Available online at www.sciencedirect.com
ScienceDirect
Computers & Industrial Engineering 50 (2006) 503–527
computers & industrial engineering
www.elsevier.com/locate/dsw

A genetic algorithm for the optimisation of assembly sequences

Romeo M. Marian *, Lee H.S. Luong, Kazem Abhary

School of Advanced Manufacturing and Mechanical Engineering, University of South Australia, Mawson Lakes, SA 5095, Australia

Available online 10 August 2006

Abstract

This paper describes a Genetic Algorithm (GA) designed to optimise the Assembly Sequence Planning Problem (ASPP), an extremely diverse, large scale and highly constrained combinatorial problem. The modelling of the ASPP problem, which has to be able to encode any industrial-size product with realistic constraints, and the GA have been designed to accommodate any type of assembly plan and component. A number of specific modelling issues necessary for understanding the manner in which the algorithm works and how it relates to real-life problems, are succinctly presented, as they have to be taken into account/adapted/solved prior to Solving and Optimising (S/O) the problem. The GA has a classical structure but modified genetic operators, to avoid the combinatorial explosion. It works only with feasible assembly sequences and has the ability to search the entire solution space of full-scale, unabridged problems of industrial size. A case study illustrates the application of the proposed GA for a 25-components product.

Keywords: Assembly sequence planning; Representation; Precedence relations; Genetic algorithms; Genetic operators; Combinatorial optimisation

1. Introduction

Assembly is an obligatory process for all multi-component manufactured goods. Assembly contributes significantly to both cost and lead-time of a product (20–50%, sometimes even more (Nof, Wilbert, & Warnecke, 1997), approaching an astounding 90% in specific areas in micro-technologies and electronics).

Assembly Sequence Planning is part of Assembly Planning. An assembly sequence is the most important part of an assembly plan and it affects other aspects of the assembly process – resources, assembly line layout, efficiency and cost – as well as various details in the product design. Automating the generation of assembly sequences and their optimisation can ensure the competitivity of manufactured goods and increase profit margins.

This paper focuses on a GA designed to optimise the assembly sequence of any type of mechanical products for problems of industrial size by overcoming the difficulties associated with the size and character of the ASPP. Other, associated aspects, necessary for the understanding of the manner in which the algorithm works

* Corresponding author. Tel.: +618 83025275; fax: +618 83023380.
E-mail address: romeo.marian@unisa.edu.au (R.M. Marian).

0360-8352/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2005.07.007

and how it relates to real-life problems, are only succinctly presented here as they were detailed in previously published papers.

ASPP is a large-scale combinatorial problem and is highly constrained. The number of potential assembly sequences is proportional to the factorial of the number of parts in the assembly–Combinatorial Explosion (CE) (Wolter, 1989, 1991). Absolute constraints – geometrical, precedence, accessibility and other types of constraints – severely limit the number of feasible assembly sequences. In general terms, automatically generating feasible assembly sequences is, in its full generality, an extraordinarily difficult task, shown to be NP-complete (Wilson & Watkins, 1990) in both two-dimensional and tri-dimensional cases (Kavraki, Latombe, & Wilson, 1993; Wilson, Kavraki, & Perez, 1995). As a result, most of the past and present work in this area have focused on restricted variants of the problem (Kaufman, Wilson, Jones, & Calton, 1996; Romney, Goddard, Goldwasser, & Ramkumar, 1995).

Assembly has an extraordinary diverse character. Assembly can address sequential or non-sequential, linear or non-linear, monotone or non-monotone, coherent or non-coherent assembly sequences or plans (Wolter, 1989) or any combination of those, involving any combination of rigid, elastic, non-elastic, solid, liquid or gaseous components or subassemblies. To be applicable in practice and useful, an assembly sequence planning and optimisation algorithm has to be general enough to accommodate any type of assembly plan and component.

In order to S/O the ASPP, all relevant aspects of the problem have to be properly modelled and the information has to be represented and stored. The quality of the modelling and representations directly impacts on the quality of the results of the subsequent algorithms.

Solving the ASPP is an essential step prior to its optimisation. It can also be considered a standalone problem in itself, when the required output is an automatically generated assembly sequence. The problem is solved, here, by generating a feasible sequence to assemble an n-part product given its description and a number of supplementary constraints (in Section 5.1.).

The optimisation algorithm – a Genetic Algorithm (GA) designed to accommodate the specific requirements of the ASPP – is a population-based search algorithm in the space of solutions and its output is a population of optimal or near-optimal assembly sequences from which the best one/s are selected.

A brief literature review in the next section shows a number of previous attempts to S/O the ASPP and pinpoints critical, limiting issues, not properly adapted to a constrained combinatorial problem. Section 3 presents the research methodology for S/O the ASPP. Section 4 succinctly shows the modelling and representation issues necessary to consider prior to S/O the ASPP.

The GA is able to S/O any type of product of industrial size and realistic constraints. It is presented, along with its operators and its implementation, in Section 5. A case study illustrates the application of the GA for a product with 25 components, in Section 6.

2. A literature review on assembly sequence planning

S/O the ASPP has been attempted, using various approaches, with mixed results. Due to CE, classic optimisation methods failed to optimise industrial-size problems.

AND/OR graphs and a hybrid A^* algorithm were used to represent and select, iteratively and interactively, the optimal assembly sequence in Archimedes, a software package (Kaufman et al., 1996; Jones, Wilson, & Calton, 1998). The planner uses Non-Directional Blocking Graphs (NDBG (Wilson, 1992)) of each subassembly to determine the assembly operations that might be performed to construct a subassembly then it adds constraints.

AND/OR graphs with weighted hyperarcs were also used to store feasible assembly operations for assembly in HighLAP – High Level Assembly Planning, another assembly software package. It selects an assembly plan minimising the costs from the initial nodes up to the goal node (Rohrdanz, Mosemann, & Wahl, 1996).

The representations used for the optimisation approaches presented above either generate and utilize the explicit AND/OR graphs to store and evaluate *all* assembly sequences for a given assembly or interactively select an assembly sequence by successively applying a number of constraints. They are limited either by the maximum number of sequences that can be stored (CE) or risk to lose valuable solutions by artificially

limiting the extent of the graph through a number of initial artificial simplifying assumptions (leading to a severely limited search space).

Simulated Annealing has been proposed by Milner et al. (1994) to search a best assembly sequence. Being based on the totality of assembly sequences (to be generated, and represented in a directed-diamond-graph), and due to CE, the method is limited to reduced search spaces.

Sebaaly and Fujimoto (1996) used a Genetic Planner for assembly automation. The information for assembly is stored in an implicit, and very compact, form in a reference and a connectivity matrix. To overcome the constrained character of the ASPP, the complete search space consisting of all possible parts combination is clustered into families of similar sequences, where every family contains only one feasible sequence satisfying the problem constraints. The assembly sequences are generated without searching the complete domain of solutions. Even if CE is overcome using this approach, by seriously limiting the search space, valuable solutions are lost.

From this brief literature review (that summarises a thorough review presented in (Marian, 2003)), it can be concluded that the ASPP has only been considered and S/O in a severely limited format, generally only for Sequential, Linear, Monotone and Coherent (SLMC) assembly plans (Wolter, 1989) and/or for reduced size problems (generally less than 15 components). For those reasons, often, due to massive artificial simplifications, the results obtained are non-representative and can not be used for industrial size products.

It is, therefore, necessary to develop a new methodology able to cope with the extraordinary diverse character of the ASPP and with its large scale when considering products of realistic size and constraints, as encountered in real life.

3. Research methodology

An assembly sequence indicates the succession of operations to assemble the product from its components. A feasible assembly sequence is generated taking into account the characteristics of the product (geometry of components, relations between components, materials of components, tolerances, etc) and other constraints, including assembly line and layout constraints.

One of the basic features of GA is that they work, in an iterative process, alternatively in the coding or model space and in the solution space. The assembly sequences and the assembly constraints are defined in the solution space, where also evaluation takes place. The genetic operators work in the model space with chromosomes, which represent assembly sequences. For this reason, modelling and representation are important issues, as they are the interface between the real-life problems, defined in the solution space, and the abstract replicas of the problem defined in the model space.

3.1. Solving the ASPP

Solving the ASPP implies finding a solution by generating a feasible assembly sequence. Solving the ASPP, due to the diversity of cases that have to be considered, is a difficult task when an algorithm that automates this operation is to be developed. Such an algorithm has to consider the scale, the complexity and generality of the problem and be able to generate a feasible assembly sequence in any realistic industrial context. Moreover, it has to permit its incorporation in the optimisation modules.

The ASPP is solved here using a guided search algorithm based on a graph-search strategy. The input for the guided search is the model of the product. The output is a population of randomly generated feasible assembly sequences (Marian, Luong, & Abhary, 1999b; Marian, 2003).

The guided search operator (implementing the algorithm) is a modified genetic operator, designed to overcome the CE by generating and working only with feasible sequences. The algorithm automatically generates feasible chromosomes – representing feasible assembly sequences – by randomly selecting, in each stage, one of the candidate assembly operations that can be performed at this particular step. To achieve this, the candidates for each step are filtered by using the precedence relations derived from absolute (feasibility) constraints, and the intrinsic precedence relations (as presented in Section 4.2.1.). The guided search operator is detailed in Section 5.1.

3.2. Optimising the ASPP

Optimising the ASPP involves searching and finding an optimum or near optimum feasible assembly sequence according to a quality function based on optimisation criteria. For the optimisation of ASPP, GA were chosen, for their ability to handle large-scale problems and the flexibility in defining the optimisation function (fitness function).

The structure of the proposed GA is classic GA (Gen & Cheng, 1997) and relies on the guided search. To handle CE and the highly constrained character of the problem, modified genetic operators are used in this case. Other approaches (penalty, reject and repairing strategy) were attempted by the authors in earlier stages of the research (Marian, Luong, & Abhary, 1999a). They proved to be effective only for assemblies with reduced number of components (<10), or for highly artificially constrained problems. In both cases, the solution space was severely limited.

