



A breakout local search (BLS) method for solving the assembly sequence planning problem



Somayé Ghandi, Ellips Masehian*

Faculty of Engineering, Tarbiat Modares University, Tehran 14115-143, Iran

ARTICLE INFO

Article history:

Received 15 July 2014

Received in revised form

7 November 2014

Accepted 16 December 2014

Available online 20 January 2015

Keywords:

Assembly Sequence Planning (ASP)

Fitness landscape analysis

Breakout local search

Multi-objective optimization

Statistical analysis

ABSTRACT

Being one of the main subproblems of the broader Assembly Planning (AP) problem, Assembly Sequence Planning (ASP) is defined as the process of computing a sequence of assembly motions for constituent parts of an assembled final product. ASP is proven to be NP-complete and thus its effective and efficient solution has been a challenge for the researchers in the field. However, despite the existence of numerous works on solving the ASP, the topology, structure, and complexity of the problem's search space (i.e., the fitness landscape) has not been studied yet. In this article, the fitness landscape of the ASP problem is analyzed for five typical assembled products through various distribution and correlation statistical measures, which reveals that locally optimal assembly sequences are distributed in the problem's landscape nearly uniformly. Based on this result, a suitable optimization algorithm called Breakout Local Search (BLS) is selected and customized for obtaining high-quality solutions to ASP. A number of ASP problems are solved by the presented BLS and other algorithms in the ASP literature, including simulated annealing, genetic algorithm, memetic algorithm, harmony search, hybrid immune systems-particle swarm optimization, as well as by other variants of local search like iterated local search and multi-start local search. Experimental results and their in-depth statistical analyses show that the BLS outperforms other algorithms by producing the best-known or optimal solutions most of the time.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

For a mechanical product A composed of n parts (i.e., $A = \{p_1, p_2, \dots, p_n\}$) with known positions and geometries, the Assembly Sequence Planning (ASP) problem concerns with finding a sequence of $l \geq n$ collision-free assembly operations $O = (o_1, o_2, \dots, o_l)$ such that the assembled parts make up the product A . Since assembly of manufactured goods constitutes over half of the total production time in manufacturing industry, and about one-third to half of labor costs (Su, 2007), ASP plays a key role in the whole lifecycle of a product and has a great impact on variation propagation, production quality, and efficiency of the assembly process (Sacerdoti, 1974).

Assembly process constraints include topological relationships of the assembly, geometrical constraints and precedence relationships of parts or subassemblies, assembly direction changes, and issues regarding tools, accessibility, stability, and safety, which strictly limit the number of feasible assembly sequences. Optimal or near-optimal assembly sequences are those that minimize assembly cost or time while satisfying the above constraints. ASP is a highly combinatorial

constraint satisfaction problem as the number of potential assembly sequences (i.e., the size of the search space of assembly sequences) is proportional to the factorial of the number of parts or components composing the whole product (Wolter, 1989). In general terms, automatically generating feasible assembly sequences is, in its full generality, an extraordinarily difficult task. It was shown to be NP-complete problem and its solution is therefore believed to require more than polynomial time (Wilson and Watkins, 1990).

Depending on special assembly process constraints, assembly sequences may have four basic properties (Homem de Mello and Lee, 1991): (1) *Sequentiality*, which refers to the maximum number of moving subassemblies with respect to one another in any assembly operation. Sequentiality also addresses the number of required hands for assembling a product, and two-handed (also known as *binary* or *sequential*) sequences are the simplest ASP problems. (2) *Monotonicity*, which refers to the need for intermediate placement(s) of at least one part of the assembly. This means that in order for the problem to be solved, some parts must be moved more than once. (3) *Linearity*, which refers to the need for simultaneous insertion of more than one part in at least one stage of the operation. So, in a linear ASP problem all assembly operations include inserting of a single part into the rest of the assembly. (4) *Coherency*, which refers to the need for attaching some parts as a subassembly before inserting it into the main assembly. In a

* Corresponding author at: Faculty of Engineering, Tarbiat Modares University, Jalale-Ale-Ahmad highway, Tehran, 14115-143, Iran. Tel.: +98 21 82884939.

E-mail addresses: s.ghandibidgoli@modares.ac.ir (S. Ghandi), masehian@modares.ac.ir (E. Masehian).

coherent assembly all parts inserted into the assembly must touch some previously-assembled part(s), and therefore it is simpler than an incoherent assembly because the latter requires grasping or maintaining the stability of a subassembly before its assembly. In this article, Sequential, Monotone, Linear, and Coherent/Incoherent assembly sequence plans are considered.

A closely related problem to assembly sequence planning is *Disassembly Sequence Planning* (DASP), which computes a sequence of collision-free operations for removing the assembly parts from the final product, having given the geometry of the final product and the positions of parts in the final product. The DASP problem has also been shown to be NP-complete, and a survey on DASP is presented in (Lambert, 2003), which covers topics such as different ways of representing the space of feasible disassembly sequences: namely, (1) component-oriented approaches that consider automatic path generation, motion and stability analyses and collision detection, (2) product-oriented approaches that perform automatic analyses of the ability to decompose a product from its assembly drawing, (3) the hierarchical-tree approach which is based on the inverse Material Requirement Planning (MRP) and relates disassembly processes to the hierarchical product structure, and (4) metaheuristic approaches that have been used to generate optimal or near-optimal disassembly sequences such as Particle Swarm Optimization (PSO) (Tseng et al., 2011a, 2011b), Ant Colony Optimization (ACO) (Mi et al., 2011, Wang and Shi, 2014), Max–Min Ant System (MMAS) (Liu et al., 2012), Genetic Algorithms (GA) (Chen et al., 2012; Elsayed et al., 2011), and Artificial Immune Systems (AIS) (Lu and Liu, 2012).

1.1. Related works

In order to meet product development requirements, numerous methods have been used to generate optimal or near-optimal assembly sequences. Assembly generation methods lie in four general categories of (1) human interaction methods, (2) geometric feasible reasoning approaches, (3) knowledge-based reasoning approaches, and (4) intelligent methods. An inclusive survey on the ASP and its methods is presented in Jiménez (2013), which overviews the elements of sequence planning such as finding a feasible sequence, determining an optimal sequence according to one or more operational criteria, representing the space of feasible assembly sequences in different ways, applying search and optimization algorithms, and satisfying precedence constraints existing between subassemblies.

Human interaction is the earliest approach in ASP which relies on user responses to program queries on the connection between a pair of parts, and the feasibility of a single assembly operation (De Fazio and Whitney, 1987). Since this approach was hardly directed toward factory automation, it was soon superseded by the geometric feasible reasoning approach.

The *Geometric feasible reasoning* approach is mainly based on the ‘assembly by disassembly’ principle, which indicates that (under certain conditions) disassembly sequence is the inverse of assembly sequence. This is an important strategy for solving the ASP since a product in its assembled state is subject to far more precedence and motion constraints on its components than in its disassembled state (Halperin et al., 2000), as a result of which the solution space size of the DASP problem is drastically reduced compared to that of its reverse (ASP) problem. There is a bijection between assembly and disassembly sequences when merely geometric constraints are concerned and all parts are rigid, but this relation does not remain correct when physics (e.g., gravity, friction) and motion control uncertainty are taken into account, or when some parts are deformable (as considered in (Rakshit and Akella, 2013)) or are tolerated (as discussed in (Latombe et al. (1997))). Some works employing the geometric feasible reasoning approach are Homem de Mello and Sanderson (1991), Ou and Xu (2013), and Morato et al.

(2013). This approach suffers from combinatorial explosion and considers only geometric information of the parts, whereas non-geometric assembly information should also be considered in assembly planning in order to generate better assembly plans.

The *Knowledge-based reasoning* approach considers both geometric and non-geometric (such as human knowledge, past experience in assembling similar parts, etc.) assembly information. In Dong et al. (2007) both geometric and non-geometric information is hierarchically organized in a connection semantics-based assembly tree, and in Hsu et al. (2011) a knowledge-based engineering system for promptly predicting a near-optimal assembly sequence is developed. The drawback of this approach is that the knowledge base is difficult to construct and it is hard to match a given assembly with an existing assembly in the knowledge base.

In recent years, intensive research efforts have been put in developing *intelligent* methods to solve the ASP problem, as they have been able to improve the efficiency of finding optimal assembly sequences while avoiding the combinatorial explosion problem as the number of assembly components increases. As an intelligent method, Artificial Neural Networks (ANN) are used to encode the precedence knowledge by expressing the precedence constraints of AND/OR graphs either as the weights of connections between neurons of a Hopfield net (Chen, 1992), or as the probability of each part to be assembled at each step in the neurons’ outputs of an $n \times n$ network (Hong and Cho, 1995). Other works using back-propagation neural networks for assembly sequence optimization are Sinanoglu and Borkl  (2005) and Chen et al. (2008).

After Bonneville et al. (1995) implemented GA in ASP for the first time in 1995, many soft computing/metaheuristic algorithms were developed to solve the problem. Although numerous papers like Smith (2004), Marian et al. (2006), Tseng et al. (2004), and Wang and Tseng (2009) have used GA, this method in its original and basic form is inappropriate to be directly implemented to solve and optimize the ASP problem because its inherently binary strings of chromosomes is incompatible with permutation-like solutions of combinatorial problems such as ASP. Also, through the crossover recombination operator, the GA tends to generate infeasible offsprings that violate precedence constraints. To address this issue, researchers have used different approaches like penalty and repair strategies. As another improvement, Gao et al. (2010) proposed the Memetic Algorithm (MA) for the ASP to combine the parallel search nature of evolutionary algorithms with local search to improve individual solutions. The results showed that their proposed approach could reach a tradeoff between exploration and exploitation of the search space.

As a major optimization algorithm, PSO has been used to the ASP problem by a number of researchers. Although the original PSO is not suitable to be directly applied to ASP problem due to its continuous-space nature, a discrete version of it was implemented in (Lv and Lu, 2010). Wang and Liu (2010) introduced a chaotic operator to diversify updated particle positions in order to reduce premature convergence and trapping in local optima. To remedy this, Yu et al. (2009) introduced a two-formula mechanism of updating velocities (versus the usual single formula) in which one of the formulas was used randomly for each particle.

The Simulated Annealing (SA) method for ASP consists of generating a random initial sequence, creating a neighboring solution, computing the energy corresponding to this new sequence, computing the Boltzmann probability of changing to the new energy state, and accepting the new sequence if this probability is larger than a random number in the interval $[0, 1]$ (in order to escape local minima). In Motavalli and Islam (1997) and Hong and Cho (1999), the SA algorithm was used with a multi-criteria objective function integrating total assembly time and number of reorientations. Motavalli and Islam (1997) and Hong and Cho (1999) applied the SA to minimize an energy function as the optimality criterion with a

part-exchange operator for generating neighboring solutions. They compared their algorithm with a neural-network-based method for the assembly of an electrical relay product.

In the Artificial Immune Systems approach to ASP, antibodies represent the assembly sequences of the product encoded in their genes as parts' numbers. Cao and Xiao (2007) implemented the AIS for ASP and concluded that it outperforms the standard GA due to the immune selection mechanism which selects individuals (antibodies) for the next generation and chooses the best antibodies (with higher fitness values) while at the same time favoring diversity; that is, by avoiding premature convergence and helping global optimization.

Wang et al. (2005) used the ACO algorithm to overcome a shortcoming of GA that highly depends on initial chromosomes. In their algorithm the positive feedback process, distributed computation, and the greedy constructive heuristic search work together to find the optimal or near-optimal assembly sequence quickly and efficiently. Since one of the common drawbacks of the original ACO pertains to the positive feedback system that accumulates only good solutions, Guo et al. (2007) focused on resolving its premature convergence.

Several other heuristic methods such as Harmony Search (HS) (Wang et al., 2013), Imperialist Competitive Algorithm (ICA) (Zhou et al., 2013), Firefly Algorithm (FA) (Yi and Jianhua, 2013), psychoclonal algorithm (Tiwarei et al., 2005), bacterial chemotaxis algorithm (Zhou et al., 2011), etc., have also been used to solve the ASP problem. A survey of works that have applied soft computing approaches in ASP has been presented in (Rashid et al., 2012), covering the years 2001–2011.

In reviewing the literature of ASP methods it was noticed that a large majority of solution methods were in the category of intelligent methods, among which the metaheuristics comprise a large portion. However, a question that arises here is that what type of metaheuristic is more appropriate for ASP. In fact, deciding upon implementing a metaheuristic with powerful intensification or diversification capabilities must be justified based on the problem's *fitness landscape* characteristics, which gives a sense about the 'shape' of the problem's solution space through describing the distribution, amplitudes, and correlation of its local optima.

Despite the large number of metaheuristics applied to solve the ASP problem, the landscape of this problem has not been analyzed in any research so far, and since the effectiveness of metaheuristics depends on the properties of the landscape associated with the problem to solve, in this paper the landscape of the ASP problem is analyzed for the first time in Section 2. Analysis of distribution and correlation measures shed light on the nature of the problem and guided us in implementing an effective and efficient algorithm called *Breakout Local Search* (BLS) algorithm for the ASP problem for the first time, which is presented in Section 3 in detail. Experimental results, comparisons with various existing methods, and detailed statistical analyses are presented in Section 4, and summarizing remarks and conclusions are provided in Section 5.

2. Fitness landscape analysis of the ASP problem

In designing metaheuristic algorithms for solving optimization problems, two inconsistent strategies may be adopted: exploitation (intensification), and exploration (diversification). In intensification the regions around local optima are explored more thoroughly in the hope to find better solutions, while in diversification non-explored regions

must be visited to ensure that all regions of the search space are evenly explored and that the search is not confined to only a reduced number of regions. In this design space, the extreme search algorithms in terms of the exploration (resp. exploitation) are random search (resp. iterative improvement local search), as illustrated in Fig. 1. In general, basic single-solution-based metaheuristics (or S-metaheuristics) are more exploitation-oriented, whereas basic population-based metaheuristics (or P-metaheuristics) are more exploration-oriented (Talbi, 2009). This does not mean of course that S-metaheuristics (resp. P-metaheuristics) do not achieve exploration (resp. exploitation). Local search (LS) and simulated annealing (SA) are examples of S-metaheuristics, and GA, PSO, ACO, NN, AIS, and MA are examples of P-metaheuristics. There are also hybrid algorithms, which usually nest an S-metaheuristic inside a P-metaheuristic, such as GA-SA (Shu-Xia and Hong-Bo, 2008), GA-TS (Li et al., 2003), etc.

In developing metaheuristics for an optimization problem, the designer must consider the properties of the problem's landscape as it influences the effectiveness and efficiency of the metaheuristic. These properties include types of solution representation, neighborhood, and the objective function, which completely define the landscape of a problem. The *Search space* of a problem can be defined as a hyper graph $G=(S, E)$, in which the set of vertices S corresponds to the solutions of the problem represented by some encoding, and the set of edges E corresponds to the move operators used to generate new solutions. If the solution s_q can be generated from the solution s_p using a move operator once, there is an edge between these solutions and they are neighbors. If the search space is considered as a floor, then the landscape of problems can be made of valleys, plains, peaks, canyons, cliffs, plateaus, basins, etc. Depending on the shape of the landscape, specific types of search methods with certain exploitation and exploration capabilities will be more effective. By computing and analyzing statistical measures such as correlation and distribution of local optima over the landscape of the ASP problem, we will be able to form an idea about the shape and ruggedness of the fitness landscape and thus propose a suitable metaheuristic that will work well in that context. The most common motivation for fitness landscape analysis is to gain a better understanding of algorithm performance on a related set of problem instances. Therefore, in order to find out what type of metaheuristics is suitable for a specific problem, the landscape of the problem's search space should be analyzed (Pitzer and Affenzeller, 2012).

In this section, our fitness landscape analysis is based on five sample assembly products with various geometries, number of parts, complexities, and dimensions. A simple hill-climbing local search algorithm is developed and run on 5000 uniformly-sampled random starting assembly sequences (the set of which is called population U), and then for each starting sequence in U (which is in the form of a permutation), the local search is run until a local optimum is achieved. As a result, 5000 (not necessarily distinct) local optima are generated, the set of which is called population O . In order to perform a local search on a random solution in Population U , it is first represented as an encoding (Section 2.1), and a neighboring solution is generated by applying a neighborhood generation operator (Section 2.2). If the fitness value of this neighbor (Section 2.3) was better than that of the current solution, then it replaces the current solution, otherwise, another random neighbor is generated and its fitness compared with the current solution. This procedure is continued iteratively until at an

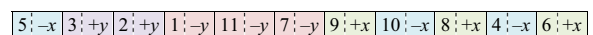


Fig. 2. A typical encoding for assembly sequence of 11 parts in 2D.

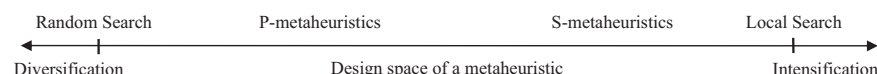


Fig. 1. Two conflicting criteria in designing a metaheuristic: diversification versus intensification (Talbi, 2009).

iteration no better neighboring solution is found for the current solution after 100 trials. Thus, we allegedly call this solution a 'local optimum' and add it to the Population O . After completely generating the Population O , some statistical measures of the distribution and correlation of local optima in the search space of the ASP problem are calculated (Section 2.4), based on which fitness landscape analysis is performed for the studied assembled products and the results reported in Section 2.5.

2.1. Solution representation

A solution s to the ASP problem can be represented by the assembling sequence and directions of the n parts of the assembly A in the form of a row vector of ordered pairs (π_i, d_i) , in which $\pi_i \in \{p_1, p_2, \dots, p_n\}$ is the i -th element of the sequence and $d_i \in \{-x, +x, -y, +y, -z, +z\}$ represents the assembly direction of part π_i (for 3D assemblies). Fig. 2 shows a typical encoding of a solution as $\langle (p_5, -x), (p_3, +y), \dots, (p_6, +x) \rangle$, which indicates that part $\pi_1 (=p_5)$ must move at direction $-x$ toward the assembly, then part $\pi_2 (=p_3)$ must move at direction $+y$ toward the assembly, and so on.

