The influence of the picking times of the components in time and space assembly line balancing problems: an approach with evolutionary algorithms

Emanuel F. Alsina*, Nicola Capodieci*, Giacomo Cabri*, Alberto Regattieri†,
Mauro Gamberi‡, Francesco Pilati‡, and Maurizio Faccio‡
*Department of Physics, Informatics and Mathematics
University of Modena and Reggio Emilia
Via Campi, 213/A, 41125 Modena, Italy
{emanuelfederico.alsina, nicola.capodieci}@unimore.it
†Department of Industrial Engineering
University of Bologna
Viale del Risorgimento, 2, 40136 Bologna, Italy
‡Department of Management and Engineering
University of Padova
Stradella San Nicola, 3, 36100 Vicenza, Italy

***Abstract*—The balancing of assembly lines is one of the most studied industrial problems, both in academic and practical fields. The workable application of the solutions passes through a reliable simplification of the real-world assembly line systems. Time and space assembly line balancing problems consider a realistic versions of the assembly lines, involving the optimization of the entire line cycle time, the number of stations to install, and the area of these stations. Components, necessary to complete the assembly tasks, have different picking times depending on the area where they are allocated. The implementation in the real world of a line balanced disregarding the distribution of the tasks which use unwieldy components can result unfeasible. The aim of this paper is to present a method which balances the line in terms of time and space, hence optimizes the allocation of the components using an evolutionary approach. In particular, a method which combines the bin packing problem with a genetic algorithm and a genetic programming is presented. The proposed method can be able to find different solutions to the line balancing problem and then evolve they in order to optimize the allocation of the components in certain areas in the workstation.**

I. Introduction

In the standardized industrial production world as well as in the production of customized products, assembly lines are flow oriented systems extremely diffused. An assembly line consists in a set of *work stations* settled arranged in series or parallel, creating a flow of assembling operation. The pieces of semi-assembled are consecutively moved from one station to the next, through a conveyor or a similar mechanical material handling equipment, supplying continuously the various stations[1].

The assembling process is divided into a set of *tasks* which are cyclically performed. Each assembly task j requires a different *operation time* t_j for its execution. The assignment of the *workload* (i.e., a subset of tasks) to each station respecting some constraints or objectives, is one of the most usual and hard problems in the field, known as the *assembly line balancing* (ALB) problem [2], [3], [4]. Generally, each station has a fixed common time to complete its tasks, called *cycle time* CT, after which the semi-assembled piece is transported from one station to the subsequent. Each station has a *station workload time*, that is the cumulative operation times of the workload assigned to the station. If the cycle time is imposed, a line is considered balanced and feasible only if the workload time of each station does not exceed the CT. Performing a task does not require only a certain time, but also a series of other factors, i.e., the completion of previous tasks, equipment of machines, components, skills of workers, and so on. ALB problems consist in assign all the tasks to the various stations, respecting the constraints of time and further. In other words, the goal is to assign to each station a group of tasks that minimizes the inefficiency of the line (its downtime) and that respects all the constraints imposed on the tasks and on the stations [5]. Due to the complexity of the problem and the great variety of possible assembly lines, the previous research activities have focused in one or more of those aspects and the fulfilling of certain restrictions in the stations.

The more simple family of problems considers the time and the precedence of some tasks respect other as the only constrains. The modeling and solving of these problems are called *simple assembly line balancing* (SALB) [6]. The purpose of this family of problems is principally to minimize the number of station (given a fixed cycle time), or minimize the cycle time (given a fixed number of stations). Or further, simultaneously minimize the cycle time and number of station achieving multi-objective cost and profit purposes.

However, the assumptions made in the SALB problems are very restricting with respect to real-world assembly line systems. When other constraints and considerations are added to the SALB, the problems are called *general assembly line balancing* (GALB) [1]. This family of problems wants to sew up the gap between the academic discussion and prac-

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.148

tical applications. GALB problems can consider for instance equipment selection and cost [7], parallel stations [8], U-shaped line layout [9], among others [10]. Belonging to this family, Bautista and Pereira [5] defined a set of problems that they call time and space constrained assembly line (TSALB), where the spatial constraint of the components necessary to operate the tasks is considered. In fact, in the assembling of big components (i.e., automotive industry) the items normally are allocated in designated areas close to each station. It is important to keep the components as near to the workplace as possible, considering the space limitations. According to Chica et al. [11] these kinds of problems contain three conflicting objectives to be accomplished: the cycle time of the assembly line, the number of the stations, and the area of these stations. Bukchin [12] considered that the components can be allocated in some traditional shelves or boxes, studying the TSALB problems focused on the dimensions of these containers for different components and their allocation along the line. However, TSALB problems do not consider different picking times of the components according to where they are allocated. These picking times can depend by their distance from the worker, the weight and manageability of the components themselves. Usually, racks for small components (screws, bolts, etc.) are arranged very close to the worker, in order to limit the number of travels to pick up a large number of small components. While bigger and often heavier components are arranged behind the worker much or less close to her. Solving a TSALB problem do not considering the picking time of the components could to culminate in a solution which assign to a single station several tasks with a lot of heavy components which increase more and more their time to be picked up. This because heavy components have limited space close to the worker. It is not just an issue of adding the pickup time to the task time, because the picking time can be different according to the area where the components are allocated. For this reason, our study considers the physical allocation of the components necessary to complete the different tasks, and the consequently optimizing of the global picking time to achieve them. This paper proposes two models to optimize the allocation of the components in different areas within the stations, ensuring the respect of the cycle time. In particular, the main contribution of this study is to consider the physical allocation of the components necessary to complete the different tasks, and the consequently optimizing of the global picking time to achieve them.