The optimisation is an iterative process, with the initial population, generated through guided search, as input. This population undergoes an evolutionary process through crossover, pseudo-mutation and selection. The result of the optimisation is a population of assembly sequences with high fitness values from which the one/s with the highest fitness (corresponding to optimal or near optimal sequences) can be selected.

4. Modelling and representation issues

The quality of a model and its representation affect the size of the model and the facility to access and handle the information abut the product. Ultimately, this dictates the level of detail and the size of the problem that can be optimised. A model of the product for assembly has to consider:

- modelling of assembly sequences – encoded as chromosomes;
- modelling the product for assembly purposes – encoding and storing assembly constraints in an implicit manner to avoid CE;
- modelling the assembly processes – what assembly planning means in mathematical terms;
- defining a framework to encode quality measures in assembly.

All those models have to fully capture the important aspects of the assembly problem and need to have an open character, so that hey can be customised to the specific need of the user. Also, they need to offer the provision to be completed/expanded, later, as new techniques will become available in optimisation and in assembly. They were detailed in previously published papers and will only be succinctly presented in the following sub-sections.

4.1. Modelling and representation of assembly sequences

The goal of modelling and representation of assembly sequences is to provide a framework able to encode any useful solution of the ASPP, namely sequential or non-sequential, linear or non-linear, monotone or non-monotone, coherent or non-coherent assembly sequences and plans or any combination of those situations (Marian, 2003).

An assembly sequence is encoded in a chromosome. In case an assembly sequence is SLMC, the chromosome encodes the addition of components to the partial assembly. A gene (a term *ai* of the sequence (Gen & Cheng, 1997)) in locus j, $j = 1\ldots n$ – counted conventionally from left to right – encodes the addition of the corresponding component c*i* in the j-th step. Any partial chromosome with k genes, $k = 1\ldots n$ represents an assembly state, where the first k components are assembled in a partial assembly and all their corresponding liaisons are established. A component, encoded as a gene, will appear in the chromosome once and only once.

Because further constraints apply, an n-term sequence of components of the product may be an illegal or infeasible chromosome, Fig. 1. A simple n-term sequence is an *illegal* chromosome if it is not a permutation. A permutation is a *legal chromosome* and designates a tentative assembly sequence, where all components are present but it is possible that the assembly cannot be realised because of constraints. A *feasible chromosome*

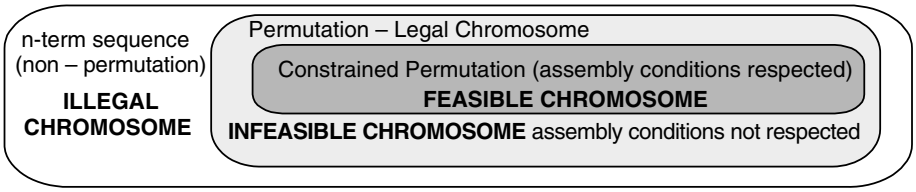

Fig. 1. Relations between chromosomes and assembly sequences (Marian, Kargas, Luong & Abhary, 2003a; Marian, Luong, & Abhary, 2003b).

represents a feasible assembly sequence. It is a *constrained permutation*, a permutation that complies with the assembly's specific conditions or constraints.

The definition of a chromosome can be broadened to encode Entities Meaningful for the Assembly Sequence (EMAS), noted *ai*. A chromosome, in this case, can encode other types of plans (non-SLMC). A gene/EMAS can be generalised to represent more than just the addition of a component to the partial assembly. Thus, non-SLMC assembly plans can be encoded. Those aspects (Marian, 2003; Marian et al., 2003a, 2003b) are as follows:

- *Non-Sequential assembly plans:* a non-sequential set of operations, like simultaneous, coordinated insertion of two or more components along different trajectories, can be isolated, aggregated and referred to as an EMAS (e.g. *ai*) and encoded as a gene. In this case a chromosome represents *a non-sequential assembly plan* (an assembly sequence with a non-sequential element);
- *Non-Linear assembly sequences:* if a gene/EMAS *ai* encodes the addition of a subassembly made elsewhere and assembled *as is*, the chromosome will encode*a non-linear assembly sequence;*
- *Non-Monotone assembly sequences:* a gene/EMAS *ai* can also encode special assembly operations (such as a manufacturing- or an assembly-like operation not involving the addition of a part, e.g. quality check on a partial assembly). This permits the encoding of production line-related information, thus allowing for the representation of *non-monotone assembly sequences*. e.g.: Fig. 2 presents a product requiring a non-monotone assembly sequence comprising addition of components c1 (a1), c2 (a2) and c3 (a3) and an assembly operation, a4. In this case the chromosome a2–a3–a1–a4 can encode in a homogeneous manner, non-homogeneous information: components c2 and c3 are assembled first, then c1 is added to the partial subassembly, then in the last step c3 is pulled in the slot of c1 (an assembly operation – a4 – with no component added). This generalization permits even encoding the addition of fluids (liquids or gases) in a piece of equipment – an assembly-like specific operation – as a gene.
- *Pseudo-Non-Coherent assembly plans:* an assembly plan is always coherent, i.e. each part that is inserted (except the first) will touch or effectively touch some other previously placed part, with two notable exceptions. The first concerns non-linear plans, but in this case the assembly is subassembly coherent and treated as above. The second exception occurs when an auxiliary fixture or tool is used temporarily, in an early

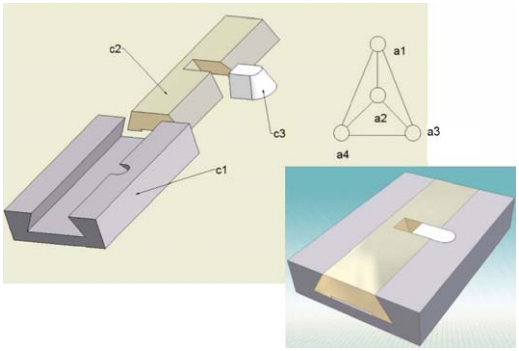

Fig. 2. An assembly realised with a non-monotone assembly sequence.

stage of the assembly process, to hold a number of parts that are not in direct contact, before a connecting part is inserted. The auxiliary fixture or tool can be considered as a component to be added to and then removed from (a 'negative fixture' is added to) the assembly. Thus, the assembly sequence is transformed from a non-coherent (due to representation) into a coherent and non-monotone sequence and can be encoded as above.

The framework briefly presented here is extremely flexible and allows for the encoding of any type of assembly plan. The degree of abstraction of the chromosome, the definition of genes of the chromosome and EMAS and specific rules governing the optimisation process are to be stated for every situation especially for non-SLMC plans, to avoid confusions.

4.2. Modelling and representation of products for assembly

Constraints in assembly are divided into several categories, the most important of which are the absolute constraints and optimisation constraints. The violation of absolute or hard constraints (Jones et al., 1998; Sebaaly & Fujimoto, 1996) produce infeasible assembly sequences. The optimisation criteria, weak constraints or quality measures, on the other hand, differentiate the quality of the assembly sequences and, if violated, do not lead to the infeasibility of the chromosome, but, instead, just add to the difficulty of the assembly process. As a result the constraining relations in assembly are either absolute or optimisation criteria.

This section briefly presents a framework for modelling and representing absolute constraints. They are represented in an implicit manner, to avoid CE. The graph model of the product will be used to represent connections/liaisons between EMAS (Marian, 2003).

The ASPP is, initially, modelled using graphs, Fig. 3. Components, operations and subassemblies/EMAS are represented as vertices in a graph, the graph of liaisons, whereas contacts, connections and relations between them are represented as edges. The graph of liaisons is then translated in an equivalent table format, the table of liaisons that enables a more straightforward development and implementation of algorithms and

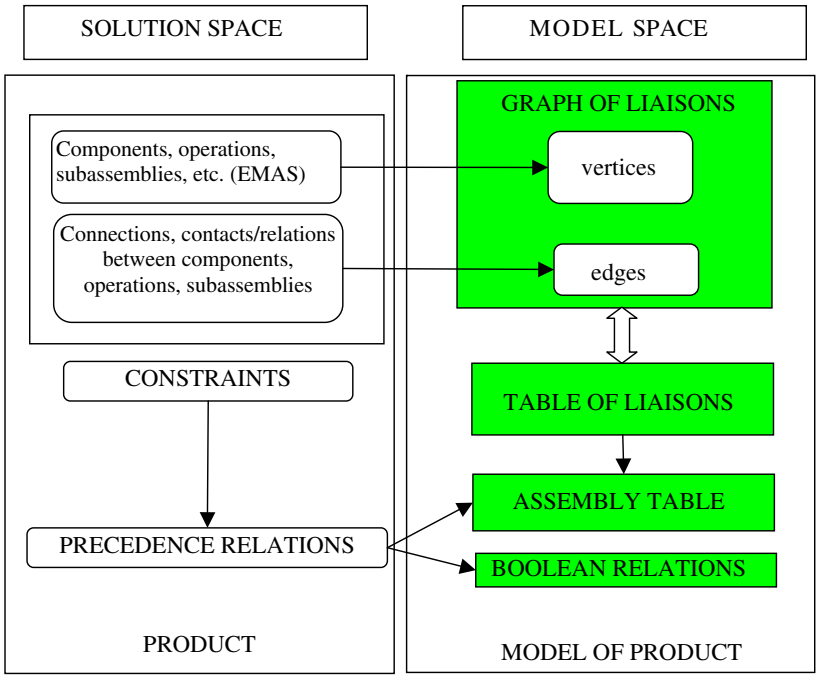

Fig. 3. Modelling and representation of products for assembly.

operators. The absolute constraints of the ASPP, defined in the solution space, are derived as precedence relations. The assembly table includes the connectivity information from the table of liaisons and precedence relations that encode absolute constraints. More precedence relations may be encoded as Boolean relations. The graph of liaisons, table of liaison, assembly table and Boolean relations constitute the database containing the relevant – necessary and sufficient for assembly planning – information about the product.

The graph of liaisons (GL) is defined as a simple, connected, undirected graph, with no loops (Wilson & Watkins, 1990), GL = G(A, L) where A = {*ai*, *i* = 1 . . .*n*} is a nonempty set of vertices *ai* representing the EMAS *ci*. L = {lij} are the edges of the graph, where *lij* = 1 if there is a liaison between *ai* and *aj* and 0 otherwise.

