Robotica (2021), **39**, pp. 2110–2130
doi:10.1017/S0263574721000199

RESEARCH ARTICLE

Simultaneous task placement and sequence optimization in an inspection robotic cell

MohammadHadi FarzanehKaloorazi[1]*, Ilian A. Bonev[1], and Lionel Birglen[2]

[1]École de technologie supérieure, 1100 Notre-Dame St W, Montreal H3C 1K3, Canada. E-mail: ilian.bonev@etsmtl.ca and
[2]Polytechnique Montréal, 2900 Édouard-Montpetit Blvd, Montreal H3T 1J4, Canada. E-mail: lionel.birglen@polymtl.ca.
*Corresponding author. Email: hamidfarzane88@gmail.com.

Received: 26 March 2020; **Revised:** 23 January 2021; **Accepted:** 27 January 2021; **First published online:** 8 April 2021

Keywords: Task placement, Cycle time, Traveling salesman problem, Particle swarm optimization (PSO), Serial robot, Redundancy

Abstract
In this article, we improve the efficiency of a turbine blade inspection robotic workcell. The workcell consists of a stationary camera and a 6-axis serial robot that is holding a blade and presenting different zones of the blade to the camera for inspection. The problem at hand consists of a 6-DOF (degree of freedom) continuous optimization of the camera placement and a discrete combinatorial optimization of the sequence of inspection poses (images). For each image, all robot configurations (up to eight) are taken into consideration. A novel combined approach is introduced, called blind dynamic particle swarm optimization (BD-PSO), to simultaneously obtain the optimal design for both domains. The objective is to minimize the cycle time of the inspection process, while avoiding any collisions. Even though PSO is vastly used in engineering problems, the novelty of our combinatorial optimization method is in its ability to be used efficiently in traveling salesman problems where the distances between the cities are unknown and subject to change. This highly unpredictable environment is the case of the inspection cell where the cycle time between two images will change for different camera placements.

1. Introduction

In today's industrial robotic cells, efficiency is a key factor to mass production. Less process cycle time leads to more production and more business value. Two common automation problems that need optimization are task sequencing and task placement.

Even though many researchers have been interested in optimizing the work flow of a multitask workcell [1, 2, 3, 4], there is still no method that is able to efficiently optimize task sequence and consider all degrees of freedom (DOF) and constraints, for example, workcell layout, collision-free trajectory planning, multiple inverse kinematics solutions (robot configurations).

The goal of this study is to minimize the process time of a turbine blade inspection workcell. The process includes the following tasks: a polished turbine blade is gripped by a serial robot and presented to a stationary camera in different poses. This paper proposes a novel algorithm to find the optimized placement of the camera and the best sequence of the inspection poses (images), simultaneously. The objective is to minimize the inspection time, while avoiding collisions and singularities [5, 6]. This method will combine the continuous camera placement and combinatorial image sequencing problems and solve them together.

Camera placement in a workcell is a redundancy resolution problem [7]. Similar to part or path placement, in camera placement, six degrees of redundancy (DOR) are introduced to the problem. Infinite solutions can be found to a redundancy resolution problem and various approaches are introduced in the literature.

For example, in ref. [8], optimum placement of the workpiece to be machined is investigated, considering the forces applied to the robot. In ref. [9], the actuator torques, energy consumption, and shaking force are minimized for a multiobjective path placement optimization involving parallel mechanisms. Finally, simultaneous placement optimization and trajectory planning were performed in ref. [10]. In that publication, a serial robot and a rotary stage (1 DOF redundancy) work together in an automated fiber placement cell. A meta-heuristic method was proposed based on particle swarm optimization (PSO) to find the best placement of the rotary stage, while optimizing the angle value of the rotary stage.

Sequence optimization is also a common problem in robotics and has been addressed in the literature. In refs. [11, 12], using genetic algorithm (GA), the sequence of visiting tasks and robot configurations are optimized. However, the task placement was not considered. In ref. [13], a GA-based method is developed to optimize the task point visit order, considering the robot configuration and the workcell layout. However in ref. [13], the robot placement is limited to position and to only four points. In ref. [14], a similar problem is address but this time by adding discrete limited orientation placement options for the robot. The issue with GA is that it is naturally limited to discrete search spaces. In these studies, a high number of population size and iterations were required. For example in ref. [13], the generation number is 500,000, and the iteration number is 222,857. Special modifications are needed in order to optimize a continuous problem by GA. Finally, in ref. [15], the task sequence and feasible robot configurations are obtained by using sequential quadratic programming, but not the task placement.

In ref. [16], the traveling salesman problem (TSP) is adapted to robotics. Travel time between any two poses is considered to be affected by the selection of the robot configuration. Again, the method is based on GA, and an encoding technique is introduced to take into account the multiple solutions of the inverse kinematic problem (IKP). In ref. [17], two decisions that need to be made simultaneously are investigated: finding the robot move cycle and finding the input part sequence. The objective is to maximize the throughput rate of the cell. The problem is formulated as a special kind of TSP. An O(n4) time algorithm that solves this problem optimally has been provided in ref. [18], and the authors extend that work and provide an algorithm of complexity O(n log n). Finally, in ref. [2], the goal is to find the appropriate order of welding tasks where the robot path is considered during sequencing. For modeling the problem, an extension of the TSP with neighborhoods, denoted as TSP-ND, is introduced. A GRASP meta-heuristic algorithm is proposed for solving it.

In a recent article [19], sequence planning for a harvesting robot is addressed, where sensing tasks are prioritized to minimize the cycle time and a discrete TSP solves the problem. In ref. [20], TSP with circular neighborhood is used to solve path planning problem for the laser welding robot and the problem is formulated as a nonlinear model. In ref. [21], three alternative initialization methods are proposed for GA-based TSP. In ref. [22], multirobot task allocation is transformed into multiple TSP and solved using Grey Wolf optimizer. Moreover, an adaptive adjustment strategy is proposed to balance exploration and exploitation. In ref. [23], classic TSP is solved with additional constraint for industrial robots. In ref. [24], a hybrid GA approached applied for shelf picking task of a 6-DOF robot.

In this paper, a novel method based on PSO is proposed to simultaneously optimize the continuous problem of camera placement and the combinatorial problem of image sequence optimization. PSO is a meta-heuristic optimization method, proven to obtain the optimal result in various engineering problems, without requiring the function to be differentiable [25]. In PSO, a population of particles search for the optimum value of the function and inform each other about the best results they have obtained. For an individual particle, personal and global (swarm) best memory, as well as the inertia from the previous movement, defines the current velocity of the particle.

In the proposed method, each particle conveys six real values (camera placement in 6-DOF space) and a set of integers (image sequence). Therefore, the problem in hand is solved for the location of the camera while the best image sequence is obtained. Even though PSO is widely used by researchers to solve continuous problems, it is less utilized for combinatorial cases. The proposed method is specifically new in term of combinatorial engineering problems.

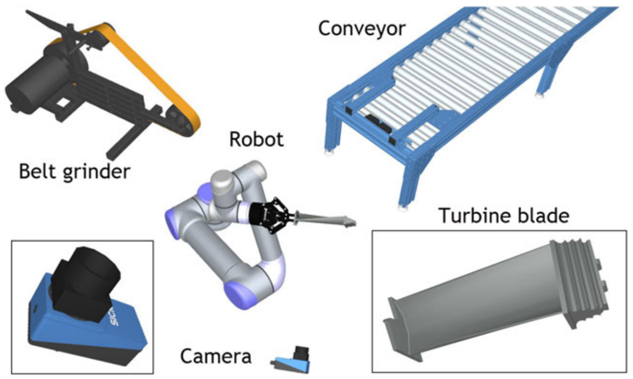

Figure 1. *The robotic workcell to polish and inspect turbine blades. The robot grabs the blade from the conveyor, brings it to belt grinder, then brings it to the camera.*

The main contribution of this paper is the proposal of a comprehensive method to simultaneously solve sequence optimization in discrete space, namely TSP, and task placement optimization in continuous space. The proposed solution is agnostic to having prior knowledge of the map. Moreover, it performs optimization in a dynamic environment where the cost of a given sequence is not fixed and depends on location in the continuous space. To the best knowledge of the authors, this approach is new and not addressed in the literature. The proposed method is not only applicable to robotics problems but also in other industries.

The remainder of this paper is as follows. First, the problem is explained in detail and the challenges are investigated. Then, the proposed methodology is discussed. Pseudo-code for the proposed algorithm is represented and explained. After that, two case studies are presented, which examine the efficiency of the algorithm. One is a pure combinatorial problem and the other is on the turbine blade inspection problem at hand. At the end, the results are discussed.

2. Problem description

In an automated robotic workcell and specifically in an inspection application, the placement of the elements involved is highly important. Imagine a turbine blade finishing setup in which the objective of the robot is to grab a machined blade, surface finish it on a grinding belt and then inspect it (Fig. 1). Each working cycle of the robot consists of two phases. In the first phase, the robot takes the blade from a conveyor and brings it to the grinding belt. In the second phase, the polished blade is to be inspected by a stationary camera. The inspection process consists of taking multiple images of the polished blade. In this section, the problem is described and the proposed solutions are postponed to the next section.

The elements of the workcell, that is, the robot, the conveyor, the polishing machine, and the camera, are to be placed with respect to the world frame. Since the polishing machine, as well as the robot, is typically bulky and fixed to the workcell, they cannot be easily moved. The only device that is relatively easy to displace is the camera. Therefore, a proper placement of the camera is needed to be able to take all the required images of the blade in a time-efficient manner. These images actually correspond to a set of predefined poses of the blade with respect to the camera. All image poses are fixed with respect to the camera frame.

Definition 1. A feasible placement of the camera is a placement in which all the required images can be taken.

If for one or more required images, the inverse kinematics of the robot has no solution, the corresponding placement of the camera is deemed unfeasible.

