'}^{1\ell'1}$. This series of effective and idle motions is continued until all processes are executed.

It is highlighted that, in addition to sequencing the processes P_i, two types of decisions must be made about their way of execution: the selection of an alternative A_{ij}, which is a consistent selection throughout the execution of the process (such as the selection of the grasp for a workpiece in a pick-and-place application, since this grasp must be applied throughout the entire processing of the workpiece), and the selection of a motion $M_{ij}^{k\ell}$ for each task separately (such as the selection of the robot joint configuration for a task, which does not influence the configuration used in other tasks on the same workpiece).

A problem can be *cyclic* or *acyclic*. In case of cyclic problems, the starting configuration (depot, in VRP terms) must be provided in the input, and it is assumed that the robot returns to this configuration in the end. For acyclic problems, the specification of a starting or a finishing configuration is optional. In the absence of a starting (respectively, finishing) configuration, it is assumed that the robot starts (finishes) at the initial (final) configuration of the first (last) task executed.

Two types of *precedence constraints* impose restrictions on the order of the processes: *process precedence* constraints $P_i \rightarrow P_{i'}$ state that process P_i must be executed before process $P_{i'}$. In contrast, *motion precedence* constraints $M_{ij}^{k\ell} \rightarrow M_{i'j'}^{k'\ell'}$ require that, if both involved motions occur in the plan, then $M_{ij}^{k\ell}$ must precede $M_{i'j'}^{k'\ell'}$. If either of the motions is left out of the plan, then the constraint is ignored. Observe that a directed cycle of process precedences renders a problem infeasible, but a directed cycle of motion precedences does not, since the cycle can be cut by leaving some of the involved motions out of the plan.

The cost of a solution is computed as the total cost accumulated while the robot travels between subsequent configurations. The proposed problem definition language includes various functions to express this cost in a user friendly way, such as the Euclidean or the Manhattan distance between configurations, travel times with trapezoid speed profiles, a special function for expressing the resource requirements of tasks and the changeovers between them, or a penalty for switching contours in cutting or drawing problems. Finally, using a matrix format with arbitrary cost values, the language captures all possible objective functions that can be computed as the sum of the elementary motion costs.

Then, the planning problem consists in sequencing the processes P_i, selecting an alternative A_{ij} for each process P_i, and choosing a robot motion $M_{ij}^{k\ell}$ for each executed task T_{ij}^{k}, in such a way that the total cost is minimized.

3.2. Illustrative example: Camera-based robotic pick-and-place

Throughout the paper, the above formal model is illustrated on a camera-based pick-and-place case study, whereas further potential applications are discussed later in Section 6. One of the greatest current challenges in industrial automation is the handling of parts arriving in bulk to the input points of an automated manufacturing or assembly system. A robot must grasp the parts lying in random poses based on appropriate sensory information. A potential solution to this challenge has been presented in [30], where various parts are first loaded onto a vibrating lighting table, their precise poses are determined by a camera system, and then a robot takes them one by one to the part holder of a press machine,[2] see Fig. 2.

In the proposed representation, one process is defined for each part. Each process contains multiple alternatives corresponding to different ways for grasping the part. Then, each alternative contains two tasks, a picking and a placing task for the given part using the given grasp. Different motions within a task correspond to different collision-free IK solutions for the given task-space poses for picking or placing.

It is highlighted that task sequencing is coupled with two types of decisions about how the tasks are executed. The selection of the grasp must be consistent between the corresponding picking and placing tasks, and hence, it must be captured on the level of alternatives. On the other hand, the selection of the IK solutions can be performed independently for each task, and accordingly, must be represented by multiple candidate motions within a task.

[2] Camera-based robotic pick-and-place (youtube link): https://youtu.be/9novNg8slN4.

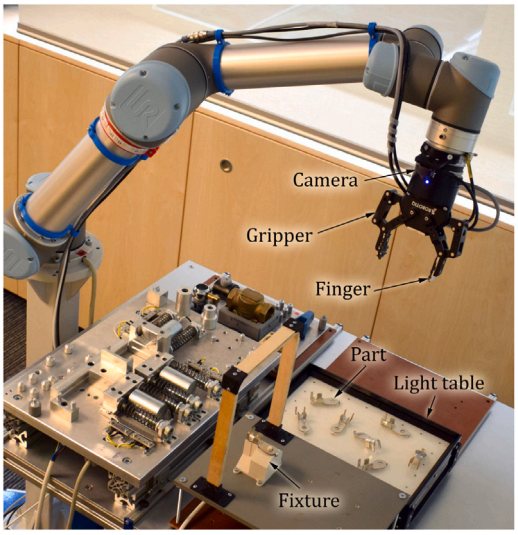


Fig. 2. Camera-based robotic pick-and-place [30].

Blocking relations between parts are captured by process precedence constraints. The objective is minimizing the total processing time by considering a trapezoid speed profile for each robot motion, with given robot joint acceleration and speed limits.

4. Language for defining task sequencing problems

In order to provide an easy-to-use interface for the proposed solver, the following language is proposed for defining the process planning and task sequencing problems. The subsections below introduce the keywords of the language that can be used in JSON, in XML, or in a custom, concise and human-readable text file format called SEQ. The main components of a problem definition are the process hierarchy, costs, resources and solver settings. A simplified camera-based pick-and-place problem is used to illustrate the SEQ format, broken into four parts. The ensemble of the four code snippets gives a complete problem specification file. Several further examples are available in the online project repository. Finally, the task sequencing problems can also be defined via an API, which enables the efficient interfacing of the ProSeqqo solver in complex planning workflows.

4.1. Process hierarchy

The following keywords are available for defining the hierarchy of processes, alternatives, tasks, motions, and configurations:

- `ConfigList: list of <ConfigID(int); Config(double[]); Name(string); ResourceID(int)>`
  The complete list of configurations in the sequencing problem. Each configuration record consists of an identifier, a vector of coordinates with common dimension over all configurations, and an optional name and resource identifier.
- `ProcessHierarchy: list of <ProcessID(int); AlternativeID(int); TaskID(int); MotionID(int); ConfigIDList(int[]), Bidirectional(bool), Name(string)>`
  The definition of the hierarchy of processes, alternatives, tasks, and motions, including the list of configurations visited during each motion. Each motion identifier must be universally unique. Tasks within an alternative are executed in increasing order of their IDs. The optional bidirectional flag can be used to override `BidirectionalMotionDefault`.
- `Cyclic: (bool)`
  Indicates whether after the execution of the processes, the robot must return to its initial configuration. If yes, then a start configuration is required. Otherwise, the start and end configurations are optional.
- `StartConfigID: (int)`
  Identifier of the start configuration. Optional, see also the keyword `Cyclic`.
- `FinishConfigID: (int)`
  Identifier of the finish configuration. Optional, see also the keyword `Cyclic`.
- `ProcessPrecedences: list of <ProcessIDBefore(int); ProcessIDAfter(int)>`
  Precedence constraints between processes. Directed cycles are not allowed.
- `MotionPrecedences: list of <MotionIDBefore(int); MotionIDAfter(int)>`
  Precedence constraints between motion. Directed cycles are allowed.
- `BidirectionMotionDefault: (bool)`
  If *true*, then the reverse of each specified motion is automatically added to every task. The identifier of the reverse motion is the opposite of the original identifier. This default value can be overridden for individual motions in the `ProcessHierarchy` records.

Fig. 3 presents a small pick-and-place problem, where the robot must start from the camera position, take two workpieces one by one from their current positions to a target position, and then return to the camera position. In the configuration list, the 6-DoF robot configuration is fully defined for the camera position, while up to 3 alternative IK solutions are available for the task-space positions corresponding to picking and placing tasks. In the process hierarchy, the three-digit identifiers are constructed from the process ID (first digit), the alternative ID (second digit) and the task ID (last digit). Hence, process 100 can be executed using the single alternative 110, which contains two tasks, the picking task 111 and the placing task 112. Task 111 can be executed using motions (configurations) 1, 2, or 3.