The present paper is organized as follows. Section II introduces the evolutionary algorithms used in the proposed approaches. In Section III a mathematical formulation of the problem is provided. The method to optimize the allocation of the components is presented in Section IV. The limitations of the proposed approaches are explored in Section V. Finally, some conclusions of this study are presented in Section VI.

II. Evolutionary Algorithms

Taking into account the large number of constraints imposed by the GALB problems, traditional search techniques for optimal solutions may not be the best approach. That can be true especially when the number of stations and tasks to take in consideration are particularly high. These traditional search techniques include a large variety of exact, heuristic, and metaheuristics procedures, mainly based on branch and bound and dynamic programming approaches [13]. We therefore investigating towards the use of a bio-inspired approach as an alternative artificial intelligence method for exploring the search space. Bio-inspired artificial intelligence is a term that groups together several computational models and algorithms that are designed in order to mimic the efficiency of complex mechanisms that are observable in nature. Genetic algorithms (GA) [14], [15] are a branch of bio-inspired intelligence; such algorithms are population based in the sense that an initially randomized population of chromosomes represents the starting point for this mechanism. Different chromosomes represent a different solution for the same optimization problem, and each chromosome is encoded as a set of parameters needed for defining such solution. During the course of a generation, the population of solutions get through the evaluation phase in which each chromosome is tested in order to evaluate its fitness value (how well that specific solution is solving the problem); the subsequent phase of a standard genetic algorithm is the selection phase in which chromosomes featuring higher fitness values are more likely to survive and to be inserted into the next iteration (generation). Generation after generation, the selected individuals undergo a process of evolution and/or mutation, so to expand the search space in the vicinity of best performing chromosomes. In order to maintain diversity and maximize the exploration capabilities of the population, individuals that are not selected are usually replaced by randomized chromosomes. Rubinovitz and Levitin [16] applied a GA to solve a SALB, recommending the use of those algorithms when solutions diversity is more important than their accuracy. The authors noted how GAs perform much faster then traditional search techniques for problems with large number of stations. Several studies have been explored further, mainly to cope with the multiple objectives of an assembly line [17], [18], [19], [20].

Depending on the dimensionality and complexity of the optimization problem to solve, GAs may have the tendency to get stuck into local optima or may require an unfeasible number of generations, therefore in this paper we also investigate the use of genetic programming (GP) [21]. GP is a specialization of GA in the sense that the chromosomes are not encoded as set of parameters needed for the specific problem, instead GP uses a set of actions represented as nodes in tree structures. Nodes can be extrapolated from known heuristics so to combine sequences of actions with the purpose of evolving programs (instead of parameters) towards the optimal solution. Practically, GP allows the induction of computer programs to solve problems without explicitly programming them. Liu et al. [22] adopted a GP structure to solve a GALB problem which considers parallel assembly lines. Baykasouglu and Özbakır [23] tried to solve SALB problems using composite task assignment rules which are discovered through a GP structure. The purpose of the authors was to evolve different heuristics in order to optimize a predefined objective function. In this way the GP can adapt to solve different problem constraints and pursue different objectives. Their results argued the ability and potential of GP for solving combinatorial optimization problems. Some other few studies have been developed to generate heuristic decision rules in manufacturing system applications [24], [25], [26], [27].

GAs and GP can be applied to almost any optimization problem [28], however in our specific case of GALB problems,

it can be useful to investigate a pre-processing phase in which we can use with the purpose of relaxing some constraints so to apply an evolutionary strategy (GA or GP) to a problem with a smaller dimension compared to the one we started. The well known Bin Packing problem [29] for instance, perfectly suits this need: having defined our problem to optimize an assembly line taking into account the cycle time CT, number of components Z, number of picking areas and respective picking times with an unknown number of assembly stations K, we can apply a 1D bin packing problem in order to find the minimum number of stations in order to satisfy the CT constraint. We can then apply an evolutionary strategy to the remaining objectives.