2.1. Optimization criteria

There may exist more than one solution for the camera placement. However, it is typically very hard to find any feasible solution for this camera placement problem, as solutions can be found only in a small region of the robot workspace. In this paper, we try to find the best solution among all the feasible camera placements. To find the best placement, various optimization objectives can be considered, such as energy consumption [26, 27], cycle time [28, 29], minimum traveled distance [30], etc. Here, the following optimization objectives are to be achieved by the proposed algorithm: find the best placement of the camera in the workcell for which (1) the cycle time is minimized, (2) there exist no collisions, and (3) all inspection poses are feasible. In what follows, these criteria are explored.

2.1.1. Collision avoidance

The collision-free placement of the camera is essential. Assuming that the placement of the robot and the belt grinder is already free of collisions, the camera needs to be located in such a way that it does not collide with neither the robot, nor the conveyor, nor the belt grinder. Furthermore, the camera has to be out of the robot path followed in the first phase (i.e., from conveyor to grinder).

Collision detection methods have been investigated for decades, and several efficient methods are being used extensively [31, 32]. Among them, 3D collision detection is very popular in robotics applications. They can be categorized into four groups: space-time volume intersection, swept volume interference, multiple interference detection, and trajectory parameterization [33]. In this paper, an open source collision detection for both still and moving CAD models is used, called Bullet. The results are simulated in RoboDK, and collision detection is double checked.

2.1.2. Cycle time

Since the placement of the camera does not affect the first phase of the automated process, only the time for inspecting the blade is considered and is simply referred to as the cycle time throughout this paper.

The exact cycle time of the robot depends on numerous factors. The actual cycle time of the robot can be quite difficult to obtain as it depends on various parameters set by the user (e.g., maximum acceleration) and on the robot controller. In terms of an iterative optimization, it is not feasible to run the real robot to get the actual cycle time (or even to run the robot manufacturer's simulation software). Therefore, we need to find the best approximation for calculating the cycle time.

2.2. Sequence of images

For any camera placement, the order of the images to be taken has a direct impact on cycle time. To find the optimal sequence, the well-known TSP has to be solved [34]. In TSP, the objective is to find the shortest path to visit each of several cities. In our inspection problem, the robot needs to move between each two inspection poses.

In a classical TSP, the distance between each pair of given cities is known in advance. Usually a $C \times C$ chart with $(C^2 - C)/2$ unique entries for the distance between each pair is given, where C is the number of cities. Moreover, the distance between two cities is constant and is not subject to change. There are three major differences between the classical TSP and the problem at hand. This fact demands certain strategy changes.

First, it is not guaranteed that, if the best sequence is obtained for a certain camera placement, this same sequence will serve the best result for another placement. Second, unlike the classical TSP, the distance chart is not given in advance. This chart needs to be calculated for each given placement of the camera. It would be extremely computationally expensive to obtain all the distances and then solve the TSP for each camera placement in an iterative solution. Finally, for each inspection pose, there are up to eight possible robot configurations to be considered (i.e., the salesman needs to visit only one of several cities in each area).

2.3. Degrees of placement

The number of independent parameters describing the placement of the camera is known to be the degree of placement (DOP), also known as degree of redundancy. In general, the camera placement has 6 DOP, that is, three translational and three rotational DOP along and around x-, y-, and z-axes, respectively. Depending on the conditions of the workcell and the vision system, the DOP can be reduced. For instance, rotation around the z-axis of the camera is equivalent to rotating a taken image in the post-processing, if the lighting conditions are uniform. Moreover, in some cases due to the symmetry of the robot's workspace, some of the DOP fall into a dependency group. For example, since the first joint of the robot is generally revolute and its axis parallel to the robot's base z-axis, the following three parameters are dependent: displacement along x- and y- axes and rotation around the z-axis. Only two of them can independently affect the cycle time. For a detailed study on parameter dependency, refer to ref. [7].

3. Methodology

The problem of inspection workcell optimization can be divided into two subproblems: (1) finding the best sequence of images and (2) finding the best placement for the camera. On one hand, the first subproblem is a combinatorial problem [34], in which the objective is to find the optimal ordered set, among a finite number of combinations, which minimizes the cycle time. On the other hand, the second subproblem is a continuous optimization problem [35], in which the objective is to find the best location of the camera in a 6D continuous space, among infinite possibilities. Even though the objectives of both subproblems are the same, they cannot be easily combined because of their different search spaces. This paper is proposing a novel approach based on a modified PSO to solve the TSP, thus solving the two subproblems simultaneously.

In what follows, first the generic cost function, constraints, and parameters discussed in Section 2.1 are brought back to perspective, specifically structured, and represented for the problem at hand. Then, the proposed modified methodology using PSO is described to combine the two subproblems. Finally, the pseudo-code of the proposed algorithm is represented.

3.1. Practical problem structure

In this section, the optimization objective (cycle time) and the optimization constraint (collision avoidance) are structured for the camera placement subproblem. Moreover, the optimization parameters for the combinatorial subproblem (sequence of images) and parameters for the continuous subproblem (camera placement) are investigated.

3.1.1. Objective function (cycle time)

As mentioned earlier, for an efficient iterative optimization one needs to calculate an estimate for the cycle time. In this paper, the Weighted Joint Travel Time (WJTT) estimation will be used [8]. The weight factor for each joint depends mostly on its maximum velocity, especially when relatively large joint motions are involved and maximum joint accelerations are used. Ideally, however, the weight factor must be adjusted experimentally.

In today's commercially available industrial robots, the (generally) fastest and simplest way to move the robot end effector from one pose to another is to use a joint-mode motion command. In joint mode, all joints move simultaneously, following linear trajectories in the joint space. Thus, in general, only one of the joints will be moving at maximal speed, and its travel time will determine the cycle time for the pose-to-pose motion. This joint is not necessarily the slowest joint – it is the one for which the product of maximum joint speed and necessary joint displacement is largest. Thus, for a total of C images to be taken, the complete cycle time that must be optimized is the sum of the $C-1$ pose-to-pose motions.

There are $C!$ sequences of images. In practice, a turbine blade inspection may require dozens of images, which means that there are millions of different sequences. Furthermore, for each image (inspection pose), there are up to eight different joint sets (robot configurations). This means that there are $8^C C!$ different sequences to verify.

3.1.2. *Degree of placement*

As mentioned in Section 2.3, although theoretically, the DOP in a camera placement problem can be less than 6, due to practical limitations (e.g., unsymmetrical workcell, uneven lighting conditions), all the 6 DOP are taken into account.

3.2. *Combining combinatorial and continuous optimization*

In this section, the PSO is first briefly addressed and then the novelty of this work to combine the combinatorial and the continuous optimization for an inspection workcell is represented.

3.2.1. *Particle swarm optimization*

PSO is a population-based stochastic method, first introduced in ref. [36]. The classical PSO algorithm is highly capable of solving continuous problems. It overperforms the other numerical optimizers in many engineering problems [37]. In PSO, a population of particles evaluate the search space and share their results among each other. In each iteration, considering the best personal memory and the best group memory, each particle decides on a new direction to search. An optimized result is obtained after a stopping criteria is reached. Similar to other stochastic optimizers, a global optimum is not guaranteed, but is very likely to reach. In what follows, the formulation of PSO is summarized. For more details refer to ref. [10].

The PSO procedure starts with an initial population (swarm) of particles, typically randomly distributed in the search space. Each particle consists of a value set called location, that is, optimization parameters, having s dimensions. Moreover, each particle stores a value for personal and a value for global best memory, along with the location of these memories. Each particle evaluates the fitness function with respect to its own location and saves it as its best personal memory. After memorizing its best personal location, each particle checks whether its own personal best is better than the current global best, and if so, global memory is updated and shared with other members of the population. For the next iteration, a velocity vector is calculated for each particle. The particle's location is changed due to the velocity and, if necessary, its personal or global best memory is updated. The above can be summarized as

$$\mathbf{v}_{i,s}^{\text{new}} \leftarrow \phi_\omega \mathbf{v}_{i,s}^{\text{old}} + \phi_p r_p (\mathbf{p}_{i,s} - \mathbf{x}_{i,s}) + \phi_g r_g (\mathbf{g}_s - \mathbf{x}_{i,s}), \tag{1}$$

where velocity $\mathbf{v}_{i,s}^{\text{new}}$ is the location update for the next iteration, velocity $\mathbf{v}_{i,s}^{\text{old}}$ is the current velocity of the ith particle for the sth dimension, working as a movement inertia ($\mathbf{v}_{i,s}$ is zero for the very first iteration), $\mathbf{x}_{i,s}$ is the current location of ith particle, $\mathbf{p}_{i,s}$ is the best location personal memory of the ith particle, and $\mathbf{g}_s$ is the best location global memory of the whole swarm. Tunable parameters ϕ_ω, ϕ_p, and ϕ_g are the learning coefficients from previous velocity, personal memory, and global memory, respectively. The tunable parameters have a value between $\{0, 1\}$ and must be chosen carefully in order to achieve reliable results. Furthermore, uniform random mutation factor r_p for the personal and r_g for the global learning ($r_p, r_g = \{0, 1\}$) improves the search ability of the swarm.

3.2.2. *Modifications to fit TSP*

As it has been discussed, the continuous optimization, that is, the placement of the camera, and the combinatorial optimization, that is, the sequence of images, need to be simultaneously resolved. Therefore, it is necessary to modify the classical PSO in order to first, be able to solve a combinatorial problem, that is, TSP, and then to do it along side with the continuous optimization. In this paper, a basic combinatorial

PSO is used, ref. [38], and some features are added to improve the performance of the algorithm. The proposed combinatorial algorithm is then tested on a few TSP benchmarks to verify its ability.

