The Matching Lego(R)-Like Bricks Problem: Including a Use Case Study in the Manufacturing Industry

Martin Zinner*, Kim Feldhoff*, Rui Song†, André Gellrich†, Wolfgang E. Nagel*
* Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden
Dresden, Germany
E-mail: {martin.zinner1, kim.feldhoff, wolfgang.nagel}@tu-dresden.de
† Technical Information Systems
Technische Universität Dresden
Dresden, Germany
E-mail: {rui.song, andre.gellrich}@tu-dresden.de

***Abstract*—We formulate and transform a real-world combinatorial problem into a constraint satisfaction problem: choose a restricted set of containers from a warehouse, such that the elements contained in the containers satisfy some restrictions and compatibility criteria. We set up a formal, mathematical model, describe the combinatorial problem and define a (nonlinear) system of equations, which describes the equivalent constraint satisfaction problem. Next, we use the framework provided by the Apache Commons Mathematics Library in order to implement a solution based on genetic algorithms. We carry out performance tests and show that a general approach, having business logic solely in the definition of the fitness function, can deliver satisfactory results for a real-world use-case in the manufacturing industry.**

***Keywords*–Constraint satisfaction problem; Combinatorial problem; Genetic algorithm; Crossover; Mutation.**

I. Introduction

We formulate a new real-world combinatorial problem, the motivation for our study. Initially, we describe succinctly the real-world problem as it has been identified at a semiconductor company and present the general strategy to solve it. In order to avoid the technical difficulties related to the industrial application, we present the equivalent problem based on LEGO® bricks. To conclude this chapter, we present the outline of the paper.

A. Motivation

Some time ago we were facing a strategic problem at a big semiconductor company. The company produces Integrated Circuits (ICs), also termed *chips*, assembles them to modules on a circuit board according to guidelines and specifications, and ships the modules as the final product to the customer. The ICs are stored in bins before the last technological process (cleaning) is performed.

The difficulties arise due to technical limitations of the tool that assembles the ICs to modules. The tool can handle at most five bins at once. This means in particular, that the ICs required to fulfill an order from the customer have to be in not more than five bins. Once the bins have been identified, the modules are assembled and shipped to the customer. If it is not possible to identify five bins in connection with a customer order, then either cost-intensive methods (rearranging the content of some bins) or time-intensive methods (waiting some days till the production process delivers new ICs) have to be applied. Hence, identifying the bins necessary to fulfill an order is crucial for the economic success of the company.

B. Current State and Challenge

There has been a selection algorithm in place, based primarily on heuristics and inside knowledge regarding the patterns of the specifications of the modules. Although the existing selection algorithm delivered satisfactory results in most of the cases, it runs for days in some cases and is not flexible enough, in particular, it cannot handle slight deviations from the existing specification patterns.

To circumvent the above inconvenient, the main aim of our study is to determine alternative selection methods, which always deliver satisfactory results within an acceptable time frame, and which are easy adaptable to meet future requirements. Our main objective is to identify and formalize the industrial problem as a mathematical model and to transform the occurring Combinatorial Problem (CP) into a Constrained Satisfaction Problem (CSP). The exact method using MATLAB did not deliver results within a satisfactory time frame. A suitable heuristic method – including Simulated Annealing (SA), Ant Colony Optimization (ACO), Genetic Algorithms (GA), etc. – to solve the CSP within the requirements had to be identified and appropriate algorithms had to be developed, which satisfy both the accuracy and performance demands.

If the general task is to find optimal solution to a set of constraints, we speak about Constrained Optimization Problem (COP). The primarily purpose of the industrial problem is to find a satisfactory solution, since from the technical perspective undercutting the requirements of the specifications does not lead to better quality. However, a straightforward extensions of the CSP towards COP is mentioned later.

C. Problem Description

The following example is artificial, it does not occur in *real life* in this manner, although it is very close to it. It is used to best describe the problem without burden the reader with the technical details of a concrete "real life" example. Later on, we will present a "real life" example from the industry and specify the respective mappings between the two models.

We describe the problem succinctly by using an analogy of building structures out of LEGO®-like pieces (bricks).

LEGO®-like pieces (also termed *blocks* or *bricks*) can be assembled to build sophisticated structures (in the following termed *objects*) like buildings, etc. Figure 1 shows how two bricks can be pooled together. The manufacturer of the bricks

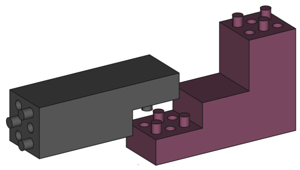

Figure 1: Illustration how two bricks, one of them a corner brick, can be pooled together.

wants to facilitate and simplify the assembling of the bricks to the final objects as well as to cut manufacturing costs and establishes a two phases strategy when designing the layout plans of the final objects. The final object is parsed into components (termed *modules* or *assemblies*) in a straightforward way, such that these modules can also be reused to assemble other objects. This strategy of representing the final object as a composition of modules is very similar to the construction of buildings out of prefabricated structural components, i.e., modules. This way, by using a modular approach, the description and the design plans of quite sophisticated objects can be kept relatively simple and manageable and the complexity and the difficulty of building the final object is delegated to the assembly of the modules. Hence, the building specification of the final object is split into two guidelines, one regarding how to assemble the required modules, one regarding how to put together the modules to form the final object.

Each brick has numerical and non-numerical characteristics. A non-numerical attribute is, for example, a unique ID which characterizes the bricks like shape, approximate dimensions, number and the arrangement of the inner tubes, etc. Another non-numerical attribute is the color of the bricks, etc. There are very tight requirements in order to be able to assemble two or more bricks. In order to cut costs the technological process to manufacture the bricks is kept simple and cost-effective to the detriment of interchangeability. Thus, the pieces are measured after the production process and the measurement values are persisted in adequate storage systems.

In order to be able to assemble the bricks, they have to fit together, i.e., some measurement values (see Figure 2 for an example) have to fulfill some constraints. The respective

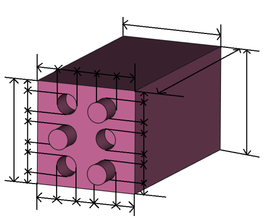

Figure 2: Exemplification of the measurements of a brick.

measurement values must match in order that the bricks can be assembled. For example, putting four bricks together, side by side and on top of each other, strict restrictions concerning perpendicularity and planarity tolerance, have to be satisfied, such that for example, the overall maximum planarity error is 0.05 mm and the maximum perpendicularity error is 0.1 angular degree. Unfortunately, these restrictions can only be evaluated when all the measurement values of the bricks chosen to build the module are at the builder's disposal. Corresponding calculation prescription are available.

Once, the modules have been assembled, the object can be put together out of the pre-assembled modules with no limitations. Furthermore, all the modules are interchangeable with similar ones.

The manufacturing of the bricks is continuous, the bricks are packed into bins after the measuring process occurred and stowed in a warehouse. The ID, the non-numerical attributes and the numerical measurement values are stored in a database and associated to the bin ID. This way, the manufacturer knows exactly the content of each bin. In order to keep the manufacturing costs low, the bins are never repacked, after a bin is full and in the warehouse.

The assembly plan for a particular structure (for example as in Figure 3) is not univocal, i.e., the number and the type of the bricks to build the envisaged structure is not unequivocally specified, the assembly plan contains more alternatives. Since

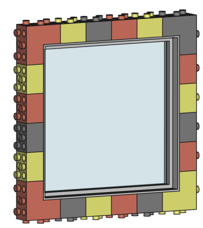

Figure 3: A frame with window as an example for a module.

the manufacturer provides detail information in digital form regarding each brick contained in the bins offered for sale, a computer program could easily verify that a house as given in Figure 4 could be built up from a particular set of bins. Unfortunately, identifying the set of bins necessary to build an

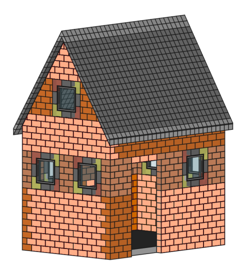

Figure 4: House as exemplification of an order composed of modules.

object (for example the house as in Figure 4) turns out to be a very hard task to accomplish. In order to keep costs down, the number of the bins to be purchased, has to be limited to the necessary ones.