III. Problem formulation

As introduced in the Section I, the assembly process is divided into a set of J tasks. Each task j requires a positive operation time t_j to be accomplished. Assigning to each station k (k = 1, 2, ..., K) a subset of tasks S_k to be completed, each station will have a station workload time $t(S_k)$ that is equal to the sum of the tasks' times assigned to this station. Each station have to respect the known cycle time CT of the assembly line, therefore the $t(S_k)$ of all the stations has to be lower or equal to CT. Each task j can be assigned only to a single station k, and all the tasks has to be assigned to one station. In this preliminary study, we hypothesize that each task is independent from each other, in other words there are no precedence constraints among the operation of different tasks. Other hypothesis include:

- Assembling is referred to a single product;
- Setting up times of stations and move time between them are negligible;
- No more than one task can be execute at the same time in the station;
- Task times are known and deterministic;
- Each station is equally equipped with respect to machines and workers.

In addition to those time constraints, space constraints have to be considered. Indeed, each task j has a list of components which are necessary to complete the task. They include different materials and semi-assembled parts that have to be carry to the work station to be assembled. Each of these components z has obviously a volume. Each station k has a limited volumetric area A_k where the components can be allocated. Hypothesizing that the components require a volume simplified to a parallelepiped, each task will require an area equal to the sum of the areas of the components necessary to complete it; and each station will require an allocation area equal to the sum of the areas of all the components of each task assigned to that station. This hypothesis does not make the model dissimilar from the reality. In fact often the components are transported along the assembly line through boxes. The required area for the components must be not larger than the available area A_k of the station k. We assume that the available area A_k is the same for all the stations. We divided this space hypothetically in three areas: Area 1, Area 2, and Area 3, as shown in Figure 1. Each area w has a different available space, in terms of volume. Each component has a different picking time PT_{zw}, according to the area where it is allocated and its manageability. The picking time in the different areas is something a priori known in the problem. A box easy to transport can have a monotonic increasing of the picking time, due only to the distance from the worker. But some unwieldiness components can have non linear picking times in the different areas. In this preliminary study, we supposed that the picking times are equal to all the stations. It means that the picking time PT_{zw} of the component z in the area w is equal in the station 1, 2, and k.

Generally, declaring the following variables:

$j = 1,\ ...,\ J$	Tasks
$z = 1,\ ...,\ Z$	Components
$k = 1,\ ...,\ K$	Assembly stations
$w = 1,\ 2,\ 3$	Allocation areas within each station

The modeling of the TSALB problem considering the allocation of the components is based on the mathematical formulation of the SALB problem provided by Patterson and Albracht [30]. In addition to those time constraints, constraints concerning the dimensional parameters of the components have to be considered. Therefore, using the following decision variables:

$$X_{jk} = \begin{cases} 1, & \text{if task } j \text{ is assigned to station } k \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$Y_{zw} = \begin{cases} 1, & \text{if component } z \text{ is stored in the area } w \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The following constraints were established in our TSALB problem:

$$\sum_{k=1}^{K} X_{jk} = 1 \quad \forall j \tag{3}$$

$$\sum_{w=1}^{3} Y_{zw} \geq A_{jz} \cdot X_{jk} \quad \forall j,\ z,\ k \tag{4}$$

$$\sum_{j=1}^{J} (t_j \cdot X_{jk} + \sum_{z=1}^{Z} A_{jz} \cdot \sum_{w=1}^{3} PT_{zw} \cdot Y_{zw}) \leq CT \quad \forall k \tag{5}$$

$$\sum_{z=1}^{Z} A_z \cdot Y_{zw} \leq A_{wk}^{max} \quad \forall w,\ k \tag{6}$$

$$X_{jk},\ Y_{zwk} \in \{0,1\} \quad \forall j,\ z,\ w,\ k \tag{7}$$

Equality 3 ensures that each task is assigned to only one assembly station. Constraint 4, where A_jz is equal to 1 if the task j requires the component z, ensures that all the components necessary to a task will be allocated within the station. Inequality 5 ensure that each station workload time does not exceed the cycle time. In this constraint, PT_{zw} is the picking time of the component z from area w. And finally,