Since the total number of possible sequences of n parts is $n!$, and the number of possible directions to assemble a particular part is $2m$, then the size of the whole search space (including feasible and infeasible solutions) equals to $n!(2m)^n$. Also, since the solution encoding has $2n$ cells, the maximum distance between two solutions in the search space (i.e., the diameter of the search space) is $2n$, as all cells in two solutions may be different.

2.2. Neighborhood generation operators

In order to conduct the fitness landscape analysis, a set of 5000 solutions are generated randomly as the initial population U , and a steepest descent (hill-climbing) local search is performed on each initial solution to reach a solution with locally maximum fitness value. For this purpose, first we need to generate neighboring solutions, which is done by four different neighborhood generation operators in this work. These operators are exchange, insertion, inversion, and flip, as explained below:

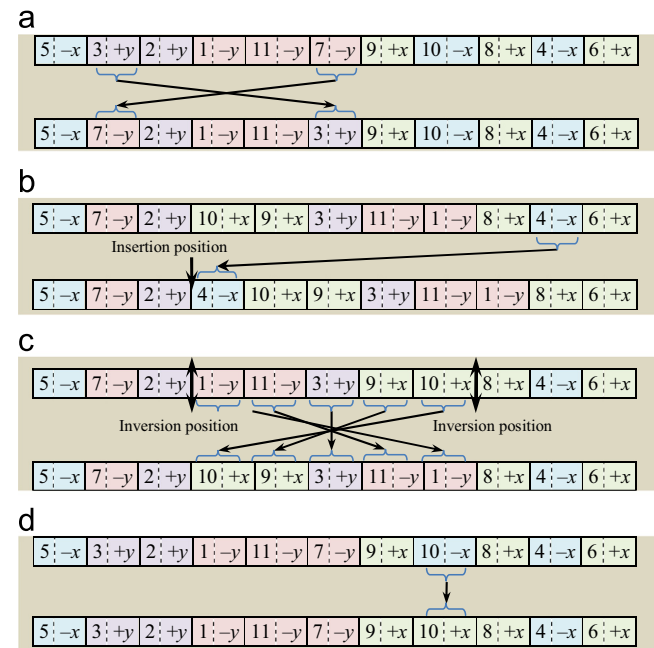


Fig. 3. Neighborhood generating operators: (a) exchange, (b) insertion, (c) inversion, and (d) flip.

- *Exchange operator*: This operator randomly selects two pairs of the sequence and exchanges (swaps) their positions in the encoding, as shown in Fig. 3(a). The neighborhood size for any solution through applying this operator is $C(n, 2) = n(n-1)/2$.
- *Insertion operator*: This operator randomly selects a pair in the sequence and relocates to another position in the string, such that all pairs after that position shift one place to the right, as shown in Fig. 3(b). The neighborhood size for any solution through applying this operator is $n \times n = n^2$, since n pairs can be selected as the start, and n positions (the position after the last pair included) can be selected as the target of the relocation.
- *Inversion operator*: This operator randomly selects two positions in the solution (two parts of the product) and reverses the position of parts between them in the sequence, as shown in Fig. 3(c). The neighborhood size for any solution through applying this operator is $\sum_{i=1}^{n-1} (n-i-1)$.
- *Flip operator*: This operator randomly selects a pair in the sequence and changes its assembly direction randomly to another direction, as shown in Fig. 3(d). The neighborhood size for any solution through applying this operator is $(2m-1)^n$, in which $m \in \{2, 3\}$ is the dimension of the assembly.

It is important for neighborhood generating operators to have the *connectivity* property, which guarantees reaching any solution from any other solution through successive application of the operators. It can be shown that for any two solutions s_p and s_q (which are in fact nodes of the search space G), our used neighborhood generation operators have the connectivity property: by successively implementing each of the exchange, insertion, or inversion operators on a given solution, all possible sequences of parts can be obtained, and successively applying the flip operator can generate all assembly directions for any part in a given sequence. Therefore, it is necessary to use the flip operator in conjunction with the other three operators in order to guarantee the connectivity property, such that when the above operators are applied randomly over an infinite time interval, there will always be a path between every two solutions in G , and the probability of reaching the global optimal solution s^* starting from any initial solution tends to 1. Nevertheless, our developed method, as all other metaheuristics, aims to reduce the search time as much as possible and improve the quality of solutions obtained.

In each iteration of the local search, the above neighborhood generation operators are selected uniformly at random and applied on the current solution representation. Therefore, the maximum number of neighbors of a solution s will be:

$$|N(s)| = \max\{n(n-1)/2, n^2, \sum_{i=1}^{n-1} (n-i-1), (2m-1)^n\}. \quad (1)$$

2.3. Fitness of solutions

Solving an ASP problem requires finding assembly sequences and directions that are both *feasible* and *high quality* as measured by a quality metric. In fact, these two properties guide us in selecting the 'best' neighbor(s) of a specific solution during the search process and hence converging to desired final solutions. The feasibility and fitness (quality) of a solution are discussed in the following subsections.

2.3.1. Feasibility of a solution

Assume $BB(\pi_i)$ be the bounding box of the parts π_j ($j=1, \dots, i-1$) assembled prior to π_i which has axes parallel to the main Cartesian directions x , y , and z . Also, let $AV(\pi_i, d_i)$ denote the assembly vector of part π_i along direction d_i , starting from a point on the boundary of $BB(\pi_i)$ and ending at the final configuration of π_i in the assembly A .

Then, a solution $s = \langle (\pi_1, d_1), (\pi_2, d_2), \dots, (\pi_n, d_n) \rangle$ to an ASP problem is *feasible* if there is no already-assembled part π_j ($\forall j < i$) blocking π_i along $AV(\pi_i, d_i)$. Put more formally, if $cl(V(\pi_i))$ denote the closure of the volume occupied by part π_i in its final configuration, and $SV_{\pi_i}^{AV(\pi_i, d_i)}$ represent the swept volume of part π_i along $AV(\pi_i, d_i)$ in the workspace, then a solution s is feasible if:

$$SV_{\pi_i}^{AV(\pi_i, d_i)} \cap \sum_{j=1}^{i-1} cl(V(\pi_j)) = \emptyset \quad \forall i = 2, \dots, n. \quad (2)$$

Since all solutions at each iteration of the local search need to be checked for feasibility, in order to expedite the search we pre-calculate all possible collisions between any two parts at all

directions and organize this information as an $n \times n \times 2m$ matrix called *Assembly Interference Matrix* (AIM). Each entry of the AIM is denoted by a binary variable $I_{p_i, p_j}^{AV(p_i, d_i)}$, which takes value 0 if part p_j does not block the movement of part p_i along direction d_i , and takes 1 otherwise, as defined below:

$$I_{p_i, p_j}^{AV(p_i, d_i)} = \begin{cases} 0 & \text{if } SV_{p_i}^{AV(p_i, d_i)} \cap cl(V(p_j)) = \emptyset \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

For all pairs of parts p_i and p_j ($i = 1, \dots, n; j = 1, \dots, n$) and for all directions $d_i \in \{-x, +x, -y, +y, -z, +z\}$ the AIM is shown in (4). An AIM is created for a sample part illustrated in Fig. 9.

$$\begin{array}{c} \overbrace{\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ I_{p_2, p_1}^{AV(p_2, -x)} & I_{p_2, p_1}^{AV(p_2, +x)} & I_{p_2, p_1}^{AV(p_2, -y)} & I_{p_2, p_1}^{AV(p_2, +y)} & I_{p_2, p_1}^{AV(p_2, -z)} & I_{p_2, p_1}^{AV(p_2, +z)} \\ I_{p_n, p_1}^{AV(p_n, -x)} & I_{p_n, p_1}^{AV(p_n, +x)} & \dots & I_{p_n, p_1}^{AV(p_n, +y)} & I_{p_n, p_1}^{AV(p_n, -z)} & I_{p_n, p_1}^{AV(p_n, +z)} \end{matrix}}^{p_1} \quad \dots \quad \overbrace{\begin{matrix} I_{p_1, p_n}^{AV(p_2, -x)} & I_{p_1, p_n}^{AV(p_2, +x)} & I_{p_1, p_n}^{AV(p_2, -y)} & I_{p_1, p_n}^{AV(p_2, +y)} & I_{p_1, p_n}^{AV(p_2, -z)} & I_{p_1, p_n}^{AV(p_2, +z)} \\ I_{p_2, p_n}^{AV(p_2, -x)} & I_{p_2, p_n}^{AV(p_2, +x)} & I_{p_2, p_n}^{AV(p_2, -y)} & I_{p_2, p_n}^{AV(p_2, +y)} & I_{p_2, p_n}^{AV(p_2, -z)} & I_{p_2, p_n}^{AV(p_2, +z)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}}^{p_n} \end{array} \quad (4)$$

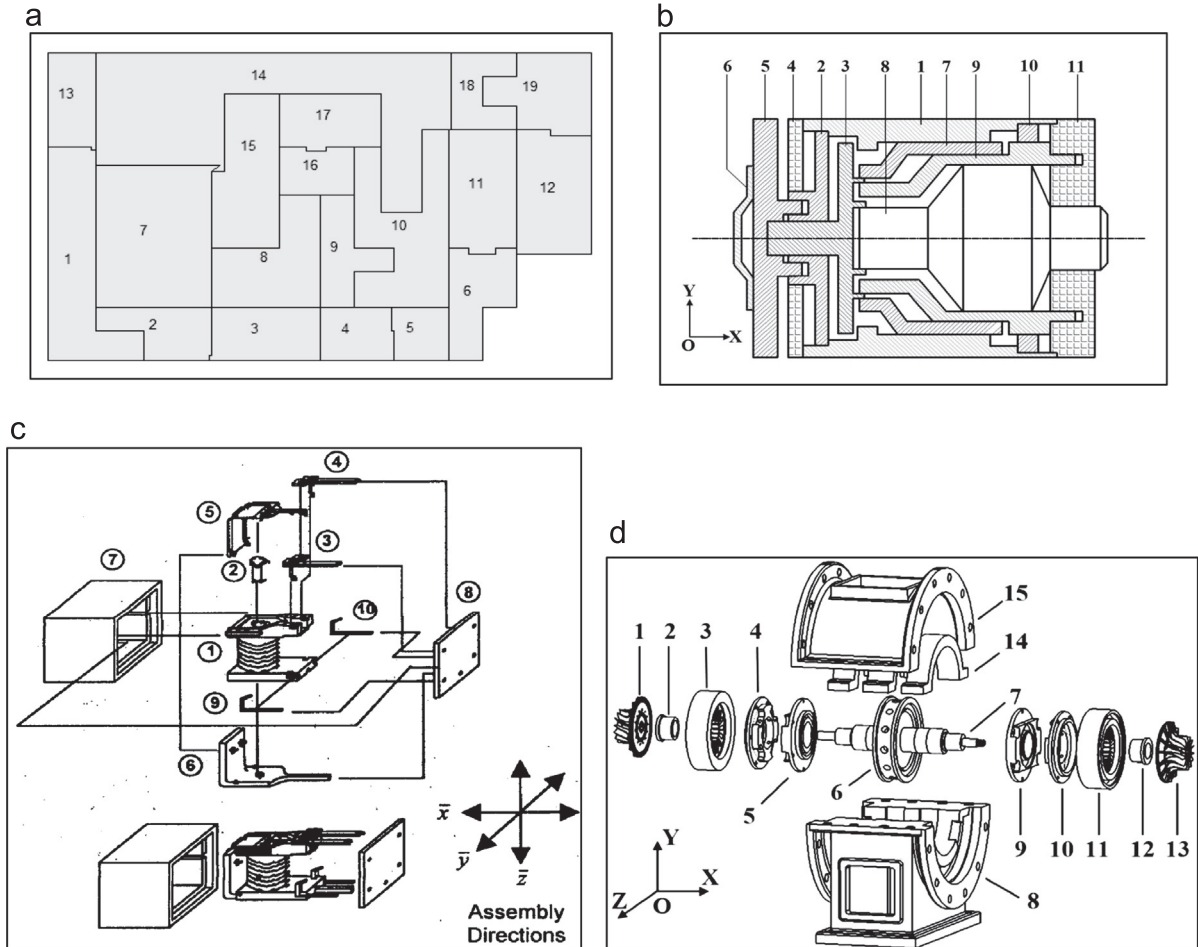
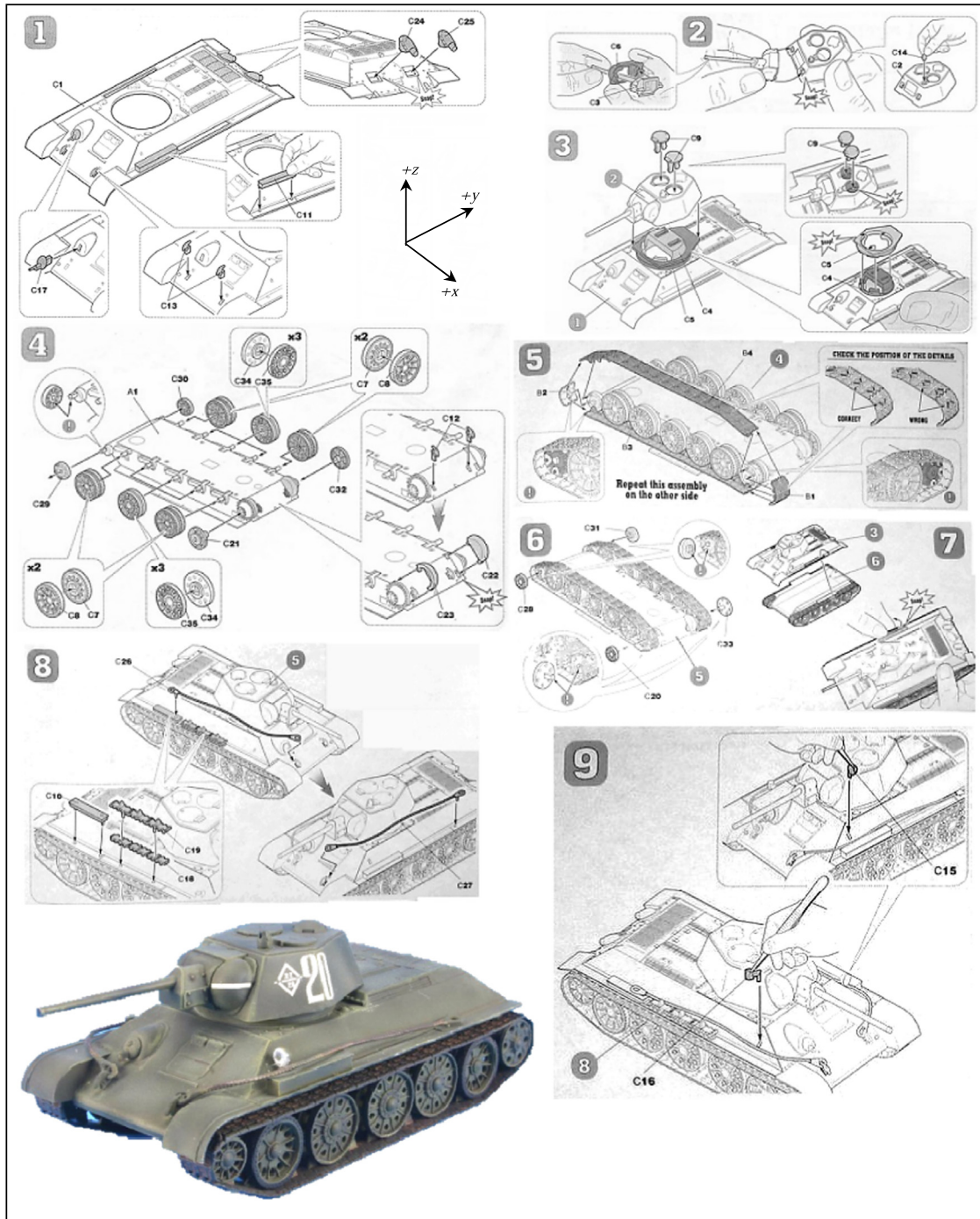


Fig. 4. Sample assembly products with various geometries, number of parts, complexities, and dimensions. (a) A puzzle-like assembly, inspired from (Smith and Liu, 2001). (b) An industrial assembly, from (De Fazio and Whitney, 1987) (c) An electrical relay, from (Hong and Cho, 1999). (d) A Generator, from (Wang and Liu, 2010). (e) Mockup of the Soviet medium tank T-34/76 from (Zvezda, 2014) and the new part numbers.