Let us suppose that the order can be assembled out of 5 bins, and the manufacturer offers 1000 bins for sale on his home page. Regrettably, the computer program can only verify if a particular set of five bins contains the bricks necessary to build the house. The brute force method to verify each set of 5 bins out of 1000 does not deliver a practical solution as elementary combinatorics show. Thus, other methods have to be applied.

D. Outline

The remainder of the paper is structured as follows: Section II gives an overview about existing work related to the described problem. Section III introduces the mathematical model and describes how the combinatorial problem can be transformed into a constrained satisfaction problem. Section IV applies the proposed selection algorithm based on genetic algorithms to an industrial use case and shows the performance of an implemented solution which is based on genetic algorithms. A short investigation regarding multi-objective optimization is considered in Section V, whereas Section VI concludes this paper and sketches the future work.

II. Related work

Generally speaking, combinatorial optimization problems are considered as being difficult [1] [2], which stimulated the development of effective approximate methods for their solutions. Combinatorial optimization problems appear in a multitude of real world applications, such as routing, assignment, scheduling, cutting and packing, network design, protein alignment, and in many fields of utmost economic, industrial, and scientific importance. The techniques for solving combinatorial optimization problems can be exact and heuristics. Exact algorithms guarantee optimal solutions, but the execution time often increases dramatically with the size of the underlying data, such that only small size of instances can be exactly solved. For all other cases, optimality is sacrificed for solvability in a limited amount of time [3].

The concept of a constraint satisfaction problem has also been formulated in the nineteen seventies by researchers in the artificial intelligence. Characteristic CSPs are the n queens problem, the zebra puzzle, the full adder circuit, the crossword puzzle, qualitative temporal reasoning, etc. Typical examples of constrained optimization problems are the knapsack problem and the coins problem [4]. Further examples of combinatorial optimization problems [5] are: bin packing, the traveling salesman problem, job scheduling, network routing, vehicle routing problem, multiprocessor scheduling, etc.

For the last decades, the development of theory and methods of computational intelligence regarding problems of combinatorial optimization was of interest of researchers. Nowadays, a class of evolutionary methods [6]–[9] is of particular interest, like simulated annealing, ant colony optimization, taboo search, particle swarm optimization, to which genetic algorithms belong [10]–[13]. Recent publications in this direction [14]–[20] prove the efficacy of applying genetic and other evolutionary algorithms in solving combinatorial optimization problems.

A genetic algorithm is an adaptive search technique based on the principles and mechanism of natural selection and of the survival of the fittest from the natural evolution. The genetic algorithms evolved from Holland's study [21] of adaptation in artificial and natural systems [5].

Typical examples of using evolutionary algorithms are the genetic algorithm approach to solve the hospital physician scheduling problem and an ant colony optimization based approach to solve the split delivery vehicle routing problem [22].

The report [23] offers an approach to use genetic algorithms to solve combinatorial optimization problems on a set of euclidean combinatorial configuration. The euclidean combinatorial configuration is a mapping of a finite abstract set into the euclidean space using the euclidean metric. The class of handled problems includes a problem of balancing masses of rotating parts, occurred in turbine construction, power plant engineering, etc.

III. The Formal Model

In the following, we will formalize the description of the combinatorial problem by introducing a mathematical model. This way, we use the advantages of the rigor of a formal approach over the inaccuracy and the incompleteness of natural languages. First, we introduce and tighten our notation, then we present the formal definition of the constraints which are considered in our formal model and which are the major components in the definition of the fitness function used to control and steer the genetic algorithm. Concluding, the combinatorial problem is defined as a constraint satisfaction problem.

A. Notation

Let $\mathfrak{V}$ be an arbitrary set. We notate by $\mathcal{P}(\mathfrak{V})$ the power set of $\mathfrak{V}$, i.e., the set of all subsets of $\mathfrak{V}$, including the empty set and $\mathfrak{V}$ itself. We notate by $\mathrm{card}(\mathfrak{V})$ the cardinality of $\mathfrak{V}$. We use a calligraphic font to denote index sets, such that the index set of $\mathfrak{V}$ is notated by $I^{\mathcal{V}}$.

The finite sets of bricks, bins, (non-numerical type of) attributes, (numerical type of) and measurements are denoted as follows:

$$\begin{aligned}
\mathfrak{S} &:= \{s_i \mid i \in I^{\mathcal{S}} \text{ and } s_i \text{ is a brick (stone)}\},\\
\mathfrak{B} &:= \{b_i \mid i \in I^{\mathcal{B}} \text{ and } b_i \text{ is a bin (carton)}\},\\
\mathfrak{A} &:= \{A^i \mid i \in I^{\mathcal{A}} \text{ and } A^i \text{ is an attribute}\},\\
\mathfrak{M} &:= \{M^i \mid i \in I^{\mathcal{M}} \text{ and } M^i \text{ is a measurement}\}.
\end{aligned}$$

Let $i \in I^{\mathcal{A}}$, $j \in I^{\mathcal{M}}$, and $k \in I^{\mathcal{S}}$. We denote by a_k^i the value of the attribute A^i of the brick s_k and by m_k^j the value of the measurement M^j at the brick s_k. We denote the list of assembly units (modules) by

$$\mathfrak{U} := \{U^i \mid i \in I^{\mathcal{U}} \text{ and } U^i \text{ is an assembly unit (module)}\}.$$

The construction (guideline) plan for a module $U \in \mathfrak{U}$ contains

a) the (three dimensional) design plan description, i.e., the position of each brick within the module,

b) the non-numerical attribute values for each brick and

c) prescriptions regarding the measurement values.

Analogously, by

$$\mathfrak{O} := \{O^i \mid i \in I^{O} \text{ and } O^i \text{ is an object}\}$$

we denote the list of objects for which there exists construction plans.

The non-numerical attributes values of the selected bricks have to match the corresponding values in the guideline plan.

We denote by

$$\mathfrak{R} := \{R^i \mid i \in I^{\mathcal{R}} \text{ and } R^i \text{ is a requirement (specification)}\}$$

the list of the requirements (specifications) of the objects.

 ISBN: 978-1-61208-752-8

B. Transformation of the CP into a CSP

Let $O \in \mathfrak{O}$ an order. Then, according to the specifications, there exists *(proxy) modules* $\hat{M}^{l_1}, \hat{M}^{l_2}, \ldots, \hat{M}^{l_k}$, such that O is an ordered list of modules, i.e., $O = (\hat{M}^{l_1}, \hat{M}^{l_2}, \ldots, \hat{M}^{l_k})$. The term *proxy* is used to denote an abstract entity according to the specifications. Analogously, each module $\hat{M}^i$ with $i \in \{l_1, l_2, \ldots, l_k\}$ is an ordered list of *proxy bricks* (stones), i.e., $\hat{M}^i = (\hat{s}_{i_1}, \hat{s}_{i_2}, \ldots, \hat{s}_{i_m})$. Hence, each module can be represented by an ordered list of proxy bricks, i.e., $O = (\hat{s}_{k_1}, \hat{s}_{k_2}, \ldots, \hat{s}_{k_n})$. This representation will be used later to define the individuals within the context of the genetic algorithms.