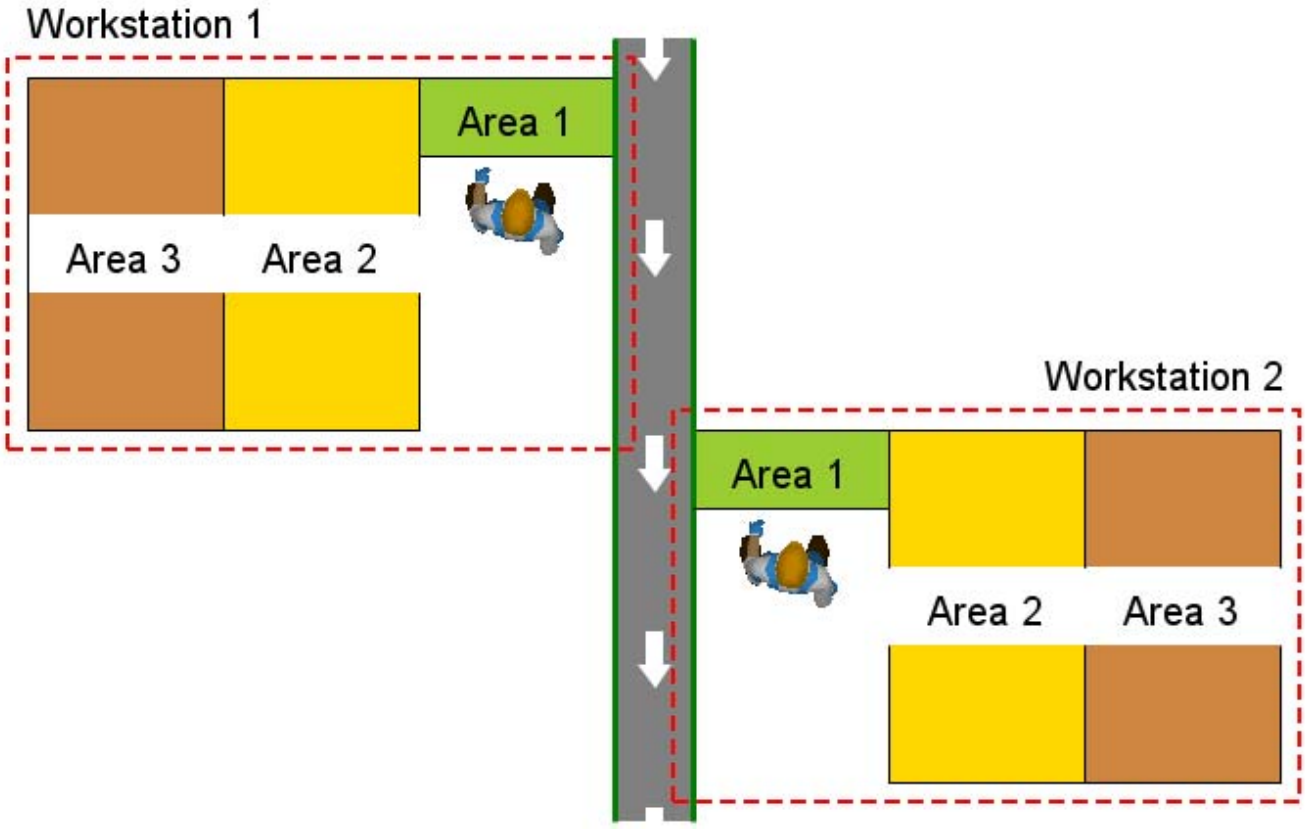

Fig. 1. A representation of an ideal assembly line with different available areas where to allocate the components.

6 is concerned with the physical area upper bound, where A_{wk}^{max} represents the limit of the allocation areas. 7 defines the domain of the decision variables. The variable Y will provide the assignment of the components to specific allocation areas, while the variable X will provide the assignment of the tasks to specific assembly stations. The objective is to minimize the the inefficiency of the line (i.e. the downtime of the stations); that in objective functions become:

$$\min(CT - \sum_{j=1}^{J}(t_j \cdot X_{jk} + \sum_{z=1}^{Z} A_{jz} \cdot \sum_{w=1}^{3} PT_{zw} \cdot Y_{zw})) \quad (8)$$

The main idea of our TSALB problem is that given a set of tasks with their temporal and spatial attributes, each task must be assigned to just one station providing that (1) there is not any station with a workload time greater than the cycle time, (2) there is not any station with a required area greater than the global available area A, and (3) the components are allocated along the line in order to optimize their global picking time.

IV. Proposed approach

In this section we propose the algorithms that can be used to solve the TSALB problem presented in Section III. Like previously stated, the way we are going to address such problem is to logically decompose the whole problem into two phases. At the end of the first phase, we can rely on several starting conditions and all of them have to obey the CT constraint: by operating this first preprocessing phase, we can focus the rest of the algorithms in order to the search of the optimal solutions without taking into account the CT constraint.

A. Bin Packing problem

This first phase is represented by solving a 1D *bin packing problem* (BPP) using time as the only dimension. BPP was already used in past to solve SALB problems [31], [32], [33]. In our case, each bin represents a station with its maximum time for processing being equal to CT: having a plurality of stations working in parallel ensures that cycle time constraint is respected. The items to pack are the tasks, with each task having its own processing time. So, simplifying the Equation 5 to consider only the execution times, the BPP trivially the following condition holds:

$$\sum_{j=1}^{J} T(t_j) \leq CT \quad (9)$$

With J the number of tasks within the same station/bin, $T(t_j)$ the time needed for complete task j. We tested an implementation of BPP able to provide an exhaustive search that exploit the concept of branch and bound and propagation algorithms preprocessed with a First Fit Decreasing

strategy [34]. The Java implementation we used for the BPP is able to solve a moderately sized problem (26 unique tasks, 75 components, real world application data) in under 10 ms using a Intel CORE i7, jre 1.8.40, 16 GB of RAM, therefore we can conclude that it should be feasible to apply a 2 or 3D BPP for similarly sized problems, so to satisfy more constraints in this initial phase; however, as we are conducting a preliminary investigation on such class of problems, we consider this phase just to be applied to the 1D model.