The process precedence $200 \rightarrow 100$ indicates that workpiece 2 blocks access to workpiece 1 entirely. Moreover, the motion precedences describe a situation where configuration `Pick_2_1` for picking workpiece 2 is also blocked by workpiece 1, and hence, becomes applicable only after workpiece 1 is removed. Observe that the ensemble of all these precedence constraints implies that `Pick_2_1` cannot be applied, and therefore, could be removed from this problem definition. Resource identifiers are added to the configurations only for the sake of illustrating the format of changeovers later.

4.2. Costs

Travel costs or times between configurations can be defined using the following keywords:

- `DistanceFunction: Euclidean/ Manhattan/ Max/ TrapezoidTime/ Matrix` The travel cost between two configurations can be computed using the Euclidean or the Manhattan distance functions, the maximum norm, by assuming a trapezoid speed profile (i.e., computing the robot travel time subject to given acceleration and speed limits), or they can be specified explicitly in a matrix format.
- `TrapezoidAcceleration: (double[])`
  Maximum acceleration for each dimension (e.g., each robot joint). Required if `DistanceFunction` = `TrapezoidTime`.
- `TrapezoidSpeed: (double[])`
  Maximum speed for each dimension (e.g., each robot joint). Required if `DistanceFunction` = `TrapezoidTime`.

```
Cyclic:                    True
StartConfigID:             0
BidirectionMotionDefault:  False
ConfigList:
0; [-1.86;-1.44; 1,69;-1.99;-1.51;-0.29]; Camera; 1
1; [-1.22;-2.28; 4.72;-0.87; 0.69; 1.59]; Pick_1_1; 2
2; [ 1.59;-0.79; 1.55; 3.60;-2.38; 1.11]; Pick_1_2; 2
3; [-1.22;-2.28;-1.56; 5.40; 0.69; 1.59]; Pick_1_3; 2
4; [-1.11;-2.53; 5.19; 2.05;-0.69;-1.57]; Pick_2_1; 3
5; [-1.11;-2.17; 4.49;-0.74; 0.69; 1.56]; Pick_2_2; 3
6; [ 1.67;-0.88; 1.77; 3.43;-2.36; 1.03]; Pick_2_3; 3
7; [ 1.22; 4.67;-2.04;-0.40; 1.72;-0.28]; Place_1_1; 2
8; [ 1.22; 4.42;-1.45; 2.28;-1.72; 2.85]; Place_1_2; 2
9; [-2.40;-1.55; 2.05;-2.58;-1.13;-0.75]; Place_2_1; 3

ProcessHierarchy:
#Pick workpiece 1
100; 110; 111; 1; [1]
100; 110; 111; 2; [2]
100; 110; 111; 3; [3]
#Place workpiece 1
100; 110; 112; 7; [7]
100; 110; 112; 8; [8]
#Pick workpiece 2
200; 210; 211; 4; [4]
200; 210; 211; 5; [5]
200; 210; 211; 6; [6]
#Place workpiece 2
200; 210; 212; 9; [9]

ProcessPrecedences:
200; 100

MotionPrecedences:
1; 4
2; 4
3; 4
```

Fig. 3. Sample problem in SEQ format (part 1): process hierarchy. Lines starting with a # symbol contain comments.

- `ConfigMatrix: <ConfigIDHeader(int[]); Costs (double[][]); NameFooter(string[]), ResIDFooter(int[])>`
  If travel costs are given explicitly in a matrix format, then first, the configuration identifiers must be enumerated, and then the distances between each pair of configurations must be specified. Optionally, configuration names and corresponding resources can be provided. If this matrix is specified, then the configuration list can be omitted.
- `OverrideCost: list of <ConfigID1(int); ConfigID2 (int); Cost(double); Bidirectional(bool)>`
  The above defined distance function can be overridden for specific pairs of configurations.
- `IdlePenalty: (double)`
  An optional penalty can be added to the above cost each time the robot aborts performing effective tasks and makes an idle movement, i.e,. if the end point of an effective motion is different from the start point of the next effective motion. Useful in cutting and drawing problems.
- `AddMotionLengthToCost: (bool)`
  When this option is enabled, the cost accumulated by changing configurations *within effective motions*, including travel costs and resource changeovers, is also added to the solution cost. Otherwise, only the cost *between effective motions* is calculated.

Fig. 4 shows an example where travel times are computed using a trapezoid speed profile, based on the given robot joint acceleration and speed limits. The automatically computed travel times are overridden for two pairs of configurations where the robot must take a complex path to avoid collisions. These travel times must be determined in a preprocessing step by path planning. If collisions are a common problem in the given application, then a reasonable alternative representation is specifying the travel times of all collision-free paths in a matrix format.

4.3. Resources

The following keywords can be applied to characterize the required resources and the changeover times between them. Note that the single resource used in each individual configuration is specified in the configuration list. Fig. 5 shows an example where the changeover times are specified in a matrix format.

- `ResourceChangeover: None/ Constant/ Matrix`

```
DistanceFunction:        TrapezoidTime
TrapezoidSpeed:          [2.2;2.2;2.2;3.3;3.3;3.3]
TrapezoidAcceleration:   [10.4;10.4;10.4;10.4;10.4;10.4]
IdlePenalty:             0
AddMotionLengthToCost:   False
OverrideCost:
4; 9; 0.75
6; 9; 0.56
```

Fig. 4. Sample problem in SEQ format (part 2): costs.

```
ResourceChangeover:            Matrix
ResourceChangeoverFunction:    Add
ChangeoverMatrix:
1; 2; 3
0.0; 12.0; 13.0
21.0; 0.0; 23.0
31.0; 32.0; 0.0
```

Fig. 5. Sample problem in SEQ format (part 3): resources.

If resource changeover takes time, then its value can be either constant or taken from a matrix. In both cases, a resource identifier must be provided for each configuration in the configuration list.

- `ResourceChangeoverFunction: Add/ Max`
  Indicates whether changeover times must be added to the travel cost between configurations (plus the potential idle penalty), or the maximum of the two values must be taken. Required if `ResourceChangeover` ≠ `None`.
- `ChangeoverConstant: (double)`
  Uniform resource changeover time. Required if `ResourceChangeover` = `Constant`.
- `ChangeoverMatrix: <ResourceIDHeader(int[]), ChangoverCostMatrix(double[][])>`
  If changeover times are specified in matrix format, then the matrix header must define the sequence of resource identifiers. This header is followed by the matrix of changeover time values, where rows correspond to the *from resource*, and columns to the *to resource*. Required if `ResourceChangeover` = `Matrix`.

4.4. Solver settings

The following keywords can be used to customize the solution strategy (see Fig. 6 for an example):

- `LocalSearchStrategy: GreedyDescent/ GuidedLocalSearch/ SimulatedAnnealing/ TabuSearch/ ObjectiveTabuSearch`
  Meta-heuristic used by the open source solver during local search. Often, guided local search results in the best solution quality, or greedy descent can be used for a quick validation of the problem definition.
- `UseMIPpresolver: (bool)`
  Use a mixed-integer linear programming (MIP) solver to generate an initial solution. This method is guaranteed to find a feasible initial solution if there exists one, but often results in worse solution quality than the default VRP initial solution. Recommended in case of precedence constraints in the model, see also Section 5.3.
- `UseShortcutInAlternatives: (bool)`
  Pre-compute shortest paths within each alternative when encoding the problem into a GTSP, see Section 5.2. Recommended in case of long chains of tasks within an alternative.
- `TimeLimit: (int)`
  Limit on the total computation time in milliseconds. Obligatory for all local search strategies except for greedy descent, where it is optional.