The term "liaison" is a relation established between two EMAS.

For a SLMC assembly process involving components, a liaison can be simplified/reduced to a connection between two components that touch each other. When a component is added to the partial assembly, all its corresponding liaisons with the partial assembly are established. An assembly operation in step *j*, *j* = 2 . . . *n* involves the addition of a component *ci* (vertex *ai*) and the establishment of all liaisons between *ci* and all components assembled previously, in steps 1 . . . (*j* − 1) (establishment of all edges – but at least one (coherence) – between *ai* and the corresponding sub-graph of the partial assembly). In graph terms, this means that the graph of liaisons is connected.

Fig. 4 shows an electric torch, a classic example encountered, with some variations, in the literature about assembly, and its graph of liaisons (c1-cap, c2-lens, c3-bulb, c4-reflector, c5-connector, c6-handle, c7 and c8 – batteries, c9-rear cap).

The graph of liaisons has the following properties (Tomescu, 1985; Wilson & Watkins, 1990):

- **GL is simple** – it is conventionally assumed that between two vertices there is only one edge representing a liaison that includes all contacts (e.g. a contact between two components may include, simultaneously a thread, a plane surface and an adhesive bond) (Marian, 2003);
- **GL has no loops** – there is no liaison joining a vertex to itself because a component cannot be assembled to itself;
- **GL is connected** – if a component belongs to the product it has at least one liaison with another component of the assembly; as a result, a vertex belonging to GL is connected through at least one edge to another vertex in GL;
- **GL is an undirected graph** – if a component *ai* can be assembled to *aj*, the reverse is also true (a liaison is commutative).

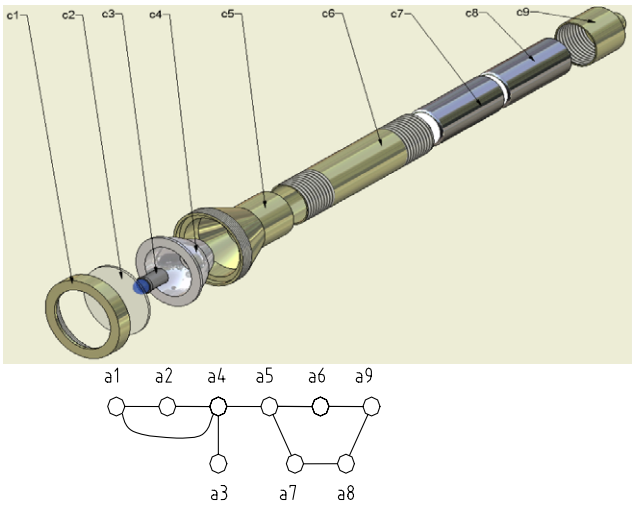

Fig. 4. The electric torch and its graph of liaisons.

Table 1
Table of liaisons for the electric torch

	a1	a2	a3	a4	a5	a6	a7	a8	a9
a1	0	1	0	1	0	0	0	0	0
a2	1	0	0	1	0	0	0	0	0
a3	0	0	0	1	0	0	0	0	0
a4	1	1	1	0	1	0	0	0	0
a5	0	0	0	1	0	1	1	0	0
a6	0	0	0	0	1	0	0	0	1
a7	0	0	0	0	1	0	0	1	0
a8	0	0	0	0	0	0	1	0	1
a9	0	0	0	0	0	1	0	1	0

The table of liaisons (used to store contact information between EMAS) is the translation in table format of liaisons of a product, in fact of the adjacency matrix of the graph of liaisons (Wilson & Watkins, 1990). GL is very intuitive for a human but is difficult to be processed by a computer, which, in turn, can easily handle the information in matrix form. Cell ij contains the value of lij, as defined in GL: if there is a liaison between components *ai* and *aj* in the graph of liaison, it will be designated by "1" in the corresponding cell ij (intersection of row *ai* with column *aj*) otherwise it is 0. If there is a liaison between *ai* and *aj*, there is also one between *aj* and *ai*. Consequently, the table of liaison is symmetric. Table 1 is the table of liaisons for the electric torch in Fig. 4.

The graph of liaisons and the coherence of the assembly process permit the definition of two broad categories of precedence relations in an assembly. **Intrinsic precedence relationships**, strictly specific to the liaisons between components and **Extrinsic precedence relationships**, specific to the actual product and assembly process: geometry, part and tool accessibility, assembly line and work cell layout, etc. (Marian, 2003; Marian et al., 2003a, 2003b).

4.2.1. Intrinsic precedence relations

Intrinsic Precedence Relations (IPR) are precedence relations in an assembly process, specific to an assembly state, derived only from the connectivity of the graph of liaisons and coherence of the assembly process (Marian, 2003).

Properties:

- IPR depend on the configuration of the product – components and liaisons between components;
- IPR depend on the state of the assembly – they are volatile, specific to a particular assembly state and change completely for the next assembly state;
- IPR can be used to determine candidate components for the next assembly state.

Example: If the first component to be assembled for the torch in Fig. 4 is c9, candidate components for next assembly state are: c8 or c6 (contact). If the second component to be added is c8, candidate for the third step are c7 or c6. If, on the other hand, c6 is assembled in the second step, candidate for the third step are c5 or c8. This example illustrates the validity of the properties listed above and shows that precedence relations can be derived from the connectivity of GL and the coherence of the assembly process.

The volatile character of IPR and the size of ASPP reduces the probability for a particular precedence relation (for a certain stage) derived from the graph of a product to be ever used in generating an assembly sequence. As a result, generating and storing all IPR for all assembly states (as it is done in explicit representations) is an expensive exercise in terms of computation time and storage capacity, especially if it can be avoided. IPR can easily be defined for each stage in the assembly process and can be used to generate feasible assembly sequences through guided search with an algorithm that takes advantage of the coherence of the graph of liaisons, to generate feasible assembly sequences (detailed in Sections 4.3 and 5.1.).

4.2.2. Extrinsic precedence relations

In most cases – except for some simple products with limited number of parts – constraints, other than those directly related to the liaisons between parts, are present and govern the assembly process. Those constraints do not have any relation with the graph of liaisons and can be derived in a separate class of precedence relations (Marian, 2003):

Extrinsic Precedence Relations (EPR) are all supplementary precedence relations that can be derived from the constraints in an assembly process and are not related to the liaisons between the components of a product.

EPR originate from constraints and requirements characterising the product – e.g. geometric and accessibility constrains – or the assembly process – e.g. assembly line layout, special operations, supply of parts. EPR are seldom assembly state-dependent and can be expressed as precedence relations between the establishment of one liaison and the establishment of another liaison. The same precedence relations can be defined in different ways, e.g. $[(a1 \wedge a2 \wedge a3) < a4]$ is equivalent to $[(a1 < a4) \wedge (a2 < a4) \wedge (a3 < a4)]$. The framework developed offers the possibility to implement useful constraints in a flexible manner, as needed.

EPR are represented by operators in the assembly table, derived from the table of liaisons. The relation between EMAS *ai* and *aj* is **'can be assembled to'** in the conditions set by the operators in the corresponding cell – *aij*. The assembly table may be asymmetric. The operators can be:

- '0' – if there is no connection/Precedence Relation (PR) between *ai* and *aj*,
- '1' – *ai* can be assembled to *aj* at any stage;
- '*xi*' – a reference liaison, can apply to individual or groups of liaisons;
- '>*xi*' – to be assembled after the reference liaison '*xi*' has been established;
- '>>*xi*' – to be assembled immediately after the reference liaison '*xi*' has been established;

The operators are described more in detail in (Marian, 2003; Marian et al., 2003a, 2003b) and can be used for single or multiple EMAS (like clusters of components).

Sometimes, precedence relations in assembly cannot be expressed as simple precedence relations between liaisons ('>' and '>>'' relations), but only as complex Boolean precedence relations between a liaison or a group of liaisons and another liaison or group of liaisons. Boolean relations are very diverse, their number is generally reduced for an assembly process and they are difficult to implement as a collection of operators. Thus, they can accompany the assembly table and be checked for every assembly state (implemented as filters).

The assembly table, operators and Boolean relations described so far permit the definition of any precedence relation between the establishment of one liaison and establishment of another liaison or state of the assembly process.

4.3. A model of the assembly process

The assembly process can be modelled as building up the graph of liaisons (Marian, 2003; Marian et al., 2003a, 2003b). Fig. 5 presents the graph of liaisons of the electric torch in two extreme states: in disassembled state, when all components are disconnected (dotted edges) and in the assembled state, when all liaisons are established (solid edges).

A similarity has been observed between the model of the assembly process of a product and the Huygens' Principle in Physics (Young, 1992) for waves. Huygens' Principle offers a geometrical method for finding, from the known shape of a wave front at some instant, the shape of the wave front at some later moment. Let's consider the analogy: fluid contained within the closed wave front $\equiv$ partial subassembly (subgraph). This wave model (component $\equiv$ vertex, liaison $\equiv$ edge) permits to determine, at each stage, what components can be added to the partial assembly in the next step so that the assembly process is feasible (generation of IPR derived from the coherence condition). As methodology is concerned, the candidate components for the next stage are determined through constraint propagation and forward checking (Russell & Norvig, 2005).

In case only liaisons are taken into account, the wave model for assembly can be expressed, in the solution space, as follows (Marian, 2003):

Each component of a partial assembly may be considered the element to which is added, in the next step, a component with which it has a liaison and has not yet been assembled. All liaisons between the newly added com-

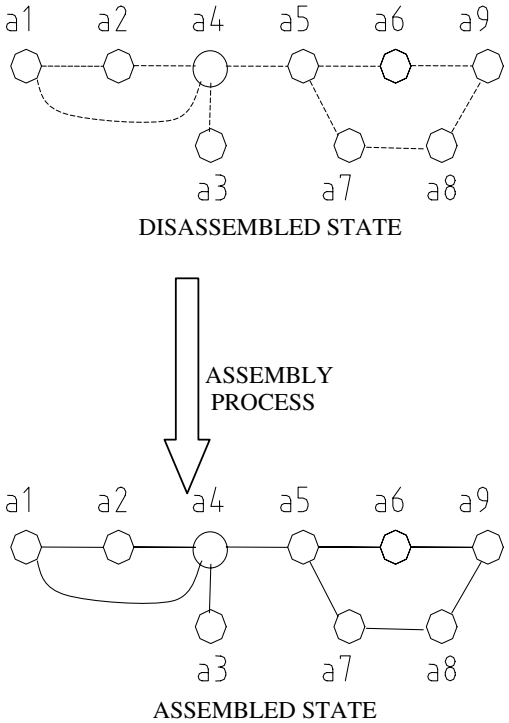

Fig. 5. The electric torch in disassembled and assembled state.

ponent and those of the partial assembly are established during this operation. The cycle is repeated until all components are assembled.