Part	A ₁	B ₁ (R)	B ₁ (L)	B ₂ (R)	B ₂ (L)	B ₃ (R)	B ₃ (L)	B ₄ (R)	B ₄ (L)	C ₁	C ₂	C ₃	C ₄
New ID	p ₁	p ₂	p ₃	p ₄	p ₅	p ₆	p ₇	p ₈	p ₉	p ₁₀	p ₁₁	p ₁₂	p ₁₃
Part	C ₅	C ₆	C ₇ (RF)	C ₇ (RB)	C ₇ (LF)	C ₇ (LB)	C ₈ (RF)	C ₈ (RB)	C ₈ (LF)	C ₈ (LB)	C ₉ (R)	C ₉ (L)	C ₁₀
New ID	p ₁₄	p ₁₅	p ₁₆	p ₁₇	p ₁₈	p ₁₉	p ₂₀	p ₂₁	p ₂₂	p ₂₃	p ₂₄	p ₂₅	p ₂₆
Part	C ₁₁	C ₁₂ (R)	C ₁₂ (L)	C ₁₃ (R)	C ₁₃ (L)	C ₁₄	C ₁₅	C ₁₆	C ₁₇	C ₁₈	C ₁₉	C ₂₀	C ₂₁
New ID	p ₂₇	p ₂₈	p ₂₉	p ₃₀	p ₃₁	p ₃₂	p ₃₃	p ₃₄	p ₃₅	p ₃₆	p ₃₇	p ₃₈	p ₃₉
Part	C ₂₂	C ₂₃	C ₂₄	C ₂₅	C ₂₆	C ₂₇	C ₂₈	C ₂₉	C ₃₀	C ₃₁	C ₃₂	C ₃₃	C ₃₄ (RF)
New ID	p ₄₀	p ₄₁	p ₄₂	p ₄₃	p ₄₄	p ₄₅	p ₄₆	p ₄₇	p ₄₈	p ₄₉	p ₅₀	p ₅₁	p ₅₂
Part	C ₃₄ (RM)	C ₃₄ (RB)	C ₃₄ (LF)	C ₃₄ (LM)	C ₃₄ (LB)	C ₃₅ (RF)	C ₃₅ (RM)	C ₃₅ (RB)	C ₃₅ (LF)	C ₃₅ (LM)	C ₃₅ (LB)		
New ID	p ₅₃	p ₅₄	p ₅₅	p ₅₆	p ₅₇	p ₅₈	p ₅₉	p ₆₀	p ₆₁	p ₆₂	p ₆₃		

Legend: L = left, R = right, M = middle, F = front, B = back.

Fig. 4. (continued)

2.3.2. Quality of a solution

In this paper, as in some other works on ASP, the objective function is to simultaneously minimize the total number of

Assembly Direction Changes (ADC) and maximize the number of Satisfied Geometric Constraints (SGC) of assembly sequences. $ADC(\pi_i)$ is a binary variable indicating the change of assembly

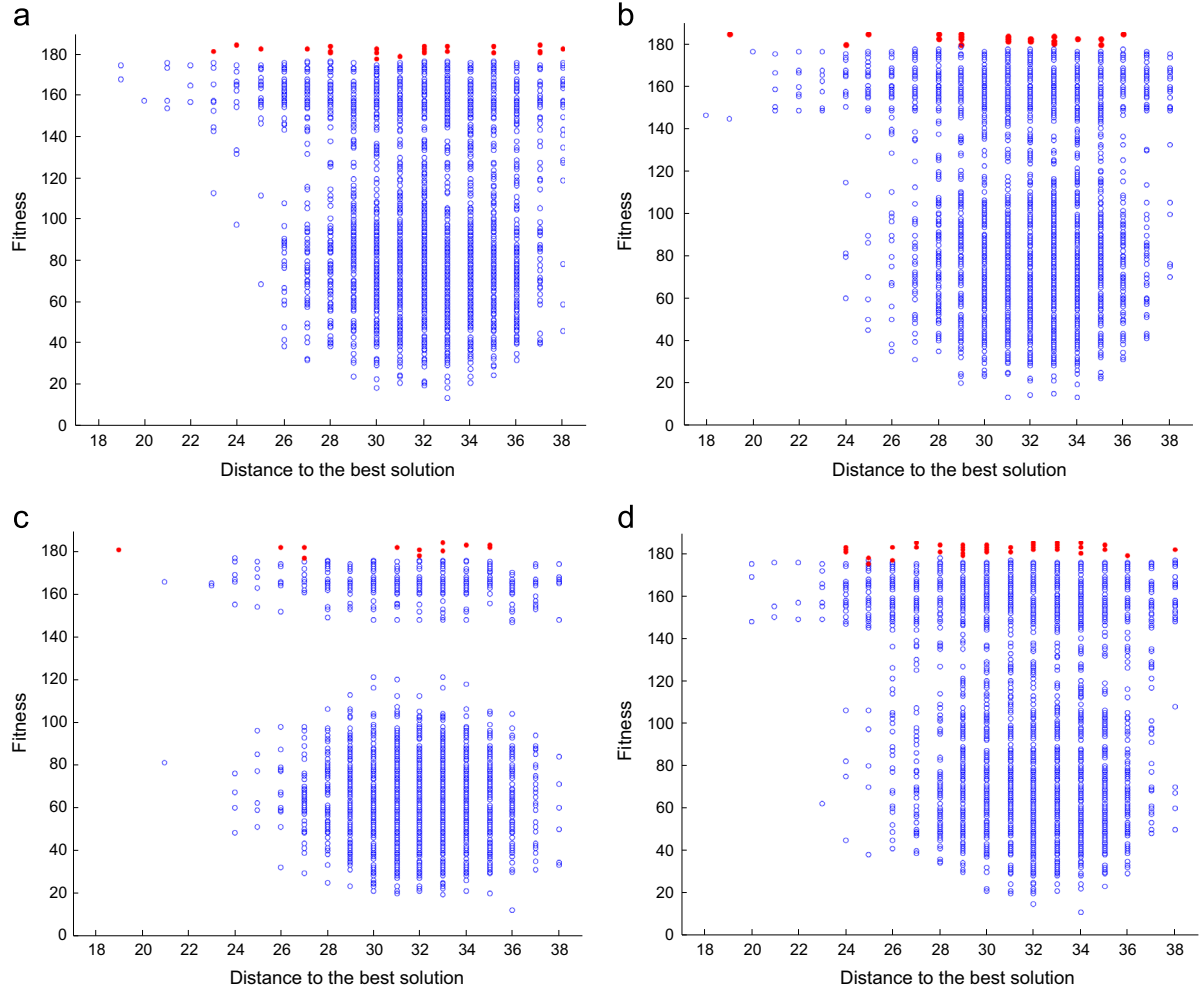


Fig. 5. F–D plots for 5000 local optimal solutions of the puzzle-like problem, classified according to (a) Exchange + Flip (XF), (b) Insertion + Flip (SF), (c) Inversion + Flip (VF), and (d) Exchange + Insertion + Inversion + Flip (ALL) operators.

direction between parts π_i and π_{i-1} in the solution encoding:

$$ADC(\pi_i) = \begin{cases} 0 & \text{if } d_i = d_{i-1} \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

Also, $SGC(\pi_i)$ is a binary variable that takes value 1 if no part assembled prior to part π_i blocks its movement along $AV(\pi_i, d_i)$. Using the AIM, this condition is expressed mathematically in (6). Obviously, $SGC(\pi_1) = 1$.

$$SGC(\pi_i) = \begin{cases} 1 & \text{if } \sum_{j=1}^{i-1} I_{\pi_i, \pi_j}^{AV(\pi_i, d_i)} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Now the total fitness function can be defined as a weighted sum of the above objectives expressed as follows:

$$\text{Maximize } f(s) = w_1 \sum_{i=2}^n (1 - ADC(\pi_i)) + w_2 \sum_{i=1}^n SGC(\pi_i). \quad (7)$$

Maximizing the first term in (7) is equivalent to minimizing the total number of assembly direction changes, and maximizing the second term is equivalent to maximizing the total number of feasible assembly operations. In order to bias the search toward feasible solutions, we set the values of the weighting factors as $w_1=1$ and $w_2=9$ empirically by testing several sets of these factors.

2.4. Landscape properties

In this paper, two groups of statistical measures are used to describe fitness landscape properties of the ASP problem, namely, (1) *Distribution* measures, that study the topology of locally optimal solutions in the search space and the objective space, and (2) *Correlation* measures, that analyze the rugosity (ruggedness) of the landscape and the correlation between the quality of solutions and their relative distance, as well as the correlation between the quality of solutions and their distance to the best-known solution. The following measures are discussed and calculated for our five sample assemblies: (i) local optima distribution, (ii) entropy in the search space, (iii), distribution in the objective space, (iv) length of the walks, (v) autocorrelation function, and (vi) fitness–distance correlation, among which the first three measures pertain to distribution measures, and the last three belong to correlation measures. Most of the material in this subsection is taken from (Talbi, 2009).

Distribution in the search space: This measure represents the concentration of a population P of solutions in the whole search space S . The normalized average distance of a population P is defined as follows:

$$0 \leq Dmm(P) = \frac{\sum_{s \in P} \sum_{t \in P, t \neq s} \text{dist}(s, t)}{|P| \cdot (|P| - 1) \cdot \text{diam}(S)} \leq 1, \quad (8)$$

where $\text{dist}(s, t)$ denotes the Hamming distance of two solutions, and $\text{diam}(P) = \max_{s, t \in P} \text{dist}(s, t)$; $\forall s, t \in P$ is the diameter of the

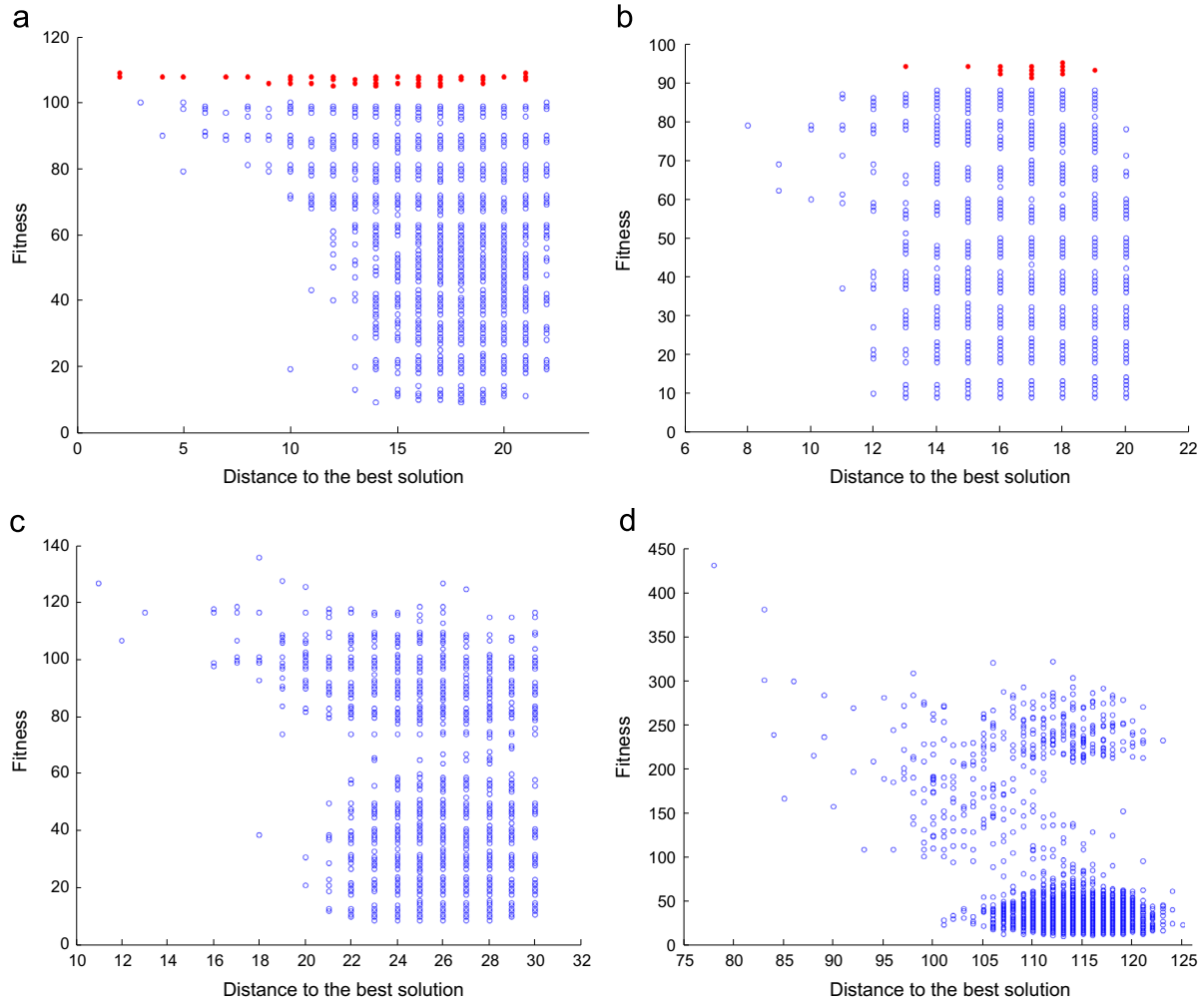


Fig. 6. Fitness–distance plots for 5000 locally optimal solutions obtained by the ALL neighborhood generation operators for the benchmark ASP problems: (a) industrial assembly, (b) electrical relay, (c) generator, and (d) tank.

Algorithm 1 (Main). Breakout_Local_Search ($AIM, M_{min}, M_{max}, N_{max}, L_{max}, maxiter$)

Inputs: AIM = The Assembly Interference Matrix
 M_{min} = Initial number of perturbation moves (Jump Magnitude)
 M_{max} = Maximum number of perturbation moves
 N_{max} = Maximum allowed number of consecutive non-improving local optima
 L_{max} = Maximum number of attempts within a local search iteration
 $maxiter$ = The maximum allowed number of iterations of the BLS algorithm

Output: s_{best} = Best solution found

```

1.  $\omega \leftarrow 0$  // Initializes counter  $\omega$  for enumerating consecutive non-improving local optima
2.  $iter \leftarrow 0$  // Initializes counter  $iter$  for enumerating the number of elapsed iterations
3.  $s_0 \leftarrow \text{Initial\_Solution}(AIM)$  // Generates the initial solution
4.  $s_{best} \leftarrow s_0$  // Records the best solution found so far
5.  $f_{best} \leftarrow f(s_0)$  // Records the best objective value obtained so far
6.  $s^* \leftarrow \text{Local\_Search}(s_0, L_{max})$  // Applies a hill-climbing local search
7.  $s_{lo} \leftarrow s^*$  // Records the last found local optimum
8.  $M \leftarrow M_{min}$  // Initializes the number of perturbation moves
9. While  $iter < maxiter$ 
10.  $M \leftarrow \text{Jump\_Magnitude}(M, M_{min}, M_{max}, N_{max}, s_{lo}, s^*, \omega)$  // Determines the number of perturbation moves
11.  $T \leftarrow \text{Perturbation\_Type}(\omega, N_{max})$  // Determines the type of perturbation moves
12.  $s_p \leftarrow \text{Perturb}(s^*, M, PT)$  // Perturbs  $s^*$  by applying  $M$  moves of type  $PT$ 
13.  $iter = iter + 1$ 
14.  $s^* \leftarrow \text{Local\_Search}(s_p, L_{max})$  // Applies a local search on the perturbed solution  $s_p$ 
15.  $s_{lo} \leftarrow s^*$  // Updates the last local optimum
16. If  $f(s^*) > f_{best}$  Then
17.  $s_{best} \leftarrow s^*$  // Records the best solution found so far
18.  $f_{best} \leftarrow f(s^*)$  // Calculated in (7)
19.  $\omega \leftarrow 0$  // Reset  $\omega$ 
20. Else
21.  $\omega \leftarrow \omega + 1$ 
22. End
23. End

```

Fig. 7. The main procedure of the BLS algorithm for the ASP problem.

population. A near zero Dmm indicates that the solutions in population P are concentrated in a small region of the search space S . The variation of average difference between Dmm of the starting population U and the final population O is defined as:

$$\Delta_{Dmm} = \frac{Dmm(U) - Dmm(O)}{Dmm(U)} \leq 1. \quad (9)$$

Entropy in the search space: Entropy is an indicator for the diversity of a given population in the search space. Weak (near zero) entropy implies concentration of solutions, whereas high entropy shows significant dispersion of the solutions in the search space. Since there is no general-purpose formulation for entropy as it depends on the optimization problem at hand, we have worked out a novel entropy formulation for the ASP problem as follows:

$$0 \leq ent(P) = \frac{\sum_{i=1}^n \sum_{j=1}^n \sum_{d=1}^{2m} N_{ij d} \cdot (|P| - N_{ij d})}{2mn^2 \cdot (|P| - 1)} \leq 1, \quad (10)$$

where $N_{ij d}$ is the number of solutions in P in which part p_i is in the j -th position of the sequence and is assembled along direction d , or mathematically, $p_i = \pi_j$ and $d = d_j$. For 3D assemblies, $d = 1, 2, 3, 4, 5, 6$ correspond to $-x, +x, -y, +y, -z, +z$ directions. When all solutions of the population P were similar, then for specific i, j , and d in the assembly sequence, $N_{ij d} = |P|$ would hold and so $ent(P) = 0$. On the other hand, if all solutions were totally different, then $N_{ij d} = 1$ would be true for every i, j , and d , and so $ent(P) = 1$. The entropy variation between the uniformly random and local optima populations is defined as follows:

$$\Delta_{ent} = \frac{ent(U) - ent(O)}{ent(U)} \leq 1. \quad (11)$$

Distribution in the objective space: In order to compute the distribution of solutions in the objective space, the *amplitude* indicator $Amp(P)$ is used to measure the difference between the fitness values of best and worst solutions in the population P , as define in (12). Small values of $Amp(P)$ denote a relatively flat search space. Relative variation of the amplitude between a

Algorithm 2: Initial_Solution(AIM)	
Output: $Asbl_{in}$ = The initial solution s_0	
1. For $i = 1$ to n	
2. For $\forall d_i \in \{-x, +x, -y, +y, -z, +z\}$	
3. $DAIM(p_i, d_i) = \sum_{j=1}^n I_{p_j, p_i}^{AV(p_i, d_i)}$	// DAIM is the Directional Assembly Interference Matrix
4. End	
5. End	
6. $d_1 \leftarrow d$	// Randomly select a direction d among all assembly directions
7. $\pi_1 \leftarrow p_i$	// p_i is the part with the minimal $DAIM(p_i, d_1)$
8. $Asbl_1 = \{(\pi_1, d_1)\}$	// The first part and direction of the assembly sequence
9. For $m = 1$ to $n - 1$	
10. For $p_i \notin \{\pi_1, \dots, \pi_m\}$	
11. For $\forall d_i \in \{-x, +x, -y, +y, -z, +z\}$	
12. $ABP(p_i, d_i) = \sum_{j=1}^m I_{p_i, \pi_j}^{AV(p_i, d_i)}$	
13. $UBP(p_i, d_i) = \sum_{p_j \in \{\pi_1, \dots, \pi_m\}} I_{p_j, p_i}^{AV(p_j, d_i)}$	
14. End	
15. End	
16. $(\pi_{m+1}, d_{m+1}) \leftarrow \min_{ABP+UBP} (p_i, d_i)$	// (p_i, d_i) is the pair with minimal $ABP(p_i, d_i) + UBP(p_i, d_i)$
17. $Asbl_{m+1} = Asbl_m \cup (p_i, d_i)$	
18. End	

Fig. 8. Procedure for generating the initial solution (sequence).