The algorithm is described by the means of TSP. Assume C cities to be traveled by a salesman. The latter has to pass each city once and only once. The route is an open chain, not a loop. The start and end city do not matter. The goal is to obtain an ordered combination of the cities that minimizes the traveled distance. Using PSO, each particle is a combination. It keeps track of its own best and global memory. The procedure starts with a randomly generated population of particles, that is, random combinations. Each particle saves its current combination as the best personal memory and also replaces the global best memory, if needed. In the next iteration, a series of *swaps* alters the current combination of each particle. A swap is done only between two cities in the sequence.

To choose which cities to swap, similar to Eq. (1), a velocity update type routine, based on learning coefficients, is executed. Learning coefficients ψ_p, ψ_g and ψ_m (personal, global, and mutation, respectively) are real numbers between $\{0, 1\}$. Five factors are related to the new velocity: personal and global best combination memories, personal and global previous velocity inertia and mutation. After a few iterations, the optimized combination is obtained. The Number of Function Calls (NFC) depends on the population size and the number of iterations. More iterations and particles are needed for larger problems. The pseudo-code of combinatorial PSO combined with continuous is represented in Algorithm 1.

Algorithm 1: The pseudo-code of the optimization algorithm, based on BD-PSO, to optimize the cost function, fnc.

```
1  Input: fnc, $S$, $C$, $\mathbf{V}_{\text{lower,upper}}$, $it_{\max}$, $n_{\text{pop}}$, $\phi_\omega$, $\phi_p$, $\phi_g$, $\psi_\omega$, $\psi_p$, $\psi_g$, $\psi_m$.
2  Output: $\mathbf{p}^{\text{best}}_{\text{cam}}$, $\texttt{seq}^{\text{best}}$.
   ———— Initialization ————
3  for $i$ from 1 to $n_{\text{pop}}$ do
4      $\mathfrak{P}_i^{\mathbf{P}} = \text{UniDist}(S, \mathbf{V}_{\text{lower,upper}})$
5      $\mathfrak{V}_i^{\mathbf{P}} = \mathbf{0}_S$
6      $\mathfrak{P}_i^{\texttt{seq}} = \text{RndInt}(C)$
7      $\mathfrak{V}_i^{\texttt{seq}} = \mathfrak{P}_i^{\texttt{seq}}$
8      $\mathfrak{P}_i best = \text{fnc}(\mathfrak{P}_i^{\mathbf{P}}, \mathfrak{P}_i^{\texttt{seq}})$
9  $\mathfrak{P}_g best = \min(\mathfrak{P}_{i\to n} best)$
   ———— Iterations ————
10 while $it_{max}$ do
11     for $i$ from 1 to $n_{\text{pop}}$ do
12         for $s$ from 1 to $S$ do
13             $\mathfrak{P}_i^{\mathbf{P}} = \text{Eq. (1)}$
14         $\mathfrak{P}_i^{\texttt{seq}} \Leftarrow \text{Alg. 2}$
15         if $fnc(\mathfrak{P}_i^{\mathbf{P}}, \mathfrak{P}_i^{\texttt{seq}}) < \mathfrak{P}_i best$ then
16             $\mathfrak{P}_i best = \text{fnc}(\mathfrak{P}_i^{\mathbf{P}}, \mathfrak{P}_i^{\texttt{seq}})$
17             if $\mathfrak{P}_i best < \mathfrak{P}_g best$ then
18                 $\mathfrak{P}_g best = \mathfrak{P}_i best$
19 $\mathbf{p}^{\text{best}}_{\text{cam}} \leftarrow \mathfrak{P}_g^{\mathbf{p}} best$
20 $\texttt{seq}^{\text{best}} \leftarrow \mathfrak{P}_g^{\texttt{seq}} best$
```

The proposed combinatorial PSO is tested on few benchmarks, and the results for ATT48 problem are represented. ATT48 is a set of 48 points in 2D space, corresponding to the capitals of the states in the United States. The optimum answer to this problem has been represented in the literature long ago. Yet, an efficient algorithm to find the best sequence is still a challenge today. The proven shortest path is 10,628 and is nearly achieved by Hybrid Discrete PSO (HDPSO) [39]. However, this result has been achieved by an excessive NFC. Typically, a value around 34,000 can be obtained by a regular cost efficient method [40]. Note that the lengths are normalized values.

In the literature to solve the TSP, the population is initialized starting from one point and connecting each city to its nearest neighbor [41]. Furthermore, during the optimization, swaps are done between the cities based on fixed and given distances [38]. This approach requires a chart of distances, known and invariable from the beginning. The problem at hand is different than the classical TSP. For the camera placement problem, the cycle time from one image to another is subject to change and depends on the camera's pose, that is, the distances between the cities can change. Therefore, those methods that require a chart of distances given at the beginning, do not work for this problem. Moreover, in our problem, the start and end city are not fixed and the salesman does not travel in a loop; therefore, we have $C!$ possibilities. Due to the aforementioned conditions, this combinatorial image sequence optimization needs more NFC compared to the classical TSP.

The proposed algorithm is called Blind Dynamic map TSP PSO (BD-PSO). *Blind* because it is independent of cities map and calculates the distances as it goes city by city. *Dynamic map* because it can handle changeable distances, that is, the map dynamically changes. The algorithm is able to find the best camera placement and the smallest cycle time, simultaneously. Put in TSP notion, imagine that there is a book of different maps for a territory. BD-PSO finds that map yields the shortest tour and provides the tour itself. Details about embedded BD-PSO into continuous PSO are represented in the following section.

4. Algorithm

In this section, the proposed algorithm is represented in a pseudo-code format. In Algorithm 1, the main routine of the algorithm is shown. First, the inputs and outputs of the algorithm are declared. The algorithm consists of two sections: initialization of the population and iterations.

4.1. Main routine

As it can be observed in the pseudo-code, the required inputs of the optimization procedure are as follows. The cost function *fnc*, that is, the fitness function, determines the fitness value of each particle $\mathfrak{P}$. Each particle is twofold. Continuous parameters, that is, the camera placement variables shown as $\mathfrak{P}^{\mathbf{p}}$, and combinatorial parameters, that is, the sequence of the images shown as $\mathfrak{P}^{\mathtt{seq}}$. The number of continuous optimization parameters S (for space) is the dimensions of the search space. The number of combinatorial optimization parameters C (for city) is the number of images. Tensor $\mathbf{V}_{\text{lower,upper}}$ is of size $2 \times S$ and consists of two vectors: lower and upper bound of the initial search field in the continuous space. Maximum number of iterations $it_{\max}$ and population size n_{pop} are standard PSO parameters.

Note that in this paper the stopping criteria are set to be the maximum number of iterations. Therefore, the total NFC is $n_{\text{pop}} \times (it_{\max} + 1)$. The rest are the tuning parameters of the optimization algorithm. Coefficients ϕ and ψ are associated with continuous and combinatorial, respectively. Coefficients ϕ_ω, ϕ_p, and ϕ_g are the inertia, personal, and global learning coefficients of camera placement, respectively. Coefficients ψ_ω, ψ_p, ψ_g, and ψ_m are the inertia, personal, and global learning coefficients and mutation factor of the image sequence optimization, respectively.

The outputs of the algorithm are the best pose of the camera $\mathbf{p}_{\text{cam}}^{\text{best}}$ and best sequence of images $\mathtt{seq}^{\text{best}}$.

4.1.1. Initialization

The initialization section consists of a loop assigning the initial value to all particles. Index i shows the individual particle. In line 4, a vector of size S with uniform distribution within the lower and upper boundary limits is assigned to continuous particles, $\mathfrak{P}^{\mathbf{p}}$. In our inspection workcell, $S = 6$ and each continuous particle consists of six values, namely the displacement along and orientation around x-, y-, and z-axes. The initial continuous velocity of each particle, $\mathfrak{V}^{\mathbf{p}}$, is a zero vector of size S.

The combinatorial part of the particle is then initialized. A string of numbers from 1 to C is randomly shuffled and assigned to $\mathfrak{P}^{\mathsf{seq}}$. Then, the velocity of combinatorial part $\mathfrak{V}^{\mathsf{seq}}$ is initialized to be the same as the particle itself, because this implies zero velocity. Next, once the pose and combination of the ith particle is set, they are passed into the cost function *fnc* and the fitness of the particle is returned.

The *fnc* is represented in Algorithm 3 and will be described in the upcoming paragraphs. The output of *fnc* is the cycle time of the particle. For the initialization, the current pose and combination is stored as the particle's personal best memory $\mathfrak{P}_i best$. Note that $\mathfrak{P}_i best$ contains both continuous and combinatorial parameters.

Finally, after all particles are assigned with the initial values and their best memory is captured, the best global memory is extracted from comparing all the members of population and the one with the minimum cycle time is stored in $\mathfrak{P}_g best$.

4.1.2. Iterations

In the iterative section, each particle decides about its search direction based on personal and global memories and explores new possibilities. From lines 10–18, the algorithm starts the iterations and stops when $it_{\max}$ number of iterations are done. In lines 11–18, for each particle i, the pose and combination is updated and the fitness value is obtained. In lines 12 and 13, the pose of the camera for each dimension S is updated based on Eq. (1). In line 14, the combination of the sequential part of particle is updated based on Algorithm 2. In lines 15–18, the fitness is evaluated with updated particles values. In lines 17 and 18, if for a particle the new pose and combination yield to a better cycle time, the personal best memory is updated with the new values. In lines 19 and 20, if the current particle's personal updated memory is better than that of its global, the global best memory is replaced with the current particle's attributes. After $it_{\max}$ number of iterations, the existing global best pose and combination ($\mathfrak{P}_g^{\mathsf{p}} best$, $\mathfrak{P}_g^{\mathsf{seq}} best$) are represented as the optimized camera placement and image sequence.

4.2. BD-PSO

In Algorithm 2, the pseudo-code of the BD-PSO is represented. In line 1, three counters are created. Counters l_p, l_g, and l_v keep track of how many times a particle has learned from its personal best combination, swarm global best combination, and previous velocity combination, respectively. All these learning counters are zero at the beginning. The BD-PSO consists of four major sections.