5. Solution approach

The above description of the robotic process planning and sequencing problem is encoded in the form of a GTSP with precedence constraints, and solved using the open-source VRP library of Google OR-Tools [31], using partly the built-in algorithms of OR-Tools and partly custom algorithms. The encoding and the algorithms are presented in the following sections.

5.1. Encoding into a GTSP with precedence constraints

In the GTSP representation of the problem, one *vertex* corresponds to each motion in the process hierarchy. Directed *edges* connect motion $M_{ij}^{k\ell}$ to motion $M_{ij}^{(k+1)\ell'}$, i.e., to all possible motions of the next task in the same alternative if $k < K(i,j)$. Moreover, the motions in the last task of every alternative, $M_{ij}^{K(i,j)\ell'}$, are connected to all possible first motions of other processes $M_{i'j'}^{1\ell'}$ with $i \neq i'$. The overall structure of the resulting GTSP is depicted in Fig. 7 for a sequencing problem with two processes.

For each directed edge, connecting motion $M = (C_1, C_2, \dots, C_N)$ to motion $M' = (C'_1, C'_2, \dots, C'_{N'})$, the edge cost $c(M, M')$ is calculated based on the configurations within the motions. If the `AddMotionLengthToCost` switch is enabled, then costs are summarized over each elementary step, i.e., $c(M, M') = \sum_{i=1}^{N-1} c(C_i, C_{i+1}) + c(C_N, C'_1)$. Otherwise, only the cost between the two motions is considered, i.e., $c(M, M') = c(C_N, C'_1)$. The cost of each elementary step is calculated based on the specified distance function, and this value is combined with the corresponding resource changeover cost, by taking the sum or the maximum of the two values, as required by the `ResourceChangeoverFunction` switch. If necessary, then the idle penalty is also added to $c(C_N, C'_1)$.

Prescribed *start* and *finish configurations* are captured by an additional process with a single motion, and a corresponding GTSP depo vertex, which is connected to vertices belonging to other processes the same way as above. If the start or the finish configuration is specified in the input, then the cost of these edges is calculated as described above. Otherwise, the cost of the corresponding edges is zero.

```
LocalSearchStrategy:          GuidedLocalSearch
TimeLimit:                    5000
UseMIPprecedenceSolver:       True
UseShortcutInAlternatives:    False
```

Fig. 6. Sample problem in SEQ format (part 4): solver settings.

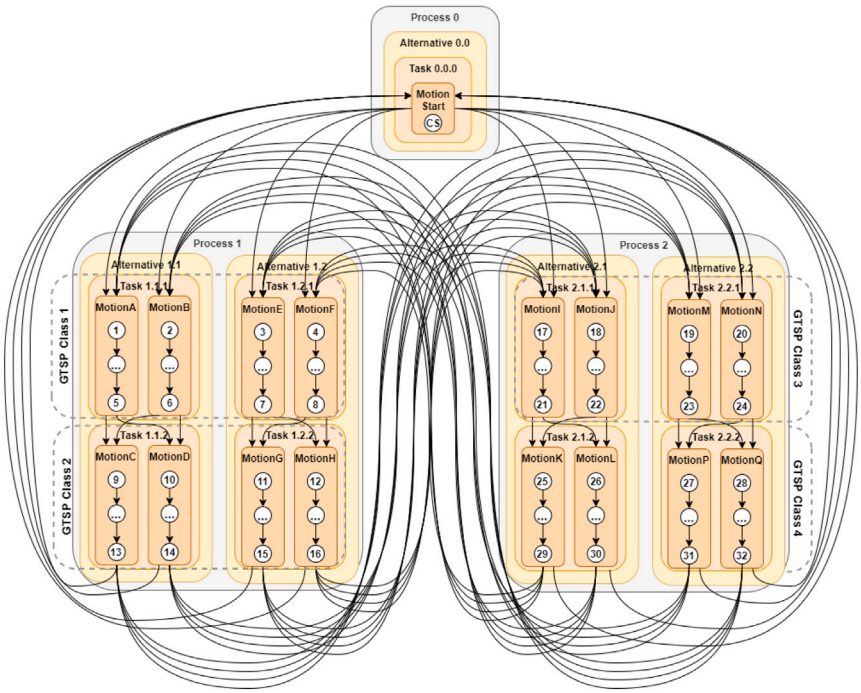


Fig. 7. GTSP representation of a sequencing problem with two processes, two alternatives per process, and two tasks per alternative, as well as an additional process for the start and finish configuration in the cyclic problem. GTSP classes are displayed with dashed lines.

Vertex classes in the GTSP are used to capture the discrete choice between alternatives within a process, and between motions within a task. This is achieved by defining a separate class for each process i and each task position k, and assigning motions $M_{ij}^{k\ell}$ with the given i and k to that class, see Fig. 7. A special case occurs when alternatives within a process contain different numbers of tasks. Assume $K(i,j) < K(i,j')$, and there is a class to be created for process i and task position k with $K(i,j) < k \leq K(i,j')$. In this case, last motions $M_{ij}^{K(i,j)\ell}$ from the shorter alternative i are added to the given class. Observe that this assignment violates the common assumption in GTSP that classes are disjoint, but this does not cause any major complication. The core requirement that exactly one vertex should be selected from each class still holds, and the disjunctive constraints in OR-Tools captures overlapping classes as well.

All process and motion precedence constraints in the sequencing problem are translated into *precedence constraints* among the vertices of the GTSP. After checking that process precedences do not contain a directed cycle, they are translated into motion precedences. Namely, a process precedence $P_i \rightarrow P_{i'}$ is converted into a set of motion precedences between all last motions of process i and all first motions of process i', i.e., $M_{ij}^{K(i,j)\ell} \rightarrow M_{i'j'}^{1\ell'}$ for each j, j', ℓ, ℓ'. Then, each motion precedence is encoded into a logical constraint in OR-Tools which states that if both involved vertices are active in a solution, then they must be executed in the given order.

5.2. Pre-computing shortest paths in the GTSP

While the above GTSP encoding is a valid and full-fledged representation of the task sequencing problem, its structure can be simplified by discovering shortest paths within each alternative. This transformation is reasonable if an alternative contains a long sequence of tasks. Namely, for each alternative, the shortest path is computed between each possible first motion vertex and each possible last motion vertex. In the transformed GTSP, one vertex corresponds to each such shortest path, and all vertices of the same process belong to the same class. The vertices of the new GTSP inherit all precedence constraints from the corresponding path in the original GTSP. The alternative representation of the sequencing problem originally introduced in Fig. 7 is displayed in Fig. 8. The transformed representation has a simpler structure, and it is shallower but wider than the original GTSP: if the number of processes, alternatives, tasks and motions are denoted by I, J, K, and L, then the original and the transformed representations contain $IJKL$ and $I(JL)^2$ vertices, respectively. Accordingly, applying the transformation can be advantageous if the problem involves long chains of tasks within the alternatives, but few alternatives within a process and few motions within a task.