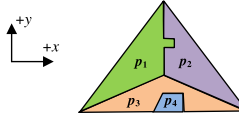
	
$DAIM = \begin{matrix} & \begin{matrix} -x & +x & -y & +y \end{matrix} \\ \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{matrix} & \begin{bmatrix} 0 & 3 & 3 & 1 \\ 3 & 0 & 3 & 1 \\ 2 & 2 & 1 & 2 \\ 2 & 2 & 0 & 3 \end{bmatrix} \end{matrix}, \quad \pi_1 = \begin{cases} p_1 & \text{if } d = -x \\ p_2 & \text{if } d = +x \\ p_4 & \text{if } d = -y \\ p_1 \text{ or } p_2 & \text{if } d = +y \end{cases}$	$AIM = \begin{matrix} & \begin{matrix} p_1 & p_2 & p_3 & p_4 \end{matrix} \\ \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{matrix}$
$ABP^{p_2}(p_2) = p_3, \quad UBP(p_2) = p_3, \quad (ABP + UBP)(p_2) = p_3$	$(p_3, +x)$ is randomly selected from $\{(p_1, -x), (p_1, +y), (p_3, +x), (p_3, +y), (p_4, +x), (p_4, -y)\} \Rightarrow o_2 = (p_2, d_2) = (p_3, +x)$
$ABP(p_2, p_3) = p_4, \quad UBP(p_2, p_3) = p_4, \quad (ABP + UBP)(p_2, p_3) = p_4$	$(p_1, +x)$ is randomly selected from $\{(p_1, +x), (p_4, +x), (p_4, +y)\} \Rightarrow o_3 = (p_3, d_3) = (p_1, +x)$
$ABP(p_1, p_2, p_3) = p_4, \quad UBP(p_1, p_2, p_3) = p_4, \quad (ABP + UBP)(p_1, p_2, p_3) = p_4$	$(p_4, +y)$ is selected $\Rightarrow o_4 = (p_4, d_4) = (p_4, +y)$
Initial solution $s_0 = \langle (p_2, +x), (p_3, +x), (p_1, +x), (p_4, +y) \rangle$ and $O = (o_1, o_2, \dots, o_4)$	

Fig. 9. Generating the initial solution (sequence) for a simple assembly taken from (Wilson and Latombe, 1994).

Table 1

The results of computing statistical measures for 20 ASP scenarios.

Product	Distribution measures								Correlation measures			
	$Dmm(O)$	Δ_{Dmm}	$ent(O)$	Δ_{ent}	$Amp(O)$	Δ_{Amp}	$Gap(O)$	$Lmm(O)$	$\rho(1)$	$\rho(2)$	$\rho(3)$	r
Exchange + Flip (XF) operators												
1	0.849	0.0000	0.247	0.0000	2.43	−0.700	−0.650	12.13	−0.0024	−0.0050	−0.0092	-4.0×10^{-7}
2	0.826	0.0040	0.244	0.0024	1.96	−0.080	−0.570	52.15	0.0015	0.0083	0.0065	-5.0×10^{-5}
3	0.857	0.0100	0.162	0.0088	1.68	0.209	−0.450	90.85	0.0081	−0.0005	0.0018	-5.0×10^{-5}
4	0.880	0.0020	0.165	0.0010	2.06	−0.010	−0.640	80.36	0.0014	0.0008	−0.0088	-1.0×10^{-5}
5	0.916	0.0015	0.163	0.0003	1.18	−0.099	−0.708	439.66	0.0011	−0.0089	0.0008	8.3×10^{-10}
Insertion + Flip (SF) operators												
1	0.847	0.0030	0.246	0.0010	1.60	−0.080	−0.497	94.08	−0.0064	−0.0030	−0.0030	-4.0×10^{-6}
2	0.825	0.0057	0.243	0.0040	1.89	0.027	−0.530	75.03	0.0029	−0.0131	0.0138	-7.0×10^{-5}
3	0.863	0.0048	0.163	0.0040	2.23	0.026	−0.587	56.42	−0.0034	0.0035	0.0007	-5.0×10^{-5}
4	0.879	0.0053	0.164	0.0030	1.42	0.292	−0.479	146.62	−0.0069	−0.0007	−0.0054	-6.0×10^{-6}
5	0.932	0.0008	0.163	0.0001	3.23	−0.225	−0.885	71.04	0.0097	0.0016	0.0062	-1.2×10^{-8}
Inversion + Flip (VF) operators												
1	0.840	0.0080	0.245	0.0040	1.17	0.240	−0.320	183.93	0.0041	−0.0030	−0.0013	-3.8×10^{-4}
2	0.826	0.0042	0.243	0.0043	1.45	0.150	−0.439	99.01	−0.0008	0.0034	−0.0093	-1.5×10^{-5}
3	0.865	0.0025	0.163	0.0020	2.34	−0.060	−0.640	40.49	−0.0025	−0.0041	−0.0059	-2.2×10^{-5}
4	0.883	0.0007	0.165	0.0001	2.25	−0.120	−0.694	52.65	−0.0081	0.0054	−0.0053	-1.0×10^{-5}
5	0.915	0.0166	0.163	0.0003	1.22	−0.066	−0.706	471.90	0.0005	0.0023	0.0032	6.9×10^{-10}
Exchange + Insertion + Inversion + Flip (ALL) operators												
1	0.845	0.0044	0.246	0.0021	1.47	0.003	−0.434	129.57	0.0020	−0.0009	0.0008	1.1×10^{-6}
2	0.829	0.0013	0.244	0.0005	2.25	−0.521	−0.653	26.01	−0.0054	−0.0036	0.0089	-2.6×10^{-5}
3	0.865	0.0018	0.164	0.0013	2.77	−0.101	−0.667	35.87	−0.0039	0.0020	−0.0043	-3.0×10^{-5}
4	0.877	0.0075	0.164	0.0041	1.35	0.462	−0.472	162.90	−0.0039	−0.0117	0.0040	-9.0×10^{-6}
5	0.930	0.0072	0.163	0.0003	2.55	−0.149	−0.832	186.22	−0.0040	−0.0026	0.0001	-1.3×10^{-9}
Average	0.867	0.0046	0.196	0.0022	1.93	−0.039	−0.593	125.35	−0.0008	−0.0015	−0.0003	-3.7×10^{-5}

starting random population U and the final population O is given by (13). Also, the average distance between a population O of local optimal solutions and the best found solution (s^*) is called the 'Gap' of the population, as defined in (14). A small Gap implies that the considered problem is easy to solve.

$$Amp(P) = \frac{|P| \cdot (\max_{s \in P} f(s) - \min_{s \in P} f(s))}{\sum_{s \in P} f(s)} \geq 0, \quad (12)$$

$$\Delta_{Amp} = \frac{Amp(U) - Amp(O)}{Amp(U)} \leq 1, \quad (13)$$

$$-1 \leq Gap(O) = \frac{\sum_{s \in O} (f(s) - f(s^*))}{|O| \cdot f(s^*)} \leq 0. \quad (14)$$

Length of the walks: The average length of the walks of a population P reveals some information about the ruggedness of the landscape. In an unsmooth landscape, the number of local optimal solutions and therefore the length of the walks between an initial random solution and a final local optimum solution are short. On the other hand, in a smooth landscape the length of the walks is longer. The average length of the walks of the population O of local optimal solutions is defined as follows:

$$Lmm(O) = \frac{\sum_{s \in O} l(s)}{|O|}, \quad (15)$$

where $l(s)$ is the number of required walks to achieve solution s from its corresponding initial solution. Indeed, it is the number of times the neighborhood generation operators are applied to the initial solution until its corresponding local optimal solution in the population O is reached. Also, $|O|$ is the cardinality of the population O that is set to 5000 for our analysis.

Autocorrelation function: The autocorrelation function $\rho(d_H)$ measures the correlation of all solution pairs with Hamming distance d_H in the search space, and investigate the effect of the distance between solutions on their fitness variation. Since construction of the

function $\rho(d_H)$ for problems with too many parts is formidable, one way to estimate this function is to create a population of solution pairs with mutual distance d_H and calculate their autocorrelation using (16). The size of this population was set to 10,000 in our analysis. If $|\rho(1)| \approx 1$, then two neighboring solutions have almost equal fitness values, and so the landscape is smooth. For a population of n pairs of solutions with distance d_H from each other, the autocorrelation function is defined as follows:

$$-1 \leq \rho(d_H) = \frac{\sum_{s,t \in S \times S, dist(s,t) = d_H} (f(s) - \bar{f})(f(t) - \bar{f})}{n \cdot \sigma_f^2} \leq 1. \quad (16)$$

Fitness–Distance correlation: The Fitness–Distance Correlation (FDC) analysis measures the correlation of the fitness of a solution and its distance to the global optimal or the best-known solution. To perform this analysis, first a population of n randomly-generated solutions is created and then the set $F = \{f_1, f_2, \dots, f_n\}$ containing the fitness value of solutions, and the set $D = \{d_{H1}, d_{H2}, \dots, d_{Hn}\}$ containing the distance of the solutions to the best-known solution are computed. Afterwards, the fitness-distance correlation coefficient r is computed as follows:

$$-1 \leq r = \frac{\frac{1}{n} \sum_{i=1}^n (f_i - \bar{f})(d_{Hi} - \bar{d}_H)}{\sigma_f \cdot \sigma_{d_H}} \leq 1, \quad (17)$$

where σ_f and σ_{d_H} are the standard deviation of the fitness and distance measures, and the numerator of (17) is the covariance between the sets F and D . If $r \approx 1$, then the direct correlation between the fitness of the solutions and their distance to the best-known solution is strong, and so the problem is easy to solve. On the contrary, $r \approx -1$ means that the problem is 'misleading', meaning that the move operator will guide the search away from the global optimum. By knowing this information, solving the problem will be not hard. The hardest situation arises when $|r| \approx 0$, which indicates that there is no correlation between the fitness of

the solutions and their distance to the best-known solution, and thus no useful rule for biasing the search in the solution space can be found. A convenient method for visualizing the FDC analysis is through using fitness–distance plot, which depicts the fitness of the solutions versus their distance to the global optimal solution. We have also used this tool for our analysis in the next section.

2.5. Landscape analysis results

Since the fitness landscape of a problem depends on the structure of the problem instance and the method that neighboring solutions are constructed, we implemented four neighborhood construction schemes (operators) on five different ASP instances (products) (as illustrated in Fig. 4) in order to reach a reliable conclusion about the shape of the general ASP problem's fitness landscape. For this purpose, $5 \times 4 = 20$ scenarios with different combinations of products and neighborhood operators were designed and the hill climbing local search algorithm was applied 5000 times for each scenario to obtain local optimal solutions (in total, 80,000 runs). Then, the statistical measures introduced above were calculated for each scenario, the results of which are reported in Table 1. In the following subsections we provide our critical analyses on the ASP landscape.

2.5.1. Distribution measures

In order to evaluate the distribution of the local optimal solutions obtained from the hill climbing approach across the landscape, the variation of the local optima distribution in the search space (Δ_{Dmm}) and the variation of entropy in the search space (Δ_{ent}) play a key role. The following combinations of these two measures reveal some facts about the scatter of local optima across the search space (Talbi, 2009):

- (1) If both Δ_{Dmm} and Δ_{ent} are low, then the search space is called 'Uniform', meaning that local optima are scattered in the search space uniformly and so applying a local search to a random initial solution would rapidly direct us toward a nearby local optimum. It has been showed that a low entropy is a low distribution, but not vice versa (Niederreiter, 1992).
- (2) If both Δ_{Dmm} and Δ_{ent} are high, then the search space is called 'Massif Central' (or 'One-Massif'), meaning that most of the local optima are concentrated in a small, dense region. Thus, S-metaheuristic algorithms will act better than P-metaheuristics in solving such search spaces considering their more exploitation-oriented nature.
- (3) If Δ_{Dmm} is high and Δ_{ent} is low, then the search space is called 'Multi-massif', meaning that local optima are concentrated in several attraction regions. Therefore, P-metaheuristics are better choices as they cover various solution clusters scattered across the search space.

The last row of the Table 1 provides the average values of the used statistical measures in all scenarios, and thus can reflect the landscape properties of the general ASP problem. It is observed that $\Delta_{Dmm} = 0.0046$ and $\Delta_{ent} = 0.0022$, which are small values and thus according to the case (1) above, imply that the local optima are scattered over the search space uniformly. On the other hand, Δ_{Amp} is a measure in the *objective space* and does not concern distribution of local optima, and its relatively small value (-0.039) shows that the landscape is almost plain in a $2n+1$ -dimensional space and the quality of local optima is not much better than initial random solutions. Note that a plain landscape may be flat or rugged (non-smooth) based on values of the correlation measures. On the other hand, the large average gap of the obtained local optimal solutions with global solutions (about 59.3% worse)

indicates that finding global optimal solutions in the ASP problem is very difficult and the basic local search method is not able to solve the problem effectively.

2.5.2. Correlation measures

Correlation measures estimate the ruggedness of the landscape together with the correlation between the quality of solutions and their distance to a global optimal solution.

$Lmm(P)$ shows the average length of the walks, and the shorter $Lmm(P)$ is, the more rugged the space will be, since in a rugged space the number of local optima is large and we encounter a local optimum within a few search iteration. However, as Table 1 indicates, the value of $Lmm(P)$ depends on the type of neighborhood generation operator, and in the studied scenarios, the exchange+flip operators produced shorter walks compared to other operators, which indicates their faster convergence to a local optimal solution and more rugged landscape. Therefore, using only this combination of operators is not recommended. For the autocorrelation measure, we calculated it for three Hamming distances of 1–3. The results showed that on average, $\rho(1)$, $\rho(2)$, and $\rho(3)$ are very small and almost equal, indicating that the variation of fitnesses between 1-neighbors, 2-neighbors, and 3-neighbors are equal on average to the variation between any two solutions, and thus the landscape is rugged. Such a weak correlation between neighboring solutions confirms that the ASP problem is hard.

The fitness–distance correlation value (r) for all instances was very small, which shows there is no correlation between the fitness of solutions and their distance to the best-known solution. Therefore, achieving the global optimal solution is very difficult and needs an effective search algorithm. For the puzzle-like assembly product (Fig. 4(a)) and for each neighborhood generating operator, we also plotted the fitness of 5000 local optimal solutions versus their distance to the best-known solution (the fitness–distance plots), as shown in Fig. 5. Solid (red) dots indicate feasible, and hollow (blue) circles show infeasible solutions. As stated earlier, the maximum Hamming distance between any two solutions of an n -part assembly is $2n$, and so for the puzzle-like product, the values on the abscissa in FD plots will be at most 38.

By analyzing the F–D plots in the Fig. 5 it can be concluded that:

1. All local optimum solutions have distances to the best-known solution in the interval [18, 38] and none of them has a distance less than 18. This fact confirms that the fitness–distance correlation (r) is low, and solutions with short distances to the best known solution do not necessarily have good fitnesses, and therefore the problem is difficult to solve.
2. At many fitness–distance points, feasible (red) and infeasible (blue) solutions overlap, which shows extracting some rules for discriminating feasible solutions among infeasible ones is difficult or impossible.
3. The number of feasible solutions for the XF, SF, VF, and ALL operators were 29, 26, 12 and 55 respectively.

Therefore, rather than just using each one of the Exchange, Insertion, and Inversion operators together with the Flip operator, the combination of the ALL operators is a better choice for neighborhood generation of the ASP problem. Based on the above analysis and for the ALL operators, the F–D plots of the benchmark assembly products in Fig. 4 are illustrated in Fig. 6. It is observed that there is no correlation between the distance to the best known solution and the fitness value of local optimal solutions. Also, the number of found feasible solutions were 92, 14, 0, and 0 for the Industrial assembly, electrical relay, Generator and tank, respectively, which exposes that finding feasible solutions for the

generator assembly is extremely difficult due to the very limited number of feasible assembly sequences.

3. Break-out local search (BLS) for the ASP problem

In the Section 2.3.2 we mentioned that the goal of the ASP problem is to find the best sequence of assembling the parts of a product while minimizing the number of assembly direction changes and maximizing the total number of satisfied geometrical constraints. On the other hand, analysis of the problem's landscape showed that it is a rugged plain with uniformly distributed local optima, and so a single-solution-based metaheuristic algorithm may work well. But since the number of feasible solutions for certain ASP instances can be as few as 0.4% of all solutions, an S-metaheuristic with effective exploration (perturbation) capabilities is needed. Such requirements properly match with the qualities of the Breakout Local Search (BLS) algorithm developed recently in (Benlic and Hao, 2013a) for solving the maximum clique problem. BLS has also been successfully applied to solve other combinatorial optimization problems like minimum sum coloring (Benlic and Hao, 2011), quadratic assignment (Benlic and Hao, 2013c), maximum cut (Benlic and Hao, 2013b), and the vertex separator problem (VSP) (Benlic and Hao, 2013d). Encouraged by the fine results of the BLS in those domains and considering its powerful exploitation and exploration capabilities, we selected it for solving the ASP problem. The general framework of the BLS algorithm and its components are detailed in the following subsections.