4.2.1. Inertia

In lines 2–5, the first section which is the effect of inertia is represented. In these lines, it is checked that if a city c in the particle sequence $\mathfrak{P}_i^{\mathsf{seq}}$ is different than the city in the velocity sequence $\mathfrak{V}_i^{\mathsf{seq}}$, possibly city c in the particle will be swapped with the one in the velocity. The velocity sequence is generated from last iterations and contains those possible swaps that were not done in the previous sequence.

In line 3, if the aforementioned condition is met, there is a chance that the swap is done, because it is done only if a randomly generated real number between 1 and 0 (rnd(0, 1)) is smaller than the inertia coefficient. This random chance swap prevents the algorithm from premature convergence. This will be observed in other learning sections as well. Without this likelihood implementation, all the particles would be the same as the global best after the first iteration. This is not desirable and we want the particles to explore new possibilities.

In line 4, the swap operation is applied. See Eq. (2) as an example for a swap in particle's combination. Assume that in this example $c = 2$ (i.e., the swap is being checked for the second city) and rnd(0, 1) $< \psi_\omega$ is true.

$$\begin{aligned} \text{Velocity combination:}\ \ \mathfrak{V}_i^{\mathsf{seq}} &= \{5,\ 3,\ 6,\ 4,\ 1,\ 2\} \\ \text{Particle combination before swap:}\ \ \mathfrak{P}_i^{\mathsf{seq}} &= \{5,\ 2,\ 4,\ 3,\ 6,\ 1\} \\ \text{Particle combination after swap:}\ \ \mathfrak{P}_i^{\mathsf{seq}} &= \{5,\ 3,\ 4,\ 2,\ 6,\ 1\}. \end{aligned} \tag{2}$$

Algorithm 2: The pseudo-code of the combinatorial BD-PSO to update the particles combination. rnd(0, 1) is a random real number generated independently each time.

```
1  $l_p = 0, l_g = 0, l_v = 0$
   ——— Inertia from previous iteration ———
2  for $c$ from 1 to $C$ do
3      if $\mathfrak{P}^{seq}_{i,c} \neq \mathfrak{V}^{seq}_{i,c}$ and $rnd(0, 1) < \psi_\omega$ then
4          in $\mathfrak{P}^{seq}_{i,c}$ swap($\mathfrak{P}^{seq}_{i,c}$, $\mathfrak{V}^{seq}_{i,c}$)
5          $l_V ++$
6  ——— Personal learn ———
7  for $c$ from 1 to $C$ do
8      if $\mathfrak{P}^{seq}_{i,c} \neq \mathfrak{P}^{seq}_{p,c}best$ and $rnd(0, 1) < \psi_p$ then
9          in $\mathfrak{P}^{seq}_{i,c}$ swap($\mathfrak{P}^{seq}_{i,c}$, $\mathfrak{P}^{seq}_{p,c}best$)
10         $l_P ++$
       —— Prepare velocity for next iteration
11     else if $\mathfrak{P}^{seq}_{i,c} \neq \mathfrak{P}^{seq}_{p,c}best$ then
12         in $\mathfrak{V}^{seq}_{i,c}$ swap($\mathfrak{V}^{seq}_{i,c}$, $\mathfrak{P}^{seq}_{p,c}best$)
13     end
   ——— Global learn ———
14 for $c$ from 1 to $C$ do
15     if $\mathfrak{P}^{seq}_{i,c} \neq \mathfrak{P}^{seq}_{g,c}best$ and $rnd(0, 1) < \psi_g$ then
16         in $\mathfrak{P}^{seq}_{i,c}$ swap($\mathfrak{P}^{seq}_{i,c}$, $\mathfrak{P}^{seq}_{g,c}best$)
17         $l_g ++$
       —— Prepare velocity for next iteration
18     else if $\mathfrak{P}^{seq}_{i,c} \neq \mathfrak{P}^{seq}_{g,c}best$ then
19         in $\mathfrak{V}^{seq}_{i,c}$ swap($\mathfrak{V}^{seq}_{i,c}$, $\mathfrak{P}^{seq}_{g,c}best$)
20     end
   ——— Mutation ———
21 if $l_v + l_p + l_g = 0$ then
22     for $\mu$ from 1 to ceil($C/5$) do
23         in $\mathfrak{P}^{seq}_{i,c}$ swap two random cities
```

As it can be observed, the second city in velocity is 3 and the second city in particle is 2. Therefore, the second city in particle is swapped to match the velocity's second city (2 and 3 are swapped in particle).

This concept is applied to every city in particle and if necessary ($\text{rnd}(0, 1) < \psi_\omega$), they will be swapped. Note that the random number is independently generated each time it is called. In line 5, the velocity learning counter is increased by 1.

4.2.2. *Personal learning*

In the second section from lines 6 to 12, the particle learns from its own personal best. In lines 7–9, as in the previous section, the particle updates its sequence if it differs from its personal best memory, and if the randomly generated number is smaller than the personal learning coefficient. In line 9, the personal learning counter increases. In lines 10 and 11, if city c in the particle is different than its counterpart in the particle's personal best, but did not have the chance to swap (the random number was larger than ψ_p), the swap is stored in the velocity. This velocity will be used in the next iteration. In this vein, in the next iteration, the particle still has a chance to take advantage of the unused swap. Therefore, in line 11,

Algorithm 3: The pseudo-code of the Cost function, fnc, to calculate the cycle time of the inspection phase.

```
1  Input: $\mathfrak{P}_i^{\mathtt{p}}$ as the pose of the camera, $\mathfrak{P}_i^{\mathtt{seq}}$ as the sequence of the images
2  Output: Cycle time
   ———— Set camera and images pose ————
3  $\mathbf{H}_{\text{base}}^{\text{cam}} = \text{pose2tran}(\mathfrak{P}_i^{\mathtt{p}})$
4  for c from 1 to C do
5      ${}_c\mathbf{H}_{\text{base}}^{\text{flange}} = \mathbf{H}_{\text{blade}}^{\text{flange}} \mathbf{H}_{\mathtt{img}_c}^{\text{blade}} \mathbf{H}_{\text{cam}}^{\mathtt{img}_c} \mathbf{H}_{\text{base}}^{\text{cam}}$
   ———— Calculations ————
6  try:
7      IKP(${}_1\mathbf{H}_{\text{base}}^{\text{flange}}$)
8      for c in $\mathfrak{P}_i^{\mathtt{seq}}$ do
9          $\mathbf{ct}_c$ = CycleTime(${}_c\mathbf{H}_{\text{base}}^{\text{flange}}, {}_{(c+1)}\mathbf{H}_{\text{base}}^{\text{flange}}$)
10         CollisionCheck(${}_c\mathbf{H}_{\text{base}}^{\text{flange}}, {}_{(c+1)}\mathbf{H}_{\text{base}}^{\text{flange}}$)
11 catch fail:
12     Collision or out of reach
13     return inf
14     break
15 return $\sum$ ct
```

the swap is done for the velocity sequence and cth city in the velocity is updated with the according city of particle's personal best.

4.2.3. Global learning

In the third section of the algorithm from lines 13 to 19, the same concept from the previous section is applied for global learning. The particle has a chance to learn from the best memory of the whole swarm. In line 14, the learning coefficient ψ_g determines the chance and in line 15 swap is done and global learning counter increases in line 16. In line 18, the possible useful swaps are stored in velocity for next iteration.

4.2.4. Mutation

In the final section of this algorithm, from lines 20 to 22, a mutation is applied to the particle. Mutation is applied when a particle has learned nothing from the previous sections, meaning that the particle's sequence is exactly the same as its previous iteration with no change. In this case, random swaps in the particle happens. It improves the efficiency of the algorithm because if a particle is the same as its previous iteration, a call to the fitness function is made with no new results. Therefore, in line 20, the algorithm checks if all the learning counters are still zero, that is, no change from the beginning, some of the cities are chosen randomly and swapped. In line 21, depending on the total number of cities, some of the cities are chosen randomly and swapped. Approximately 20% of the cities are randomly swapped.

4.3. Cost function (fnc)

The fitness function fnc of the optimization algorithm is represented in Algorithm 3. The inputs of fnc, as shown in line 1, are both the continuous and combinatorial part of a particle, that is, the pose of the camera and the sequence of the images, respectively. The output of fnc, that is, the fitness value, is a single number as the cycle time of the blade inspection operation. The function fnc is twofold:

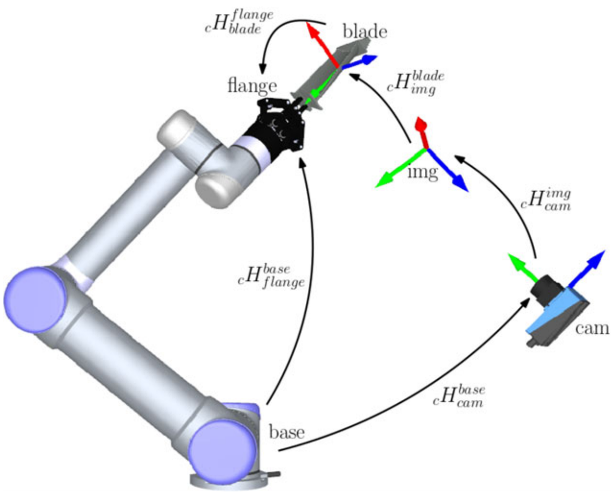

Figure 2. *Transformations between important frames used in Algorithm 3.*

(1) set camera's pose and consequently the inspection poses and (2) perform the cycle time calculation and the collision check.