Graphically, the assembly process is equivalent to the addition of vertices and edges to the graph of liaisons until all vertices and edges have been included. When representing the closed curve containing the partial assembly, the assembly process is similar to the propagation of a wave, starting with one vertex and ending with the whole graph. The order in which the vertices are added to the partial graph represent the assembly sequence.

The assembly process starts with the first component/EMAS to be assembled. The graph of liaisons is, at this stage, a null graph – has only the first vertex – and no edges (the graph has no loops). The second EMAS is added to the partial assembly (consisting of the first EMAS) and their liaison is established in the graph of liaisons. The corresponding vertex and edge are added. The process is repeated; in each step all liaisons between the added component and the partial assembly are established. The assembly sequence is this succession of components added until the product is assembled. Relevant information for this model and for generating assembly sequences comprise the components/EMAS to be assembled, liaisons between components/EMAS and a number of conditions to be satisfied – constraints or precedence relations. The assembly process presented graphically in Fig. 6 can be visualised, in the graph representation, as the propagation of a wave. It corresponds to the following feasible assembly sequence: a4–a3–a2–a1–a5–a6–a7–a8–a9.

The wave model of assembly is the foundation for the guided search algorithm, used to generate feasible assembly sequences, and is widely applied in the genetic operators presented below. It determines, at each stage of the assembly process, which EMAS could be assembled, should no other constraints exist, to obtain a feasible assembly sequence. The existence of supplementary constraints further reduces the number of candidate EMAS for each stage. In this way, the generation of relevant EMAS is done using an informed search, confined strictly to the promising regions of the search space, thus being very efficient (Russell & Norvig, 2005). The actual procedure is detailed in Section 5.1.

The use of the wave model and the guided search algorithm, together with the modelling and implicit representations of products for assembly permits the automatic generation of feasible assembly sequences for products of industrial size virtually instantly. It took less than 5 ms for generating a feasible assembly sequence for a 25-component product (measured by generating a population of 1000 feasible chromosomes) on a

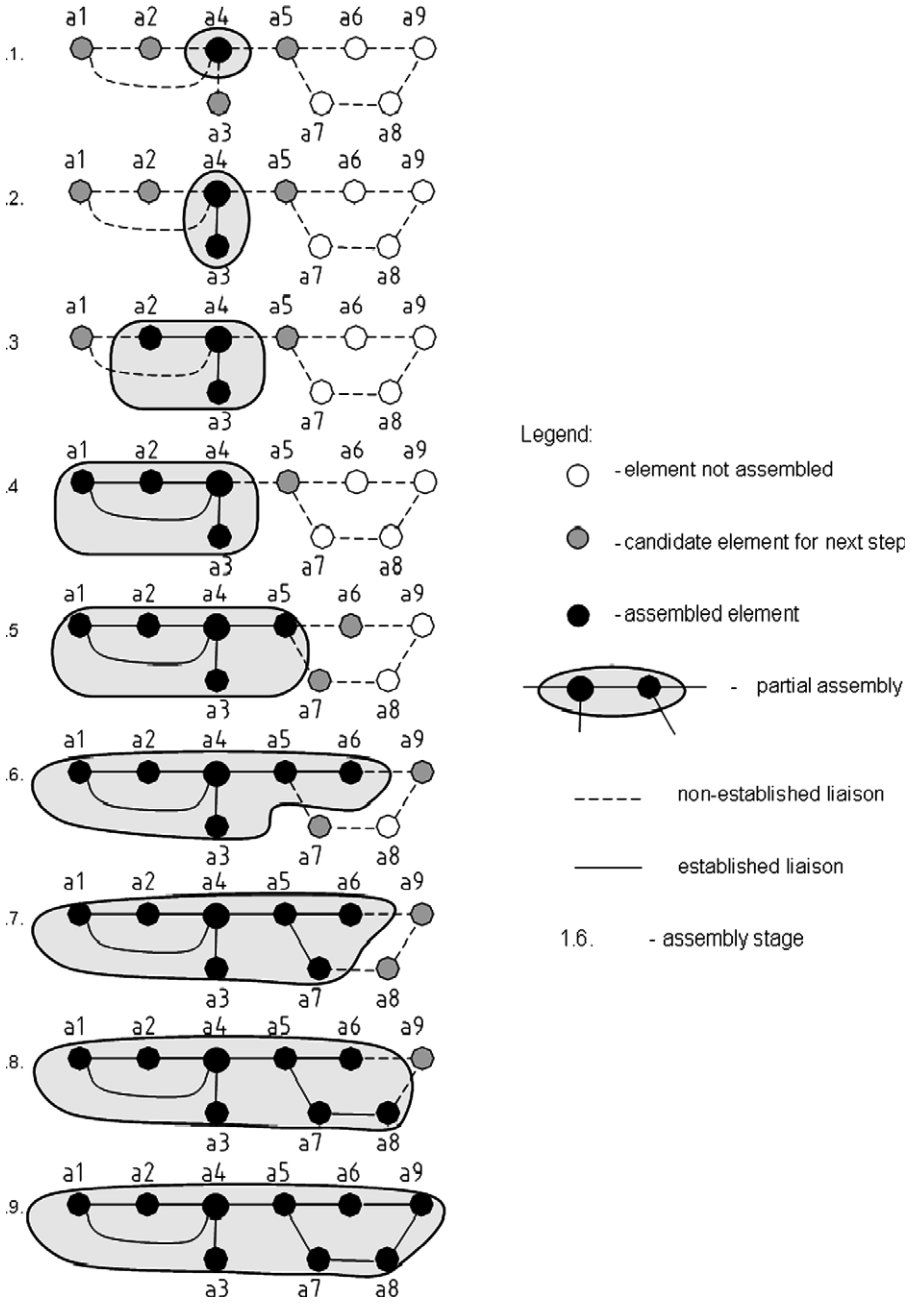

Fig. 6. The wave model of assembly process for the electric torch (Marian et al., 2003a, 2003b).

machine with a Pentium Mobile 1.6 GHz processor. Moreover, the CE is avoided; this approach transforms the combinatorial problem of randomly generating a feasible assembly sequence in a polynomial one (by generating and working, in each step, with all EMAS that would produce feasible assembly sequences).

4.4. A framework to encode quality measures in assembly

A vital issue in any optimisation process is the definition of an objective function able to discriminate between the more and the less desirable solutions of the problem. This issue becomes critical when the quality of a solution cannot be easily expressed as a mathematical function.

The framework used to encode quality measures in assembly (Marian et al., 2003a, 2003b; Luong, Marian, & Abhary, 2005) permits the quantification of the quality of assembly sequences, avoids subjective approaches, is flexible and simple, and permits single criterion or multiple criteria to be considered. The value of an assembly sequence depends on the optimisation criterion/criteria specifically selected/used in any particular optimisation (flexibility) and its use is not restricted to a particular optimisation technique.

For the ASPP optimised using GA, the objective function is designated as the Fitness Function (FF). The FF has to attach a value to each assembly sequence, proportional to the worth of the sequence in the assembly process. The FF has to differentiate, using quantitative and measurable criteria, the quality of an assembly plan.

The FF, as a final indicator, is a composite mathematical cost function of the quality of an assembly process. The FF for the assembly of a product is the sum of partial fitness values corresponding to each assembly operation.

4.4.1. The fitness function for a single optimisation criterion

For a single optimisation criterion, 1 the FF is defined as

$$\mathrm{FF}_1 = \frac{\mathrm{PFF}_1(1) + \mathrm{PFF}_1(2) + \cdots + \mathrm{PFF}_1(n)}{n}, \quad 0 \leqslant \mathrm{PFF}(i) \leqslant 1. \tag{1}$$

where $\mathrm{PFF}_1(i)$ is the partial FF for the criterion 1 associated with the assembly of a component/EMAS in locus i.

$\mathrm{FF}_1(i)$ is defined between 0, corresponding to an ideally bad assembly step, and 1, corresponding to an ideally good assembly step.

The FF is defined as a sum of partial fitness functions, PFF(i), each corresponding to an assembly task. It indicates the facility to add an element or subassembly to the existent partial assembly. The value of the FF depends on the definition of the optimisation criterion and the position of the gene in the assembly sequence but is non-dimensional. This makes comparisons between optimisation with different criteria independent of the actual criteria and allows optimisation with different criteria to be combined together for multi-criteria optimisation.

$\mathrm{PFF}_i(i)$ can be obtained from any linear or non-linear, differentiable or non-differentiable, continuous or non-continuous function. The reason for this extreme elasticity in defining the fitness function is that GA only need a value of the fitness assigned to each individual in the population, not the way this value is obtained or varies from an individual to its neighbour.

The definition of the FF can use the geometrical and physical cost evaluation for assembly planning as presented in (Mosemann, Röhrdanz, & Wahl, 1997). Care should be taken to correctly assign the function so that the highest value corresponds to the best assembly situation.

4.4.2. The fitness function for multi-criteria optimisation

For multi-criteria optimisation, the FF is defined as: $\mathrm{FF}_\Sigma = \alpha 1 \cdot \mathrm{FF}_1 + \alpha 2 \cdot \mathrm{FF}_2 + \cdots + \alpha \mathrm{k} \cdot \mathrm{FF}_k$where $\alpha 1 + \alpha 2 + \ldots + \alpha \mathrm{k} = 1$.