Let $O = (\hat{s}_{k_1}, \hat{s}_{k_2}, \ldots, \hat{s}_{k_n})$ be a module. We say that the ordered list $(s_{k_1}, s_{k_2}, \ldots, s_{k_n})$ with $s_i \in \mathfrak{S}\ \forall i \in \{k_1, k_2, \ldots, k_n\}$ is an assignment (embodiment) of O. This means especially that the abstract unit of the specification is materialized within the production process. We set

$$\mathfrak{E} := \{E^i \mid i \in I^{\mathcal{E}} \text{ and } E^i \text{ is an assignment (embodiment)}\}.$$

Let $R \in \mathfrak{R}$ be a requirement (specification) of a specific module $U \in \mathfrak{U}$ and let $\{\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_n\}$ be the proxy bricks of the specification. The design plan of the specification provides the three-dimensional assembly plan of the proxy bricks. Additionally, the specifications provide information regarding the restriction the bricks have to fulfill in order to be eligible. For each $j \in \{1, 2, \ldots, n\}$ the specifications contain the values $\{\hat{a}^i_j \mid i \in I^{\mathcal{A}}\}$ of the attributes $\mathfrak{A} := \{A^i \mid i \in I^{\mathcal{A}}\}$ at the proxy brick $\hat{s}_j$. We use the symbol $\hat{a}^i_j$ to denote the value of the attribute a^i of the proxy (placeholder) brick $\hat{s}_j$.

This means especially, that the brick s_j can substitute the proxy brick $\hat{s}_j$ if the values of the corresponding attributes coincide, i.e., $a^i_j = \hat{a}^i_j$ for all $i \in I^{\mathcal{A}}$.

More formally, the attributes must satisfy certain constraints:

$$C_A : \mathfrak{A} \times \mathfrak{S} \to \{yes, no\},$$
$$\{s^i_j \mid i \in I^{\mathcal{A}},\ j \in I^{\mathcal{S}}\} \mapsto C_A(a^i_j).$$

$C_A(a^i_j) = yes$ if the attribute constraint is satisfied for the brick s_j i.e., $a^i_j = \hat{a}^i_j$, else $C_A(a^i_j) = no$.

On the other side, the (numerical) measurement values must also satisfy certain constraints (restrictions). For example, the standard deviation of the respective measurement values for some bricks of a specific module should not surpass some given limits. Formally, C_M can be represented as:

$$C_M : \mathfrak{M} \times \mathfrak{U} \to \{yes, no\},$$
$$\{m^i_j \mid i \in I^{\mathcal{M}},\ j \in I^{\mathcal{U}}\} \mapsto C_M(m^i_j).$$

$C_M(m^i_j) = yes$ if the constraint is satisfied for the bricks belonging to the module U^j, else $C_M(m^i_j) = no$.

In order to be able to reduce the constraints to brick level, i.e., to be able to decide whether the constraint is satisfied for a specific brick or not, we use the restriction C^S_M of C_M namely $C^S_M := C_M\big|_S$ such that $C^S_M(s^i_j) = yes$ if $s_j \in U^j$ and $C_M(m^i_j) = yes$; else $C^S_M(m^i_j) = no$. Let $U \in \mathfrak{U}$ be a module. The above means especially, that the measurement constraint on brick level is satisfied for $s \in U$ if the measurement constraint is satisfied (on module level) for U.

Since the measurement values do not really characterize the modules (they must only fulfill the requirements regarding the constraints), we introduce equivalence classes on the set of modules. Two modules belong to the same class if

a) they have both the same design plan,

b) the component bricks fulfill the same (non-numerical) attributes and

c) the prescriptions regarding the measurement values are satisfied for both modules.

Accordingly, two modules belonging to the same equivalence class are interchangeable.

Hence, all the bricks needed for a module must be selected in order to be able to finally decide if the constraints are satisfied or not.

As already mentioned, each object $O \in \mathfrak{O}$ should be assembled out of bricks contained in a reduced number of bins. We set $MaxB^O$ for the maximum number of bins as mentioned above.

Let $i \in I^{\mathcal{B}}$, $j \in I^O$, let $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ be the content of the bin b_i and let $\{s_{j_1}, s_{j_2}, \ldots, s_{j_l}\}$ be an assignment of $O^j \in \mathfrak{O}$. We set $b^j_i := 1$ if $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\} \cap \{s_{j_1}, s_{j_2}, \ldots, s_{j_l}\} \neq \emptyset$ else 0. This means especially, that $b^j_i := 1$ if the bin contains bricks belonging to the respective assignment of O^j.

Analogously, the constraints regarding the bins (cartons) can be regarded formally as:

$$C_B : \mathcal{P}(\mathfrak{B}) \times \mathfrak{O} \to \{yes, no\},$$
$$\{\mathbb{B} \in \mathcal{P}(\mathfrak{B}),\ O^j \in \mathfrak{O}\} \mapsto C_B(\mathbb{B}, O^j).$$

Let $\mathbb{I}$ an index set, such that $\mathbb{B} = \{b_i | i \in \mathbb{I}\}$. Then $C_B(\mathbb{B}, O^j) = yes$ if

$$\sum_{i \in \mathbb{I}} b^j_i \leq MaxB^{O^j},$$

i.e., the bricks of the order O^j are contained in no more than $MaxB^{O^j}$ bins. Additionally, $C_B(\mathbb{B}, O^j) = no$ if the above condition is not satisfied.

Similar to the measurement constraint C_M, we reduce C_B to brick level. Let $\mathbb{S} := \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ be an assignment of O^i. Let $\mathbb{B} = \{b_{l_1}, b_{l_2}, \ldots, b_{l_n}\}$ be a set of bins, such that each bin contains at least one $s \in \mathbb{S}$ and there is no brick $s \in \mathbb{S}$ which is not contained in one of the bins of the set $\mathbb{B}$. In this sense, $\mathbb{B}$ is minimal regarding the assignment $\mathbb{S}$. Then, for the restriction C^S_B on S of C_M we have $C^S_B(s) = yes$ if $C_B(\mathbb{B}, O^i) = yes$, i.e., all the bricks of the assignment $\mathbb{S}$ are stored in no more than $MaxB^{O^j}$ bins. Additionally, $C^{S,U}_B(s) = no$ if $C_B(\mathbb{B}, O^i)$ is not satisfied.

Until now, we considered the constraints related to the architecture of the object, i.e., related to the attributes of a particular brick, the measurement values of the bricks belonging to a module, and the restrictions regarding the bins which contain the bricks. We can condense the constraints mentioned above, such that they relate only to bricks. This means especially, that the measurement constraint are satisfied for a brick, if there is a group of bricks (module, or order), such that the given measurement constraint is satisfied as described above. We set accordingly:

$$\mathfrak{C} := \{C^i \mid i \in I^C \text{ and } C^i \text{ is a distinct constraint}\}$$

the list of distinct constraints.

The constraints can be considered as a function. Please recall that I^S is the index set of $\mathfrak{S}$.

$$C : \mathfrak{S} \times I^C \to \{yes, no\},$$
$$\{s_k^i \mid i \in I^C,\, k \in I^S\} \mapsto C(s_k^i).$$

Please consider, that the above representation can be misinterpreted, such that the constraint is exclusively a property of the respected brick. This is not the case, for example the measurement constraints fulfilled or not for the bricks assigned to a module. Hence, if one brick is changed, then the constraints of all the bricks belonging to a module can be invalidated.

We define now formally the weights, (i.e., w is a weight if $w > 0$) which are necessary to be able to model the importance of the constraints within the genetic algorithm. We set

$$\mathfrak{W} := \{w_i \mid i \in I^C \text{ and } w_i \text{ is a weight}\}$$

the list of weights. This means especially, that each constraint has an associated weight.

The fitness function [24] characterizes the quality of an assignment of an order, such that a value closer to 1 means a better quality. It plays an important role in the decision, whether an assignment fulfills the specifications or not.