The BPP algorithm provides two important output in this first phase: (1) the minimum number of stations needed in order to satisfy CT, and (2) the list of tasks to be executed in each station particular station. The minimum number of stations K can be also mathematically verified, taking into account the execution time of each task, picking times, and cycle time of the line:

$$K = \left\lceil \frac{\sum_j \left(t_j + \sum_z \left(\min_x \left(PT_{zw} \right) \right) \right)}{CT} \right\rceil \tag{10}$$

In Equation 10, the number of stations is defined as the sum of the execution times of all the tasks and the lowest picking time of each component; the result is then divided by the cycle time of the line. Concerning the tasks assigned to each station, BPP algorithm is biased to fill the first bins with smaller items, while the last bins are more likely to be filled with bigger items. More specifically to our case study, it is convenient to think that the stations are ideally located one next to the other on a straight line: the leftmost station is usually occupied with many tasks with small time requirements, while the rightmost station is most likely responsible of a small number of very time demanding tasks. This particular output configuration enables us to quickly find alternative and equivalent solutions: by exchanging tasks assigned to different stations that are characterized by a similar time requirement and/or by shifting low time demanding tasks from the leftmost bins to the rightmost bins (whenever it is possible) we are creating different starting conditions for the following phase of the algorithm (see Fig. 2). It is trivial to verify that the CT constraint is easily satisfiable in each newly created solution; we create as many solutions depending on the size of the problem and for future reference we indicate as S_{BPP} the set of solutions found with this mechanism.

B. Genetic algorithm

A genetic algorithm can be now applied in order to solve the picking time minimization objective. It is important to notice that up until now, we left the notion of components and picking areas out of any equation as we were only interested in the CT constraint: we can now forget about that constraint and therefore we will not take into account the different tasks for each station. Instead of thinking about tasks, we are now considering the list of N components that are needed for the completion of every task. We start by picking a specific solution $s \in S_{BPP}$ so that we can model each station as a series of n tuples with n being the number of different picking areas, three in our case. Each picking area features a variable number of components z_x, and both the components and the picking areas influence the total picking time that the

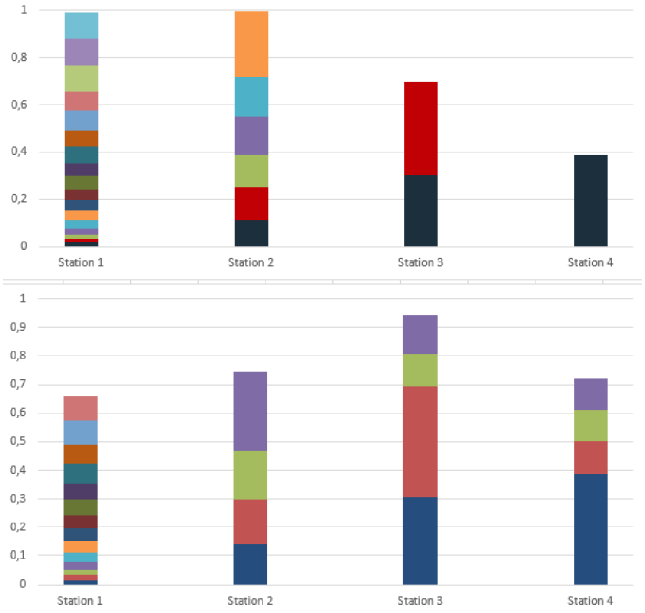

Fig. 2. The above graph displays the solution of BPP applied to our case study: it appears that four stations are a minimum requirement. It is important to note how tasks with larger time requirements ended up in the rightmost bins (different colors indicates different tasks. On the Y axis we can see the proportional time occupancy normalized from 0 to 1). The second graph below is an example of equivalent solution, obtained from the original one by switching tasks with similar time requirements among different stations.

human operator has to spend in order to process a specific component. To summarize all of this, for each of the station k we can encode a chromosome as seen in 11:

$$\begin{array}{l} Station_1 \langle c_0, \ldots, c_j \rangle \, p_0 \\ \vdots \quad \vdots \\ \vdots \quad \langle c_k, \ldots, c_n \rangle \, p_{n-1} \\ \vdots \\ Station_w \langle c'_0, \ldots, c'_j \rangle \, p_0 \\ \vdots \\ \langle c'_k, \ldots, c'_n \rangle \, p_{n-1} \end{array} \tag{11}$$