4.3.1. Set camera and image pose

In the first section, in line 3, using the pose2tran() function, the camera's pose is converted from the 6×1 pose vector of $\mathfrak{P}^{\mathbf{p}}$, that is, translation and orientation, into a homogeneous 4×4 transformation matrix represented with respect to the global base frame. In lines 4 and 5, each image pose, which is given in the camera's frame, is represented as a transformation matrix with respect to the base. In line 5, the transformation matrix from robot's flange to the base frame is obtained. In line 5, the transformation matrices are as follows: camera's location with respect to the base ($\mathbf{H}_{\text{base}}^{\text{cam}}$), cth image with respect to camera ($\mathbf{H}_{\text{cam}}^{\texttt{img}_c}$), reference frame of the blade with respect to cth image ($\mathbf{H}_{\texttt{img}_c}^{\text{blade}}$) , and robot's flange with respect to blade's reference frame ($\mathbf{H}_{\text{blade}}^{\text{flange}}$). Note that $\mathbf{H}_{\text{cam}}^{\texttt{img}_c}$, $\mathbf{H}_{\texttt{img}_c}^{\text{blade}}$ and $\mathbf{H}_{\text{blade}}^{\text{flange}}$ are invariable and only $\mathbf{H}_{\text{base}}^{\text{cam}}$ can change. Consequently, the pose of the flange with respect to the base (${}_c\mathbf{H}_{\text{base}}^{\text{flange}}$) is obtained for cth image, as shown in Fig. 2.

4.3.2. Calculations

At this point all we need is to compute the cycle time and check for collisions. In lines 6–10, the algorithm *tries* to calculate the cycle time and checks for collisions. In line 7, the IKP is solved for the first image in the sequence. Since this is being done in a *try and catch* block, if any of the functions in the *try* block fails, it is *caught* in line 13 and the fitness value of the particle is returned as infinite. In lines 9 and 10, for each image c, the cycle time is calculated (line 9) and stored in $\mathbf{ct_c}$. Furthermore in line 10, all possible collisions are checked.

The try block may fail in two general cases: (1) if there is a collision or (2) if the IKP has no solution. The IKP may fail in two cases: when the Cartesian target $\mathbf{H}_{\text{base}}^{\text{flange}}$ (1) is out of reach for the robot or (2) is at a singularity. In line 15, in case of no failure, the sum of the cycle times for all the images is returned as the fitness value of the particle. Otherwise, inf is returned and basically the current state of the particle is useless and will be ignored by the swarm.

4.4. Cycle time and collision check

In Algorithm 4, two subroutines which were used in the previous section are explained, namely cycle time and collision check. Note that in this algorithm, synchronous point-to-point (PTP) motion for the

Algorithm 4: The pseudo-code of two routines: CycleTime() and CollisionCheck().

```
———————— CycleTime ————————
1  Input: ${}_{c}\mathbf{H}_{\text{base}}^{\text{flange}}$ and ${}_{(c+1)}\mathbf{H}_{\text{base}}^{\text{flange}}$
2  Output: $ct$
3  $\mathbf{q}_{\text{st}} = \text{IKP}({}_{c}\mathbf{H}_{\text{base}}^{\text{flange}})$
4  for $m$ from 1 to 8 do
5      $\mathbf{q}_{\text{end}} = \text{IKP}\Big|_{m}\left({}_{(c+1)}\mathbf{H}_{\text{base}}^{\text{flange}}\right)$
6      $\mathbf{D}_{m} = \boldsymbol{\omega}(\mathbf{q}_{\text{st}} - \mathbf{q}_{\text{end}})$
7  $\mathbf{d} = \max_{\text{Column}} \mathbf{D}$
8  return fail if out of reach
9  return min $\mathbf{d}$ as $ct$
———————— CollisionCheck ————————
10 Input: Cad model of all parts in workcell
11 Output: fail or pass
12 for step from $\mathbf{q}_{\text{st}}$ to $\mathbf{q}_{\text{end}}$ each ( max $(\mathbf{q}_{\text{end}} - \mathbf{q}_{\text{st}})/10$) degree do
13     Check for geometrical collision
14     return fail if collision
```

Table 1. Weights used in WJTT.

J1	J2	J3	J4	J5	J6
0.35	0.25	0.2	0.1	0.05	0.05

robot movements during inspection is assumed, because the path used to go from the start to the end position is irrelevant and only minimizing the time is of interest.

4.4.1. Cycle time

In order to calculate the cycle time, two sets of joint values are needed: start and end one. In line 1, the resultant matrix transformation to get the pose of cth image, ${}_{c}\mathbf{H}_{\text{base}}^{\text{flange}}$ and the image after, ${}_{(c+1)}\mathbf{H}_{\text{base}}^{\text{flange}}$ are given as inputs. In line 2, the output of this subroutine is the cycle time, ct. In line 3, the IKP is solved for the image pose corresponding to the beginning of movement and robot's joint values are stored in $\mathbf{q}_{\text{st}}$. In lines 4–6, the joint value difference between start and end joint set is obtained for all eight robot configurations. In line 5, the IKP is solved for the end joint set at the mth configuration of the robot and the solution is stored in an array called $\mathbf{q}_{\text{end}}$. In line 6, the difference in the joint values is calculated and stored in an array called $\mathbf{D}$. Note that the difference is first multiplied to the weight vector $\boldsymbol{\omega}$ (represented in Table 1) to apply the joint response time. In line 7, the set of smallest joint value difference among all the configurations is stored in $\mathbf{d}$. In case the IKP fails, the loop breaks and failure is returned to superior calculation routine. Otherwise, in line 9, the minimum cycle time value among the six joints is returned as the cycle time.

The working mode of the robot may change from one image to another. The method behind choosing the right working mode (also referred to as configuration) is illustrated in Fig. 3. In this figure, from left to right, the *approach point* is a fixed point in space, also referred to as *home position*. After the robot has polished the blade, it repositions at the approach point. From the approach point, the IKP is solved for all eight possible configurations of the robot in order to reach the pose for image 1. The configuration that requires the smallest cycle time is chosen. In Fig. 3, the smallest cycle time is shown by a dotted circle around the approach point. In the illustrated example, the 3rd working mode results in smallest cycle

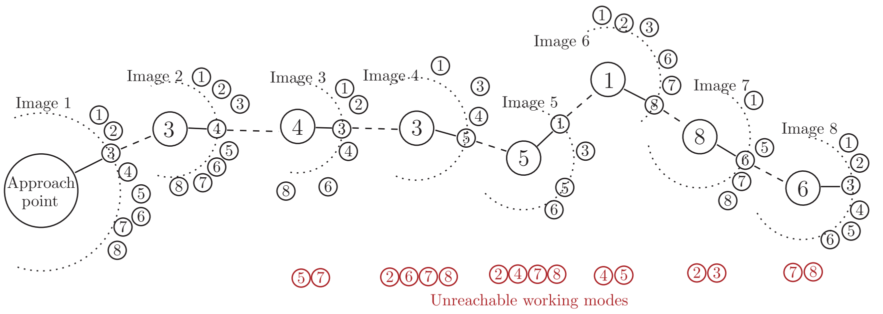

Figure 3. *Working mode selection for each image.*

time. Then, from the 3rd working mode to reach image 2, the 4th working mode results in the smallest cycle time. For some working modes, it might happen that no solution can be found for the robot. The working mode selection process continues for all images. Note that this is a local optimum solution and each working mode is only guaranteed to be the best choice regarding the previous solution.

4.4.2. Collision check

In the second subroutine, all possible collisions in the workcell are checked. Inputs are the CAD models of all the parts in the workcell. The output is only a flag indicating if the collision check is passed. If only one of them fails, the whole cycle fails. This task is performed by an external CAD engine collision check program in line 11. However, it is important to check for the collisions not only at the beginning and ending point but also along the joint mode pose-to-pose movement. Therefore, in line 12, joint values between start $\mathbf{q_{st}}$ and end $\mathbf{q_{end}}$ joint sets are divided into approximately 10 steps. Of course, if the resolution between $\mathbf{q_{st}}$ and $\mathbf{q_{end}}$ is higher, the results are more reliable, but the calculation time will increase as well.

5. Case Study and Discussion

In this section, the proposed algorithm is tested to see whether it is reliable to solve combined (continuous and combinatorial) problems. First, the continuous factor is eliminated in order to verify the algorithm's power only in combinatorial problems, where a map of 13 cities is considered for the TSP and the performance of the algorithm is analyzed. Then, the case study of this paper, that is, turbine blade inspection cell, is resolved using the proposed algorithm.

5.1. Combinatorial test case of 13 cities

The city map of this combinatorial problem test case is shown at the top of Fig. 4. These points (cities) are chosen in a fashion that it is easy to predict the optimal path, which is a circular route. Furthermore, the points are chosen so that they are not symmetrically or homogeneously distributed. It is worth recalling the fact that BD-PSO algorithm is blind in terms of knowing the distances between the cities in advance. Therefore, no decision is made based on proximity of two points or predicting what would be the consequence of certain swaps. The solution purely evolves by manipulating a sequence of numbers, that is, $\texttt{seq} = [1, 2, \ldots, 13]$.