3.1. General framework

BLS can be considered as an Iterated Local Search (ILS) algorithm that uses information of the search history for perturbing its solutions. The basic idea behind BLS is to use local search in each iteration to discover a local optimum and then employ adaptive diversification strategies to move from the current local optimum to another in the search space. Thus, exploration of new search areas is achieved by continually and adaptively applying weak to strong perturbations, depending on the search state. BLS starts from an initial solution s_0 and applies a hill-climbing local search to reach a local optimum s^* . At each iteration of the local search algorithm, the current solution is replaced by a randomly generated neighboring solution that improves the objective function. A local optimum is said to be reached if after L_{max} attempts no improving neighbor was found. At this point, BLS tries to escape ('break out') from the current local optimum and move to a neighboring solution perturbed by a number of dedicated moves to the current optimum s^* . Each time a local optimum is perturbed, the new solution is used as the starting point for the next round of the local search procedure. The whole procedure iterates $maxiter$ times. Details of the BLS method are presented in Fig. 7.

As indicated in Fig. 7, the BLS algorithm requires the following five subroutines to be executed:

- Initial_Solution(): For generating a starting sequence (point) for the search.
- Local_Search(): A hill-climbing local search procedure for exploiting a part of the neighborhood of a solution in order to improve the current solution, and terminates if a better solution is not found.
- Jump_Magnitude(): Determines the number M of perturbation moves (or jump magnitude).

- Perturbation_Type(): Determines the type of perturbation moves (PT) among alternative neighborhood generation operators (exchange, insertion, inversion, and flip).
- Perturb(): Applies the selected perturbation PT operators M times to the current local optimum. The perturbed solution becomes the new starting point for the next round of the Local_Search().

Details of the above procedures are provided below.

3.2. Initial solution generation

Although an initial solution for the BLS algorithm can be constructed by creating a random permutation of parts associated with random assembly directions, we propose a novel greedy heuristic algorithm for generating an initial solution which prioritizes assembly operations according to the number of satisfied geometric constraints before and after adding a particular part to the current subassembly. In this way, the probability of creating a 'more feasible' initial solution increases significantly. Details of the proposed initial solution generation algorithm are as follows:

At first, the *Directional Assembly Interference Matrix* (DAIM) is constructed using the AIM, as below:

$$DAIM(p_i, d_i) = \sum_{j=1}^n I_{p_j, p_i}^{AV(p_i, d_i)}; \quad \forall i = 1, \dots, n; \quad d_i \in \{-x, +x, -y, +y, -z, +z\}. \quad (18)$$

Then, among all directions, a random direction d_1 is selected for assembling the first part of the sequence, and based on that, the first part to be assembled (π_1) is selected to be the one that causes minimal violation of the precedence relations for assembling other parts along the direction d_1 , as follows:

$$\pi_1 = \{p_i \mid DAIM(p_i, d_1) \leq DAIM(p_k, d_1); \quad k = 1, \dots, n\}. \quad (19)$$

Assume that a subassembly with m parts (together with their assembly directions) is defined as $Asbl_m = \{(\pi_1, d_1), (\pi_2, d_2), \dots, (\pi_m, d_m)\}$ and we intend to select the next part of the assembly, i.e., (π_{m+1}, d_{m+1}) . For this purpose, we define two more matrices: (1) 'Assembled Blocking Parts (ABP)', for recording the number of violated precedence relations if p_i is added to $Asbl_m$ for each direction, and (2) 'Unassembled Blocked Parts (UBP)', for recording the number of violated precedence relations when all unassembled parts $p_j \notin \{\pi_1, \dots, \pi_m\}$ are added to the subassembly after p_i for each direction. Mathematically, each element of these matrices equals to:

$$ABP(p_i, d_i) = \sum_{j=1}^m I_{p_i, \pi_j}^{AV(p_i, d_i)}, \quad \forall i = 1, \dots, n; \quad d_i \in \{-x, +x, -y, +y, -z, +z\},$$

$$UBP(p_i, d_i) = \sum_{p_j \notin \{\pi_1, \dots, \pi_m\}} I_{p_j, p_i}^{AV(p_j, d_i)}, \quad \forall i = 1, \dots, n;$$

$$d_i \in \{-x, +x, -y, +y, -z, +z\}. \quad (20)$$

Now, the next assembly part and direction pair (π_{m+1}, d_{m+1}) is the one for which the sum of the numbers of violated precedence relations for already assembled and still unassembled parts is minimum:

$$(\pi_{m+1}, d_{m+1}) = \{(p_i, d_i) \mid ABP(p_i, d_i) + UBP(p_i, d_i) \leq ABP(p_k, d_k) + UBP(p_k, d_k); \quad p_k \notin \{\pi_1, \dots, \pi_m\}\}. \quad (21)$$

When two or more parts have equal $ABP + UBP$ sums, the part is selected that has smaller ABP value. The aforementioned routine is continued until all parts of the product are added to the assembly sequence. The pseudocode of this algorithm (Algorithm 2) is presented in Fig. 8, and a representative example for a simple product with four parts is shown in Fig. 9.

Algorithm 3: Local_Search (s, L_{max})

Output: s^* = A local optimum solution

```

1.  $ni = 0$  // Initialize counter  $ni$  of enumerating non-improving iterations
2. While  $ni < L_{max}$ 
3.   Generate a random  $s' \in N(s)$  // Generate a neighbor by randomly applying the Insertion or Flip operators
4.   If  $f(s') > f(s)$  Then // Calculated in (7)
5.      $s \leftarrow s'$  // Replace  $s$  with the better neighbor  $s'$ 
6.      $ni \leftarrow 0$  // Reset  $ni$ 
7.   Else
8.      $ni \leftarrow ni + 1$ 
9.   End
10. End
11.  $s^* \leftarrow s$  // Record  $s^*$  as the local optimum

```

Fig. 10. Procedure of the hill-climbing local search performed in each iteration of the main BLS algorithm.

Algorithm 4: Jump_Magnitude ($M, M_{min}, M_{max}, N_{max}, s_{lo}, s^*, \omega$)

Output: M = The Jump Magnitude

```

1. If  $\omega > N_{max}$  then // Search seems to be stagnating
2.    $M \leftarrow M_{max}$  // Strong perturbation applied
3. Else if  $\omega \leq N_{max}$  AND  $s^* = s_{lo}$  then // Search returned to the previous local optimum
4.    $M \leftarrow M + 1$  // Increment the jump magnitude
5. Else // Search escaped from the previous local optimum
6.    $M \leftarrow M_{min}$  // Reinitialize the jump magnitude
7. End

```

Fig. 11. Procedure of determining the magnitude of jumping from the current local optimum.

Algorithm 5: Perturbation_Type (ω, N_{max})

Output: PT = Perturbation Type

```

1. If  $\omega / N_{max} < 0.25$  then
2.    $PT \leftarrow \text{Flip}$  // The Flip operator is selected to mildly perturb the solution
3. Else if  $\omega / N_{max} < 0.50$  then
4.    $PT \leftarrow \text{Exchange}$  // The Exchange operator is selected to moderately perturb the solution
5. Else if  $\omega / N_{max} < 0.75$  then
6.    $PT \leftarrow \text{Insertion}$  // The Insertion operator is selected to strongly perturb the solution
7. Else
8.    $PT \leftarrow \text{Inversion}$  // The Inversion operator is selected to radically perturb the solution
9. End
10. If  $\omega / N_{max} > 1$  then // The best solution not improved after a certain number of visited local optima
11.    $\omega \leftarrow 0$  // Reset  $\omega$ 
12. End

```

Fig. 12. Procedure of determining the perturbation type (neighborhood generation operator) of a solution.

Table 2

Best values of the parameters of the proposed BLS algorithm.

Parameter	Description	Value
$maxiter$	Maximum total number of iterations of the BLS algorithm	100
N_{max}	Maximum number of BLS iterations with non-improving local optima	$0.4n$
L_{max}	Maximum number of iterations of the hill climbing local search	50
M_{min}	Minimum number of perturbation moves (jump magnitude)	$0.2n$
M_{max}	Maximum number of perturbation moves (jump magnitude)	$0.8n$
w_1	Weight of the Assembly Direction Changes (ADC) in the objective function (7)	1
w_2	Weight of the Satisfied Geometric Constraints (SGC) in the objective function (7)	9
p_F	Probability of selecting the flip operator for neighborhood generation	0.3
p_S	Probability of selecting the Insertion operator for neighborhood generation	0.7

3.3. Local search and neighborhood generation

The BLS algorithm utilizes a simple hill-climbing local search procedure to move towards the final solution (lines 6 and 14 of Algorithm 1). In each iteration of the local search, a neighborhood generation operator (either Insertion or Flip) is applied to the current solution s in order to obtain a new solution, which replaces the current solution if it

improves the objective function. In each iteration, if the current solution is not improved after L_{max} attempts, then it is deemed a 'local optimum'. Selection between Insertion and Flip neighborhood generation operators is based on the roulette wheel selection strategy, which opts for the Flip operator with a probability of p_F ($=0.3$) and the Insertion operator with a probability of p_S ($=0.7$). Pseudocode of the implemented local search procedure (Algorithm 3) is shown in Fig. 10.

Table 3
Parameters of the implemented SA, GA, MA, IPSO, HS, MLS, and ILS algorithms.

Algorithm	Parameter description	Value
SA	θ_0 : Initial temperature	2
	Neighborhood generation approach	Swap+Exchange operators
	$r(k)$: Number of non-improving solutions at a particular iteration k	$\lceil 2k/n(n-1) \rceil$
	Cooling schedule	$\theta_k = \frac{\theta_0}{1 + \ln(r(k))}$
GA, MA	PS : Population size	100
	w_c : Crossover probability	0.8
	Crossover type	Partially Matched Crossover (PMX)
	w_m : Mutation probability	0.1
IPSO	Mutation type	Exchange operator
	c_2 : Acceleration coefficient in the velocity equation of the PSO algorithm	2
	M : Number of particles in the population	1000
	L : Number of immune regulating particles	50
HS	p_c : Crossover probability	0.6
	HMS : Harmony memory size	5
	$HMCr$: Harmony memory considering rate	0.95
	PAR : Pitch adjusting rate	0.5
MLS	BW : Bound width	0.1
	Neighborhood generation approach	Swap+Inversion operators
	New solution acceptance criterion	Only if better
ILS	Neighborhood generation approach	Swap+Exchange operators
	Perturbation approach	Inversion operator
	New solution acceptance criterion	Only if better

Table 4
Comparison of the results of GA, MA, IPSO, HS and BLS methods for the industrial assembly in Fig. 4(b).

Algorithm	The best found solution (s^*) after 30 iterations	Average of all generated solutions				
	Sequence	$f(s^*)$	$\sum_{i=2}^n ADC(\pi_i)$	\bar{f}	$n - \sum_{i=2}^n ADC(\pi_i)$	$\sum_{i=1}^n SGC(\pi_i)$
GA (Gao et al., 2010)	$\langle (1, +x), (2, +x), (3, +x), (4, +x), (7, -x), (9, -x), (8, -x), (10, -x), (5, +x), (6, +x), (11, -x) \rangle$	98.0	3	72.99	6.63	7.34
MA (Gao et al., 2010)	$\langle (1, +x), (2, +x), (3, +x), (4, +x), (5, +x), (6, +x), (7, -x), (10, -x), (9, -x), (8, -x), (11, -x) \rangle$	100.0	1	68.86	8.95	6.66
IPSO (Zhang et al., 2013)	$\langle (1, +x), (2, +x), (3, +x), (4, +x), (5, +x), (6, +x), (7, -x), (9, -x), (8, -x), (10, -x), (11, -x) \rangle$	100.0	1	95.52	8.76	9.64
HS (Wang et al., 2013)	$\langle (6, -x), (5, -x), (4, -x), (2, -x), (3, -x), (1, -x), (7, -x), (10, -x), (9, -x), (8, -x), (11, -x) \rangle$	110.0	0	75.61	9.01	7.33
BLS (proposed)	$\langle (11, +x), (8, +x), (9, +x), (7, +x), (10, +x), (1, +x), (3, +x), (2, +x), (4, +x), (5, +x), (6, +x) \rangle$	110.0	0	97.19	7.07	9.94

3.4. Jump magnitude determination

As the basic hill climbing local search algorithm gets stuck to the first found local optimum (which may be very inferior to the global optimum), the BLS method implements a perturbation mechanism to ‘jump’ to new, hopefully better, local optima. At each main iteration of the BLS algorithm, a local optimum (s^*) is found by the Local_Search() procedure and then perturbed weakly or strongly, depending on the state of search. The magnitude (weakness or strength) of perturbation (M) is expressed as the number of times a neighborhood generation operator is consecutively applied to the solution s^* , which fluctuates between M_{min} and M_{max} . Determining the Jump Magnitude is realized through a simple algorithm presented in Fig. 11, in which the counter ω for enumerating consecutive non-improving local optima plays a decisive role. Since in the early iterations of the search, $\omega < N_{max}$ (where N_{max} is the maximum allowed number of consecutive non-improving local optima), BLS performs a weak perturbation by selecting a small jump magnitude, i.e., $M=M_{min}$, which is hopefully just strong enough to escape the current local optimum s^* and fall under the influence of a neighboring local optimum. If the jump was not sufficient to escape s^* (i.e., the search returns to

the last local optimum found, or $s^*=s_{lo}$), BLS perturbs s^* more strongly by setting $M=M+1$. After visiting local optima for N_{max} time without improving the best solution found so far BLS applies the strongest perturbation ($M=M_{max}$) in order to drive the search categorically toward a new and distant region in the search space. After deciding the jump magnitude, the types of perturbations must be determined, as described next.

3.5. Perturbation type determination

The Breakout Local Search method employs the Flip, Exchange, Insertion, and Inversion operators (introduced in Section 2.2) to perturb solutions and explore new regions of the search space. The selection among different types of perturbation operators is done based on the ratio ω/N_{max} : the larger the ratio, the stronger the perturbation should be. Generally, at the beginning of the search, $\omega/N_{max} < 0.25$, and so a mild perturbation seems sufficient for escaping the current local optimum, which is realized by applying a Flip operator for just changing assembly directions. When $0.25 \leq \omega/N_{max} < 0.50$, the Exchange operator is selected as the Perturbation Type (PT) to moderately perturb the solution. For when $0.50 \leq \omega/N_{max} < 0.75$ and $\omega/N_{max} \geq 0.75$ the Insertion and

Inversion operators are selected to bring about strong and radical perturbations, respectively. Once a search stagnation is detected, that is, the best found solution has not been improved after N_{max} visits of local optima (i.e., $\omega/N_{max} > 1$), BLS applies the Inversion operator to break out towards distant regions of the search space and resets the ω . Fig. 12 shows pseudocode of the Perturbation_Type procedure.

Subsequently, through the Perturb procedure, the selected perturbation operator is applied M (jump magnitude) times to yield the perturbed solution s_p . Afterwards, the Local_Search procedure is run starting from s_p until a new local optimum s^* is reached and compared with the best solution found so far (s_{best}). In case of an improvement in the objective function, s_{best} is replaced with s^* ; otherwise, ω is incremented and the next BLS iteration starts.

The parameters of the proposed BLS algorithm were determined by extensive experiments with different sets of values, and their best values are presented in Table 2.

4. Experimental results

In this section, the results of implementing the BLS algorithm for assembly sequence planning are presented. The program was coded in MATLAB and run on a PC with an Intel Pentium® IV 2.8 GHz CPU and 2.45 GB of RAM. The performance of the BLS algorithm was compared to that of various algorithms existing in the ASP literature, including Simulated Annealing (SA) (Hong and Cho, 1999), Genetic Algorithms (GA) and Memetic Algorithms (MA) (Gao et al., 2010), Immune System-Particle Swarm Optimization hybrid method (IPSO) (Zhang et al., 2013), and Harmony Search (HS) (Wang et al., 2013).

We also implemented Multi-start Local Search (MLS) and Iterative Local Search (ILS) methods to solve the ASP problem for the purpose of evaluating the effectiveness of the perturbation mechanism of the BLS. Both MLS and ILS algorithms used the same objective function, hill climbing local search (with Insertion and Flip operators), and initial solution and neighborhood generation methods, as in the BLS method. However, instead of employing the perturbation mechanism in BLS, the MLS simply starts from a random solution in each iteration, and the ILS iterates by conducting a new local search from a solution that is a perturbation of the last local optimum via the Inversion operator. Parameters of the implemented algorithms, as originally proposed by their developers (except for the MLS and ILS algorithms that were tuned by us), are listed in Table 3.

The following performance metrics were considered for comparing the aforementioned algorithms:

- $f(s^*)$: Fitness value of the best found solution (larger is better);
- $\sum_{i=2}^n ADC(\pi_i)$: Number of reorientations of the best found solution (smaller is better);
- $\bar{f}(s^*) = \sum_{i=1}^{30} f(s_i^*)/30$: Average of best fitness values in 30 runs of the algorithm, in which $f(s_i^*)$ is the best found fitness value in the i -th run of the algorithm (larger is better);
- SD : Standard deviation of best fitness values in 30 runs of the algorithm (smaller is better);
- SR : Percent of feasible solutions (i.e., success rate) in 30 runs of the algorithm (larger is better). This metric is important since according to our landscape analyses, usually more than half of the existing solutions in the search space of the

Table 5

The results of solving the Products 1 and 2 ASP problems by the BLS, SA, GA, MA, MLS, and ILS methods.