Random permutations of components within different picking areas give us our initial population of chromosomes (represented as in 11) that we use as a starting point for the genetic algorithm. Do note that permutations are only admissible within the same station. The search process begins with the first evaluation of every individual against the proposed objective function, hence for the selection of fittest individuals any selection operator normally used in literature is feasible. Similar conclusions can be drawn for mutation operators and elitism. The self-imposed constraint to contain mutations and generations of new individuals within the same station have pros and cons: the biggest advantage is represented by the fact that by restricting migration of single components only

within different picking areas of the same station we are still satisfying the CT constraint; on the other side, the limitation of this approach is that by doing so we are limiting the exploration space as we are optimizing locally each station instead of moving towards a global optimal solution that concerns every station. On this latter remark, it has to be highlighted that such a genetic algorithm can be easily implemented to run in a parallel way w.r.t. every element $\in S_{BPP}$, so to minimize the possibility of being stuck in local optima.

C. Genetic Programming

In this section we speculate on how to use genetic programming as an alternative to genetic algorithms in order to evolve programs as set of actions able to solve the remaining constraints imposed by our objective function. Chromosomes here are therefore encoded as set of nodes in tree-like structures in which we can randomly generate, select and evolve and so forth for a typically large number of generations. The set of possible actions to combine are divided into different layers, knowing that the first layer (symbolized by the choice of a station) will always be encoded into the root node of our evolutionary program. We constrain the subsequent layers to have implications: a specific layer l_a that implies layer l_b means that if an heuristic belonging to l_a is chosen, the subsequent or precedent heuristic must belong to l_b: if we take a look at table I a summary of layers, some of their example heuristics and their respective implications is shown.

Once the a first station is chosen, different chromosomes can be formed by randomly picking heuristics belonging to the other layers. If, for instance, *Move to Area* is selected after the root node, it means that the program will attempt to move to a specific component to a specific picking area within the same station, and those two artifacts (area and components) will be picked by following the heuristics indicated in the layers *Move to area* and *Select Component(s)*. We can therefore think that constructing a program as two phase process: the first phase a general structure of the program will be laid down, while in the second phase the generic structure will be translated as a set of actions, therefore shaping the terminal nodes of the tree-structure representing the program as an individual of the chromosomes population (see Fig.3 for an example situation).

Once an initial population of programs have been created, the following steps of the GP closely follows a standard genetic algorithm: each program is tested iterating over a maximum amount of steps or until the current action is found to be unfeasible due to breaking previously imposed constraints (such as the CT). We select, evolve, mutate and re-initialize the population over each generation by following known genetic operators. As far as mutation operators are concerned, limits must be imposed on the maximum amount of actions that forms the program so to avoid excessively bloated solutions. It has to be pointed out that it is possible that consecutive sequences of actions may not bring any changes to the components/task distribution over the different picking areas/stations: choosing to erase such sequences while creating or evolving the programs may results in increasing computational complexity and therefore it is advisable not to detect those situations. Moreover, due to the fact that testing such sequences against the objective function will result in lower fitness values, programs that are largely constituted by these ineffective sequences are significantly less likely to survive for the subsequent generations.

V. Limitations

In this paper we proposed the main idea of a method to address the TSALB problems, considering in addition the picking times of the components within the workstations. As what we are presenting here is a speculation of the best approach to take in such a complex problem, we are still investigating towards the following points:

- All the heuristics and actions shown in table I regarding GP, are still under investigations and as of now, just heuristics with deterministic outcomes are considered: in the future it may be useful to study how the algorithm reacts with the insertion of stochastic actions (e.g., pick a random station/component etc...).
- The bin packing problem can suggest solutions able to respect the time constraint but it does not satisfy the spacial constraint, thus failing to find a solution at the proposed TSALB problem. In a moderate sized dimension of the problem, it could be feasible to run a 2D BPP solver algorithm with space being the second dimension. When such extension becomes computationally intractable, we can think about taking into account the space constraint as a variable that influences the fitness of each genetically evolved individual. By doing this, the solutions which do not respect the space constraint will obtain a low fitness score, therefore such inefficient individuals will not be considered in future generations. We are still investigating the implementation details of these different mechanisms.