The purpose of the following function inv is purely technical, it is used to switch the values of the boolean values 1 and 0 to be used in the definition of an example of the fitness function, i.e., $inv : \{yes, no\} \to \{0, 1\}$ such that $inv(yes) = 0$ and $inv(no) = 1$.

Please find below an example for the fitness function. Let $I^{\mathbb{S}} \subset I^S$ and let $w_i \in \mathfrak{W}$ for all $i \in I^C$. Then:

$$F : \mathfrak{C} \times I^C \to (0, 1],$$
$$\{s_k^i \mid i \in I^C,\, k \in I^{\mathbb{S}}\} \mapsto \frac{1}{1 + \sum\limits_{i \in I^C, k \in I^{\mathbb{S}}} w_i \cdot inv(C(s_k^i))}. \quad (1)$$

Problem formulation (Combinatorial problem)

Let $O \in \mathfrak{O}$ a given object and let $n \in \mathbb{N}$.

Choose n bins from the warehouse, such that the object can be assembled out of the bricks contained in these bins according to the existing construction plans.

The construction plan for an object $O \in \mathfrak{O}$ specifies the lists of (non equivocally determined) modules, including the design plan, such that the object can be build out of these modules. Hence, it can be unambiguously decided, whether the n cartons contain the necessary bricks to assemble them to modules, which can be put together to form the required object. Let us suppose that $n \ll \mathrm{card}(\mathfrak{C})$, i.e., the number of bins in the warehouse exceeds the number of bins to be chosen by orders of magnitude. The difficulties of solving the problem in a straightforward way lie in the very large number of possibilities to combine n bins out of $\mathrm{card}(\mathfrak{B})$. Therefore, other strategies have to be used.

To summarize: the specification of an object (for example a house composed of bricks), contains very strict requirements regarding the components. The assembly plan specifies the strict order in which the bricks have to be assembled. Hence, the bricks must satisfy some attributes (like shape, type, color, etc., in order to satisfy the requirements of the construction plans. Moreover, some bricks have to fit together (for example the window frame) so they can be assembled in the order given by the construction plans. If the above requirements are satisfied for all the units (modules), the object can be assembled. Furthermore, the selected bricks have to be selected from a restricted number of bins (cartons). The latter makes the task so difficult.

From a formal point of view, the associated constraint satisfaction problem of the combinatorial problem can now be formulated:

Problem formulation (Constraint satisfaction problem)

Let $O \in \mathfrak{O}$ be an order with the representation $O = (\hat{s_1}, \hat{s_2}, \ldots, \hat{s_k})$. Set $w_i = 1$ for all $i \in I^C$.

Find an index set $\{l_1, l_2, \ldots, l_k\} \subset I^S$ such that $(s_{l_1}, s_{l_2}, \ldots, s_{l_k})$ is an assignment of O, having $F((s_{l_1}, s_{l_2}, \ldots, s_{l_k})) = 1$.

IV. USE CASE: AN EXCERPT

In the following, we present a real-life use case [25] we came across at an international semiconductor company. We describe the problem by using the specific terminology in the semiconductor industry, utilizing them with care and only when it is inevitable and undeniable necessary. We describe the fundamentals of the genetic algorithms and show the way it is used to solve our problem. Finally, we conclude by presenting some performance tests.

A. Problem Description

The company manufactures integrated circuits (ICs, also termed chips), which are subsequently assembled on circuits boards to salable entities, termed modules. In order to keep production cost low, the specification of the ICs do not impose very tight constraints on the attributes of the ICs, such that the same IC can be used for different types of modules. On the contrary, the specification regarding the modules are very stringent, in order that the module should be fully functional at the customer side. As soon as the ICs are manufactured, a good dozen of electrical properties are measured and persisted in a data repository.

Usually, four to six ICs are assembled on the module. The specification of the modules contains the design (i.e., number and positioning) of the ICs on the integrated circuit board, the type of the IC (article, number of pins, etc.), and several constrains regarding the interaction of the ICs of the module. In order for the module to be fully functional, the corresponding measurement values of the ICs have to be in a narrow range. For example, for a specific measurement, the values of the voltage of the ICs have to be between 2.1 volt and 2.5 volt in order that the IC is not scrapped and can be used for further processing. Unfortunately, not all the ICs having the corresponding measurement value in the range as described above, can be assembled to a module. The values differ too much from each other, and the module will not work properly at the customer side. To circumvent this impediment further constraints are needed. These constraints apply on all ICs of the module or just on a subset of it. For example, an often used constraint is limiting the standard deviation of the voltage to 0.1 within one module.

The ordering unit (termed *work order*) contains the description of the modules, the customer expects to be shipped together at once. There are no additional constraints on the ICs regarding the work order.

As soon as the manufacturing process of the ICs has been finished, the ICs are packed in boxes. That way the cleaning of the ICs can be performed and the ICs can be assembled to modules. The boxes are then transferred to the warehouse.

The difficulties of the semiconductor company to honor each the order in general are also due to technological restrictions, since the tool that assembles the ICs can handle at most five boxes, i.e., all the ICs necessary to fulfill a work order should be located in five boxes. Due to the fact that the work orders always contain the same number of ICs, independent of the specification of the modules, the minimal number of boxes which are needed to meet the requirement of the work order is four with 9 percent surplus of ICs. If we rephrase the above in a more concise form, the challenge is: Find five boxes in the warehouse, such that it contains the ICs needed to fulfill the requirements for a work order.

B. Used Methods

Simple combinatorics show that the brute force method, i.e., go through all the possibilities and check if the selected boxes fulfill the requirements, is not implementable for practical systems. Fortunately, there is an implementation in place for the selection strategy, based on heuristics, local optimum, and inside knowledge of the pattern of the modules. This way, we have a very good way to compare the results of the genetic algorithm with alternative solutions. Regretfully, our attempt to deliver exact solutions on the problem using MATLAB were not crowned by success due to the large amount of data and to the restricted computing power of the machines we used. The disadvantages of the already existing solution for the selection strategy were partly also the issues that made it possible to set up such a solution:

a) the unpredictability that the selection strategy delivers a solution within the expected time frame;

b) the inflexibility to even minor changes in the design and specifications of the modules, thus, the unpredictability that the software can be used in the future;

c) heavy maintenance efforts due to the sophisticated and architecture and implementation;

d) lack of the proprietary knowledge and documentation of the implementation on the low level side;

e) impossibility to reuse the existing code with reasonable efforts for further development and enhancements.

The concepts of the genetic methods are straightforward and easy to understand. The main idea is that we start with an initial population of individuals, and as time goes by, the genes of the descendants are improved, such that at least one individual satisfies the expectations. The individual incorporates the requirements of the problem. The expectation in the end is that these requirements are finally satisfied. Each individual owns genes, part of it is inherited by his descendants.

We define the individuals as an abstraction of the work order, such that each gene of the individual is the abstraction for an IC of the warehouse. Accordingly, the individual satisfies the requirements if the ICs can be assembled to modules, such that the corresponding work order is fulfilled. The initial population is randomly generated out of the ICs in the warehouse. The criterion, which determines to what degree the individual fulfills the requirement of the associated work order, is the fitness function. The fitness function takes values between 0 and 1, a greater value means that the individual is more close to fulfill the specification of the work order. To achieve a value of 1 is the ultimate goal. It means that the corresponding individual satisfies the requirements to fulfill the associated work order. Hence, the definition of the fitness function is one of the most sensible parts of the genetic algorithms and the setup of this function should be considered very carefully.

Actually, the strategy of the genetic algorithm resembles very much to the evolution of the mankind. People marry, have children by passing their genes to them, divorce and remarry again, have children, and so on and so forth. The expectation is that the descendants have more “advantageous genes”, regardless of how the term “advantageous genes” is defined.