In Fig. 4, the evolution of particles is depicted throughout the iterations. For this optimization, 20 particles are evaluated for 20 iterations. In total, $400 = 20 \times 20$ NFC. Each column in the figure indicates an iteration, therefore, there are 20 particles represented in each column. For each particle, a broken line is drawn with respect to the sequence represented by the particle. For example, on the top of the figure, the drawing of the minimal route is shown. In the first column located on NFC 0, the initial random generation of 20 particles in the swarm is illustrated. The cost function, that is, path length, obtained for the initial generation is between 12.2 and 19 (Fig. 5). In the second column, a sensible drop in

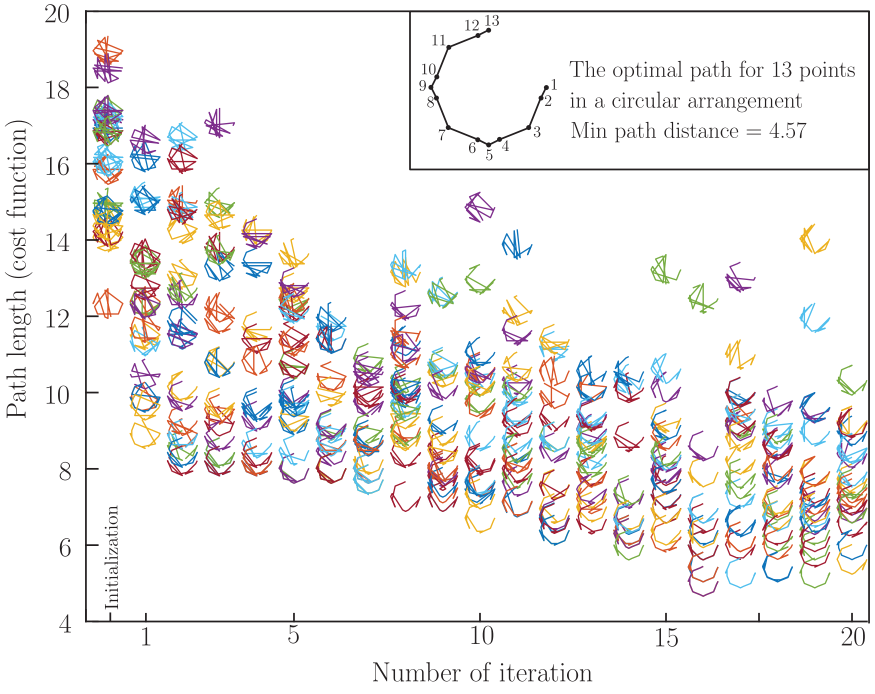

Figure 4. *A test case to measure reliability of the BD-PSO algorithm.*

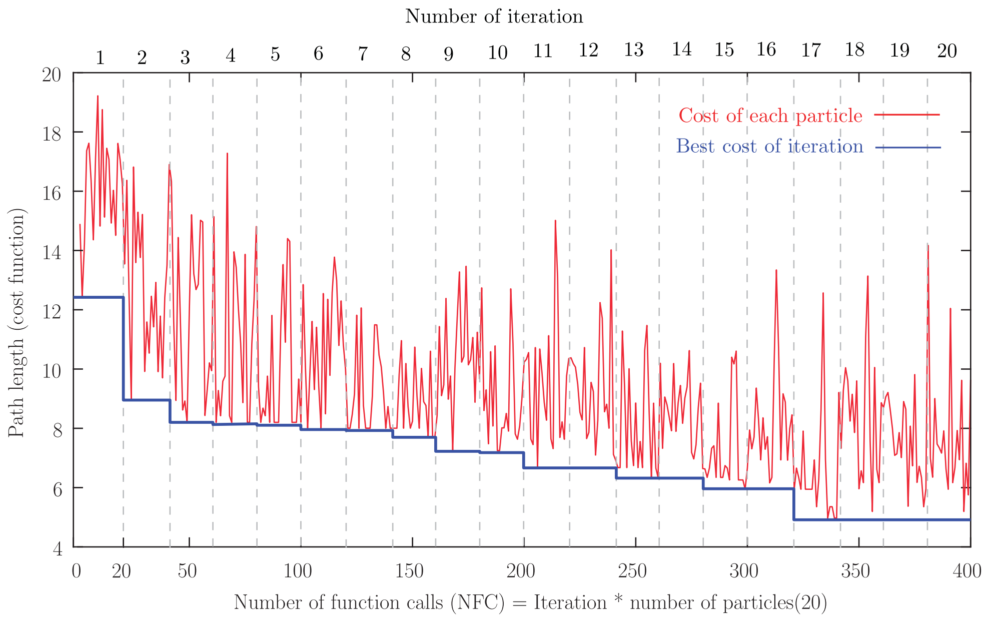

Figure 5. *Convergence plot of 13-point test case using BD-PSO. From NFC 0 to 20 is the initial random generated population. Spikes of longer path length in the plot show the mutation-based attempt to find new solutions.*

route length is observed between 9 and 17. From the first column to the eighth, all the particles are converging to a solution, because there is always either a global, personal, or velocity learned swap done ($l_g + l_p + l_v \neq 0$). From the eighth iteration, a diverging behavior is observed, as seen in Figs. 4 and 5. That is where the mutation has started. This happens when some of the particles are brought

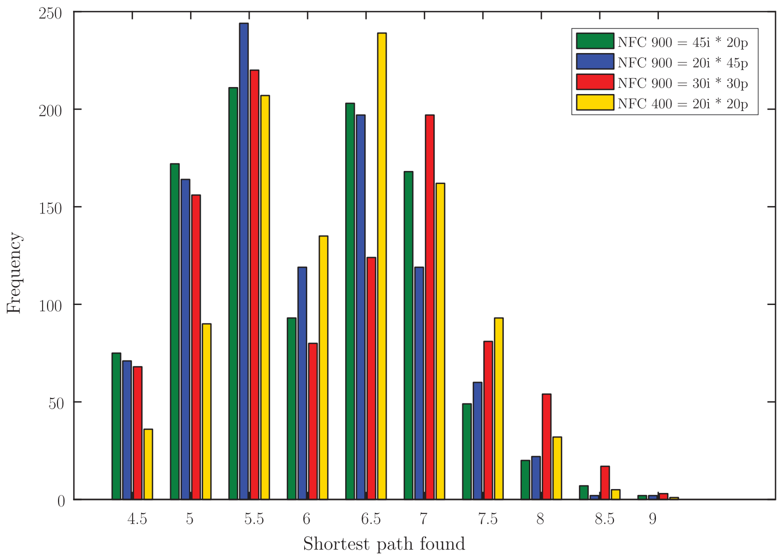

Figure 6. *Histogram of comparison between four combinations of population size and max number of iterations. For each combination, the algorithm is tested 1000 times and distribution of the result shows the repeatability of the algorithm.*

forward to the next iteration without a single change. In order to avoid calling the cost function for a previously calculated sequence, at least one random swap is done in the particle. For this reason in the later iterations, more outliers are emerging to explore new opportunities.

For this optimization the stopping criteria is set to the maximum number of iterations. As it can be observed, after 400 NFC the result of path length is 4.87. Even though the optimal solution is known to be 4.57, the obtained result is quite satisfactory, because without any optimization, $13! = 6, 227, 020, 800$ NFC is needed to find the best solution. Yet after only 400 NFC, a near optimal solution is found. Moreover, this is done in a blind fashion, without knowing the distances between each two points.

In order for readers to implement, reproduce, and compare results, the hyper parameters used in this case study are provided in what follows. Size (S) of the search space is 0, since this example is only testing combinatorial optimization algorithm. The number of combinatorial optimization is 13, as in number of cities. Maximum number of iterations $it_{\max}$ and population size n_{pop} are both 20. Coefficients ϕ_w, ϕ_p, and ϕ_g, associated with continuous inertia, personal, and global learning values are irrelevant in this example, because only the combinatorial algorithm is tested. For combinatorial learning coefficients ψ_w, ψ_p, ψ_g, and ψ_m, associated with inertia, personal, global and mutation values are 0.3, 0.6, 0.3, and 0.08.

Finally, a comparison is done between various combinations of population size and maximum number of iterations in Fig. 6. Furthermore, this study compares 1000 runs of the algorithm to obtain its repeatability. In Fig. 6, four different combinations are compared, where i stands for the maximum number of iterations and p for the population of swarm. For example, NFC 900 = $45i \times 20p$ means 45 iterations are done for a population of 20 particles. Evidently, better results are obtained with more calls to the cost function, that is, NFC 900 gives a better overall result than NFC 400. However at a fixed NFC, it would be interesting to examine whether a larger p is more important than a larger i. As a future work, a full analysis of the method, for example, fine tuning the learning coefficients and NFC combinations, using more TSP benchmarks, is suggested.

5.2. Turbine blade inspection test case

In this section, the proposed algorithm is applied on a simulated turbine blade polish and inspection workcell. As it can be observed in Fig. 1, a 6-DOF serial robot (UR5 from Universal Robots) is located in

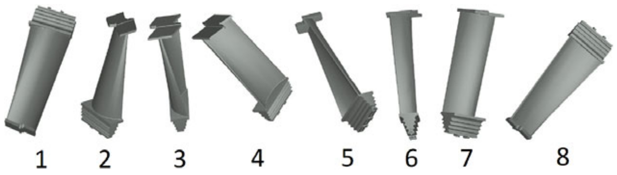

Figure 7. *Eight images needed to be taken from different angles of the turbine blade for the sake of inspection.*

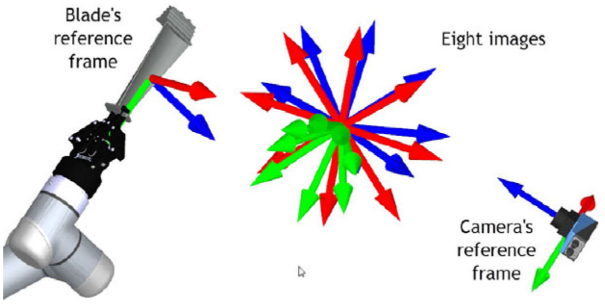

Figure 8. *Eight reference frames corresponding to the eight images.*

the middle of the workcell. It grabs a turbine blade from the conveyor and takes it to belt grinder machine (phase 1). After the polishing operation is done, the robot takes the blade in front of the camera for inspection (phase 2). The location of the camera and the sequence of the inspection poses are optimized with the proposed algorithm.

In this example, eight images from different angles are needed (Fig. 7). Therefore, there are eight inspection poses of the end-effector with respect to the camera. These poses are illustrated in Fig. 8. Eight reference frames correspond to the poses.

In Fig. 9, a comprehensive evolution of particles during the optimization process is depicted. On the top the figure, the shortest path of an 8-point map is shown. This is a representation of the sequence of eight images.