5.3. Initial solution

The above defined problem is solved using local search. While the algorithms provided by OR-Tools are efficient on GTSP problems, the handling of precedence constraints – especially motion precedences – is a challenge. At this point, the difference between PCGTSP, a problem model commonly studied in the literature, and the proposed GTSP model with precedence constraints must be highlighted: in PCGTSP, the precedence constraints are defined between the given disjoint classes of vertices, i.e., the sequencing and the vertex selection subproblems are connected only via the solution cost, but feasibility is

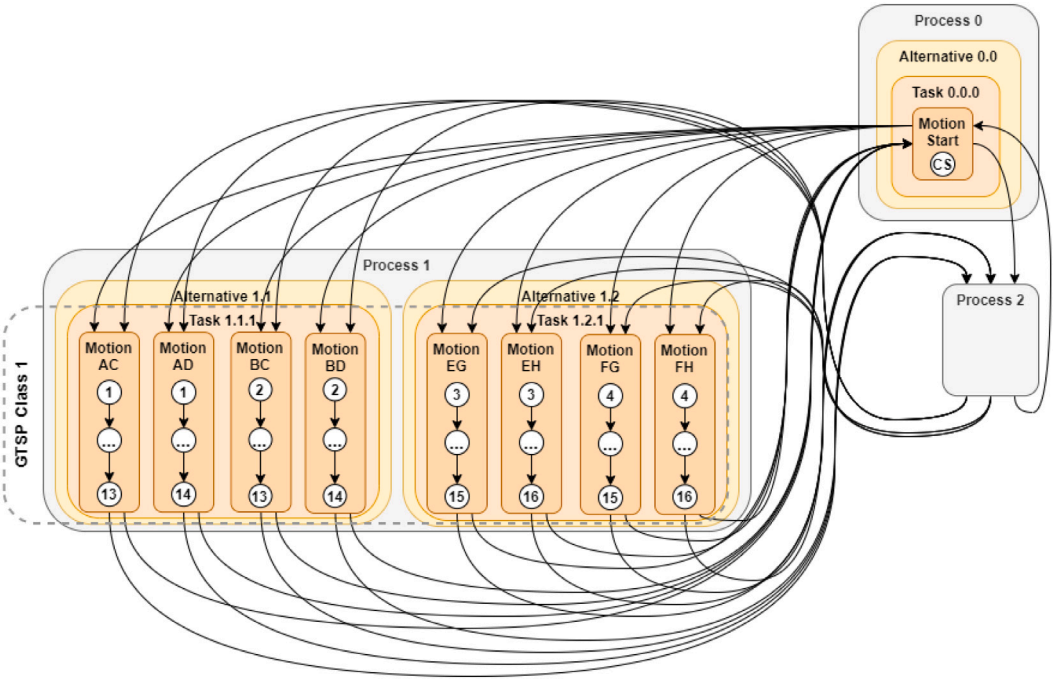


Fig. 8. Alternative GTSP representation of the problem in Fig. 7 after pre-computing all shortest paths. The detailed content of *Process 2* is not shown in the figure.

independent of vertex selection. In contrast, in the proposed model, motion precedence constraints are defined between vertices, which implies that the interdependence of the sequencing and vertex selection sub-problems is much stronger, as they jointly determine the feasibility of a tour. An example of this requirement is provided in the building blocks problem in Section 6.5. The consequences are twofold: first, the open-source solvers for PCGTSP, such as [6], are not applicable to the proposed model; and second, finding a feasible solution itself is a hard combinatorial problem.

For this reason, the problem of finding a feasible solution is formulated as a mixed-integer linear program (MIP). Variables in the MIP include binary decision variables x_{ij}^{kl} for each motion M_{ij}^{kl}, indicating if the given motion is selected, as well as continuous variables p_i capturing the position of process P_i in the solution. $\mathcal{P}$ and $\mathcal{R}$ denote the set of process and motion precedence constraints, respectively:

$$\sum_{j,\ell} x_{ij}^{\min(k,K(i,j))\ell} = 1 \qquad \forall i,k \tag{1}$$

$$p_i + 1 \leq p_{i'} \qquad \forall (i,i') \in \mathcal{P} \tag{2}$$

$$p_i + 1 \leq p_{i'} + I(2 - x_{ij}^{kl} - x_{i'j'}^{k'l'}) \qquad \forall (M_{ij}^{kl}, M_{i'j'}^{k'l'}) \in \mathcal{R} \tag{3}$$

$$\sum_{\ell} x_{ij}^{k\ell} = \sum_{\ell} x_{ij}^{(k+1)\ell} \qquad \forall i,j,k < K(i,j) \tag{4}$$

$$1 \leq p_i \leq I \qquad \forall i \tag{5}$$

$$x_{ij}^{kl} \in \{0,1\} \qquad \forall i,j,k,\ell \tag{6}$$

Constraint (1) states that exactly one motion must be selected from each process and each task position (unless the given task position is empty in an alternative, which is achieved by using $x_{ij}^{\min(k,K(i,j))\ell}$ instead of $x_{ij}^{k\ell}$, just as it is done in the GTSP formulation). Inequalities (2) and (3) ensure that process and motion precedence constraints are satisfied. Yet, a motion precedence implies a restriction on the sequence of the corresponding processes only if both motions are selected in the actual solution, which condition is captured by a so-called *big-M* constraint. If a motion is selected from a given task, then a motion from the next task of the same alternative must also be selected (4). The solution of this MIP serves as an initial solution during local search. Yet, for problems without precedence constraints, it is recommended to use the built-in algorithms of OR-Tools, which often result in better initial solutions by applying common VRP insertion heuristics.

5.4. Local search

In the improvement phase of the local search, the planner relies on the VRP-specific meta-heuristics offered by the underlying solver, according to the choice of the user. The available strategies include *Greedy Descent*, running until the first local minimum, *Guided Local Search* [32], which tries to escape local minima by a modified objective that penalizes certain features of the solution, *Simulated Annealing* [33], also accepting worsening moves with a given probability, or *Tabu Search* [34], which builds a short-term memory about search history to avoid returning to previously visited solutions. In experiments on problems from various applications, Guided Local Search has shown the best performance. OR-Tools combines multiple common VRP neighborhood functions according to the status of the search, including *Or-Opt*, *Two-Opt*, the *Lin–Kernighan* heuristic, and various other, more complex neighborhoods.

6. Experimental evaluation

6.1. Overview of case studies

This section introduces five, apparently very different robotic applications, presents how the arising task sequencing and process planning problems can be captured using the proposed approach, and investigates the performance of the approach in each of the applications. A brief overview of the applications is given in Table 2.

Computational experiments were performed with ProSeqqo version 1.0, available open source on GitHub.[3] ProSeqqo has been implemented in C#, and it is built on top of Google OR-Tools version 9.0.9048. The experiments were run on a laptop computer with Intel i7-10510U 2.30 GHz CPU and 16 GB RAM, under a 64-bit Windows 10 operating system.

[3] https://github.com/sztaki-hu/proseqqo.

Table 2
Overview of sample applications. Problem size involves the number of processes/alternatives per process/tasks per alternative/motions per task. Letter *B* denotes bidirectional motions.

	Choice	Size	Cost function	Additional features
Pick-and-place, Section 6.2	IK solution	3-30/1/ 2/1–10	6D trapezoid time	Model: process prec. Solver: shortest paths
Drawing, Section 6.3	Direction	265-643/ 1/1/1B	2D Euclidean	Model: idle penalty
Engraving, Section 6.4	Direction	250-4048/ 1/1/1B	2D Euclidean	Model: idle penalty
Building blocks, Section 6.5	Grasp	25-100/ 1/2/2	3D Euclidean	Model: process & motion prec. Solver: initial sol.
Grinding, Section 6.6	Direction	19/1/1/ 1B	Matrix (collision-free path time)	Model: process prec. Solver: initial sol.