VI. Conclusion and future works

Time and space assembly line balancing problems model a close version to existing real-world situations of assembly lines. The existing approaches to solve TSALB problems do not take in consideration an important real scenario of the physical allocation of the components used during the executions of the tasks. In this paper we have presented a method which addresses this issue, balancing of an assembly line trying to optimize the global picking time of its components. In particular, the proposed approach is able to produce solutions which respect of the time constraint and then evolve them in order to optimize the allocation of the components in different areas, complying the spatial constraint. We ideologically divided our method in two phases. The first phase is represented by solving a 1D Bin Packing Problem using time as the only dimension. Each bin represents a station with its maximum time for processing being equal to the cycle time of the entire line. The items to pack are therefore the operation times of the single tasks. In the second part, the optimization of the components' allocation and the conformity of the solutions with the spatial constraint are considered. We proposed two different evolutionary approaches at this point: a genetic algorithm and a genetic programming. The genetic algorithm requires a variety of initial solutions given by the bin packing problem, in order to avoid a limited exploration of the solution space. The genetic programming approach, on the other hand, sets some nodes in tree-like structures in

TABLE I. LAYERS, IMPLICATIONS AND PROPOSED HEURISTICS FOR GP

Layer	Example heuristics	Implies
Select Station	Pick station having the largest/smallest number of tasks, Pick station having the largest/smallest number of components ...	-
Move Task	Pick task having the most/least time demanding task, pick task having the largest/smallest number of components...	Select Station
Select Component(s)	Pick component having the largest/smallest picking time ...	Move to area
Move to area	Pick closest/2nd/3rd ... closest area to human operator	Select components

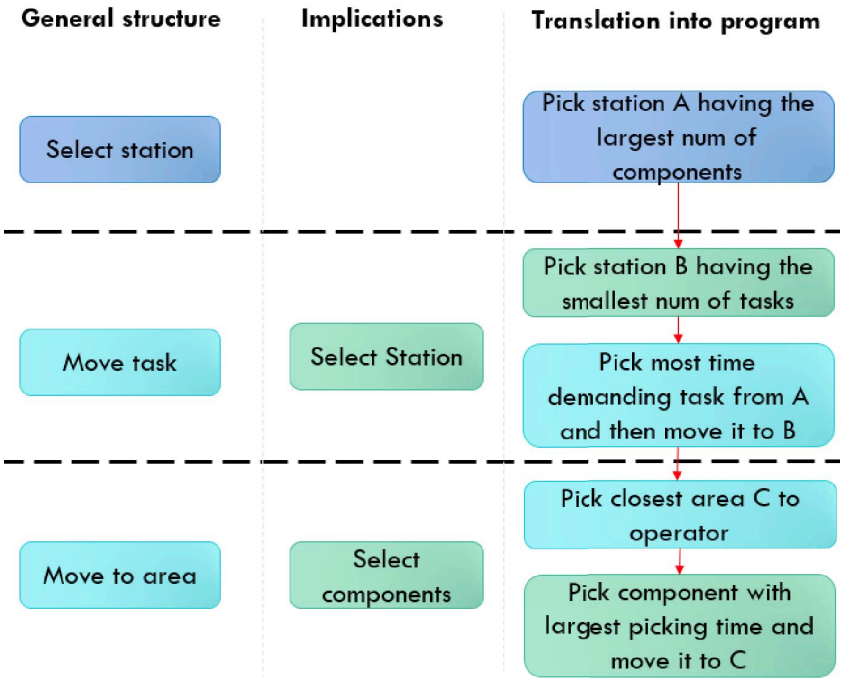

Fig. 3. Example situation of a randomly generated program. A first phase creates the program general structure that allows us to identify implications. From these implications we are able to translate the general structure in a program as a sequence of actions.

which we can randomly generate a series of heuristics which solve the remaining constraints imposed by our objective function. Selecting, evolving, mutating and reinitializing the heuristics, following known genetic operators, the approach can be able to find sub-optimal solutions for the proposed TSALB problem. As we pointed out in the previous section, the presented work represents just a speculative preliminary investigation of solving methodologies for such a complex problem; however the suggested evolutionary approaches look promising as they have been already used for solving problems characterized by less constraints. First of all, as future work, a concrete implementation of the presented approach tested against real world data will be conducted in order to prove the validity of this presented approach. After that, some other points can be explored, such as the assembling of mixed products, availability of different allocation areas for each station, different picking times depending on the stations, and so on.

REFERENCES

[1] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *European Journal of Operational Research*, vol. 168, no. 3, pp. 694–715, 2006.

[2] M. Chica, J. Bautista, Ó. Cordón, and S. Damas, "A multiobjective model and evolutionary algorithms for robust time and space assembly line balancing under uncertain demand," *Omega*, vol. 58, pp. 55–68, 2016.

[3] C. A. Yano and R. Rachamadugu, "Sequencing to minimize work overload in assembly lines with product options," *Management Science*, vol. 37, no. 5, pp. 572–586, 1991.

[4] M. Faccio, M. Gamberi, A. Persona, A. Regattieri, and F. Sgarbossa, "Design and simulation of assembly line feeding systems in the automotive sector using supermarket, kanbans and tow trains: a general framework," *Journal of Management Control*, vol. 24, no. 2, pp. 187–208, 2013.

[5] J. Bautista and J. Pereira, "Ant algorithms for a time and space constrained assembly line balancing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2016–2032, 2007.

[6] A. Scholl and C. Becker, "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," *European Journal of Operational Research*, vol. 168, no. 3, pp. 666–693, 2006.