Establishing the fitness function is one of the most important strategical decision to be taken when setting up the genetic algorithm. In our case, there are a few constraints (more than one) which affect the quality of the individuals. Implementations which try to find a Pareto optimal state [26], [27] (i.e., a state from which it is impossible to make an individual better, without making at least one individual worse) use strategies as tournament selection [28] or the improved Pareto domination tournament [27].

As already mentioned, the starting population is selected aleatorically. Once, the first generation is constituted, the preparations to generate the next generation are met. Unfortunately, the Apache Commons Mathematics Library does not support multi-objective optimization problems, hence our algorithms cannot use the strategy of the *Niched Pareto Genetic Algorithm* (NPGA) [27].

Instead, for each individual, the fitness function is calculated such that the suitability to fulfill the expectations, is evaluated for each individual. The higher the computed value is, the better fitted are the individuals. Let us suppose, that the initial population is composed of 500 individuals. We use some of the concepts provided by Apache Commons Mathematics Library in our implementation, among others the *elitism rate,* which specifies the percentage of the individuals with the highest fitness value to be taken over / cloned to the new generation. We use an elitism rate of 10 percent, i.e., the 50 best individuals will be taken without any changes of the genes to the new generation.

The population of each generation remains constant in time. In order to choose the remaining 450 individuals (parents) to generate the next generation, we use the *tournament selection* [28] including the implementation of the Apache Commons Mathematics Library. The tournament strategy can be configured by the *arity* of the tournament strategy, which specifies the number of individuals who take part in the tournament. For our purpose, five individuals in the tournament proved to be efficient. Accordingly, five individuals are selected randomly out of the total population of 500 individuals to take part in the tournament. Out of the individuals taking part in the tournament, the fittest individual is selected as a parent for the new generation. This way, 500 parents are selected out of a population of 500 individuals. These parents are paired aleatorically and they always have two descendants. This way, the next generation is created. Accordingly, the size of each generation remains constant.

We use two major strategies in order to improve the quality of the genes of the descendants, the *crossover strategy* described in [29] and the *mutation strategy*. Generally speaking, during the crossover phase, the two descendants receive the partly interchanged genes of their parents. Additionally, some particular genes can suffer mutations. The general strategy to generate the descendants is based on random decisions.

We describe in brief the creation strategy of the new generation. Some parameters are freely configurable, in order to assure best performance. Thus, the *crossover rate,* i.e., the threshold of the probability that a crossover is performed, has to be set in advance. Then, a crossover is performed if a randomly generated number is less than the crossover rate. Same is true regarding the mutation rate.

The crossover policy is quite straightforward. The position and length of the genes to be crossed over are randomly generated and the two descendants have receive the interchanged genes of their parents. In our case, this policy has been improved, such that by in the end, the number of the bins of at least one descendant is using, is lower than (or if this is not possible equal to) the number of the bins of their parents. This way, the reduction of the number of bins an individual is using, is enforced by the crossover policy itself.

The mutation policy is also very intuitive. In addition to the mutation rate, which defines in the end, whether mutation is applied after the crossover phase or not, the exchange rate indicates whether a slot (IC) is to be renewed. Analogously, the mutation policy can be configured such that the number of bins the descendant is using is reduced or in worst case, kept constant.

C. Performance Results

The benchmarks were performed on a Intel® Core$^{\text{TM}}$ i5-6500 CPU (quad core CPU 3.2 GHz, 16 GB RAM) running on Windows 10 and Eclipse 3.7.0 using Java SE Runtime Environment 1.6.0_22. The genetic algorithm was implemented using the Apache Common Library, version 3.0. The test data is a subset of the production environment and contained 5518 ICs in 261 boxes, having 28 measurements on average. The restricted test data is a subset of the production environment and contained 27,590 ICs in 1,305 boxes. Due to the incomplete set of production data, only the two most critical modules are considered for selection. After taking into account the attributes corresponding to the specifications of the two modules regarding the ICs (article, number of pins, etc.) only eleven boxes contain ICs to be considered for the selection process. We term *pre-selection* the method to restrict the number of boxes by excluding those boxes which do not contain selectable elements. In this way, the search area can be drastically reduced and thus, the performance of the selection algorithm can be substantially improved.

We use a generation size of 500 individuals, an elitism rate of 10 percent and an arity value of 5. The number of generation is limited to 1000 and the runtime of the selection algorithm is limited to 300 seconds. The other parameters like the crossover rate and the mutation rate are configured on a case by case basis. Regarding the fitness function, the following configuration parameters have proved themselves as good choice: attribute weight = 1; measurement weight = 2; bin weight = 5. This means especially, that fulfilling the bin constraints is the most difficult one. In summary, we use the configuration parameters given in Table I for the performance tests.

TABLE I: SETTINGS OF CONFIGURATION PARAMETERS.

Configuration parameter	Value
Population limit	500 individuals
Generation limit	1000 generations
Crossover policy	Bin reduction
Crossover rate	78 %
Mutation rate	13 %
Runtime limit	300 seconds

The prediction of the results of the selection algorithm is hardly possible, since we use random strategies to generate the initial population, to select the parents for the next generation, to determine the crossover and mutation policy. Moreover, parameters like the elitism rate and the arity have to be configured. Hence, the interaction between many factors that can influence the success and performance of the selection algorithm is not obvious.

It is not the aim of this study to deliver the possible best solution in an acceptable time frame and to improve the performance of the algorithm. Instead, our objective is to deliver an acceptable solution, i.e., a solution that fulfills the required constraints, for the industry to a crucial problem regarding their production problems. For example, there is no technological benefit of tightening the measurement constraints; the bin constraint was set up in such a way that seeking a lower value is not possible due to the fixed number of ICs of a work order and to the maximal capacity of the bins. Hence, the acceptable solution is also the best possible solution.

Nevertheless, we tried to improve the selection algorithm by testing the influence of the parameters, we find out to be decisive. This was also the case for the parameters of the fitness function as described above.

As already mentioned, we have an algorithm in place, which can find

a) a suboptimal solution in a heuristic way,
b) determine exactly whether the group of bins contain ICs which satisfies the specification of a particular work order.

 ISBN: 978-1-61208-752-8

Figure 5 shows that the selection algorithm using pre-selection delivers the expected results, finding individuals having 19 modules. The *success rate,* i.e., the probability that the selection algorithm reaches with an individual the given number of modules, is over 60 percent and thus, high enough for practical systems. The pre-selection strategy is very straightforward and easy to implement. Thus, no practical system would renounce to it. Nevertheless, when neglecting the benefit of reducing the search space by using pre-selection, the results of the genetic algorithm are not always as promising as with pre-selection. In order to evaluate worst-case scenarios, we used work that posed a lot of difficulties to select with the heuristic algorithm in place. As illustrated in Figure 5, the

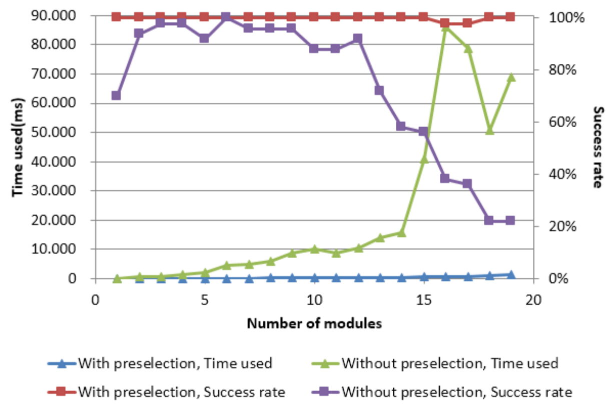

Figure 5: Success rate and time used depending on the number of modules with and without pre-selection.

success rate to select 19 modules as in the previous case, is at 60 percent. This means especially, that the successful run of the genetic algorithm heavily depends on the random numbers that were generated.