In order to normalize and systematise the FF, FF i for each criterion i and the multicriteria FF_Σ are defined between 0 and 1. The reason is to have the possibility to make comparisons with an ideal assembly sequence for an idealised product. The relative importance of each criterion can be adjusted through the coefficients αi. The fitness of the best assembly sequence may be far from the ideal value of 1 because the inherent difficulty to build certain assemblies due to the specific constraints. It is also a rough measure of the overall difficulty of the assembly derived from the difficulties to assemble each EMAS (Marian, 2003).

5. The genetic algorithm for the optimisation of the ASPP

The ASPP has a number of particularities, as shown above, and a novel GA, which possesses special features, was required to optimise it.

The structure of the proposed GA is classic, as presented in Fig. 7, but with modified Genetic Operators (GO), (Marian, 2003). The Operators are also named, to facilitate references to them. The GO used in this GA are developed for the particularities of the ASPP. This was necessary as no suitable GO, to comply with the requirements of the ASPP, could be found in the literature.

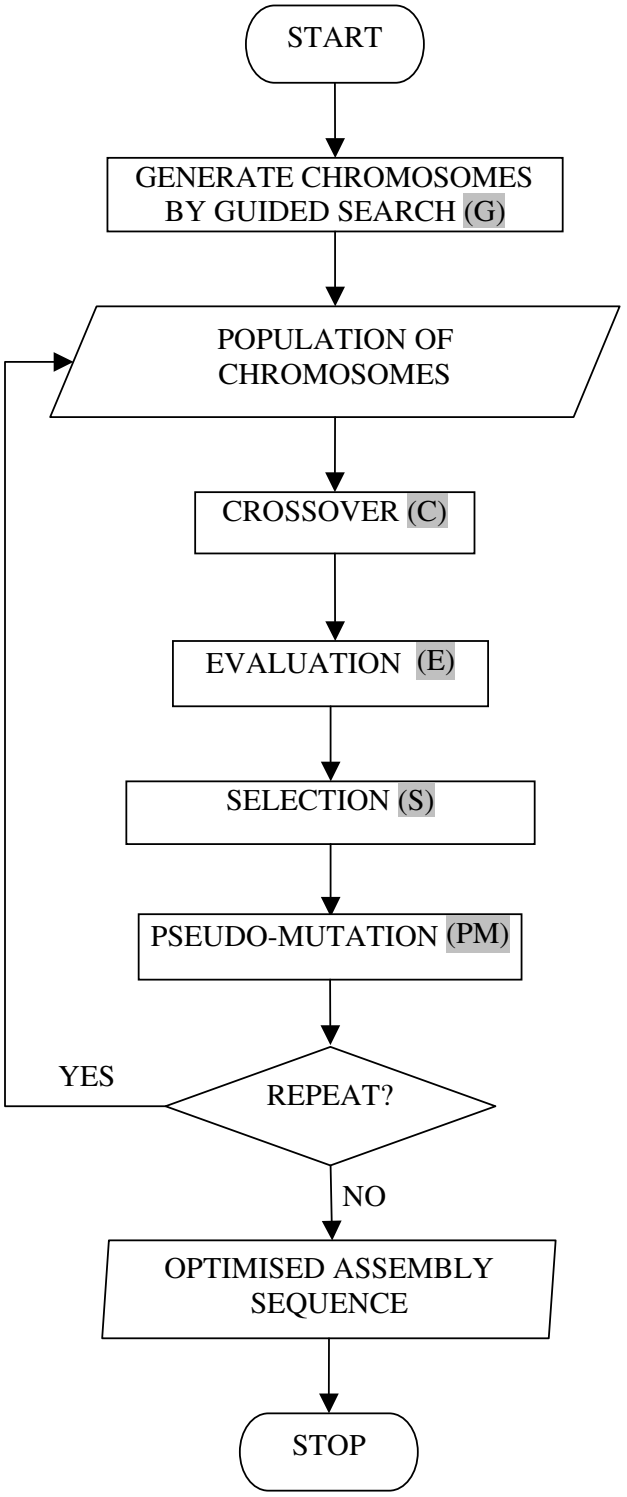

Fig. 7. Structure of the GA for optimising assembly sequences.

A major, conceptual, difference between a classic, complete GA (Gen & Cheng, 1997) and the GA used to optimise the ASPP and presented in Fig. 7 is the absence of a mutation operator *per se*, due to the extreme difficulty to develop for the ASPP.

In certain conditions, using a sufficiently extended initial population and an extended sampling space of both parents and offspring, the premature convergence can be avoided without the need to use mutation. However, for robustness of the algorithm, a pseudo-mutation operator, which is, functionally, similar to a classical mutation operator, is recommended.

A very simple alternative to mutation, a pseudo-mutation (PM), is used, instead, for the proposed GA. It consists of randomly replacing a feasible chromosome in the population with a new, feasible chromosome, generated through guided search, at a rate similar to normal mutation rates. This operation successfully replaces the mutation operator in its function. It avoids premature convergence and permits the exploration of new regions of the search space. At the same time, it is simple and straightforward to implement (duplicates, more or less, the generation of the initial population of chromosomes) and does not differ, in its results, from a classical mutation operator used in ASPP. The PM is applied to the population after evolution, i.e. after the selection process has taken place.

5.1. Automatic generation of initial population of chromosomes by guided search

The solutions of the ASPP are generated through guided search, when all precedence relations (both IPR and EPR) are considered.

The input for the guided search operator is the model of the product. The necessary and relevant information about the product is extracted from the complete model of the product for assembly purposes.

Even it might seem that the whole model of the product, as described above, is slightly redundant (graph of liaisons and table of liaisons contain equivalent information, just presented in a different form), all its components are necessary, useful and used in a different, complementary, manner. The table of liaisons can be simply and directly implemented in a computer program, in the form of a matrix. On the other hand, the graph of liaisons is extremely useful during the coding, debugging and validation stage, when developing and implementing the optimisation of the ASPP in a computer program, due to its very visual quality.

IPR and EPR are implemented as test filters which, applied to candidate components/EMAS for assembly in a particular stage, eliminate those that would produce infeasible chromosomes.

Conceptually, the guided search for chromosomes considering IPR and EPR is as follows, (Marian, 2003):

Input for guided search: the model of the product;

- **Step G1**: The first gene is randomly chosen from vertices of the product;
- **Step G2**: Test vertex chosen in Step 1 against relevant EPR:
  - **if** first vertex does not satisfy EPR, **repeat** Step 1;
- **Step G3**: Delete column of first vertex in the table of liaisons;
- **Step G4**: Select candidates for next assembly stage:
  - **if** $aij \neq 0, j = 1 \ldots n$, in the corresponding rows of the vertices *ai* already assembled of the table of liaisons, **then** *aj* are candidates for next assembly stage and are aggregated in a list;
- **Step G5**: Test vertices *aj* selected in Step 4 for EPR:
  - **if** candidates *aj* do not satisfy EPR, **then** eliminate them from candidates' list;
- **Step G6**: Randomly select a vertex from the candidates' list to become the current gene;
- **Step G7**: Delete column of vertex selected in Step 6;
- **Step G8**: Repeat Steps G4–G7 until all vertices have been placed;
- **Step G9**: Repeat Steps G1–G8 for the size of the population.

Output of guided search: the initial population of feasible chromosomes.

In **Step G1** the algorithm generates a perfectly random first gene from vertices of the EMAS of the assembly process. In **Step G2** this first vertex is tested to determine if the corresponding EMAS satisfies

EPR (step 2). If not, it is rejected and first 2 steps are repeated until a suitable vertex is found, to become the first gene.

Generating and testing the first gene proved to be faster and easier to implement than defining a list of candidate vertices for the first gene. The list of candidate vertices can also be obtained by checking every vertex for all the EPR that appear in the assembly process and selecting a group of potential vertices from which to randomly select the first gene. Those two methods are perfectly equivalent as final result. There is, however, a notable difference in the associated difficulty to implement them and the computational costs involved, hence the selection of the generate-and-test method.

In **Step G3** the column of the vertex selected to be the first gene is deleted. This will ensure that the vertex appearing in gene 1 cannot be selected again in that chromosome permutation.

In **Step G4**, candidate vertices for next assembly state are selected. Those candidates are the vertices *aj* for which the cells $aij \neq 0$, where *ai* are the vertices already assembled (genes already selected). This means that candidate vertices are only those that have a liaison with the vertices already assembled (generation and immediate use of relevant IPR). The list of candidates comprises the vertices that satisfy IPR for that particular assembly stage.

In **Step G5**, each candidate vertex from the list is checked against all relevant EPR (whether implemented in the assembly table or as Boolean relations). The candidates that do not satisfy EPR are removed from the list. The remaining candidates satisfy both IPR and EPR.

In **Step G6**, a vertex is randomly selected from the candidates' list. Due to the manner the candidates were picked, the selected vertex corresponds to a feasible assembly step.

In **Step G7** the column of the vertex selected in Step 6 is deleted. Again, this will ensure that the vertex appearing in the previous gene cannot be selected again in that chromosome (permutation).

Steps G4–G7 are repeated until all vertices are placed (**Step G8**). The result of the algorithm at this stage is a chromosome.

Steps G1–G8 are repeated until all the chromosomes of the initial population are generated.

The output of the algorithm is the initial population of feasible chromosome.

This algorithm has the following particularities:

- the chromosome is a permutation. This is enforced by deleting the column of each vertex once it is assembled. This avoids duplications of genes, as each vertex can be chosen only once. And, because the algorithm runs until all vertices are placed, each vertex will be selected.
- all genes are generated perfectly at random from the ones that can be assembled at each particular step. Indeed, in Step 1, the first vertex is randomly chosen (then checked, and the operation is repeated). Also, in Step 6, the vertex is randomly chosen from the list of candidates.
- IPR are respected. The candidate vertices for gene 2 and onwards are the ones that have at least one contact with a gene already assembled (Step 4). This ensures the coherence of the assembly process.
- EPR are respected. Candidate vertices are the ones that satisfy all precedence relations (Steps 2 and 5). Thus, the generated chromosome is feasible.

The algorithm is very effective. At each iteration it adds a gene to the chromosome, except for the selection of the first gene. In case of the first gene, it might only require an iteration to generate a valid vertex (if no EPR are defined or if a non-constrained vertex is chosen during first iteration). It is, however, possible that several iterations might be needed, depending on the degree of constraint, the number of EPR and the chance to select the right vertex (a stochastic process).