Algorithm	The best found solution in 100 iterations			On all 30 runs				Average of all generated solutions			
	Sequence, s^*	$f(s^*)$	$\sum_{i=2}^n ADC(\pi_i)$	$\bar{f}(s^*)$	STD	SR (%)	\overline{FFE}	\bar{f}	$n - \sum_{i=2}^n ADC(\pi_i)$	$\sum_{i=1}^n SGC(\pi_i)$	
Product 1 (puzzle-like assembly)											
BLS	$\langle (8, -x), (18, -x), (9, -x), (19, -x), (10, -x), (16, -y), (15, -y), (17, -y), (14, -y), (7, +x), (5, +x), (13, +x), (2, +x), (12, +y), (4, +y), (11, +y), (6, +y), (1, +y), (3, +y) \rangle$	187	3	184.2	1.93	90	11103.5	173.18	13.46	17.81	
SA	$\langle (16, +y), (9, +y), (15, +y), (13, +y), (8, +y), (4, +y), (18, +y), (12, -x), (19, -x), (5, -x), (10, -x), (17, -y), (2, -y), (14, -y), (7, +x), (11, +y), (3, +y), (1, +y), (6, +y) \rangle$	186	4	184.0	0.67	80	52296.5	167.96	7.48	17.74	
GA	$\langle (14, -y), (7, +x), (17, +y), (10, +y), (12, +y), (5, +y), (13, +y), (11, +y), (6, +y), (2, +y), (16, +y), (15, +y), (8, +y), (4, +y), (3, +y), (1, +y), (18, -x), (19, -x), (9, -x) \rangle$	178	3	172.4	4.55	0	9101.0	161.10	8.79	16.56	
MA	$\langle (8, -x), (9, -x), (10, -x), (18, -x), (13, -x), (5, -x), (11, -x), (19, -x), (2, -x), (16, -y), (15, -y), (17, -y), (14, -y), (4, +x), (7, +x), (1, +x), (12, +y), (3, +y), (6, +y) \rangle$	187	3	178.3	4.85	20	49101.0	169.33	8.66	17.14	
MLS	$\langle (16, +y), (15, +y), (8, +y), (9, +y), (5, +y), (10, -x), (18, -y), (17, -y), (14, -y), (13, -y), (12, -x), (19, -x), (11, +y), (4, +x), (2, +x), (7, +x), (1, +x), (6, +y), (3, +y) \rangle$	184	6	176.7	3.40	30	28599.4	144.99	8.26	14.79	
ILS	$\langle (14, +x), (2, +x), (11, +y), (17, +y), (10, +y), (16, +y), (5, -x), (18, -x), (12, -x), (9, +x), (13, +x), (7, +x), (4, +y), (6, +y), (15, +y), (8, +y), (3, +y), (1, +y), (19, -x) \rangle$	185	5	181.2	3.43	70	26636.8	160.49	10.96	16.46	
Product 2 (Industrial assembly)											
BLS	$\langle (6, -x), (5, -x), (4, -x), (2, -x), (3, -x), (1, -x), (10, -x), (7, -x), (9, -x), (8, -x), (11, -x) \rangle$	110	0	109.7	0.48	100	5894.6	101.11	9.23	10.23	
SA	$\langle (1, -x), (7, -x), (9, -x), (8, -x), (10, -x), (11, -x), (3, +x), (2, +x), (4, +x), (5, +x), (6, +x) \rangle$	109	1	108.8	0.63	100	22236.7	103.98	7.47	10.72	
GA	$\langle (4, -x), (2, -x), (3, -x), (1, -x), (7, -x), (10, -x), (9, -x), (8, -x), (11, -x), (5, +x), (6, +x) \rangle$	109	1	102.6	10.64	70	9101.0	93.78	8.19	9.51	
MA	$\langle (7, -x), (9, -x), (8, -x), (11, -x), (10, +x), (1, +x), (3, +x), (2, +x), (4, +x), (5, +x), (6, +x) \rangle$	109	1	109.0	0.00	100	49101.0	101.76	9.66	10.21	
MLS	$\langle (6, -x), (5, -x), (4, -x), (2, -x), (3, -x), (1, -x), (7, -x), (9, -x), (10, -x), (8, -x), (11, -x) \rangle$	110	0	93.2	33.32	80	23366.4	71.22	7.99	7.03	
ILS	$\langle (11, +x), (8, +x), (9, +x), (7, +x), (10, +x), (1, +x), (3, +x), (2, +x), (4, +x), (5, +x), (6, +x) \rangle$	110	0	108.8	0.42	100	21091.4	90.47	9.06	9.05	

Table 6
The results of solving the Products 3 and 4 ASP problems by the BLS, SA, GA, MA, MLS, and ILS methods.

Algorithm	The best found solution in 100 iterations			On all 30 runs				Average of all generated solutions		
	Sequence, s^*	$f(s^*)$	$\sum_{i=2}^n ADC(\pi_i)$	$\bar{f}(s^*)$	STD	SR (%)	FFE	\bar{f}	$n - \sum_{i=2}^n ADC(\pi_i)$	$\sum_{i=1}^n SGC(\pi_i)$
Product 3 (Electrical relay)										
BLS	$\langle(6, -z), (1, -z), (2, -z), (3, -z), (10, -y), (5, -y), (9, +y), (4, -x), (8, -x), (7, +x)\rangle$	96	4	94.1	2.72	80	10243.9	84.17	6.04	8.68
SA	$\langle(6, -z), (1, -z), (3, -z), (2, -z), (5, -z), (9, +y), (10, -y), (4, -x), (8, -x), (7, +x)\rangle$	96	4	93.2	3.13	40	23888.2	85.35	4.04	9.04
GA	$\langle(6, +y), (3, +y), (9, +y), (10, -z), (2, -z), (5, -z), (1, -z), (4, -z), (7, +x), (8, -x)\rangle$	88	3	81.3	3.56	0	9101.0	75.82	6.71	7.68
MA	$\langle(6, -y), (2, -y), (3, -y), (5, -y), (10, -y), (4, -y), (9, -x), (8, -x), (1, +x), (7, +x)\rangle$	89	2	88.5	0.53	0	69101.0	84.26	7.08	8.58
MLS	$\langle(6, -z), (2, -z), (9, -z), (3, -z), (5, -z), (10, -z), (4, -z), (7, +x), (1, +x), (8, -x)\rangle$	89	2	87.9	0.74	0	24183.0	63.12	6.34	6.31
ILS	$\langle(6, -z), (1, -z), (3, -z), (2, -z), (9, +y), (5, -y), (10, -y), (4, -x), (8, -x), (7, +x)\rangle$	95	5	91.2	3.29	40	19651.2	75.13	6.58	7.62
Product 4 (Generator)										
BLS	$\langle(7, -x), (9, -x), (10, -x), (11, -x), (12, -x), (13, -x), (6, +x), (5, +x), (4, +x), (3, +x), (2, +x), (1, +x), (14, +x), (15, +x), (8, +y)\rangle$	148	2	146.0	2.83	90	11355.5	124.09	10.46	12.63
SA	$\langle(7, +x), (9, +x), (10, -x), (11, -x), (6, +x), (5, +x), (4, +x), (3, +x), (8, +y), (12, -x), (13, -x), (2, +x), (1, +x), (14, +x), (15, +x)\rangle$	145	5	134.2	6.73	20	30515.8	114.86	6.85	12.00
GA	$\langle(9, -z), (3, -z), (10, -x), (11, -x), (5, +y), (8, +y), (4, +y), (6, +y), (2, +x), (1, +x), (7, +x), (12, -x), (13, -x), (14, -x), (15, -x)\rangle$	119	4	102.8	5.19	0	9101.0	93.53	9.54	9.33
MA	$\langle(6, +y), (3, +y), (4, +z), (9, -x), (10, -x), (11, -x), (7, -x), (12, -x), (13, -x), (5, +x), (2, +x), (1, +x), (14, +x), (15, +x), (8, +y)\rangle$	128	4	112.5	8.45	0	69101.0	104.60	11.83	10.31
MLS	$\langle(13, +y), (12, +y), (11, +z), (9, -x), (6, -x), (5, -x), (3, -x), (8, -x), (10, -x), (2, +x), (1, +x), (14, +x), (15, +x), (7, +x), (4, +y)\rangle$	120	3	111.4	5.56	0	28741.6	83.19	9.50	8.19
ILS	$\langle(7, -x), (10, -x), (11, -x), (9, +x), (6, +x), (12, -x), (13, -x), (5, +x), (4, +x), (3, +x), (2, +x), (1, +x), (14, +x), (15, +x), (8, +y)\rangle$	137	4	121.7	9.25	0	24562.1	92.71	9.53	9.13

- ASP problem are infeasible due to violating precedence relationships between parts of an assembly;
- (vi). FFE : Average number of Fitness Function Evaluations (smaller is better);
 - (vii). \bar{f} : Average of fitness values of all generated solutions in 30 runs of the algorithm (larger is better);
 - (viii). $n - \sum_{i=2}^n ADC(\pi_i)$: Average of the same directions in one assembly sequence in 30 runs of the algorithm (larger is better);
 - (ix). $\sum_{i=1}^n SGC(\pi_i)$: Average number of satisfied geometrical constraints in one assembly sequence in 30 runs of the algorithm (larger is better).

While in some works on ASP, infeasible assembly sequences (those that violate geometrical constraints) are discarded and therefore satisfiability of geometrical constraints do not appear in the objective function, in this article we incorporated the number of satisfied geometrical constraints (SGC) into the objective function. However, due to the large weight given to the SGC ($w_2=9w_1$) in the objective function (7), larger values of the metric (ix) leads to larger \bar{f} .

For comparisons, we solved the industrial assembly problem in Fig. 4(b) by the BLS method and compared its performance metrics with those obtained by the GA and MA (Gao et al., 2010), IP SO (Zhang et al., 2013), and HS (Wang et al., 2013) population-based metaheuristics, as reported in Table 4. The stopping criterion of the algorithms was set to 30 iterations. The global optimal assembly sequence, which is feasible and has no reorientations, was generated after 25 iterations of the BLS method with a fitness value (=110) better than the other algorithms, except for the HS, which was equally fit. The BLS also outperformed the other algorithms in the average number of satisfied geometrical constraints and the average fitness value thanks to the effective methods used for generating the initial solution, neighboring solutions, and perturbed solutions. The results in Table 4 show that considering the landscape shape of the ASP problem, the single-solution-based BLS method was more successful than the population-based methods.

In order to further compare the performance of the algorithms, we solved all the five problems shown in Fig. 4 by our designed

BLS, MLS, ILS, as well as the SA (Hong and Cho, 1999), GA and MA (Gao et al., 2010) methods, the results of which are presented in Tables 5–7. Here the stopping criterion of the algorithms was set to 100 iterations. The results in the above Tables reveal the following facts:

1. The BLS outperformed the other methods in all five benchmark problems in terms of best fitness value $f(s^*)$, the average of the best fitness value $\bar{f}(s^*)$, and the success rate. This fact shows the reliability of this algorithm in optimality and feasibility aspects.
2. While the average FFE value of the BLS algorithm was in 1.3 times that of the GA (which had almost the least average FFE among others), the BLS dominated GA in all criteria of all solved problems.
3. The MA had the best average same-direction assembly sequences in all (except for the puzzle-like assembly) problems thanks to its guided local search which is based on reducing the number of assembly direction changes by sequencing components whose assembly directions are the same.
4. The Swap+Exchange operators used by the SA algorithm caused more minor perturbations to the solutions and hence premature convergence to a relatively good (but not the best) solution, which led to superior average fitness values of all generated solutions (\bar{f}) and average number of satisfied geometrical constraints $\sum_{i=1}^n SGC(\pi_i)$ for the simpler industrial assembly and electrical relay problems. For problems with more complex interference relations (i.e. the puzzle-like assembly) or larger number of parts (i.e. the generator and tank assemblies), the somewhat premature convergence of the SA algorithm generated solutions with larger gaps from the best known solution, while the BLS algorithm had better performance in the \bar{f} and $\sum_{i=1}^n SGC(\pi_i)$ criteria for these problems.

4.1. Statistical analysis

In order to statistically analyze and verify the significance of the outperformance of the BLS algorithm, two nonparametric statistical tests have been conducted: (1) the “Friedman two-way ANOVA by

Table 7

The results of solving the Product 5 (Mockup of the Soviet medium tank T-34/76) ASP problem by the BLS, SA, GA, MA, MLS, and ILS methods.

Algorithm	The best found solution in 100 iterations		On all 30 runs				Average of all generated solutions			
	Sequence, s^*	$f(s^*)$ $\sum_{i=2}^n ADC(\pi_i)$	$\bar{f}(s^*)$	STD	SR (%)	FFE	\bar{f}	$n - \sum_{i=2}^n ADC(\pi_i)$	$\sum_{i=1}^n SGC(\pi_i)$	
BLS	$\langle (1, -z), (28, -z), (29, -z), (41, -z), (40, -z), (17, +x), (52, +x), (16, +x), (20, +x), (54, +x), (60, +x), (53, +x), (39, +x), (58, +x), (21, +x), (59, +x), (47, +x), (18, -x), (56, -x), (50, -x), (19, -x), (55, -x), (61, -x), (48, -x), (62, -x), (23, -x), (57, -x), (22, -x), (63, -x), (9, -x), (7, -x), (8, +x), (6, +x), (3, -y), (2, -y), (5, +y), (4, +y), (38, +x), (46, +x), (51, -x), (49, -x), (10, -z), (31, -z), (30, -z), (43, -y), (42, -y), (35, +y), (13, -z), (14, -z), (11, -z), (24, -z), (32, -z), (25, -z), (12, +y), (15, +y), (26, -z), (36, -z), (37, -z), (27, -z), (44, -z), (34, -z), (45, -z), (33, -z) \rangle$	617 13	427.1	16.13	10	21639.8	403.37	55.15	38.68	
SA	$\langle (57, -y), (47, +x), (39, +x), (52, +z), (20, -x), (55, -x), (54, -z), (19, -z), (63, -x), (61, -x), (50, +z), (60, +x), (22, -x), (53, +z), (48, -x), (18, -x), (23, -x), (56, -z), (17, -z), (59, +x), (8, +x), (3, -y), (2, -y), (4, +y), (49, -x), (51, -x), (58, +z), (38, +x), (46, +x), (30, -z), (35, +y), (42, -y), (41, -y), (43, -y), (16, -y), (31, -y), (13, -z), (14, -z), (11, -z), (32, -z), (12, +y), (9, +y), (15, +y), (25, -z), (24, -z), (7, +z), (21, -z), (6, -z), (62, -x), (36, -z), (28, -z), (27, -z), (29, -z), (5, -z), (26, -z), (40, -z), (37, -z), (44, -z), (34, -z), (45, -z), (10, -z), (33, -z), (1, -z) \rangle$	459 27	358.3	92.18	0	46763.3	208.59	25.25	20.38	
GA	$\langle (50, -z), (54, -z), (18, +y), (63, +z), (61, +z), (23, +x), (17, +x), (58, +x), (59, +x), (39, +x), (57, +x), (22, -x), (56, -x), (21, -x), (16, -z), (48, -z), (60, +z), (43, -y), (42, -y), (53, +y), (1, +y), (35, +y), (12, +y), (46, +y), (45, +y), (20, +y), (62, +x), (19, +x), (9, +x), (10, -z), (44, -z), (31, -z), (55, -z), (34, -z), (27, -x), (32, -x), (38, -x), (52, +y), (6, +y), (30, +y), (24, +x), (4, -y), (40, -y), (26, +z), (36, +z), (47, +z), (5, +z), (7, +z), (49, +z), (14, +x), (28, +x), (2, -x), (3, -x), (13, -x), (8, -x), (51, -z), (29, -z), (33, -z), (11, -z), (25, -x), (37, -x), (15, +z), (41, +z) \rangle$	232 20	203.2	19.05	0	9101.0	136.72	29.59	11.90	
MA	$\langle (17, +y), (53, +y), (52, +y), (19, +y), (22, +y), (47, -z), (55, -z), (50, -z), (16, -z), (57, -z), (56, -z), (48, -z), (39, -z), (62, +x), (8, +x), (38, +x), (10, -z), (20, -z), (30, -z), (54, -z), (49, -z), (13, -z), (1, -z), (42, -z), (3, -z), (9, -z), (4, -z), (11, -z), (32, -z), (25, -z), (24, -z), (46, -z), (35, -z), (36, -z), (43, -z), (26, -z), (29, -z), (37, -z), (12, -z), (40, -z), (14, -z), (61, -z), (51, -z), (27, -z), (7, -z), (23, -z), (63, -z), (28, -z), (58, -z), (21, -z), (31, -z), (45, -z), (44, -z), (34, -z), (33, -z), (60, -z), (6, -y), (2, -y), (18, -y), (41, -y), (15, -y), (5, -y), (59, -y) \rangle$	329 4	285.8	34.44	0	69101.0	212.28	55.53	17.42	
MLS	$\langle (58, -z), (54, -y), (55, -y), (48, -z), (39, -z), (17, -z), (59, -x), (20, -x), (62, -x), (57, -z), (22, -x), (50, -x), (19, -z), (47, -z), (63, -x), (38, +x), (3, +x), (46, +x), (51, -x), (31, -z), (35, +y), (10, +x), (29, +x), (30, -y), (43, -y), (53, -y), (32, -z), (52, +y), (60, +y), (40, -z), (16, -z), (24, -z), (12, +y), (49, +y), (26, -z), (61, +x), (1, +x), (4, +x), (7, +z), (28, +z), (15, +z), (36, -z), (21, +y), (2, +y), (11, +x), (41, +x), (37, -z), (42, -z), (56, -z), (27, -z), (34, -z), (33, -z), (14, -x), (5, -x), (13, +y), (45, +y), (25, -x), (18, -y), (9, -y), (23, +x), (44, +x), (6, +z), (8, +z) \rangle$	303 30	272.5	17.17	0	66832.1	140.96	38.39	11.40	
ILS	$\langle (55, -x), (57, +x), (50, +x), (48, +x), (56, +x), (17, -z), (23, -z), (52, +z), (47, +z), (53, +z), (39, +x), (22, +x), (63, -x), (61, -x), (58, +x), (62, -x), (54, -z), (41, -z), (28, -z), (16, +x), (59, +x), (40, +x), (8, +y), (5, +y), (31, -z), (30, -z), (32, -z), (18, -x), (19, -x), (20, +z), (13, +z), (11, +z), (60, +y), (15, +y), (3, +y), (7, -y), (2, -y), (25, -z), (46, -z), (1, -z), (14, -z), (24, -z), (38, -z), (27, -z), (35, +y), (21, +y), (9, +y), (37, -z), (45, -z), (44, -z), (26, +y), (51, -z), (12, +x), (10, +x), (49, +y), (29, +y), (42, +y), (36, +y), (6, +y), (4, -z), (34, -z), (33, -z), (43, -z) \rangle$	329 23	314.0	10.66	0	63432.0	190.13	42.86	16.35	

Table 8Ranks and p -values achieved by the Friedman test for all algorithms and benchmark ASP problems.