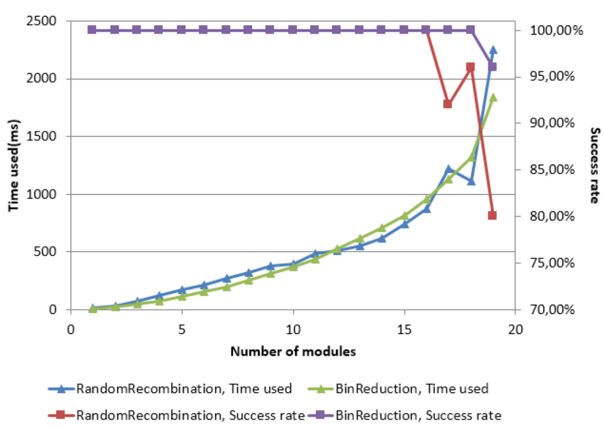

Figure 6: Success rate and time used when selecting the random recombination crossover policy or the bins reduction policy.

Figure 6 shows the difference between the random recombination crossover policy and the bins reduction policy. The boxes reduction crossover policy tries to reduce the number of boxes of the new individuals by focusing on the common bins of the parents. As a conclusion, using business logic over general approach, the general approach is as expected slower and has a lower success rate. This is the price to pay for using a more general solution over a customized one.

Figures 7 and 8 show the influence of the crossover rate and

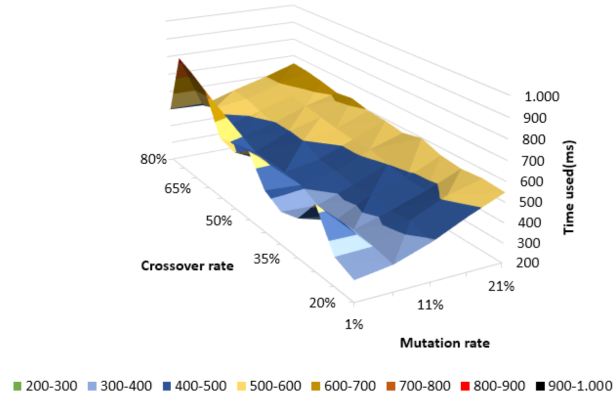

Figure 7: Time used depending on the crossover rate and the mutation rate.

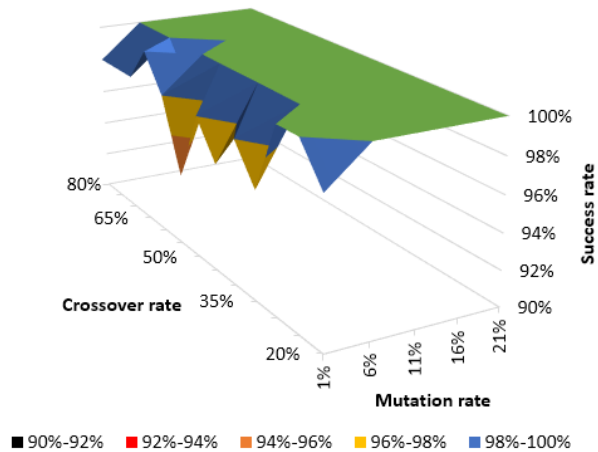

Figure 8: Success rate depending on the crossover rate and the mutation rate.

the mutation rate to the success rate and the wall clock time. As not obvious at first glance, a smaller crossover rate and a higher mutation rate gives better values for the success rate. Keeping the crossover rate and the mutation rate low, better run time performance is achieved. Generally speaking, high mutation rate can destroy the structure of good chromosomes, if used randomly [30]. The above remark does not hold in our case, since we do not exchange ICs randomly, but according to our strategy to minimize the number of bins.

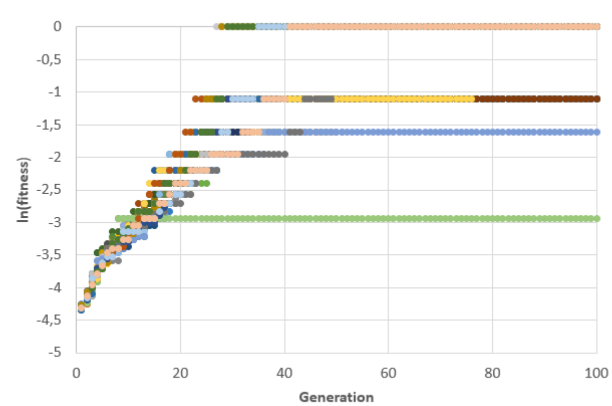

Figure 9: Logarithmic representation of the values of the fitness function depending on the number of generations (50 attempts).

The tendency of the convergence of the fitness function is visualized in Figure 9. The graph shows that in the end all 50 threads converge after some generations, but only a subset to the envisaged value. Recall that the maximum value of the fitness function is per definition equal to 1, the higher the value of the fitness function, the better the solution. The values of the fitness functions are discrete, $\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots\}$.

V. Resuming on Multi-Objective Optimization

Our preferred implementation framework is *Apache Commons Mathematics Library*, version 3.0 [31]. However, a very similar combinatorial grouping problem, the *Bin Packing Problem (BPP)* is investigated [32], by using the off the shelf *jMetal* framework [33]. The (one-dimensional) BPP [34] is defined as follows: given an unlimited number of bins with an integer capacity $c > 0$ each, a set of n items, $N = \{1, 2, ..., n\}$, and an integer weight $w_i, 0 < w_i \leq c$ for each item $i \in N$ assign each item to one bin, such that the total weight of the items in each bin does not exceed c and the number of bins used is minimized.

Luo et al. [32] extends the base implementation of jMetal to problems with dynamic variables. This was necessary, since the number of the genes in chromosomes is fixed in the base implementation of jMetal. However, the number of the genes in the specific implementation of the chromosomes for BPP – termed group based representation – is fluctuating; they vary in length depending on how many bins are used in every solution. Accordingly, the adopted implementation of BPP includes specific adaptations and enhancements of the basic primitives of jMetal, including those for chromosomes, crossover and mutation. The need for dynamic variables is justified by difficulties to use other solutions due to the fitness function.

In order to evaluate the performance of their algorithms – termed GABP –, Luo et al. [32] use public bench data as well as self-created big data sets. The performance of GABP does not differ very much from some of the known implementation of BPP. The main benefit of GABP is the implementation in a generic framework. However, the problem described in this article, the *Matching Lego(R)-Like Bricks Problem (MLBP)* is new to our knowledge, we are now aware of any implementation of a similar problem. The nearest problem to the MLBP seems to be BPP.

It seems that Luo et al. [32] used the fitness function as given below (termed cost function) [35] for their *group-based encoding scheme*:

$$f_{BPP} = \frac{1}{N_u} \cdot \sum_{i=1}^{N_u} \left(\frac{fill_i}{c}\right)^k \tag{2}$$

with N_u being the number of bins used, $fill_i$ the sum of sizes of the objects in the bin i, c the bin capacity, and k a constant, $k > 1$. In other words, the cost function to maximize is the average over all bins used, of the k-th power of the bin's utilization of it's capacity. The authors state that experiments show that $k = 2$ gives good results. As Falkenauer and Delchambre [35] point out that one of the major purpose of the fitness function is to guide the algorithm in the search.

Although, both BPP and MLBP yield a reduction of the number of bins, the fitness function of the algorithms are different. The strategy at BPP is to pack the bins as full as possible, i.e., a bad use of the capacity of the bins leads to the necessity of supplementary bins [35]. On the contrary, the suggestion for the fitness function for MLBP is given by the need to fulfill the constraints.