5.2. The crossover operator

Crossover is the main genetic operator. It operates on pairs of chromosomes at a time and generates offspring by combining both chromosomes' features (Gen & Cheng, 1997). The crossover swaps a part of the parents' genetic information to produce the offspring chromosomes.

Particularly, for the crossover implemented in the GA for the optimisation of the ASPP, its input is a pair of randomly selected parent chromosomes and the output is a pair of offspring that combine the parents' fea-

tures. The hard condition imposed in this case is the feasibility of the children chromosomes. This prevents, in this case, the simple swap of the partial chromosomes at the right of the cut point.

In the GA for optimising the ASPP, the crossover operator has to respect the following two conditions:

- the children have to be feasible;
- the children chromosomes have to combine and keep most – if not all, conditions permitting – the parents' properties, i.e. they have to inherit (incorporate) most of the genetic information from their parents;

Due to the constrained character of the ASPP and due to the integer representation used for chromosomes, a suitable crossover operator could not be found in the literature and used. Thus, a modified crossover operator had to be developed.

The crossover operator relies heavily on the guided search. It is, in essence, a development of the guided search on which supplementary conditions are imposed. It operates with feasible chromosomes and, from pairs of feasible parent chromosomes undergoing crossover, the result is pairs of feasible offspring chromosomes with the maximum of the parents' features. The crossover operator is presented, conceptually, below (Marian, Luong, & Abhary, 2000; Marian, 2003):

Input for crossover: a population of feasible parent chromosomes from the current generation and the model of the product, as shown in the previous section.

- **Step C1:** Randomly select pairs of parent chromosomes;
- **Step C2:** For the first pair of parent chromosomes randomly select the cut point;
- **Step C3:** For the first parent chromosome;
  - **Step C4:** For first locus at the right hand side of the cut point:
    - Determine, **by guided search**, the candidate vertices for the gene;
    - **If** the gene in the corresponding locus from the other parent is amongst candidates, **Then** choose it **Else** place any other candidate gene;
  - **Step C5:** Repeat Step C4 for all loci to the end of the chromosome;
- **Step C6:** Repeat Steps C4–C5 for the second parent chromosome;
- **Step C7:** Repeat Steps C2–C6 for the remaining pairs of parent chromosomes;

Output of crossover: a population of feasible offspring (children) chromosomes.

In **Step C1** the algorithm generates perfectly random pairs of parent chromosomes. In the implementation of the GA, the pairs of parent chromosomes for each generation are chosen using a randomly generated permutation of the size of the population. The current generation is then ordered, i.e. each term of the permutation selects the chromosome corresponding to the position of the term.

In **Step C2** the cut point is randomly selected, from gene 2 to gene $n-1$, where n is the length of the chromosome.

Step C3 is more complex and includes, in itself, a number of substeps. For the first parent chromosome:

Step C4: For the first locus at the right hand side of the cut point, determine, by **guided search**, the candidate vertices for the gene. This implies, essentially, going through **Steps G3–G5** (from guided search) to determine the candidate vertices for that particular locus. It is important to point out that **Step G3** has to concern all vertices assembled up to the cut point, not only the first vertex assembled.

At this stage, the gene is to be selected. If the corresponding gene from the other parent chromosome is amongst the candidate vertices, it is selected; otherwise, another gene is randomly selected from candidates.

Step C5 repeats Step C4 for all loci to the end of the chromosome. In this case, only the columns of added vertices are to be cleared, at each step;

Step C6 repeats steps C4–C5 for the second parent chromosome;

Step C7: Repeat Steps C2–C6 for the remaining pairs of parent chromosomes.

This algorithm has the following particularities:

- the offspring chromosomes are feasible. This is guaranteed by the fact that, during crossover, candidate vertices for a locus are determined through guided search (Step C4, incorporating Steps G3–G5 from the guided search module) and the vertex finally chosen will be from that set of candidates;
- the operations involved in the crossover are perfectly stochastic. The stochastic character of the crossover is guaranteed by the following:
- in Step C1, the pairs of parent chromosomes are chosen perfectly at random. using a randomly generated permutation;
- in Step C2, the cut point for the parent chromosomes is selected perfectly at random;
- in case the corresponding vertex from the other parent chromosome is not amongst candidates, the vertex for the current gene is selected perfectly at random from the candidates selected through guided search;
- the chromosomes keep maximum of the parents' features. This is guaranteed by the choice, as a first option, in each locus at the right of the crossover site, of the vertex from the corresponding locus of the other parent chromosome (subject to feasibility).

5.3. The fitness evaluation operator

Each chromosome in the current, extended, population – parent and children – undergoes the evaluation of its fitness. This is a preparatory step for the evolution, so that the better chromosomes have a better chance to be selected.

The fitness function used to evaluate the fitness of the chromosomes is as defined in Section 4.4. The fitness evaluation is as follows (Marian, 2003):

Input for evaluation: the current extended population of parents and children;

Step E1: for the first chromosome in the extended population;
- **Step E2:** For the first locus of the chromosome,
  - compute the partial fitness of the corresponding gene;
- **Step E3:** repeat Step E2 for each gene of the chromosome;
- **Step E4:** compute the Fitness value for the whole chromosome;

Step E5: repeat Steps E1–E4 for the entire population.

Output of evaluation: the list of fitness values associated with the extended population of parent and children chromosomes.

The operator has the following particularities:

- the fitness value associated with a chromosome depends on the way the fitness function is defined;
- the fitness value associated with a chromosome depends on the fitness associated with every gene, respectively the ease to assemble each EMAS (partial FF);

5.4. The selection operator

The sampling mechanism (how chromosomes are selected from the sampling space) used is the stochastic sampling, associated with the Holland's proportionate selection or roulette wheel selection (Holland, 1975). The selection probability or the survival probability for each chromosome is proportional to its fitness value: a chromosome with fitness value f_i is allocated $f_i/\sum \mathrm{FF}$ offspring, where $\sum \mathrm{FF}$ is the sum fitness value of the population. A string with a fitness value higher than the average has a higher chance of selection in the offspring, while a string with a fitness value less than average is less likely to be selected in the next generation. It is possible that a chromosome is selected more than once in the offspring generation.

The selection operator is as follows (Marian, 2003):

Input for selection: the extended population of parent and children chromosomes, their fitness values, of and the size of the initial population;

Step S1: compute the sum of FF, S, from the fitness of each chromosome of the extended population;
Step S2: Generate a random number R;
- **Step S3:** Compute S0 = S*R
  - **Step S4:** S0 =S0-FF(i); $i = i + 1$;
- **Step S5:** Repeat Step 4 until S0 < 0;

Step S6: Select the ith chromosome from the extended population for the new generation;
Step S7: Repeat Steps S2–S6 until all chromosomes have been selected.

Output of selection: the new generation of feasible chromosomes;

The selection operator has the following particularities:

- the selection process is perfectly stochastic, as R is generated at random, in Step S2;
- the probability for a chromosome to be selected is proportional with its fitness value and this probability is equal at each iteration;
- the offspring chromosomes are feasible. This is guaranteed by the fact that they are selected from an extended population of feasible chromosomes;

5.5. The pseudo-mutation operator

The pseudo-mutation operator is, essentially, a modified guided search operator limited to a reduced number of chromosomes. The pseudo-mutation operator is as follows (Marian, 2003):

Input for the pseudo-mutation:

– the current generation of children chromosomes, after selection;
– the pseudo-mutation rate;

Step PM1: Randomly determine the number of chromosomes to be replaced;
Step PM2: Randomly choose a chromosome from current population;
Step PM3: Replace the chromosome chosen in Step PM1 with a new chromosome, generated through guided search (G);
Step PM4: Repeat Steps PM1–PM3 until the number of chromosomes to be changed (given by the pseudo-mutation rate) has been reached;

Output of the pseudo-mutation: – the current generation of chromosomes;

The operator has the following particularities:

- the number of chromosomes to be replaced at each iteration with new chromosomes is determined in a stochastic manner (Step PM1);
- the chromosomes to be replaced are selected perfectly at random (Step PM2);
- the new chromosomes are generated perfectly at random guided search (Step PM3);
- the offspring chromosomes are feasible. This is guaranteed by the fact that they are generated through guided search, respecting both IPR and EPR, as shown in the Guided Search module;

The pseudo-mutation rate is a sensitive variable. For a given, strictly limited population (less than 100 chromosomes, for the studied case), this sensitivity is accentuated. A mutation rate that is too low will permit to some super-chromosomes to take over and the optimisation process converges quickly towards

local optima (exploiting behaviour). At the other extreme, if the pseudo-mutation rate is too high, the probability to replace the good chromosomes with just feasible chromosomes may endanger the convergence of the algorithm, so that it cannot settle in regions of global optima (characteristic of an exploratory behaviour).

It was found, during the run of the GA proposed in the case study that a mutation rate between 1.75% and 2.25% was good for consistent convergence. The actual pseudo-mutation rate for the example presented in Section 6 was 2%.

For extended populations, of the order of 300–500 chromosomes, the pseudo-mutation rate is less important, the GA having, naturally, an exploratory character. In fact, as pointed out at the beginning of the section, in those cases, for large populations, the pseudo mutation is not really indispensable.

5.6. The implementation of the genetic algorithm

The algorithm has been implemented in a computer program coded in VBA, Visual Basic for Applications, in the Microsoft Office suite. VBA was chosen, essentially, for availability and portability reasons. The machine used to run the program was a notebook with a Pentium Mobile 1.6 GHz processor, with 1 GB DDR RAM.

In general, for a product with 25 components/vertices, as described in the next section, a population of 70 chromosomes, evolving over 80 generations, with a crossover rate of 100% and a pseudo-mutation rate of 2%, the following values and run times are characteristic for a non-compiled code, for the implementation and the computer described above:

- the initial population (up to 250 chromosomes) is generated in less than one second;
- the evolution through one generation takes typically under 1 s;
- the evolution over 80 generations takes typically under 70 s;
- the output of the algorithm is recorded in a *.txt* file, from which the information is processed further;
- the size of the output .txt file is in the order of 1.2 MB, which corresponds to about 500 pages of raw data.