6.2. Camera-based robotic pick-and-place

The detailed description of the camera-based pick-and-place application is presented in Section 3.2. The goal of the computational experiments was evaluating the efficiency of different GTSP formulations (i.e., with or without pre-computing the shortest paths) and of different local search algorithms on a large set of randomly generated, but realistic problem instances. Namely, instances were generated using the simulation model of the physical pick-and-place workcell, and parts were placed on the vibrating table in random poses. Although the approach would allow using multiple candidate grasps corresponding to multiple alternatives in the model, a single feasible grasp (alternative) was used for the given part geometry. The candidate robot configurations for picking the parts were computed using a closed-form IK solver using the nominal kinematics of the UR robot, and colliding configurations were filtered out using the collision detection and path planning library [35]. This resulted in 1–10 picking configurations per part, and accordingly, 1–10 motions in the first, picking tasks of each process in the model. All parts had to be placed into the same pose at the entry of a press machine, using one of the 4 collision-free robot configurations, i.e., motions in the second, placing task of the processes. Five such instances were generated for each problem size of 3–30 parts, resulting in 80 instances altogether. Two alternative GTSP formulations, with and without pre-computed shortest paths were generated for each instance, and both formulations were solved using a greedy descent (GD) algorithm to local optimality, and using guided local search (GLS) with four different time limits between 0.1 s and one minute. It is noted that the instances were solved using tabu search and simulated annealing, too, but these algorithms were clearly dominated by GLS, hence, their results are omitted in the paper. Moreover, the exact optimal solutions were available for the small instances with 3–4 parts from an exact solver.

The results are presented in Table 3, where each row contains aggregated results for the ten instances with the same number of parts. In the upper part of the table, the GTSP model without pre-computed shortest path (PSP=N), whereas in the lower part, the GTSP with pre-computed paths (PSP=Y) was used. Column *GTSP vertices* displays the average number of vertices in the GTSP representation. Then, the average value of the local optimal solution by GD and the corresponding computation time are presented. Finally, the values of the solutions found by GLS with different time limits are shown. The computation of the shortest paths in the small GTSP took insignificant time.

For all small problems where the exact optimum is known, GLS could find that optimal solution in at most 0.1 s for three parts, or 1 s for four parts, using either GTSP formulation. For most of the larger problems, the value of the exact optimum is unknown, but GLS could find high-quality solutions for these instances as well. Without pre-computed shortest paths, the local optima found by GD were surprisingly weak, 10%–32% worse than the best known solution. In contrast, shortest paths could effectively simplify problem structure, resulting in high-quality local optima only 0.4%–3.9% worse than the best known solution. These gaps were reduced effectively by GLS, to at most 5.5% (PSP=N) or 4.5% (PSP=Y) in 0.1 s, or to at most 4.1% (PSP=N) or 2.0% (PSP=Y) in ten seconds.

Various conclusions can be drawn from these experiments. First, in applications with multiple tasks in an alternative, it is definitely worth pre-computing the shortest paths, and the performance gap between the two GTSP representations will most likely increase further in problems with longer chains of tasks. Second, GLS showed robust performance and found high-quality solutions quickly. Third, the above computation times facilitate the application of the solver in online planning scenarios as well.

6.3. Robotic cartoon drawings

While the robotic cartoon drawing application was originally built as a popular science demonstration, the involved process planning and task sequencing problem illustrates various real industrial problems from the domains of cutting, welding, and painting. In this application, a visitor's picture is taken, and after appropriate image processing, it is drawn on a white board by a UR5 robot using a marker pen, see Fig. 9 and the video demonstrations (using a previous version of the sequence planner).[4,5] Since force feedback is applied when pushing the pen against the board, lifting up and then re-positioning the pen takes considerable time. A special challenge is that the sequencing problem must be solved online, with as little computational time as possible.

The problem consists in sequencing the drawn lines and choosing their directions. The drawing of each line is captured as a separate process, with a single alternative, a single task, and a single but bidirectional motion. The objective is minimizing the 2D Euclidean distance traveled plus the *idle penalty* for lifting up the pen.

Experiments on the robotic cartoon drawing application investigated solution quality with different algorithms and different time limits. Table 4 displays the results for five instances, corresponding to the four faces and the Christmas greetings message displayed in Figs. 9 and 10, solved with GD until reaching a local minimum, and with GLS using four different time limits ranging from one second to ten minutes. The indicated objective values include a combination of the total distance traveled (including both the lines drawn and the transitions between them) and the idle penalty. The latter penalizes the time required for re-positioning the pen and the potential discontinuity of the lines after re-positioning. Cartoons consisted of 265–643 lines, corresponding to the same number of tasks in the model. The local minimum found by GD was typically 3%–8% worse than the best solution known, except for cartoon B, where it was nearly 22% worse. Solution quality was similar after a 1 s run of GLS, but the quality gap decreased to 3%–6% after 10 s even for the notorious cartoon B. Other local search strategies were dominated by GLS.

[4] Robotic cartoon drawing (youtube link): https://youtu.be/8ULIP_5nEJ0.
[5] Christmas greetings (youtube link): https://youtu.be/NhOibxWIpH0.

Table 3
Results for the camera-based pick-and-place application. Column PSP indicates if shortest paths were pre-computed in the GTSP formulation (Y) or not (N).

Parts	PSP	GTSP	GD		GLS			
		vertices			0.1 s	1 s	10 s	1 min
3	N	28.8	10.73	(0.02 s)	9.73	9.73	9.73	9.73
4		36.0	15.64	(0.03 s)	12.65	12.60	12.60	12.60
5		54.3	18.88	(0.05 s)	15.44	15.38	15.38	15.38
10		117.5	37.72	(0.08 s)	31.01	30.75	30.09	29.90
15		180.0	57.11	(0.15 s)	45.59	45.27	44.44	43.36
20		249.8	75.29	(0.34 s)	61.40	61.18	60.25	58.32
25		305.0	94.58	(0.46 s)	76.26	75.63	75.34	74.11
30		358.2	114.64	(0.57 s)	92.35	91.64	91.16	88.39
3	Y	61.2	9.78	(0.00 s)	9.73	9.73	9.73	9.73
4		74.0	12.73	(0.00 s)	12.63	12.60	12.60	12.60
5		131.2	15.44	(0.01 s)	15.40	15.38	15.38	15.38
10		304.0	30.66	(0.03 s)	30.64	30.39	30.16	30.02
15		474.0	45.02	(0.05 s)	44.93	44.58	43.93	43.56
20		673.2	60.58	(0.09 s)	60.58	60.26	59.67	58.79
25		814.0	75.57	(0.12 s)	75.76	75.43	75.15	73.67
30		946.8	90.76	(0.18 s)	91.59	90.61	89.27	87.61

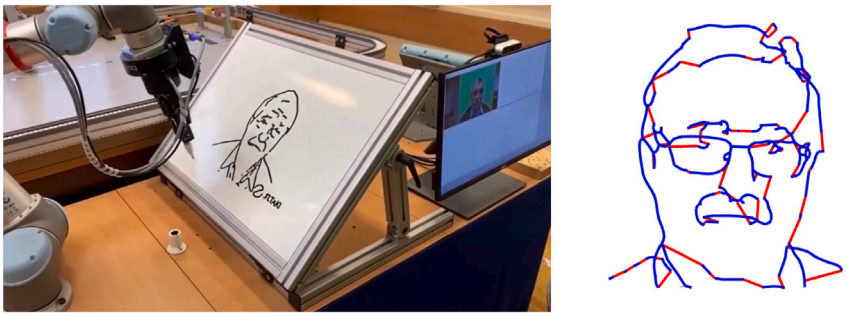


Fig. 9. Robotic cartoon drawing by a UR5 collaborative robot, and the task sequence for a sample portrait (cartoon C).