No.	Criterion	Algorithm						p -value
		BLS	SA	GA	MA	MLS	ILS	
1	$\bar{f}(s^*)$	25.61	20.71	5.12	13.73	10.36	17.47	0.000
2	\bar{f}	24.87	22.42	10.32	21.05	3.6	10.74	0.000
3	$n - \sum_{i=2}^n ADC(\pi_i)$	21.59	3.68	12.71	23.76	11.62	19.64	0.000
4	$\sum_{i=1}^n SGC(\pi_i)$	24.68	24.90	10.62	18.43	3.42	10.95	0.000

ranks” test (Friedman, 1937) for detecting any significant difference between behaviors of all algorithms, and (2) the “Kruskal–Wallis one-way ANOVA by ranks” test (Kruskal and Wallis, 1952) for detecting a significant difference between the behaviors of a pair

of algorithms. Note that parametric tests could not be used here because of their assumptions on independency, normality, and homoscedasticity of the variables (Derrac et al., 2011).

The Friedman two-way ANOVA is a multiple comparisons nonparametric test and a counterpart to the parametric two-way ANOVA used to detect significant differences between the means of data from several groups. In our context, the null hypothesis (H_0) of the Friedman’s test will be: “The six algorithms BLS, SA, GA, MA, MLS, and ILS have equivalent performances in solving the five benchmark ASP problems”, and the Alternative Hypothesis (H_1) claims the opposite of H_0 . An important result of the Friedman test is the relative performance ranks of the algorithms, the most superior one being ranked first (for maximization criteria), and so on. The average relative rank r_j^i of each algorithm j in solving each benchmark problem i for 30 times is determined according to its performance in some selected criteria, and the global rank R_j of

Table 9
Ratios of mean ranks and p -values computed by the Kruskal–Wallis test for comparing the BLS algorithm with the other five algorithms in terms of four performance criteria.

Problem	Measure	Performance criteria																			
		$\bar{f}(s^*)$					\bar{f}					$n - \sum_{i=2}^n ADC(\pi_i)$					$\sum_{i=1}^n SGC(\pi_i)$				
		SA	GA	MA	MLS	ILS	SA	GA	MA	MLS	ILS	SA	GA	MA	MLS	ILS	SA	GA	MA	MLS	ILS
1	Rank	36.3 24.7	44.9 16.1	39.4 21.6	44.2 16.8	41.62 19.38	42.9 18.1	44.9 16.1	39.88 21.12	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	31.6 29.4	44.9 16.1	42.3 18.7	44.9 16.1	44.9 16.1
	p	0.100	0.000	0.017	0.000	0.003	0.001	0.000	0.014	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.762	0.000	0.002	0.000	0.000
2	Rank	40.5 20.5	43.2 17.8	40.6 20.4	39.4 21.6	40.4 20.6	21.2 39.8	41.8 19.2	25.8 35.2	44.9 16.1	44.9 16.1	44.9 16.1	41.2 19.8	23.5 37.5	44.9 16.1	36.3 24.7	16.1 44.9	43.2 17.8	32.3 28.7	44.9 16.1	44.9 16.1
	p	0.005	0.000	0.001	0.012	0.005	0.016	0.003	0.227	0.000	0.000	0.000	0.000	0.005	0.069	0.000	0.100	0.000	0.001	0.623	0.000
3	Rank	33.2 27.8	44.9 16.1	43.5 17.5	44.4 16.6	38.6 22.4	27.3 19.2	44.1 16.1	29.6 31.3	44.9 16.1	44.9 16.1	44.7 16.3	22.9 38.1	17.4 43.6	28.4 32.6	23.9 37.1	24.4 36.8	43.8 19.2	32.8 28.8	44.9 16.1	44.9 16.1
	p	0.455	0.000	0.000	0.000	0.024	0.406	0.000	0.821	0.000	0.000	0.000	0.000	0.049	0.001	0.597	0.088	0.112	0.001	0.545	0.000
4	Rank	44.4 16.6	44.9 16.1	43.5 17.5	44.4 16.6	38.6 22.4	41.8 19.2	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	40.7 20.3	17.4 43.6	44.9 16.1	37.9 23.1	37.8 23.2	44.9 16.1	44.9 16.1	44.9 16.1
	p	0.000	0.000	0.000	0.000	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.001	0.000	0.049	0.000	0.000	0.000	0.000
5	Rank	42.1 18.9	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	29.3 31.7	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1	44.9 16.1
	p	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.762	0.000	0.000	0.000	0.000	0.000	0.000
O_{BLS}		4	5	5	5	5	3	5	3	5	5	5	4	1	4	4	2	5	3	5	5
U_{BLS}		0	0	0	0	0	1	0	0	0	0	0	1	3	0	1	1	0	0	0	0

that algorithm is calculated by computing the average of its relative ranks.

Another output is the p -values related to the similarity between algorithms, which indicate the maximum errors incurred when rejecting the H_0 and accepting the H_1 . For example, a p -value of zero indicates that the algorithms are definitely (with 100% of confidence) different. A commonly used confidence level for the p -value is $\alpha=0.05$; thus, if the calculated p -value of a statistical test is larger than 0.05, we can reject the H_1 regarding the difference of algorithms. Table 6 summarizes the numerical results of performing the Friedman test for the six algorithms and the five benchmark problems. The p -values in the Table 8 show that performances of the algorithms in all criteria are significantly different at a confidence level of 100.0%. Also, values of the four important criteria vary from one problem to another.

Now that there are significant differences between the performances of all algorithms, we conduct a Kruskal–Wallis test to characterize these differences for each pair of the algorithms. The Kruskal–Wallis one-way analysis of variance by ranks test is a pairwise comparisons nonparametric test and an analog of the parametric one-way ANOVA (Kruskal and Wallis, 1952). Here we conducted $5 \times 5 \times 4 = 100$ tests using the best solutions of 30 runs of each algorithm as the test population of the respective method, and with a null hypothesis stating that “The performance of the BLS algorithm is equivalent to the performance of algorithm i in problem j and criterion k ”. If the null hypothesis for a specific problem and a specific criterion is rejected (i.e., when p -value $< \alpha$), then the BLS algorithm stochastically dominates the compared algorithm.

For each pair of BLS and another algorithm, the Kruskal–Wallis test first sorts all the performance values regardless of their generating algorithms and computes the relative rank of each value (in case of a tie their average value is set as their common rank) and assigns to its respective algorithm. The sum of the ranks for each algorithm is calculated and through a formula representing the variance of the ranks among algorithms (with an adjustment for the number of ties), an H statistic is calculated, which approximately has a Chi-square distribution. The p -value is then determined as $\Pr(\chi^2_1 \geq H)$, and if it is smaller than the confidence level α , then the Null hypothesis is rejected. Numerical results of performing the Kruskal–Wallis test for the six algorithms, five benchmark problems, and four performance criteria are reported in Table 9.

In each column of Table 9, since all the four performance criteria are of maximization type, ranks larger than 1 (shown in boldface) indicate that the BLS was better than the compared algorithm and criterion for a specific problem, and $(1-p)$ denote the confidence level of stating that one of the algorithms outperform the other one.

The O_{BLS} and U_{BLS} parameters in the last two rows count the times BLS outperformed and underperformed relative to a competing algorithm in a specific criterion.

It is interesting to observe that most underperformances of the BLS relative to other algorithms occurred in the third criterion ($n - \sum_{i=2}^n ADC(\pi_i)$) of the Table 9. The reason is that in the BLS method, whenever the ratio ω/N_{max} increases, more severe perturbation methods (via Insertion and Inversion operators) are applied with larger magnitudes, as a result of which the number of assembly direction changes (ADC) tends to increase as well. Moreover, due to the small weight of the term $w_1 \sum_{i=2}^n (1 - ADC(\pi_i))$ in the objective function (7) (that is, $w_1=1/9w_2$), the BLS is less sensitive to the assembly direction changes criterion compared to the average number of satisfied geometrical constraints, and so it accepts solutions with large ADCs but small SGCs. That is why BLS generally produces highest percentage of feasible solutions with relatively large number of assembly direction changes.

Further analysis of the test results for each compared algorithm is as follows:

- SA:** The BLS algorithm outperformed the SA in all criteria for products 4 and 5, with at least 99.7% confidence. In the first (puzzle-like) product, while BLS was also significantly better than the SA method in the first three criteria, the same claim cannot be made with a sufficiently high confidence level for the fourth criterion (the average number of satisfied geometrical constraints, $\sum_{i=1}^n SGC(\pi_i)$), due to the high p -value of 0.762. For the product 2, BLS outperformed the SA in the first ($\bar{f}(s^*)$) and third ($n - \sum_{i=2}^n ADC(\pi_i)$) criteria with at least 99.5% confidence, but was inferior to it in the second and fourth criteria because of the fact No. 4 mentioned below Table 4. For the product 3, although the BLS overtook SA in the third criterion, no conclusion can be made about the superiority of either BLS or SA due to p -values larger than the acceptance level.
- GA:** The BLS algorithm outperformed the GA in all criteria for all products with at least 99.3% confidence, except in the third criterion of the third (electrical relay) product, which is due to the fact that all the solutions produced by the GA for this product were infeasible (as indicated by its zero success rate SR % in the Table 6) but with small assembly direction changes.
- MA:** For the product 1, the BLS outperformed the MA method in all four criteria with at least 98.3% confidence. For the products 2 and 3, while the BLS overtook MA in the first criterion (with 99.9% confidence), MA obviously performed better in the third criterion due to the fact No. 3 mentioned below Table 4. On the other hand, since p -values of the second and fourth criteria are

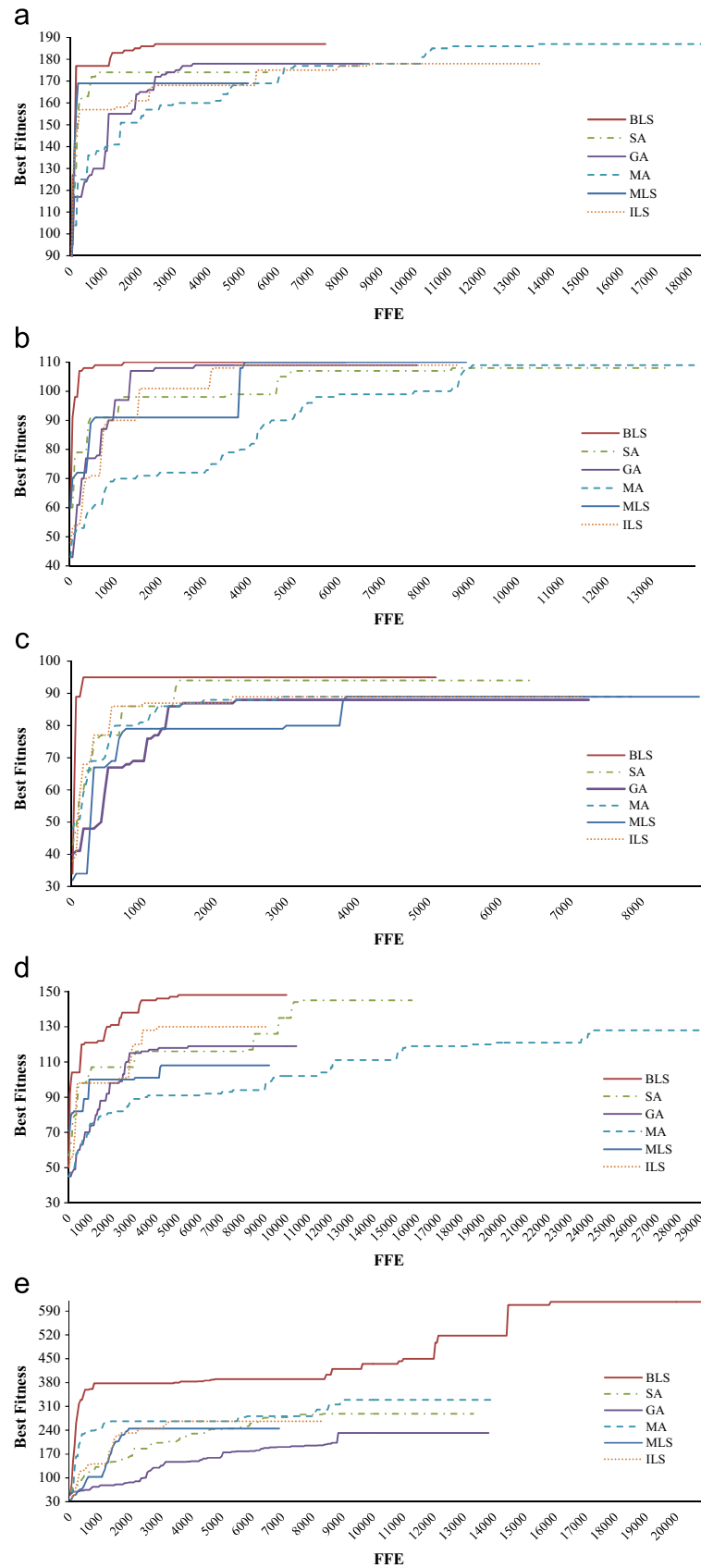


Fig. 13. The convergence curves of the BLS, SA, GA, MA, MLS, and ILS algorithms for the (a) puzzle-like assembly, (b) industrial assembly, (c) electrical relay, (d) generator, and (e) tank mockup.

Table 10FFE_{max} and NCCA results for all compared algorithms and products.

Problem	$f(s^*)$	Criterion	Algorithm					
			BLS	SA	GA	MA	MLS	ILS
1	187	NCCA	0.989	0.922	0.906	0.931	0.902	0.919
		f_{best}	187	174	178	187	169	178
		FFE _{max}	7400	5700	8500	18450	5150	13650
2	110	NCCA	0.999	0.935	0.948	0.852	0.922	0.931
		f_{best}	110	108	109	109	110	109
		FFE _{max}	6150	13500	7750	13950	8850	8650
3	96	NCCA	0.992	0.948	0.872	0.903	0.868	0.910
		f_{best}	95	94	88	89	89	89
		FFE _{max}	5100	6450	7250	7850	8800	7200
4	148	NCCA	0.948	0.845	0.734	0.731	0.698	0.801
		f_{best}	148	145	119	128	108	130
		FFE _{max}	10000	15750	10450	29000	9200	9100
5	617	NCCA	0.772	0.394	0.286	0.469	0.346	0.384
		f_{best}	617	288	232	329	245	266
		FFE _{max}	20800	13300	13800	13900	6900	8300

much larger than the typical cutoff value $\alpha=0.05$, the performance differences between the BLS and the MA methods are not significant in these criteria. For products 4 and 5 the BLS dominated the MA in the first, second, and fourth criteria with over 99.99% confidence, and in the third criterion it lost to the MA for the same reason mentioned in the fact No. 3 below Table 4.

4. **MLS:** The BLS algorithm dominated the MLS in all criteria for all products with at least 98.8% confidence, except in the third criterion of the third (electrical relay) product, in which the underperformance of the BLS still cannot be proved due to the large p -value (0.597).
5. **ILS:** The BLS algorithm outperformed the ILS in all criteria for all products with at least 90.0% confidence, except in the third criterion of the third product (electrical relay). This specific product cannot be assembled feasibly with less than 4 assembly direction changes (which is rather large for a 10-part product), and while the BLS algorithm produced feasible solutions with large ADCs in 80% of its runs, ILS produced less feasible solutions (only in 40% of its runs) with smaller average ADCs with 91.2% confidence for the third product.

The analysis of Kruskal–Wallis tests is concluded with reference to the values of O_{BLS} and U_{BLS} in the last two rows of the Table 9, as follows:

- (1) The developed BLS method proved to be better than the other algorithms in producing highest-quality solutions ($f(s^*)$ and \bar{f} criteria) in all five benchmark problems on average. This confirms the aforementioned fact No. 1.
- (2) Although the BLS method did not outperform the MA and SA algorithms in the average number of same-direction assembly operations and the average number of satisfied geometrical constraints criteria, respectively, it was better with respect to the weighted sum of these criteria, which is equivalent to the objective function (7).
- (3) The BLS was very successful in solving the benchmark problems 1, 4, and 5 (with 19 times outperformance over all algorithms out of 20 tests for each product). On the other hand, in case of products 2 and 3, the BLS could not surpass the MA and SA methods in 3 out of 5 mutual tests for each of the methods. The reason might be that the implemented MA and SA methods were originally developed, tailored, and parameter-tuned for the product 2 (Gao et al., 2010) and product 3 (Hong and Cho, 1999), respectively, which resulted in efficient and effective performance. These two

methods, however, did not produce superior results for the products 4 and 5.