The optimisation of the same problem for a population of 150 chromosomes evolving over 150 generations took under 4 min, for the same implementation.

The implementation of the GA is further detailed in the case study.

6. A case study

This section presents the application of the GA developed for optimising the ASPP to an industrial-size product.

The example used for this case study is a modified hydraulic linear motor (HLM), the original of which has been designed and built as a prototype by one of the authors (Marian, 1996). It was chosen for its size, complexity and mix of precedence relations that appear in its construction. An important factor in the decision to use the HLM as an example is the intimate familiarity with its conception, construction and assembly process (direct involvement, from design to testing).

6.1. Description of the product for illustrating the application of the GA

The product considered for illustrating the application of the GA is the Assembled Body of the HLM 25 (ABHLM25). The number of pistons has been doubled from the original construction, such that the total number of parts is 25. ABHLM25 is shown in Fig. 8 and is composed of:

- a body, c1;
- 8 lower bushes, c2...c9 – to be assembled/pressed from the lower side upwards;
- 8 middle bushes c10...c17 – to be assembled/pressed from the upper side downwards;
- 8 upper bushes c18...c25 – to be assembled from the upper side downwards.

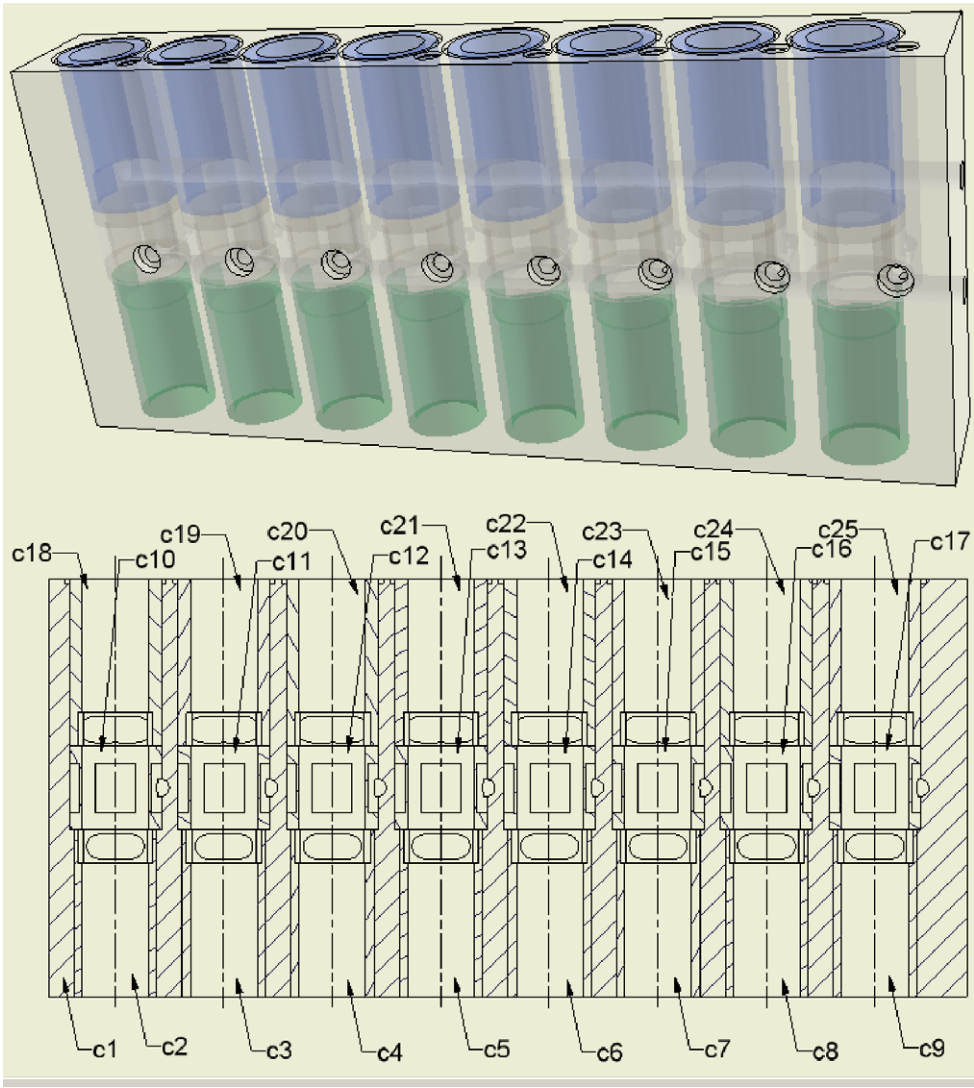

Fig. 8. Assembled body of the HLM25.

This example has been chosen to demonstrate the capabilities of the GA, to work with industrial-size, real-life products. Because of the similarities with the original HLM, all technological data can be transferred directly to this construction.

6.2. The model of the ABHLM 25

The precedence conditions for the ABHLM25 and their implementation are detailed in Table 2. The model used to store the information about the ABHLM25 consists of:

- the graph of liaisons, presented in Fig. 9;
- the table of liaisons, presented in Table 3;
- the assembly table presented in Table 4.

This model fully characterises the product for S/O the ASPP.

Table 2
Assembly conditions and their implementation for ABHLM25

Condition(s)	Implementation, in assembly table
Upper bushes to be assembled after middle bushes are assembled to body (geometric conditions derived into precedence conditions)	- Liaisons between middle-bushes – body are allocated references x1..x8; - Liaisons upper bushes-middle bushes are to be done after x1..x8, so they are allocated (>x1)..(>x8); **- SYMMETRIC RELATIONS;**

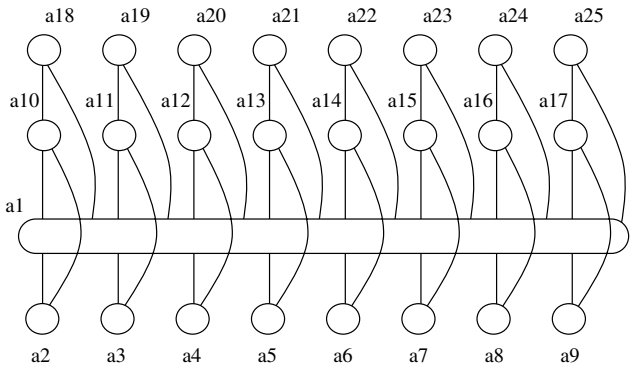

Fig. 9. Graph of liaisons of the ABHLM25.

Table 3
Table of liaisons for the ABHLM25

	a1	a2	a3	a4	a5	a6	a7	a8	a9	A10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25
a1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
a2	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a3	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a4	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a5	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
a6	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
a7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
a8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
a9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
a10	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
a11	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
a12	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
a13	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
a14	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
a15	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
a16	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
a17	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
a18	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a19	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a20	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a21	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
a22	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
a23	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
a24	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
a25	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Table 4
Assembly table for ABHLM25

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25
a1	0	1	1	1	1	1	1	1	1	x1	x2	x3	x4	x5	x6	x7	x8	>x1	>x2	>x3	>x4	>x5	>x6	>x7	>x8
a2	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a3	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a4	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a5	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
a6	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
a7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
a8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
a9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
a10	x1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
a11	x2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
a12	x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
a13	x4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
a14	x5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
a15	x6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
a16	x7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
a17	x8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
a18	>x1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a19	>x2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a20	>x3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a21	>x4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
a22	>x5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
a23	>x6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
a24	>x7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
a25	>x8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

6.3. Quality measures and the FF for the ABHLM25

The quality measures for this product are extrapolated from the HLM with 4 pistons. A penalty function approach has been used to compute the fitness value for assembly sequences.

The penalty function approach was chosen because the penalties are easy to define, realistically capture the difficulties associated with the assembly process and the number of penalties to consider is relatively reduced. Also, the evaluation is simple and straightforward, a most desirable feature for a population-based search.

The fitness function FF is defined as follows:

$$\mathrm{FF} = \frac{\mathrm{PFF}_1(1) + \mathrm{PFF}_1(2) + \ldots + \mathrm{PFF}_1(25)}{25}$$

The partial fitness functions PFF(i) are defined as: PFF(i) = (1 − PF(i))where PF(i) is a penalty function corresponding to adding a particular component to the partial subassembly in Step i.

It is assumed that components are assembled vertically, downwards, on an assembly press. The bushes have to be assembled (pressed) in the body, even if the reverse (body pressed on a bush), as awkward as it might seem, is possible. This means there might be necessary to rotate the block during assembly. The body, being heavy and difficult to manipulate (in order of 25 kg), should be the base component, to which other components, the bushes (0.2–0.4 kg) are added/pressed into, not vice-versa. Assembling body to bushes, rotating the body or changing the type of bushes to be assembled carries a penalty. The penalty values are as follows:

1. if the body is assembled to a bush, the corresponding penalty value is 0.9;
2. changing type of part: carries a penalty of 0.15;
3. rotating block: carries a penalty of 0.5;

The value of those penalties, from experience, is consistent with the difficulty to perform the related operations at the shop-floor level.

6.4. Results and discussions

A typical set of results of running the GA are summarised in Fig. 10. It plots the evolution of the maximum, minimum and average values for the FF (FFM, FFm and FFa). The results correspond to a population of 70 chromosomes, evolving over 80 generations.

It can be seen that FFM and FFa, corresponding to the maximum and the average fitness values have a strong ascendant trend for the first 20 generations, then a fairly stable ascendant trend for the rest of the generations or until they reach the maximum value. The best chromosomes appear in generation 70.

The minimum value of the fitness is very noisy, due, in part, to the new chromosomes introduced by the pseudo-mutation.

The maximum value that can be achieved by the FF for the penalties presented above is FFmax = (25–0.15– 0.5–0.15–0.15)/25 = 0.962 and is the fitness of $(8!)^3$ = 65,548,320,768,000 'best' sequences, corresponding to the following succession of assembly steps: a1 – the first component to be assembled; a2–a3–a4–a5–a6–a7–a8–a9 – in any order, in a continuous sequence; a10–a11–a12–a13–a14–a15–a16–a17 – in any order, in a continuous sequence; a18–a19–a20–a22–a23–a24–a25 – in any order, in a continuous sequence;

Those best assembly sequences imply the change of the type of part after the assembly of the body (penalty of 0.15), a rotation of the body after the assembly of the lower bushes (penalty of 0.5), change of type of part from lower to middle bushes (penalty of 0.15) and change of type of part from middle to upper bushes (penalty of 0.15).