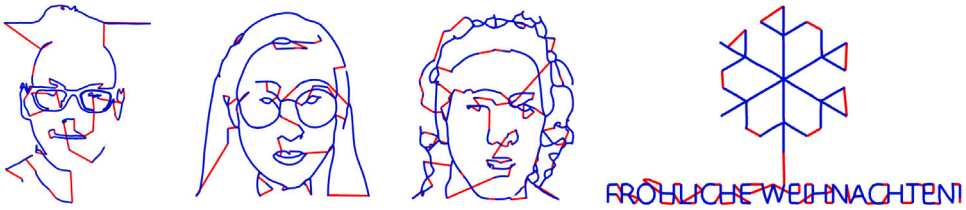


Fig. 10. Best solutions found for cartoons A, B, D, and E. Blue lines indicate the actual lines drawn, whereas red lines stand for idle movements. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
Results for the cartoon drawing application.

Cartoon	Lines	GD		GLS			
				1 s	10 s	1 min	10 min
A	265	811.83	(1.05 s)	811.83	790.55	764.02	751.80
B	485	3899.34	(4.47 s)	4081.54	3399.34	3387.65	3203.63
C	614	3970.43	(6.98 s)	4312.00	3970.43	3927.73	3818.12
D	643	5204.70	(7.18 s)	5364.63	5185.53	5074.26	5033.43
E	296	3502.24	(1.99 s)	3595.68	3476.95	3411.11	3310.68

6.4. Robotic laser engraving

Somewhat similarly to the previous application, the goal is to create a 2D image from lines on different objects by laser engraving, such as the Celtic knot drawing in Fig. 12. While the problem model is identical to that of the previous application, the special challenge is the handling of the large number of lines (e.g., up to 4000) in the raw input. The problem is relevant both for one-of-a-type products, with as low computation times as possible, and for mass production, where large computation times can also be allowed. Accordingly, experiments focused on the trade-off between computation time and execution time by varying the resolution and the time limit for the solver.

The focus of the experiments on the laser engraving application was the evaluation of the performance of the planner on huge problem instances. For this purpose, the problem of engraving the Celtic knot motif shown in Fig. 11 was investigated with different resolutions: the 4048 line segments in the original motif were heuristically merged in multiple steps, resulting in 250 segments with the lowest resolution. It is noted that the motif itself remains identical, while having less line segments and less opportunities to switch between segments has a two-fold effect on solution quality: the optimal solution might be lost, but

Fig. 11. Laser-engraved wooden coin product, and the corresponding task sequencing problem.

Table 5
Results for the laser engraving application.

Lines	GD		GLS				
			1 s	10 s	1 min	10 min	30 min
250	485.59	(1.15 s)	485.59	481.84	468.93	465.45	464.27
466	493.43	(4.56 s)	507.80	493.43	488.37	464.76	463.28
883	485.33	(33.50 s)	514.10	510.13	485.17	470.68	463.37
1800	480.43	(89.91 s)	515.03	512.55	481.38	480.43	480.43
4048	491.42	(546.19 s)	–	501.28	496.27	491.42	491.42

smaller instances can be managed more efficiently within the same time frame. The objective was minimizing the total travel distance plus a penalty for switching between line segments. The best solutions found with the different resolutions are displayed in Fig. 12.

The detailed results are presented in Table 5. The best solution over all experiments was achieved after 30 min with GLS, using 466 lines, i.e., a relatively low resolution. This solution is also an upper bound on all instances with higher resolution, while in theory it may happen that with 250 lines, the exact optimum is worse than this. Compared to this solution, GD terminates in a 3.7%–6.5% worse local minima, which takes one second with the lowest, and almost one minute with the highest resolution. In one second, GLS achieves a 4.8%–11.2% gap (but it fails to find a solution with the highest resolution), which is gradually decreased to 1.2%–7.1% after one minute. In order to achieve the best solution with a given, fixed time limit, it is recommended to use a low resolution (e.g., 250 lines) if the limit is at most one minute, which corresponds to the online usage scenario for a one-of-a-kind product. In an offline planning scenario before large-series production, it can be beneficial to improve the resolution of the sequencing model as well (e.g., using ca. 500 lines in case of a 10–30 min time limit).

6.5. Robotic building blocks

The building blocks application is a student project, originally focused on the identification of objects, their poses, and the potential ways of grasping them using a vision camera. The building blocks must be grasped using a two-finger gripper and taken from their identified source poses to the specified target poses, without applying an intermediate buffer. An interesting feature from the sequence planning point of view is the interdependence of the task sequence and the grasping modes: e.g., the green cylinder in Fig. 13 can be grasped using an east–west (EW) orientation of the gripper only if the blue cube is removed first. At the same time, it can be grasped using a north–south (NS) orientation if the red cube is removed beforehand. These correspond to conditional precedence constraints. On the other hand, some precedence constraints are independent of the grasping modes, e.g., between two blocks placed on the top of each other. Similar precedence and conditional precedence constraints stem from the target poses. While the physical implementation has been tested with few, up to 10 blocks only, a simulated environment has been applied to investigate the performance of the proposed planner on large instances with up to one hundred blocks and a thousand motion precedence constraints between them.

The sequencing problem originating from this application has been modeled in the 3D task space, with one process standing for each block, which contains one alternative and two tasks for picking and placing the block. The two grasping modes, NS and EW, are captured by two motions within the task. Note that the current blocks have a 90° rotational symmetry, which implies that the target configuration is realized correctly independently of the chosen grasping mode. For asymmetric blocks, the grasps could be modeled as different alternatives within the process. Conditional precedences are modeled as motion precedences, whereas classical precedence constraints as process precedences.

Experiments on the building blocks problem addressed the evaluation of the solver on instances with a large number of motion precedence (i.e., conditional precedence) constraints, where finding a feasible solution in itself is challenging. Special attention was paid to the performance of the MIP proposed in Section 5.3 for computing an initial solution. Artificial instances of the building blocks problem were generated, in which a building had to be dismantled to build up another building from the same blocks. Both buildings were geometrically dense structures, implying a large number of motion precedence constraints, but the existence of a feasible solution was guaranteed by construction (yet, in additional tests on infeasible instances, the MIP could also prove quickly that no solution existed).

The results are presented in Table 6, where each row displays average results over five instances with a given problem size. Columns *Blocks* and *Prec* contain the number of building blocks and the average number of motion precedence constraints, respectively. Column *MIP time* shows the computation time for finding an initial solution. Then, column *GD* presents the average value of the local optimum found by GD and the required computation time. Finally, column *GLS* shows the values of the solutions constructed by GLS with four different values of the time limit. MIP solution times are not accounted for in these limits. The results indicate that the MIP approach could find a first solution quickly, in 3.35 s even for the largest problem size with 100 building blocks. There was a significant variance among the solution times within a given problem size as well, e.g., an outstanding instance with only 25 blocks and 1.2 s solution time resulted that the average MIP solution time was higher for 25 blocks than for 50 blocks. Due to the large number of precedence constraints, order flexibility was very low, and therefore, local search could only slightly decrease robot travel times compared to these initial solutions: GD found local optima in 0.16 s (for 25 blocks) or 54.63 s (for 100 blocks) that were only 0.09%–0.31% worse than the best known solutions, found by GLS in 30 min.

6.6. Robotic grinding of furniture parts

In the last application scenario, the goal is the robotic belt grinding and polishing of cast aluminium furniture parts [36], see Fig. 14. The surface of the part is decomposed into nine longitudinal stripes, and each stripe must undergo up to three surface finishing tasks: rough grinding, fine grinding, and polishing. Five stripes need all the three tasks, whereas four stripes need only polishing, resulting in 19 tasks altogether. For technological reasons, all rough grinding tasks must precede all fine grinding tasks, which in turn must precede all polishing tasks.

Each task corresponds to a robot motion specified in the 6D joint configuration space of the robot, which guides the part along a contact trajectory between the given stripe of the part surface and the tool. The direction of the motion can be reversed. Idle motions between the effective tasks can be rather complicated due to the difficult part geometry and the densely populated workcell. Hence, all possible 38×38 idle motions between the effective path end points were pre-computed using the library [35].