In Fig. 9, the evolution of the particles to optimize the inspection workcell using BD-PSO is depicted. Each octagonal shape drawing represents the sequence of the particle. For example, the upper particle in the first column (with highest cycle time value of around 565) represents the following image sequence: $\{2, 7, 5, 8, 3, 6, 1, 4\}$. One can match the particle to the large octagonal (top right of Fig. 9) to obtain this sequence. Each column represents one iteration. The first column is the initial randomly generated population. The cost function is the cycle time of taking all eight images. The population size is 30 particles. As it can be noticed from the figure, the first column has less than 30 particles. This is due to the fact that for some particles (pose of camera and combination), there is no solution found, either because of collisions or because of workspace limits. From the first column (for which the cost is between 300 and 570) to the second column (cost between 210 and 425) a drastic drop in cycle time is observed. This drop continues up to the fourth iteration, where the mutation feature starts to act on particles. After that, there is always mutations in the particle to avoid repeated calculations.

It is worth comparing the impact of image sequence and camera pose on the cycle time to see which one is more effective to reduce the cycle time. For instance, as it can be seen in Fig. 9 between 25 and 35 NFC, for the same sequence at the bottom, there exist a slight difference in cycle time among the particles. These differences are caused by the camera pose adjustment and ranges in about 207–216 (9 unit difference in cycle time). Comparing the difference in the sequence, which ranges between 160 and 187 (about 17 units in cycle time), it is safe to say that the image sequence optimization is much more important than the camera location adjustment. However, finding a camera location that results in a solution is still very important. Moreover, the camera placement may be of more importance given a

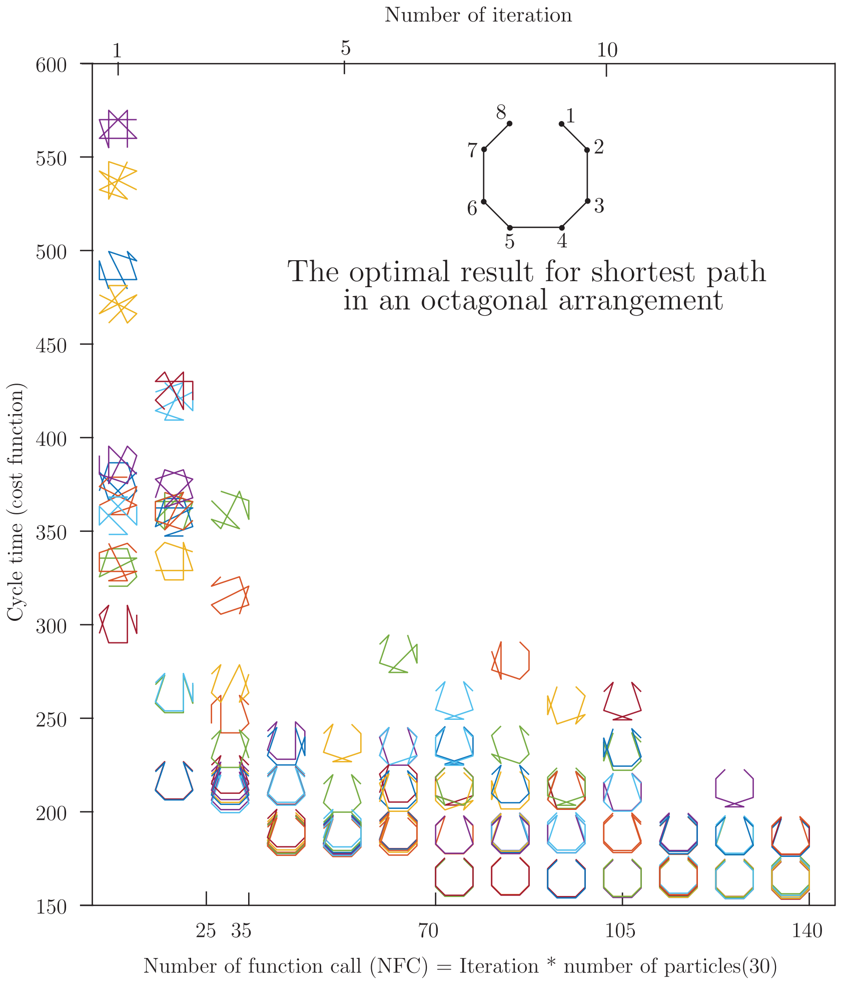

Figure 9. *Evolution of the particles to optimize the inspection workcell using BD-PSO. Each octagonal shape drawing represents the sequence of the particle.*

different set of images. Eventually, a near optimal solution is obtained in about 70 NFC (0.17% of 8! total possibilities), as shown in Fig. 10. The final placement of the camera is represented in Fig. 11 and the obtained sequence is $\{1, 2, 3, 4, 5, 6, 7, 8\}$.

In order for readers to implement, reproduce, and compare results, the hyper parameters used in this case study are provided in what follows. Size (S) of the search space is 6, since it is a six-dimensional problem, namely, translations along and rotations around x, y, and z axes. The number of combinatorial optimization is 8, as in number of images. Maximum number of iterations $it_{\max}$ is 12, and population size n_{pop} is 20. Coefficients ϕ_w, ϕ_p, and ϕ_g, associated with continuous inertia, personal and global learning values are 0.5, 0.5, and 0.65. For combinatorial learning coefficients ψ_w, ψ_p, ψ_g, and ψ_m, associated with inertia, personal, global, and mutation values, values are 0.3, 0.4, and 0.03.

It is worth noting that this approach can be extended to general cases and used for higher degrees of freedom mechanisms, both in continuous and discrete spaces. However, more number of function calls are expected and the speed of convergence highly depends on the computation power. Moreover, calculation time is not increased linearly. This approach is recommended to solve optimization problems related to robots up to 6 DOF ($S \leq 6$) and combination optimization of no more than roughly 20 images ($C \leq 20$).

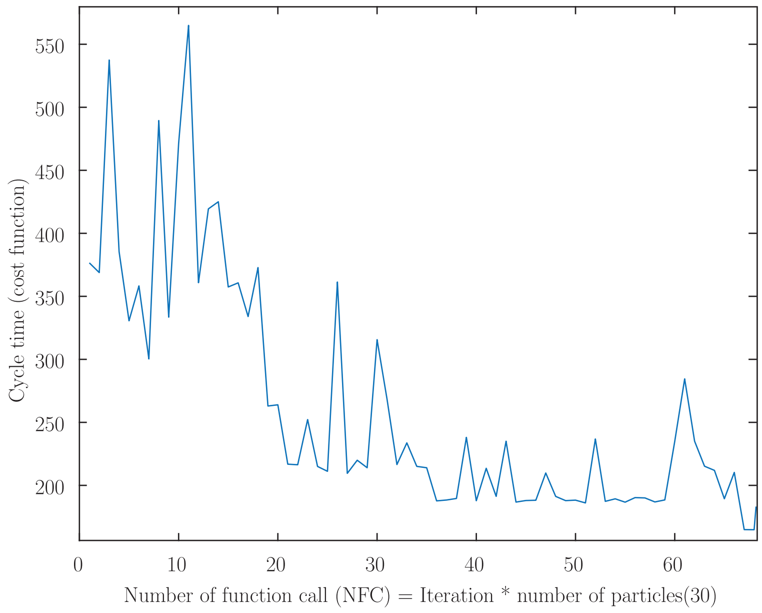

Figure 10. *Convergence of cycle time in the optimization of inspection workcell.*

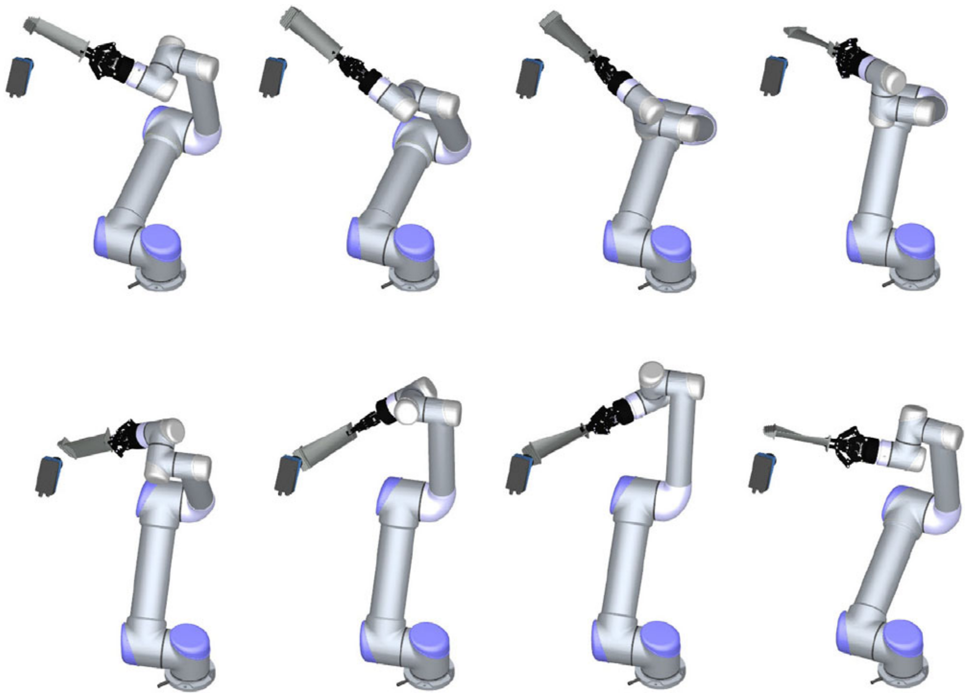

Figure 11. *8 robot poses to perform image inspection for 8 complicated images.*

6. Conclusion

In this paper, an evolutionary based optimization method, called BD-PSO, was proposed to simultaneously solve the continuous problem of camera placement optimization and the combinatorial problem of image sequence optimization. The continuous search space of camera placement includes 6 DOF.