Since the convergence of metaheuristic algorithms to a final solution is an important issue in their design and implementation, we are interested in comparing the six algorithms under study in that respect too. A proper tool is plotting the convergence behavior of each algorithm in solving each benchmark problem as it improves its best-found solution through successive iterations. However, the notion of ‘iteration’ is not the same in single-solution-based and population-based metaheuristics: in the former, an iteration is elapsed when a single solution is replaced by a new, preferably better, solution; while in the latter, an iteration is completed when a bunch of solutions evolve together, creating an updated population of solutions. Therefore, in order to set up a fair basis for comparing our studied S- and P-metaheuristics, we will use the number of Fitness Function Evaluations (FFE) as an indicator of the algorithms’ progress toward their best solutions. While the runtime of an algorithm is not adequately accurate for measuring its computational effort (due to its dependency to many factors of hardware and software), the FFE provides a rather coherent measure of the algorithms’ computational burden. Fig. 13 illustrates the convergence curve of each method (for its best run in 30 experiments) in the form of a plot of its best-found objective function value as the FFE grows. As the stopping criterion, we let each algorithm to evaluate its fitness function for 5000 times after the last improvement of its best-found solution.

In order to analyze the above convergence plots quantitatively, we implement the average Normalized Convergence Curve Area (NCCA) index proposed in (Masehian et al., 2013), which is a measure of an algorithm’s convergence toward its optimal solution. In fact, by calculating the area under a convergence curve we can infer how fast a method improves the objective function: for minimization (resp. maximization) problems, a relatively small (resp. large) area implies that the algorithm succeeded in reducing (resp. increasing) the objective function value at its early iterations. For a given algorithm, the NCCA measures the area under the convergence curve (CCA) of a complete run of the algorithm, which is normalized to lie in the interval (0, 1) as follows:

$$NCCA = \frac{CCA}{f(s^*) \cdot FFE_{max}} = \frac{\sum_{k=1}^{FFE_{max}} f_k}{f(s^*) \cdot FFE_{max}}, \quad (22)$$

in which $f(s^*)$ is the best-known (global best) fitness value and FFE_{max} stands for the maximum number of fitness function

evaluations, which is counted from the beginning of the search until 5000 non-improving solutions are generated. Table 10 presents the NCCA and FFE_{max} of the six compared methods after solving all the five benchmark ASP problems.

The Table shows that in all products the BLS algorithm attained the highest NCCA and f_{best} values among all algorithms. Also, the BLS achieved the best global solution $f(s^*)$ of each problem (except for the problem 3 where it ultimately reached $f_{best}=f(s^*)=96$ after 6750 non-improving fitness function evaluations). The performance gap between the BLS and other algorithms is very notable especially in the problem 5 (the Tank mockup with 63 parts, which is the largest among all products), and is an evidence of the BLS algorithm's efficiency and effectiveness in solving large-size, real-world ASP problems.

5. Conclusions

Assembly Sequence Planning (ASP) is the process of computing a sequence of assembly motions for constituent parts of an assembled final product. The problem is well studied and several works in the literature have been dedicated to its solution, as the problem is proved to be NP-complete. Nevertheless, the fitness landscape of the ASP problem has not been analyzed previously and therefore the selection of solution methods has not been justified fully in the existing research. In this paper, the fitness landscape of the ASP problem is analyzed through several distribution and correlation measures, which showed that the problem's landscape is plain rugged with local optima scattered uniformly in the search space. As a result, it was concluded that a local search method with effective exploitation and exploration capabilities is suitable for solving the ASP problem, and as such, the Breakout Local Search method developed in 2013 was selected for this purpose. The distinctive characteristic of the BLS algorithm is its ability to perturb local optimal solutions in a hope to lead the search toward unexplored areas and attain better local optima.

In this paper, we developed a greedy heuristic procedure for generating a proper initial solution that is tried to be as feasible as possible. Also, different combinations of neighborhood generation operators are proposed and their effectiveness is analyzed. The objective function used for evaluating the quality of an assembly sequence is a weighted sum of the number of same-direction assembly operations and the number of satisfied geometrical constraints in a sequence. By solving five real-world ASP cases, the proposed BLS method was compared to other methods like SA, GA, MA, IPSO, and HS, which have reported results in the literature, as well as to Multi-start and Iterated Local Search methods (MLS and ILS), which we tailored for ASP originally. Computational experiments showed that the BLS outperforms nearly all of the competing methods, especially in the fitness of the best found solution and the average fitness of all solutions criteria, thanks to the careful selection of the initial solution, neighborhood generation operators, perturbation magnitude, perturbation operators, and algorithm parameters.

The performance of the BLS algorithm was then verified by conducting Friedman two-way and Kruskal-Wallis one-way ANOVA tests to find out any significant difference with other algorithms. A detailed analysis of the test results per each product and competing method reconfirmed our findings in the Experimental Results Section. As a result, based on the computational results, statistical tests, and convergence curve analysis, we can confidently maintain that the developed BLS-based ASP solver can help design and manufacturing engineers in finding higher quality assembly sequences more quickly and reliably, which in turn reduces the product's lead time and overall production costs.

While nearly all existing works in the field of ASP deal with merely rigid parts, most real-world assembled products like ships,

aircrafts, and automobiles are composed of rigid and flexible parts, and so automatic generation of assembly sequence plans for such products requires the deformability of flexible parts to be taken into account. Accordingly, our future research addresses assembly sequence planning of products with flexible parts as well, which requires incorporating parameters like elasticity, force, and tolerated geometry of such parts in the model.

References

- Benlic, U., Hao, J.-K., 2011. A multilevel memetic approach for improving graph k-partitions. *Evol. Comput. IEEE Trans.* 15, 624–642.
- Benlic, U., Hao, J.-K., 2013a. Breakout local search for maximum clique problems. *Comput. Oper. Res.* 40, 192–206.
- Benlic, U., Hao, J.-K., 2013b. Breakout local search for the max-cut problem. *Eng. Appl. Artif. Intell.* 26, 1162–1173.
- Benlic, U., Hao, J.-K., 2013c. Breakout local search for the quadratic assignment problem. *Appl. Math. Comput.* 219, 4800–4815.
- Benlic, U., Hao, J.-K., 2013d. Breakout local search for the vertex separator problem. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. AAAI Press, pp. 461–467.
- Bonneville, F., Perrard, C., Henrioud, J.-M., 1995. A genetic algorithm to generate and evaluate assembly plans. In: *Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, ETFA*, pp. 231–239.
- Cao, P.-B., Xiao, R.-B., 2007. Assembly planning using a novel immune approach. *Int. J. Adv. Manuf. Technol.* 31, 770–782.
- Chen, C.P., 1992. Design of a real-time AND/OR assembly scheduler on an optimization neural network. *J. Intell. Manuf.* 3, 251–261.
- Chen, J., Zhang, Y., Liao, H., 2012. Disassembly Sequence Planning Based on Improved Genetic Algorithm. In: Jin, D., Lin, S. (Eds.), *Advances in Multimedia, Software Engineering and Computing*, Vol. 2. Springer, Berlin Heidelberg, pp. 471–476. http://dx.doi.org/10.1007/978-3-642-25986-9_74.
- Chen, W.-C., Tai, P.-H., Deng, W.-J., Hsieh, L.-F., 2008. A three-stage integrated approach for assembly sequence planning using neural networks. *Expert Syst. Appl.* 34, 1777–1786.
- De Fazio, T., Whitney, D.E., 1987. Simplified generation of all mechanical assembly sequences. *IEEE J. Robot. Autom.* 3, 640–658.
- Derrac, J., García, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* 1, 3–18.
- Dong, T., Tong, R., Zhang, L., Dong, J., 2007. A knowledge-based approach to assembly sequence planning. *Int. J. Adv. Manuf. Technol.* 32, 1232–1244.
- Elsayed, A., Kongar, E., Gupta, S., Sobh, T., 2011. An online genetic algorithm for automated disassembly sequence generation. In: *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, pp. 657–664.
- Friedman, M., 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.* 32, 675–701.
- Gao, L., Qian, W., Li, X., Wang, J., 2010. Application of memetic algorithm in assembly sequence planning. *Int. J. Adv. Manuf. Technol.* 49, 1175–1184.
- Guo, J., Wang, P., Cui, N., 2007. Adaptive ant colony algorithm for on-orbit assembly planning. In: *Proceedings of the 2nd IEEE Conference on Industrial Electronics and Applications, ICIEA*, pp. 1590–1593.
- Halperin, D., Latombe, J.-C., Wilson, R.H., 2000. A general framework for assembly planning: the motion space approach. *Algorithmica* 26, 577–601.
- Homem de Mello, L.S., Lee, S., 1991. *Computer-Aided Mechanical Assembly Planning*. Kluwer Academic, Norwell, MA, ISBN: 978-1-4615-4038-0.
- Homem De Mello, L.S., Sanderson, A.C., 1991. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation* 7 (2), 228–240.
- Hong, D., Cho, H., 1995. A neural-network-based computational scheme for generating optimized robotic assembly sequences. *Eng. Appl. Artif. Intell.* 8, 129–145.
- Hong, D., Cho, H., 1999. Generation of robotic assembly sequences using a simulated annealing. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 1247–1252.
- Hsu, Y.-Y., Tai, P.-H., Wang, M.-W., Chen, W.-C., 2011. A knowledge-based engineering system for assembly sequence planning. *Int. J. Adv. Manuf. Technol.* 55, 763–782.
- Jiménez, P., 2013. Survey on assembly sequencing: a combinatorial and geometrical perspective. *J. Intell. Manuf.* 24, 235–250.
- Kruskal, W.H., Wallis, W.A., 1952. Use of ranks in one-criterion variance analysis. *J. Am. Stat. Assoc.* 47, 583–621.
- Lambert, A.J., 2003. Disassembly sequencing: a survey. *Int. J. Prod. Res.* 41, 3721–3759.
- Latombe, J.-C., Wilson, R.H., Gazals, F., 1997. Assembly sequencing with tolerated parts. *Comput.-Aided Des.* 29, 159–174.
- Li, J.-R., Khoo, L.P., Tor, S.B., 2003. A Tabu-enhanced genetic algorithm approach for assembly process planning. *J. Intell. Manuf.* 14, 197–208.

- Liu, X., Peng, G., Liu, X., Hou, Y., 2012. Disassembly sequence planning approach for product virtual maintenance based on improved max–min ant system. *Int. J. Adv. Manuf. Technol.* 59, 829–839.
- Lu, C., Liu, Y.-C., 2012. A disassembly sequence planning approach with an advanced immune algorithm. *Proc. Inst. Mech. Eng. Part C: J. Mech. Eng. Sci.* 226, 2739–2749.
- Lv, H., Lu, C., 2010. An assembly sequence planning approach with a discrete particle swarm optimization algorithm. *Int. J. Adv. Manuf. Technol.* 50, 761–770.
- Marian, R.M., Luong, L.H., Abhary, K., 2006. A genetic algorithm for the optimisation of assembly sequences. *Comput. Ind. Eng.* 50, 503–527.
- Masehian, E., Akbaripour, H., Mohabbati-Kalejahi, N., 2013. Landscape analysis and efficient metaheuristics for solving the n-queens problem. *Comput. Optim. Appl.* 56, 735–764.
- Mi, X., Zhen, X., Zhou, S. & Fan, W., 2011. Research and implementation on visualization system of disassembly sequence planning based on Ant colony algorithm. *Computer Supported Cooperative Work in Design (CSCWD)*, 2011 15th International Conference on IEEE, pp. 581–585.
- Morato, C., Kaipa, K., Gupta, S.K., 2013. Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters. *Comput.-Aided Des.*
- Motavalli, S., Islam, A.-U., 1997. Multi-criteria assembly sequencing. *Comput. Ind. Eng.* 32, 743–751.
- Niederreiter, H., 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, ISBN: 978-0-898712-95-7.
- Ou, L.-M., Xu, X., 2013. Relationship matrix based automatic assembly sequence generation from a CAD model. *Comput.-Aided Des.* 45, 1053–1067.
- Pitzer, E., 2012. And Affenzeller M. A comprehensive survey on fitness landscape analysis. In: Fodor, J., Klempous, R., SuárezAraujo, C. (Eds.), *Recent Advances in Intelligent Engineering Systems volume 378 of Studies in Computational Intelligence*. Springer, Berlin Heidelberg, pp. 161–191.
- Rakshit, S. and Akella, S. The Influence of Motion Paths and Assembly Sequences on the Stability of Assemblies. *IEEE Transactions on Automation Science and Engineering*, Vol. PP, no.99, pp.1–13, <http://dx.doi.org/10.1109/TASE.2014.2345569>.
- Rashid, M.F.F., Hutabarat, W., Tiwari, A., 2012. A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. *Int. J. Adv. Manuf. Technol.* 59, 335–349.
- Sacerdoti, E.D., 1974. Planning in a hierarchy of abstraction spaces. *Artif. Intell.* 5, 115–135.
- Shu-Xia, L., Hong-Bo, S., 2008. GSSA and ACO for assembly sequence planning: a comparative study. In: *IEEE International Conference on Automation and Logistics, ICAL*, pp. 1270–1275.
- Sinanoglu, C., Börklü, H.R., 2005. An assembly sequence-planning system for mechanical parts using neural network. *Assem. Autom.* 25, 38–52.
- Smith, S.S.-F., 2004. Using multiple genetic operators to reduce premature convergence in genetic assembly planning. *Comput. Ind.* 54, 35–49.
- Smith, S.S.-F., Liu, Y., 2001. The application of multi-level genetic algorithms in assembly planning. *J. Ind. Technol.* 17, 1–9.
- Su, Q., 2007. Computer aided geometric feasible assembly sequence planning and optimizing. *Int. J. Adv. Manuf. Technol.* 33, 48–57.
- Talbi, E.-G., 2009. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, Hoboken, New Jersey, ISBN 978047049690.
- Tiwari, M., Kumar, A., Mileham, A., 2005. Determination of an optimal assembly sequence using the psychoclonal algorithm. *Proc. Inst. Mech. Eng. Part B: J. Eng. Manuf.* 219, 137–149.
- Tseng, H.-E., Li, J.-D., Chang, Y.-H., 2004. Connector-based approach to assembly planning using a genetic algorithm. *Int. J. Prod. Res.* 42, 2243–2261.
- Tseng, Y.-J., Huang, F.-Y., Wang, J.-C., 2011a. A particle swarm optimization and case-based reasoning model for sub-disassembly formation in the multi-plant disassembly sequence planning. In: *Proceedings of the 2011 17th International Conference on Concurrent Enterprising (ICE)*, IEEE, pp. 1–9.
- Tseng, Y.-J., Yu, F.-Y., Huang, F.-Y., 2011b. A green assembly sequence planning model with a closed-loop assembly and disassembly sequence planning using a particle swarm optimization method. *Int. J. Adv. Manuf. Technol.* 57, 1183–1197.
- Wang, J., Liu, J., Zhong, Y., 2005. A novel ant colony algorithm for assembly sequence planning. *Int. J. Adv. Manuf. Technol.* 25, 1137–1143.
- Wang, L., Hou, Y., Li, X., Sun, S., 2013. An enhanced harmony search algorithm for assembly sequence planning. *Int. J. Model. Identif. Control* 18, 18–25.
- Wang, W.-P., Tseng, H.-E., 2009. Complexity estimation for genetic assembly sequence planning. *J. Chin. Inst. Ind. Eng.* 26, 44–52.
- Wang, W.L., Shi, X.J., 2014. Disassembly sequence planning based on poisoning Ant Colony Algorithm. *Key Eng. Mater.* 572, 340–343.
- Wang, Y., Liu, J., 2010. Chaotic particle swarm optimization for assembly sequence planning. *Robot. Comput. Integr. Manuf.* 26, 212–222.
- Wilson, R.H., Latombe, J.-C., 1994. Geometric reasoning about mechanical assembly. *Artif. Intell.* 71, 371–396.
- Wilson, R.J., Watkins, J.J., 1990. *Graphs: An Introductory Approach: A First Course in Discrete Mathematics*. Wiley, New York.
- Wolter J.D. On the automatic generation of assembly plans. In: *Proc. of the 1989 IEEE International Conference on Robotics and Automation (Vol. 1)*, Scottsdale, AZ, pp. 62–68.
- Yi, Z.B.L.M.Z., Jianhua, M., 2013. Research on assembly sequence planning based on firefly algorithm. *J. Mech. Eng.* 11, 025.
- Yu, H., Yu, J.P., Zhang, W.L., 2009. An particle swarm optimization approach for assembly sequence planning. *Appl. Mech. Mater.* 16, 1228–1232.
- Zhang, H., Liu, H., Li, L., 2013. Research on a kind of assembly sequence planning based on immune algorithm and particle swarm optimization algorithm. *Int. J. Adv. Manuf. Technol.*, 1–14.
- Zhou, W., Yan, J., Li, Y., Xia, C., Zheng, J., 2013. Imperialist competitive algorithm for assembly sequence planning. *Int. J. Adv. Manuf. Technol.* 67, 2207–2216.
- Zhou, W., Zheng, J.-R., Yan, J.-J., Wang, J.-F., 2011. A novel hybrid algorithm for assembly sequence planning combining bacterial chemotaxis with genetic algorithm. *Int. J. Adv. Manuf. Technol.* 52, 715–724.
- Zvezda, C. 2014. *Миниатюры-ООО Звезда*. Available at (<http://www.zvezda.org.ru/?lng=1&nav=2&p=59&set=6101>). Also at (<http://www.hobbyandleisure.co.uk/hlstore/catalog/zvezda-5001-soviet-medium-tank-3476-mod-1943-172-t34-p-6660.html>).