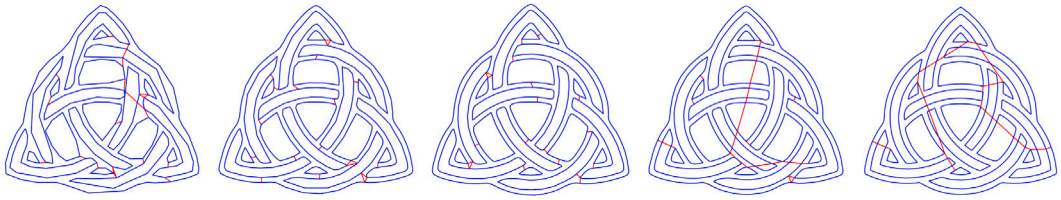


Fig. 12. Best solutions found for the same Celtic knot motif with different resolutions (250, 466, 883, 1800, 4048 lines). Blue lines indicate the lines drawn, whereas red lines stand for idle movements. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 6
Results for the building blocks application.

Blocks	Prec	MIP	GD		GLS			
		time (s)			30 s	1 min	5 min	10 min
25	269	0.31	5 145	(0.16 s)	5 141	5 141	5 141	5 140
50	588	0.15	10 636	(4.18 s)	10 622	10 618	10 606	10 603
75	1088	2.38	16 079	(22.82 s)	16 075	16 068	16 060	16 045
100	1360	3.35	21 148	(54.63 s)	21 174	21 154	21 119	21 105

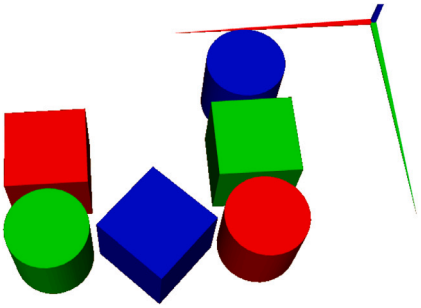


Fig. 13. A small sample instance of the building blocks problem. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The process planning problem then consists in sequencing the above tasks and selecting the direction of each effective motion. Each of the above tasks was represented as separate processes in the model, with appropriate process precedence constraints between them. Each process contains a single alternative and the single task, with a single bidirectional motion. The durations of the pre-computed collision-free idle motions were provided in matrix format as input for the planner. This way, the objective is minimizing the total transition time, which is equivalent to minimizing the total processing time, since the duration of the effective tasks is fixed.

In this application, a single real industrial problem instance was available, corresponding to the surface finishing of a metallic furniture part. The surface was decomposed into 5 stripes, each of which underwent 1–3 finishing steps, resulting in 19 tasks altogether. The objective was minimizing the transition times between the effective tasks, assuming trapezoid speed profiles in the 6D robot joint configuration space on each section of the pre-computed collision-free paths between the 38×38 motion end points.

The solutions computed using different local search strategies, including GD, GLS, and tabu search (TS), and with different time limits are compared to that by a human expert using simulation software in Table 7. The initial solution was computed using the MIP approach presented in Section 5.3. Solving this small-sized MIP took negligible time. Even the simple GD approach could decrease the transition times by 10% w.r.t. the human expert's solution. TS found a very good solution, corresponding to a 23.5% gain in only 5 s, but it could not improve that further. In contrast, GLS kept on enhancing the solution for several minutes, finishing with a 23.7% improvement compared to the human expert's solution. The validity of this best solution, including the avoidance of collisions, was also confirmed in simulation experiments and by the human expert. Hence, the application of the planner results in a substantial performance improvement in the industrial robotic grinding cell.

Table 7
Results for the robotic grinding application.

Time limit	Human expert	GD	GLS	TS
5 s	38.93	35.16	32.47	29.79
10 s	(one working day)	(0.27 s)	32.47	29.79
1 min			32.47	29.79
10 min			29.70	29.79
30 min			29.70	29.79

6.7. Discussion of the results

The sections above demonstrated how rather different robotic applications can be captured using the proposed problem definition language and solved using the proposed solver. In the five applications, task sequencing was coupled with different types of decisions on how the tasks are executed (e.g., the selection of IK solutions in pick-and-place, or determining line directions in drawing, engraving, and grinding), the problems were defined in different dimensions (e.g., 2D or 3D task space or 6D robot joint configuration space), and involved optimizing different performance measures (e.g., travel times assuming trapezoid speed profiles, travel distances, or cycle time using the pre-computed durations of collision-free trajectories).

The ProSeqqo solver computed high-quality solutions in all these applications. For small-to-medium problem sizes, e.g., with up to 30 parts in pick-and-place, the solver found close-to-optimal solution with computation times allowing online planning. For larger problems, e.g., with thousands of lines in laser engraving, reasonable solutions could still be found in a matter of seconds, but these solutions could be enhanced further by allowing higher computation times.

From the local search algorithms of Google OR-tools, GLS provided a consistently good performance in all applications. Yet, finding the appropriate GTSP representation by pre-computing the shortest paths was crucial in the pick-and-place application, which was the only investigated application with a chain of multiple configurations to visit within each process. Moreover, the proposed MIP approach was necessary for finding feasible initial solutions in all applications involving precedence constraints. It should be noted the construction of a feasible initial solution is indeed a combinatorial problem in case of motion precedences (e.g., in the building blocks application), whereas the solver can be extended in the future with quick initial heuristics for problems with process precedences only.

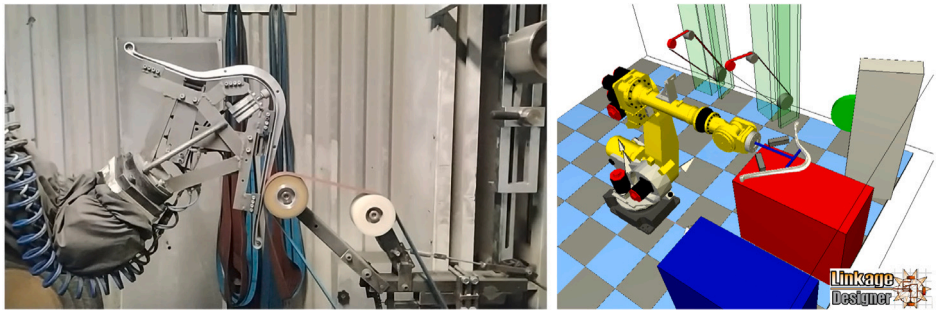


Fig. 14. Robotic grinding and polishing of furniture parts [36].

7. Conclusions

This paper introduced a generic problem definition language and a corresponding solver for task sequencing in industrial robotics. It was shown that the model covers most of the robotic task sequencing problems studied in recent literature, and hence, can alleviate the need for developing custom models and algorithms for such applications. The solver translates the problem description into a GTSP model with precedence constraints, slightly extending classical PCGTSP models, and solves it using a combination of built-in algorithms of Google OR-Tools and custom techniques. The solver has been made available open source.

The applicability and effectiveness of the solver was demonstrated in five rather different robotic applications: the sequencing problems are coupled with different decisions on the directions or the IK solutions applied for executing the task, they can be solved in the task space or in the robot joint space, and they also differ in the problem size and the time frame available for solving the problems.

Future research should focus on stronger support for common sub-problems, such as an initial solution heuristic for GTSP with precedences, or the aggregation of elementary line segments into larger contours before sequencing in cutting, engraving, or drawing applications with thousands of elementary segments. Extension to multi-robot problems by conversion to richer VRP models is also of interest. Finally, the possibilities of publishing the solver in a software-as-a-service model will be investigated.

CRediT authorship contribution statement

László Zahorán: Methodology, Software, Validation, Visualization, Writing – original draft. **András Kovács:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing.