[7] J. Bukchin and M. Tzur, "Design of flexible assembly line to minimize equipment cost," *Iie transactions*, vol. 32, no. 7, pp. 585–598, 2000.

[8] P. M. Vilarinho and A. S. Simaria, "A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations," *International Journal of Production Research*, vol. 40, no. 6, pp. 1405–1420, 2002.

[9] R. Gamberini, A. Grassi, M. Gamberi, R. Manzini, and A. Regattieri, "U-shaped assembly lines with stochastic tasks execution times: heuristic procedures for balancing and re-balancing problems," *Simulation Series*, vol. 36, no. 2, p. 137, 2004.

[10] N. Boysen, M. Fliedner, and A. Scholl, "Assembly line balancing: Which model to use when?" *International Journal of Production Economics*, vol. 111, no. 2, pp. 509–528, 2008.

[11] M. Chica, O. Cordón, S. Damas, and J. Bautista, "A new diversity induction mechanism for a multi-objective ant colony algorithm to solve a real-world time and space assembly line balancing problem," *Memetic computing*, vol. 3, no. 1, pp. 15–24, 2011.

[12] Y. Bukchin and R. D. Meller, "A space allocation algorithm for assembly line components," *IIE Transactions*, vol. 37, no. 1, pp. 51–61, 2005.

[13] N. Kriengkorakot and N. Pianthong, "The assembly line balancing problem," *KKU Enginieering Journal*, vol. 34, no. 2, pp. 133–140, 2007.

[14] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[15] H. John, "Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence," 1992.

[16] J. Rubinovitz and G. Levitin, "Genetic algorithm for assembly line balancing," *International Journal of Production Economics*, vol. 41, no. 1, pp. 343–354, 1995.

[17] Y. K. Kim, Y. J. Kim, and Y. Kim, "Genetic algorithms for assembly line balancing with various objectives," *Computers & Industrial Engineering*, vol. 30, no. 3, pp. 397–409, 1996.

[18] J. F. Gonçalves and J. R. De Almeida, "A hybrid genetic algorithm for assembly line balancing," *Journal of Heuristics*, vol. 8, no. 6, pp. 629–642, 2002.

[19] M. Chica, O. Cordon, and S. Damas, "An advanced multiobjective genetic algorithm design for the time and space assembly line balancing problem," *Computers & Industrial Engineering*, vol. 61, no. 1, pp. 103–117, 2011.

[20] S. O. Tasan and S. Tunali, "A review of the current applications of genetic algorithms in assembly line balancing," *Journal of intelligent manufacturing*, vol. 19, no. 1, pp. 49–69, 2008.

[21] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*. Springer, 2005, pp. 127–164.

[22] Y.-j. LIU, T.-l. GU, Z.-b. XU, and L. CHANG, "Parallel assembly sequence planning based on improved genetic programming," *Computer Integrated Manufacturing Systems*, vol. 6, p. 010, 2013.

[23] A. Baykasoğlu and L. Özbakır, "Discovering task assignment rules for assembly line balancing via genetic programming," *The International Journal of Advanced Manufacturing Technology*, vol. 76, no. 1-4, pp. 417–434, 2015.

[24] C. Dimopoulos and A. M. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Advances in Engineering Software*, vol. 32, no. 6, pp. 489–498, 2001.

[25] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[26] D. Jakobović and L. Budin, "Dynamic scheduling with genetic programming," in *Genetic Programming*. Springer, 2006, pp. 73–84.

[27] J. Park, S. Nguyen, M. Zhang, and M. Johnston, "Evolving ensembles of dispatching rules using genetic programming for job shop scheduling," in *Genetic Programming*. Springer, 2015, pp. 92–104.

[28] C. Dimopoulos and A. Zalzala, "Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons," *Evolutionary Computation, IEEE Transactions on*, vol. 4, no. 2, pp. 93–113, 2000.

[29] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[30] J. H. Patterson and J. J. Albracht, "Technical noteassembly-line balancing: Zero-one programming with fibonacci search," *Operations Research*, vol. 23, no. 1, pp. 166–172, 1975.

[31] T. Wee and M. J. Magazine, "Assembly line balancing as generalized bin packing," *Operations Research Letters*, vol. 1, no. 2, pp. 56–58, 1982.

[32] P. Schaus, Y. Deville *et al.*, "A global constraint for bin-packing with precedences: Application to the assembly line balancing problem." in *AAAI*, 2008, pp. 369–374.

[33] C. Boutevin, M. Gourgand, and S. Norre, "Bin packing extensions for solving an industrial line balancing problem," in *Assembly and Task Planning, 2003. Proceedings of the IEEE International Symposium on*. IEEE, 2003, pp. 115–121.

[34] C. McGeoch, *A Guide to Experimental Algorithmics*. Cambridge Univerisity Press, 2012.