The combinatorial solution was the shortest traveling path in a TSP for a blind and dynamic map space, meaning that the distances between the cities are not known (blind) and can change as a function of the camera placement (dynamic). Details of the algorithm were represented where the learning coefficients for personal, global, previous velocity, and mutation are defined for both aspects of the problem. The proposed algorithm was verified by comparing the result to a TSP benchmark. A near optimal result of (4.87 compared to 4.57) was obtained after 6.42×10^{-8} NFC fraction of total possibilities. A case study was introduced for a turbine blade polishing and inspection workcell.

Optimal design of workcell layout was obtained. In this optimization, the objective was to minimize the total cycle time of the process, and collision avoidance was respected as a limit of the solutions. In future works, for high number of images (more than 20), fine TSP solution is suggested to be performed at the end when the camera location is fixed.

Acknowledgments. The authors gratefully acknowledge the financial support of the Canada research chairs program.

References

[1] S. Alatartsev, S. Stellmacher and F. Ortmeier, "Robotic task sequencing problem: A survey," *J. Intell. Robot. Syst.* **80**(2), 279–298 (2015).

[2] A. Kovács, "Integrated task sequencing and path planning for robotic remote laser welding," *Int. J. Production Res.* **54**(4), 1210–1224 (2016).

[3] E. Kolakowska, S. F. Smith and M. Kristiansen, "Constraint optimization model of a scheduling problem for a robotic arm in automatic systems," *Rob. Auto. Syst.* **62**(2), 267–280 (2014).

[4] K. Vicencio, B. Davis and I. Gentilini, "Multi-goal Path Planning Based on the Generalized Traveling Salesman Problem with Neighborhoods," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)* (IEEE, 2014) pp. 2985–2990.

[5] M. Kaloorazi, M. T. Masouleh and S. Caro, "Interval-Analysis-Based Determination of the Singularity-Free Workspace of Gough-Stewart Parallel Robots," *2013 21st Iranian Conference on Electrical Engineering (ICEE)* (IEEE, 2013) pp. 1–6.

[6] M.-H. F. Kaloorazi, M. T. Masouleh and S. Caro, "Determination of the maximal singularity-free workspace of 3-DOF parallel mechanisms with a constructive geometric approach," *Mech. Mach. Theory* **84**, 25–36 (2015).

[7] M. H. FarzanehKaloorazi, I. A. Bonev and L. Birglen, "Parameters Identification of the Path Placement Optimization Problem for a Redundant Coordinated Robotic Workcell," *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (American Society of Mechanical Engineers, 2018) pp. V05BT07A087–V05BT07A087.

[8] S. Caro, C. Dumas, S. Garnier and B. Furet, "Workpiece Placement Optimization for Machining Operations with a KUKA KR270-2 Robot," *2013 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2013) pp. 2921–2926.

[9] R. Ur-Rehman, S. Caro, D. Chablat and P. Wenger, "Multi-objective path placement optimization of parallel kinematics machines based on energy consumption, shaking forces and maximum actuator torques: Application to the orthoglide," *Mech. Mach. Theory* **45**(8), 1125–1141 (2010).

[10] M. FarzanehKaloorazi, I. A. Bonev and L. Birglen, "Simultaneous path placement and trajectory planning optimization for a redundant coordinated robotic workcell," *Mech. Mach. Theory* **130**, 346–362 (2018).

[11] J. Zhang and A.-P. Li, "Genetic Algorithm for Robot Workcell Layout Problem," *WCSE'09. WRI World Congress on Software Engineering, 2009*, vol. 4 (IEEE, 2009) pp. 460–464.

[12] P. T. Zacharia and N. A. Aspragathos, "Optimization of Industrial Manipulators Cycle Time Based on Genetic Algorithms," *2004 2nd IEEE International Conference on Industrial Informatics, 2004. INDIN'04* (IEEE, 2004) pp. 517–522.

[13] K. Baizid, R. Chellali, A. Yousnadj, A. Meddahi and T. Bentaleb, "Genetic Algorithms Based Method for Time Optimization in Robotized Site," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2010) pp. 1359–1364.

[14] K. Baizid, A. Meddahi, A. Yousnadj, R. Chellali, H. Khan and J. Iqbal, "Robotized Task Time Scheduling and Optimization Based on Genetic Algorithms for Non Redundant Industrial Manipulators," *2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE)* (IEEE, 2014) pp. 112–117.

[15] A. Tubaileh, I. Hammad and L. Kafafi, "Robot cell planning," *Eng. Tech.* **26**, 250–253 (2007).

[16] P. T. Zacharia and N. Aspragathos, "Optimal robot task scheduling based on genetic algorithms," *Rob. Comput. Integr. Manuf.* **21**(1), 67–79 (2005).

[17] Y. P. Aneja and H. Kamoun, "Scheduling of parts and robot activities in a two machine robotic cell," *Comput. Oper. Res.* **26**(4), 297–312 (1999).

[18] N. G. Hall, H. Kamoun and C. Sriskandarajah, "Scheduling in robotic cells: Classification, two and three machine cells," *Oper. Res.* **45**(3), 421–439 (1997).

[19] P. Kurtser and Y. Edan, "Planning the sequence of tasks for harvesting robots," *Rob. Auto. Syst.* **131**, 103591 (2020). https://doi.org/10.1016/j.robot.2020.103591.

[20] A. Nedjatia and B. Vizvárib, "Robot path planning by traveling salesman problem with circle neighborhood: Modeling, algorithm, and applications," arXiv preprint arXiv:2003.06712 (2020).

[21] A. Oliinyk, I. Fedorchenko, A. Stepanenko, M. Rud and D. Goncharenko, "Implementation of Evolutionary Methods of Solving the Travelling Salesman Problem in a Robotic Warehouse," **In:** *Data-Centric Business and Applications* (Springer, 2020) pp. 263–292.

[22] J. Li and F. Yang, "Task assignment strategy for multi-robot based on improved grey wolf optimizer," *J. Ambient Intell. Humanized Comput.* **11**, 6319–6335 (2020). https://doi.org/10.1007/s12652-020-02224-3.

[23] M. Bottin, G. Rosati and G. Boschetti, "Working Cycle Sequence Optimization for Industrial Robots," *The International Conference of IFToMM ITALY* (Springer, 2020) pp. 228–236.

[24] S. Tian, Y. Chen, Q. Gu, R. Hu, R. Li and D. He, "Optimal Path Planning for a Robot Shelf Picking System," *2020 39th Chinese Control Conference (CCC)* (IEEE, 2020) pp. 3898–3903.

[25] J. Kennedy, "Particle Swarm Optimization," **In:** *Encyclopedia of Machine Learning* (Springer, 2011) pp. 760–766.

[26] G. Field and Y. Stepanenko, "Iterative Dynamic Programming: An Approach to Minimum Energy Trajectory Planning for Robotic Manipulators," *1996 IEEE International Conference on Robotics and Automation, 1996. Proceedings*, vol. 3 (IEEE, 1996) pp. 2755–2760.

[27] A. R. Hirakawa and A. Kawamura, "Trajectory Planning of Redundant Manipulators for Minimum Energy Consumption without Matrix Inversion," *1997 IEEE International Conference on Robotics and Automation, 1997. Proceedings*, vol. 3 (IEEE, 1997) pp. 2415–2420.

[28] K. Chan and A. Zalzala, "Genetic-Based Minimum-Time Trajectory Planning of Articulated Manipulators with Torque Constraints," *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, (IET, 1993) pp. 4–1.

[29] V. Pateloup, E. Duc and P. Ray, "Corner optimization for pocket machining," *Int. J. Mach. Tools Manuf.* **44**(12–13), 1343–1353 (2004).

[30] L. Tian and C. Collins, "Motion planning for redundant manipulators using a floating point genetic algorithm," *J. Intell. Rob. Syst.* **38**(3–4), 297–312 (2003).

[31] M. H. F. Kaloorazi, M. T. Masouleh and S. Caro, "Collision-free workspace of parallel mechanisms based on an interval analysis approach," *Robotica* **35**(8), 1747–1760 (2017).

[32] M. H. F. Kaloorazi, M. T. Masouleh and S. Caro, "Collision-Free Workspace of 3-RPR Planar Parallel Mechanism via Interval Analysis," **In:** *Advances in Robot Kinematics* (Springer, 2014) pp. 327–334.

[33] P. Jiménez, F. Thomas and C. Torras, "3D collision detection: A survey," *Comput. Graphics* **25**(2), 269–285 (2001).

[34] E. L. Lawler, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics (1985).

[35] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.* **11**(4), 341–359 (1997).

[36] J. Kennedy, "The Particle Swarm: Social Adaptation of Knowledge," *IEEE International Conference on Evolutionary Computation, 1997* (IEEE, 1997) pp. 303–308.

[37] R. Hassan, B. Cohanim, O. De Weck and G. Venter, "A Comparison of Particle Swarm Optimization and the Genetic Algorithm," *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, p. 1897.

[38] K.-P. Wang, L. Huang, C.-G. Zhou and W. Pang, "Particle Swarm Optimization for Traveling Salesman Problem," *2003 International Conference on Machine Learning and Cybernetics*, vol. 3 (IEEE, 2003) pp. 1583–1585.

[39] T.-D. Wang, X. Li and X. Wang, *Simulated Evolution and Learning: 6th International Conference, SEAL 2006, Hefei, China, October 15–18, 2006, Proceedings*, vol. 4247 (Springer, 2006).

[40] J. B. Odili and M. N. Mohmad Kahar, "Solving the traveling salesman's problem using the african buffalo optimization," *Comput. Intell. Neurosci.* **2016**, 3, (2016).

[41] F. Huilian, "Discrete particle swarm optimization for tsp based on neighborhood," *J. Comput. Inf. Syst.* **6**(10), 3407–3414 (2010).

Cite this article: M. FarzanehKaloorazi, I. A. Bonev and L. Birglen (2021). "Simultaneous task placement and sequence optimization in an inspection robotic cell", Robotica **39**, 2110–2130. https://doi.org/10.1017/S0263574721000199