By comparing the formula (1) for the fitness function of MLBP with the formula (2) as above, it is obvious that both formulas are very similar, both are expected to be maximized, contain summation over parameters of the respective problems and the possibility to optimize the execution time of the respective algorithms through clever setting of constants. In this respect, both MLBP and BPP substantially benefit from an ingeniously designed fitness function to ensure fast convergence towards the optimization goals [35].

Although, in concept MLBP is merely a constraint satisfaction problem, the implementation as described in this article, can be easily adapted to simulate an optimization problem. Within the current algorithm, the number of required bins is fixed. This assumption is perfectly reasonable from an industrial perspective, since the number of the objects in the bins is more or less the same and as the number of objects needed for a work order is known, the minimal number of bins necessary to fulfill the work order is thus determined. Besides, the maximum number of bins that can be accessed by a machine is fixed, less beans means just a vacant working place. However, by setting the bin constraints to a lower value, – while keeping the other constraints unchanged – and by modifying the exit criteria accordingly, the algorithm will loop – delivering better solutions according to the guidelines of the fitness function – until the new exit criteria are met. Thus, a local extremum relating to the fitness function is found. The local extremum is not automatically a solution to the constraint satisfaction problem, this has to be validated. The exit criteria only ensure that the best local value of the fitness function and corresponding physical entities are found. In this respect, the MLBP and BPP are closely related problems.

There are other systems, which provide frameworks of evolutionary algorithms, such as EvA2, OPT4j, ECJ, MOEAT, for a discussion and bibliography see [35]. Moreover, a remarkable attempt to obtain a deeper understanding of the structure of the BPP by using Principal Component Analysis and repercussions on the performance of the heuristic approaches to solve them, was undertaken [36].

The aim of jMetal was to set up a Java-based framework in order to develop meta heuristics for solving Multi-objective Optimization Problems (MOP). jMetal provides a rich set of Java classes as base components, which can be reused for the implementation of generic operators, thus making a comparison of different meta heuristics possible [37] [38].

Unfortunately, the Apache Commons Mathematics Library deployed in our use case, does not support multi-objective optimization mechanisms. This means especially, that multi-objective optimization have to be simulated by single-objective optimization. For example, optimization criteria for MLBP are a) reducing the number of bins, b) fulfillment of the measurement constraints, c) fulfillment of the attribute constraints. These criteria are independent of each other and ideally within the multi-objective optimization they can be optimized independently, such that an improvement of a criterion does not lead to a degradation of another one. For practical purposes, the fitness function, see formula (1), can be used for simulating multi-objective optimization by choosing adequate weight function. Thus, by choosing the values $w = 5$ for the criterion (a), $w = 2$ for criterion (b) and $w = 1$ for criterion (c) we achieve fast convergence and hence reduced execution time, but for example by improving criterion (a) we cannot avoid the degradation of criterion (b) or (c). By using frameworks which support multi-objective mechanism, better convergence of the genetic algorithm is expected.

The jMetal project was started in 2006 [37] and since then it underwent significant improvements and major releases [39], such that the redesigned jMetal should be useful to researchers of the multi-objective optimization community, such as evolutionary algorithm, evolution strategies, scatter search, particle swarm optimization, ant colony optimization, etc., [40]. Improvements regarding a new package for automatic tuning of algorithm parameter settings have been introduced [41] in order to facilitate accurate Pareto front approximations.

In addition, jMetal in conjunction with Spark – which is becoming a dominant technology in the Big Data context – have been used to solve Big Data Optimization problems by setting up a software platform. Accordingly, a dynamic bi-objective instance of the Traveling Salesman Problem based on near real-time traffic data from New York City has been solved [42] [43].

VI. Conclusion and Future Work

The main challenge, which led to the results of this paper, was to investigate whether a real-life combinatorial problem, which popped up at a semiconductor company, can be solved within a reasonable time. Moreover, the solution should be flexible, such that the solution is not restricted to the existing specification of the modules.

We established an abstract formal model, such that the implementation of the use case is fully functional within the boundaries of this model. In this sense, new constraints can be added to the existing ones, no inside knowledge regarding the structure of the module is needed, as it is the case for the heuristic algorithm in place.

We set up a genetic algorithmic approach based on the Apache Commons Mathematics Library, implemented it and validated the results. Some decisive policies like the crossover and mutation policy have been additionally implemented and new optimizations like the bin reduction crossover policy have been set up to improve the convergence of the genetic algorithm. The performance results were satisfactory for an industrial application.

The current implementation does not combine good genes (taking into account all the constraints, like measurement, etc.) of the parents. Instead, the crossover strategy is based on random decisions. Additional research is necessary in this direction to find a good balance between a more general suitability (random decisions) and good convergence (adjusted crossover policy). For the time being, only the fitness function contains proprietary information regarding the production process, any other decision is aleatoric.

The implemented basic framework is very flexible, it has many configuration possibilities like the elitism rate and the arity. As a consequence of the random variables, many convergence tests with various configuration assignment have to be performed, in order to ensure satisfying results. Furthermore, of crucial importance for the successful completion of the algorithm is the design of the fitness function, especially the values of the weights.

The convergence tests show that not every execution will succeed to find the best solution delivered, this is due to the random numbers used through out the genetic algorithm. This is exemplified by the success rate. Thus, the selection algorithm based on genetic strategies always delivers local maxima, which may substantially differ from the global one. Our attempt to find the optimal solution using MATLAB for a reduced set of ICs failed due to the long execution time.

As already mentioned, the fitness function plays an outstanding role during the selection of the best candidates for the next generation. This means especially, that two candidates having the same value of their fitness function are considered of the same quality. This assertion is not accurate enough, due to the use of three different weights, whose interdependence can hardly be anticipated. Each weight represents the significance of one aspect of the quality of a candidate. To circumvent this dilemma, Pareto optimality [27] can be used to solve the challenge of the multi-objective function. In this case, a new framework [37] is needed, since Apache Commons Mathematics Library does not support multi-objective mechanism. Genetic algorithms, if configured properly, can be used to solve our constraint satisfaction problem. The delivered solution may substantially differ from the optimal one.

The current problem is not defined as an optimization problem, the constraints of a work order are either satisfied or not. Accordingly, two different solutions of the same work order, which satisfy the constraints, are of the same quality. However, the genetic algorithm is based internally on an optimization process – the higher the value of the fitness function, the better the solution. The constraints are used as exit criterion for the genetic algorithm. In this way, the optimization is stopped arbitrarily, considering that a better solution is out of scope.

Moreover, during the production process multiple work orders have to be honored simultaneously. The current strategy at the semiconductor company adopted a sequential one. We can reformulate the problem as an optimization problem: Given a list of work orders, find the maximum number of work orders that can be satisfied simultaneously.

The elapsed time till the genetic algorithm of MLBP converges is in range of seconds, by all means satisfactory for the investigated industrial application. As expected, not all the execution threads converge to the same solution, and not all the threads find an optimal solution, as shown in some cases less than 60 percent. Therefore, starting a bunch of threads within the genetic algorithm increases the chance towards better solutions.

Acknowledgment

We acknowledge the assistance and helpful comments of the anonymous referees.

REFERENCES

[1] B. Korte and J. Vygen, Combinatorial Optimization: Theory and Algorithms, ser. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2018.

[2] P. M. Pardalos, D.-Z. Du, and R. L. Graham, Handbook of Combinatorial Optimization. Springer, 2013.

[3] J. Puchinger and G. Raidl, "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification," Lecture Notes in Computer Science, vol. 3562, 06 2005, pp. 41–53, doi: 10.1007/11499305_5.