The number of best sequences could be further reduced, by using supplementary penalties, but in this case this would complicate matters without being of much benefit.

The results obtained by optimising the assembly sequence with GA were confirmed in practice for the smaller HLM. The sequence used at the time for assembling it was the equivalent one of the 'best assembly sequences' obtained by applying the GA in this case.

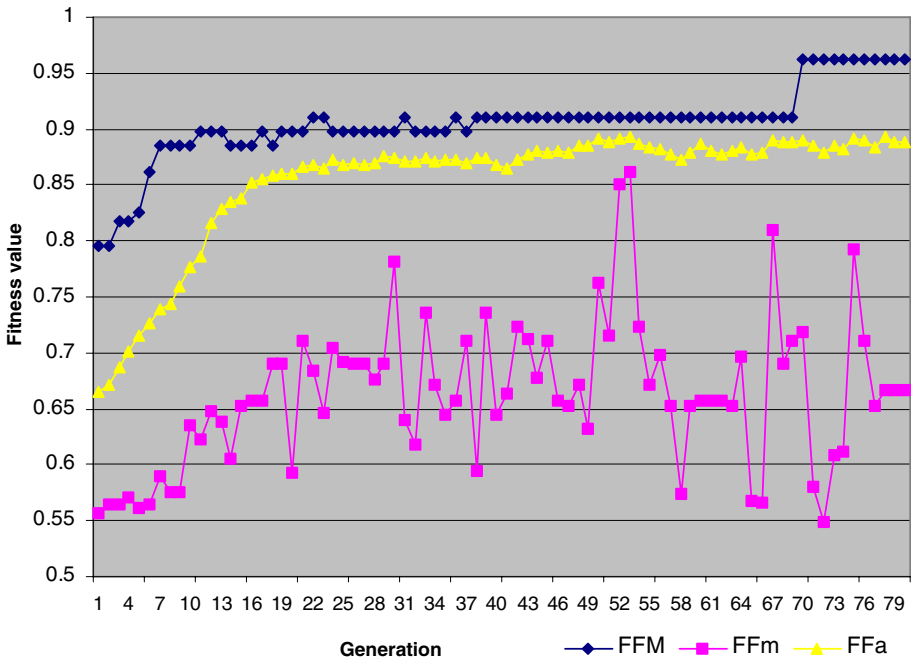

Fig. 10. Evolution of the FF during a representative optimisation run.

7. Concluding remarks

The paper presented a GA able to optimise the ASPP, an extremely diverse, large scale, highly constrained, combinatorial problem.

The algorithm has a classic structure, but the genetic operators are modified and adapted for the specific set of tasks they have to fulfill and the characteristics of the ASPP. In particular, the algorithm works only with feasible chromosomes. This characteristic is strongly enforced. Each genetic operator has been conceived and designed around this condition. Except for the evaluation and the selection modules/operators, which are more or less, conceptually, classical, all other operators are specially designed and implemented so that they can handle the large scale and the constrained character of the ASPP.

The framework for modelling and representing assembly sequences and plans as chromosomes permits the encoding of any known assembly plan. Any combinations of sequential/non-sequential, linear/non-linear, monotone/non-monotone, coherent/pseudo-non-coherent plans, involving any combination of rigid, elastic, non-elastic, solid, liquid or gaseous components, with fixed or variable geometry and volume, or subassemblies can be modelled and represented and, where necessary, can include consideration of operations, fixtures and tools. The constraints are modelled and represented as precedence relations in the assembly table and as Boolean relations.

The initial population is obtained by randomly generating feasible chromosomes using a guided search algorithm. The algorithm is very effective. At each iteration, it produces a gene and adds it to the chromosome.

The results obtained by optimising the assembly sequence with GA were confirmed in practice. The hydraulic motor from which the case study was derived, for example, was built by one of the authors and the assembly sequence used at the time is one of the 'best assembly sequences' obtained by applying the GA. The proposed GA has been successfully implemented in a computer program in VBA, in the Microsoft Office suite.

To the authors' best knowledge, no assembly sequence planner developed to this day, can attempt to genuinely solve and optimise (with the potential to explore any region of the search space) assembly problems of 25 elements. All previous attempts only consider problems that are significantly simplified and drastically reduced to, in fact, the equivalent of a much lower count of components and severely limited search spaces.

The computer code developed to implement the GA, though written in VBA and non-compiled (generally considered a very slow implementation), managed to achieve excellent times for the optimisation of the ASPP considering the size of the product, the degree of constraint, the size of the population and the number of generations. The capacity of the algorithm to work in those conditions proves its robustness.

Those results, coupled with the polynomial complexity of the proposed optimisation algorithm, lead to the assumption that the technique can be safely extrapolated for generating feasible assembly sequences and for the optimisation of the assembly sequences/plans of most industrial-size products that are today planned (and not optimised) manually.

References

Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York: John Wiley & Sons.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Jones, R. E., Wilson, R. H., & Calton, T. L. (1998). On constraints in assembly planning. *IEEE Transactions on Robotics and Automation, 16*(6), 849–863.

Kaufman, S. G., Wilson, R. H., Jones, R.E., Calton, T.L. & Ames A.L. (1996). The archimedes 2 mechanical assembly planning system. In *Proceedings of the 1996 international conference on robotics and automation* (pp 3361–3368). Minneapolis: IEEE.

Kavraki, L., Latombe, J.-C., & Wilson, R. H. (1993). On the complexity of assembly partitioning. *Information Processing Letters, 48*(5), 229–235.

Luong, L. H. S., Marian, R. M., & Abhary, K. (2005). Assembly sequence optimisation using genetic algorithms. In C. T. Leondes (Ed.). *Intelligent knowledge-based systems* (Vol. 5, pp. 234–271). Boston: Kluwer Academic Publishers.

Marian, R., Luong, L. H. S. & Abhary, K. (1999a). Applications of genetic algorithms in design for Assembly. In H. Parsaei (Ed.), *Proceedings EDA Conference '99, (pp 92–98)*. Vancouver.

Marian, R., Luong, L. H. S. & Abhary, K. (1999b). Optimisation of assembly sequences using genetic algorithms. In B. Katalinic (Ed.), *Proceedings 10th International DAAAM Symposium* (pp 315–316). Viena: DAAAM.

Marian, R., Luong, L. H. S. & Abhary, K. (2000). A new crossover technique for assembly sequence planning using GA. In *Proceedings of the computer integrated manufacturing CIM 2000 conference*, Vol. 2, (pp 723–733). Singapore.

Marian, R. M. (1996). *Research and contributions concerning the improvement of a hydraulic linear motor with piston-valves.* Research Report in preparation for Ph.D., Cluj-Napoca, Romania: Technical University Cluj-Napoca.

Marian, R. M. (2003). *Optimisation of assembly sequences using genetic algorithms*. Ph.D. Thesis. Adelaide, Australia: University of South Australia.

Marian, R. M., Kargas, A., Luong L. H. S. & Abhary K. (2003a). A genetic algorithm for the optimisation of assembly sequences. In *Proceedings of the 32nd international conference on computers and industrial engineering* (Vol. 2, pp. 659–667). Limerick, Ireland.

Marian, R. M., Luong, L. H. S. & Abhary K. (2003b). Assembly sequence planning and optimisation using genetic algorithms. Part I: automatic generation of feasible assembly sequences, *Applied Soft Computing* (2/3F), 223–253.

Milner, J. M., Graves, S. C. & Whitney D. E. (1994). Using simulated annealing to select least-cost assembly sequences. In *Proceedings IEEE international conference on robotics and automation, Vol. 3* (pp 2058–2063). Sad Diego: IEEE Computer Society Press.

Mosemann, H., Röhrdanz, F. & Wahl, F. (1997). Geometrical and physical cost evaluation for robot assembly sequence planning. In *Proceedings IEEE international conference on intelligent engineering systems* (pp 499–504). Budapest, Hungary.

Nof, S., Wilbert, W., & Warnecke, H.-J. (1997). *Industrial assembly*. London: Chapman & Hall.

Rohrdanz, F., Mosemann & Wahl, F. (1996). HighLAP: a high level system for generating, representing and evaluating assembly sequences. In *Proceedinge IEEE international joint symposium on intelligence and systems* (pp. 134–141). Rockville, MD: IEEE.

Romney, B., Goddard, C., Goldwasser, M., & Ramkumar, G. (1995). An efficient system for geometric assembly sequence generation and evaluation. In *Proceedings ASME international computers in engineering conference* (pp. 699–712). Boston: ASME.

Russell, S. J., & Norvig, P. (2005). *Artificial intelligence – A modern approach* (2nd ed.). NJ: Upper Saddle River.

Sebaaly, M. F., & Fujimoto, H. (1996). A genetic planner for assembly automation. In *Proceedings of international conference on evolutionary computation* (pp. 401–406). Nagoya: IEEE.

Tomescu, I. (1985). *Problems in combinatorics and graph theory*. New York: John WIley & Sons.

Wilson, R. H. (1992). *On geometric assembly planning*, Ph.D. Thesis. Stanford, CA: Stanford University.

Wilson, R. H., Kavraki, L., & Perez, T.-L. (1995). Two-handed assembly sequencing. *International Journal of Robotics Research, 14*(4), 335–350.

Wilson, R. J., & Watkins, J. J. (1990). *Graphs – An introductory approach*. New York: John Wiley & Sons Inc.

Wolter, J. D. (1989). On the automatic generation of assembly plans. In *Proceedings international conference on robotics and assembly planning* (pp. 62–68). Scottsdale: IEEE.

Wolter, J. D. (1991). A combinatorial analysis of enumerative data structures for assembly planning. In *Proceedings international conference on robotics and automation* (pp. 611–618). Sacramento: IEEE.

Young, H. D. (1992). *University physics*. Massachussetts: Reading.