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

The authors thank their colleagues Bence Tipary, Gábor Erdős, János Csempesz and Tamás Cserteg for sharing information about the sample applications. This research was supported by the Ministry for Innovation and Technology, Hungary and the National Research, Development and Innovation Office, Hungary within the framework of the National Lab for Autonomous Systems, Hungary and the ED_18-2-2018-0006 grant on “Research on prime exploitation of the potential provided by the industrial digitalisation”. A. Kovács acknowledges the support of the János Bolyai Research Fellowship, Hungary.

References

[1] G. Laporte, H. Mercure, Y. Nobert, Generalized travelling salesman problem through n sets of nodes: the asymmetrical case, Discrete Appl. Math. 18 (2) (1987) 185–197.
[2] F. Suarez, T. Lembono, Q.C. Pham, RoboTSP – A fast solution to the robotic task sequencing problem, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 1611–1616.
[3] E.M. Arkin, R. Hassin, Approximation algorithms for the geometric covering salesman problem, Discrete Appl. Math. 55 (3) (1994) 197–218.
[4] I. Gentilini, F. Margot, K. Shimada, The travelling salesman problem with neighbourhoods: MINLP solution, Optim. Methods Softw. 28 (2) (2013) 364–378.
[5] L. Gambardella, M. Dorigo, An ant colony system hybridized with a new local search for the sequential ordering problem, INFORMS J. Comput. 12 (3) (2000) 237–255.
[6] M. Khachay, A. Kudriavtsev, A. Petunin, PCGLNS: A heuristic solver for the precedence constrained generalized traveling salesman problem, in: N. Olenev, Y. Evtushenko, M. Khachay, V. Malkova (Eds.), Optimization and Applications (OPTIMA 2020), in: Lecture Notes in Computer Science, vol. 12422, Springer, 2020, pp. 196–208.
[7] R. Salman, F. Ekstedt, P. Damaschke, Branch-and-bound for the precedence constrained generalized traveling salesman problem, Oper. Res. Lett. 48 (2) (2020) 163–166.
[8] B.D. Backer, V. Furnon, P. Shaw, P. Kilby, P. Prosser, Solving vehicle routing problems using constraint programming and metaheuristics, J. Heuristics 6 (4) (2000) 501–523.
[9] S. Alatartsev, S. Stellmacher, F. Ortmeier, Robotic task sequencing problem: A survey, J. Intell. Robot. Syst. 80 (2) (2015) 279–298.
[10] M. Bottin, G. Rosati, G. Boschetti, Working cycle sequence optimization for industrial robots, in: V. Niola, A. Gasparetto (Eds.), Advances in Italian Mechanism Science, Springer, 2021, pp. 228–236.
[11] Y. Edan, T. Flash, U. Peiper, I. Shmulevich, Y. Sarig, Near-minimum-time task planning for fruit-picking robots, IEEE Trans. Robot. Autom. 7 (1) (1991) 48–56.
[12] Y.-L. Lai, P.-C. Shen, C.-C. Liao, T.-L. Luo, Methodology to optimize dead yarn and tufting time for a high performance CNC by heuristic and genetic approach, Robot. Comput.-Integr. Manuf. 56 (2019) 157–177.
[13] S. Pellegrinelli, T. Tolio, Pallet operation sequencing based on network part program logic, Robot. Comput.-Integr. Manuf. 29 (5) (2013) 322–345.
[14] K. Baizid, A. Yousnadj, A. Meddahi, R. Chellali, J. Iqbal, Time scheduling and optimization of industrial robotized tasks based on genetic algorithms, Robot. Comput.-Integr. Manuf. 34 (2015) 140–150.
[15] E. Glorieux, P. Franciosa, D. Ceglarek, Coverage path planning with targetted viewpoint sampling for robotic free-form surface inspection, Robot. Comput.-Integr. Manuf. 61 (2020) 101843.
[16] Y. Li, L. Zeng, K. Tang, C. Xie, Orientation-point relation based inspection path planning method for 5-axis OMI system, Robot. Comput.-Integr. Manuf. 61 (2020) 101827.
[17] T.R. Weller, D.R. Weller, L.C. de Abreu Rodrigues, N. Volpato, A framework for tool-path airtime optimization in material extrusion additive manufacturing, Robot. Comput.-Integr. Manuf. 67 (2021) 101999.
[18] N. Volpato, L.C.G. ao, L.F. Nunes, R.I. Souza, K. Oguido, Combining heuristics for tool-path optimisation in material extrusion additive manufacturing, J. Oper. Res. Soc. 71 (6) (2020) 867–877.
[19] G. Reinhart, U. Munzert, W. Vogl, A programming system for robot-based remote-laser-welding with conventional optics, CIRP Ann. – Manuf. Technol. 57 (1) (2008) 37–40.
[20] S.L. Villumsen, M. Kristiansen, A framework for task sequencing for redundant robotic remote laser processing equipment based on redundancy space sampling, Proc. Manuf. 11 (2017) 1826–1836.
[21] K. Castelino, R. D'Souza, P.K. Wright, Toolpath optimization for minimizing airtime during machining, J. Manuf. Syst. 22 (3) (2003) 173–180.
[22] R. Dewil, P. Vansteenwegen, D. Cattrysse, Construction heuristics for generating tool paths for laser cutters, Int. J. Prod. Res. 52 (20) (2014) 5965–5984.

[23] A. Kovács, Integrated task sequencing and path planning for robotic remote laser welding, Int. J. Prod. Res. 54 (4) (2016) 1210–1224.
[24] S. Alatartsev, M. Augustine, F. Ortmeier, Constricting insertion heuristic for traveling salesman problem with neighborhoods, in: Proc. 23rd Int. Conf. Automated Planning and Scheduling, 2013, pp. 2–10.
[25] E. Kolakowska, S.F. Smith, M. Kristiansen, Constraint optimization model of a scheduling problem for a robotic arm in automatic systems, Robot. Auton. Syst. 62 (2) (2014) 267–280.
[26] A.M. Dalavi, P.J. Pawar, T.P. Singh, Tool path planning of hole-making operations in ejector plate of injection mould using modified shuffled frog leaping algorithm, J. Comput. Des. Eng. 3 (3) (2016) 266–273.
[27] H. Yang, H. Shao, Distortion-oriented welding path optimization based on elastic net method and genetic algorithm, J. Mater Process. Technol. 209 (9) (2009) 4407–4412.
[28] R. Dewil, P. Vansteenwegen, D. Cattrysse, A review of cutting path algorithms for laser cutters, Int. J. Adv. Manuf. Technol. 87 (5) (2016) 1865–1884.
[29] R. Dewil, İ. Küçükoğlu, C. Luteyn, D. Cattrysse, A critical review of multi-hole drilling path optimization, Arch. Comput. Methods Eng. 48 (2) (2019) 449–459.
[30] B. Tipary, A. Kovács, G. Erdős, Planning and optimization of robotic pick-and-place operations in highly constrained industrial environments, Assem. Autom. 41 (5) (2021) 626–639.
[31] L. Perron, V. Furnon, Google OR-tools version 9.0, https://developers.google.com/optimization/.
[32] C. Voudouris, E. Tsang, Guided local search and its application to the traveling salesman problem, European J. Oper. Res. 113 (2) (1999) 469–499.
[33] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[34] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.
[35] L. Zahorán, A. Kovács, Efficient collision detection for path planning for industrial robots, in: Proc. StuCoSReC 2019, 6th Student Computer Science Research Conference, 2019, pp. 19–22.
[36] G. Erdős, I. Paniti, B. Tipary, Transformation of robotic workcells to digital twins, CIRP Ann. – Manuf. Technol. 69 (1) (2020) 149–152.