[4] Krzysztof Apt, Principles of Constraint Programming. Cambridge University Press, 2003, Retrieved: October 2019. [Online]. Available: https://doi.org/10.1017/CBO9780511615320

[5] A. L. Corcoran, Using LibGA to Develop Genetic Algorithms for Solving Combinatorial Optimization Problems. The Application Handbook of Genetic Algorithms, Volume I, Lance Chambers, editor, pages 143-172 CRC Press, 1995.

[6] E. Andrés-Pérez et al., Evolutionary and Deterministic Methods for Design Optimization and Control With Applications to Industrial and Societal Problems. Springer, 2018, vol. 49.

[7] D. Greiner et al., Advances in Evolutionary and Deterministic Methods for Design, Optimization and Control in Engineering and Sciences. Springer, 2015, vol. 1, no. 1.

[8] D. Simon, Evolutionary Optimization Algorithms. John Wiley & Sons, 2013.

[9] A. Pétrowski and S. Ben-Hamida, Evolutionary Algorithms. John Wiley & Sons, 2017.

[10] O. Kramer, Genetic Algorithm Essentials. Springer, 2017, vol. 679.

[11] T. Bäck, D. B. Fogel, and Z. Michalewicz, Evolutionary Computation 1: Basic Algorithms and Operators. CRC press, 2018.

[12] R. K. Belew, Adaptive Individuals in Evolving Populations: Models and Algorithms. Routledge, 2018.

[13] T. Bäck, D. B. Fogel, and Z. Michalewicz, Evolutionary Computation 2 : Advanced Algorithms and Operators. CRC Press, 2000, Textbook - 308 Pages.

[14] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic Research: a Comprehensive Survey," Artificial Intelligence Review, 2018, pp. 1–43.

[15] T. Jiang and C. Zhang, "Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases," IEEE Access, vol. 6, 2018, pp. 26 231–26 240.

[16] H. Zhang, Y. Liu, and J. Zhou, "Balanced-Evolution Genetic Algorithm for Combinatorial Optimization Problems: the General Outline and Implementation of Balanced-Evolution Strategy Based on Linear Diversity Index," Natural Computing, vol. 17, no. 3, 2018, pp. 611–639.

[17] X. Li, L. Gao, Q. Pan, L. Wan, and K.-M. Chao, "An Effective Hybrid Genetic Algorithm and Variable Neighborhood Search for Integrated Process Planning and Scheduling in a Packaging Machine Workshop," IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2018.

[18] I. Jackson, J. Tolujevs, and T. Reggelin, "The Combination of Discrete-Event Simulation and Genetic Algorithm for Solving the Stochastic Multi-Product Inventory Optimization Problem," Transport and Telecommunication Journal, vol. 19, no. 3, 2018, pp. 233–243.

[19] J. C. Bansal, P. K. Singh, and N. R. Pal, Evolutionary and Swarm Intelligence Algorithms. Springer, 2019.

[20] G. Raidl, J. Puchinger, and C. Blum, "Metaheuristic hybrids," Handbook of Metaheuristics, Vol. 146 of International Series in Operations Research and Management Science, 09 2010, pp. 469–496, doi: 10.1007/978-1-4419-1665-5_16.

[21] J. Holland, Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor. 2nd Edition, MIT Press, 1992.

[22] G. P. Rajappa, "Solving combinatorial optimization problems using genetic algorithms and ant colony optimization," 2012, PhD diss., University of Tennessee, Retrieved: October 2019. [Online]. Available: https://trace.tennessee.edu/utk_graddiss/1478

[23] S. Yakovlev, O. Kartashov, and O. Pichugina, "Optimization on Combinatorial Configurations Using Genetic Algorithms," in CMIS, 2019, pp. 28–40.

[24] T. Weise, "Global Optimization Algorithms – Theory and Application," 2009, Retrieved: October 2019. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[25] Rui Song, "Substrate Binning for Semiconductor Manufacturing," Master's thesis, Dresden University of Technology, Faculty of Computer Science, Institute of Applied Computer Science, Chair of Technical Information Systems, 2013.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," IEEE transactions on evolutionary computation, vol. 6, no. 2, 2002, pp. 182–197.

[27] J. rey Horn, N. Nafpliotis, and D. E. Goldberg, "A Niched Pareto Genetic Algorithm for Multiobjective Optimization," in Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence, vol. 1. Citeseer, 1994, pp. 82–87.

[28] B. L. Miller, D. E. Goldberg et al., "Genetic Algorithms, Tournament Selection, and the Effects of Noise," Complex systems, vol. 9, no. 3, Champaign, IL, USA: Complex Systems Publications, Inc., 1995, pp. 193–212.

[29] R. Poli and W. B. Langdon, "A New Schema Theorem for Genetic Programming with One-Point Crossover and Point Mutation," Cognitive Science Research Papers-University of Birmingham CSRP, 1997.

[30] F.-T. Lin, "Evolutionary Computation Part 2: Genetic Algorithms and Their Three Applications," Journal of Taiwan Intelligent Technologies and Applied Statistics, vol. 3, no. 1, 2005, pp. 29–56.

[31] M. Andersen et al., "Commons Math: The Apache Commons Mathematics Library," URL http://commons. apache. org/math/, online, 2011, Retrieved: October 2019.

[32] F. Luo, I. D. Scherson, and J. Fuentes, "A Novel Genetic Algorithm for Bin Packing Problem in jMetal," in 2017 IEEE International Conference on Cognitive Computing (ICCC). IEEE, 2017, pp. 17–23.

[33] J. J. Durillo and A. J. Nebro, "jMetal: A Java Framework for Multi-Objective Optimization," Advances in Engineering Software, vol. 42, no. 10, Elsevier, 2011, pp. 760–771.

[34] K. Fleszar and C. Charalambous, "Average-Weight-Controlled Bin-Oriented Heuristics for the One-Dimensional Bin-Packing Problem," European Journal of Operational Research, vol. 210, no. 2, Elsevier, 2011, pp. 176–184.

[35] E. Falkenauer and A. Delchambre, "A Genetic Algorithm for Bin Packing and Line Balancing," in Proceedings 1992 IEEE International Conference on Robotics and Automation. IEEE, 1992, pp. 1186–1192.

[36] E. López-Camacho, H. Terashima-Marín, G. Ochoa, and S. E. Conant-Pablos, "Understanding the Structure of Bin Packing Problems Through Principal Component Analysis," International Journal of Production Economics, vol. 145, no. 2, Elsevier, 2013, pp. 488–499.

[37] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, "jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics," Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, ETSI Informática, Campus de Teatinos, Tech. Rep. ITI-2006-10, 2006.

[38] A. J. Nebro and J. J. Durillo, "jMetal 4.3 User Manual," Available from Computer Science Department of the University of Malaga, 2013, Retrieved: October 2019. [Online]. Available: http://jmetal.sourceforge.net/resources/jMetalUserManual43.pdf

[39] A. J. Nebro, J. J. Durillo, and M. Vergne, "Redesigning the jMetal Multi-Objective Optimization Framework," in Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM, 2015, pp. 1093–1100.

[40] J. J. Durillo, A. J. Nebro, and E. Alba, "The jMetal Framework for Multi-Objective Optimization: Design and Architecture," in IEEE Congress on Evolutionary Computation. IEEE, 2010, pp. 1–8.

[41] A. J. Nebro-Urbaneja et al., "Automatic Configuration of NSGA-II with jMetal and irace," ACM, 2019.

[42] J. A. Cordero et al., "Dynamic Multi-Objective Optimization with jMetal and Spark: a Case Study," in International Workshop on Machine Learning, Optimization, and Big Data. Springer, 2016, pp. 106–117.

[43] C. Barba-González, J. García-Nieto, A. J. Nebro, and J. F. A. Montes, "Multi-objective Big Data Optimization with jMetal and Spark," in EMO, 2017.