Development of an application for balancing product flow lines through genetic algorithms

Manuel F. Silva*

INESC TEC – INESC Technology and Science (formerly INESC Porto) and ISEP/IPP – School of Engineering,
Polytechnic Institute of Porto,
Rua Dr. António Bernardino de Almeida, 4200-072 Porto, Portugal
Email: mss@isep.ipp.pt
*Corresponding author

Cecília Reis

ISEP/IPP – School of Engineering,
Polytechnic Institute of Porto,
Rua Dr. António Bernardino de Almeida,
4200-072 Porto, Portugal
Email: cmr@isep.ipp.pt

Ricardo Pimenta

ISEP/IPP – School of Engineering,
Polytechnic Institute of Porto,
Rua Dr. António Bernardino de Almeida,
4200-072 Porto, Portugal
Email: rpimentta@gmail.com

Abstract: When defining the layout for a production line, it is necessary to assign tasks to workstations, so that the work is performed in a feasible sequence and approximately equal amounts of time are needed at each workstation, a process called line balancing. Therefore, the need for balancing production lines involves the distribution of sequential activities for jobs in order to allow high labour and equipment utilisation and minimise the idle time. Line balancing problems are complex to treat, being used distinct methodologies to perform it. This paper describes an application for line balancing using two genetic algorithms (the first obtains solutions to the problem and the second optimises those solutions), associated with a graphical interface for the problem data input and visualisation of results. Results demonstrate advantages over heuristic methods as it is possible to obtain more than one solution and it is more practical to use the developed application.

Keywords: assembly line; balancing; genetic algorithms; optimisation.

Reference to this paper should be made as follows: Silva, M.F., Reis, C. and Pimenta, R. (2016) 'Development of an application for balancing product flow lines through genetic algorithms', *Int. J. Business Excellence*, Vol. 9, No. 3, pp.310–331.

Biographical notes: Manuel F. Silva graduated, received his MSc and PhD in Electrical and Computer Engineering from the Faculty of Engineering of the University of Porto, Portugal, in 1993, 1997 and 2005, respectively. Presently, he is an Adjunct Professor at the Institute of Engineering of the Polytechnic Institute of Porto, Department of Electrical Engineering. His research focuses on modelling, simulation, motion and time analysis, genetic algorithms, and robotics.

Cecília Reis graduated in Electrical Engineering – Industrial Control from Institute of Engineering of the Polytechnic Institute of Porto, Portugal, in 1992 and received her Master's in Electrical and Computer Engineering from the Faculty of Engineering of the University of Porto, Portugal, in 1995. In October of 2007, she concluded her PhD in Engineering Sciences at the University of Trás-os-Montes and Alto Douro, Portugal. Presently she is a Professor at the Institute of Engineering of the Polytechnic Institute of Porto, Department of Electrical Engineering. Her research interests include digital systems, evolutionary computation and information systems.

Ricardo Pimenta graduated in Electrical Engineering, and then received his Master's in Electrical Engineering – Industrial Control, from the Institute of Engineering of the Polytechnic Institute of Porto, Portugal, in 2009 and 2011, respectively. Presently he works in RobotSol, as the Project Manager and Robotics Integrator. His main interests are in robotics and automation.

This paper is a revised and expanded version of a paper entitled 'Assembly line balancing using genetic algorithms' presented at the Second International Conference on Business Sustainability, 2011 – Management, Technology and Learning for Individuals, Organisations and Society in Turbulent Markets, Póvoa de Varzim, Portugal, 22–24 de June de 2011.

1 Introduction

A production line consists in multiple workstations in sequence, each with a set of associated tasks, devoted to the manufacture or assembly of goods or products. Each task under consideration has a set of precedence relations with other tasks, specifying that some of them must be completed before the task can start. Each task has also associated an execution time. The execution time of each workstation must fulfil the Takt time (*TT*), in order for the line to have a steady flow while achieving the demand, which is called line balancing problem (LBP).

Production line balancing is an analysis method which aims to equally distribute the work to be performed by the various work stations such that the number of workers or workstations in the line is minimised.

There are two main types of LBP:

- LBP-1: given the *TT*, minimise the number of workstations.
- LBP-2: given the number of workstations, minimise the cycle time.

The LBP-1 is mainly present when a new production flow line system has to be installed and the external demand can be estimated. By opposition, LBP-2 leads to the maximisation of the production of an already existing flow line. This is of importance if any changes take place on the production process or on the demand structure (Scholl and Voß, 1996).

Among the methods used to solve LBP, are trial and error methods, heuristics methods (Jonnalagedda, 2010), computational methods for evaluation different options until a good solution is found and optimisation methods (Li et al., 2006).

This work describes an application, that was developed in the C# programming language, to solve LBP using genetic algorithms (GAs). This application allows the input of data regarding the problems to be solved in an easy and user friendly way, allows solving the problem of balancing the production line using GA, and allows the user to analyse the results, once again, in an easy and friendly way. Whenever possible, the application generates a solution (or a set of solutions) feasible and with an efficiency equal or greater than the one of the solutions that can be found using heuristic methods.

Bearing these ideas in mind, the remaining of this paper is organised as follows. Section 2 describes in detail the LBP. Section 3 presents a brief description of GA. Based on this theoretical foundation, Section 4 introduces the developed GA for the application under consideration, Section 5 presents the results achieved by this application when used to solve an LBP and Section 6 analyses and discusses the obtained results. Finally, in Section 7 are presented the main conclusion of this work and are proposed some ideas for future developments of the developed GA and application.

2 Defining the LBP

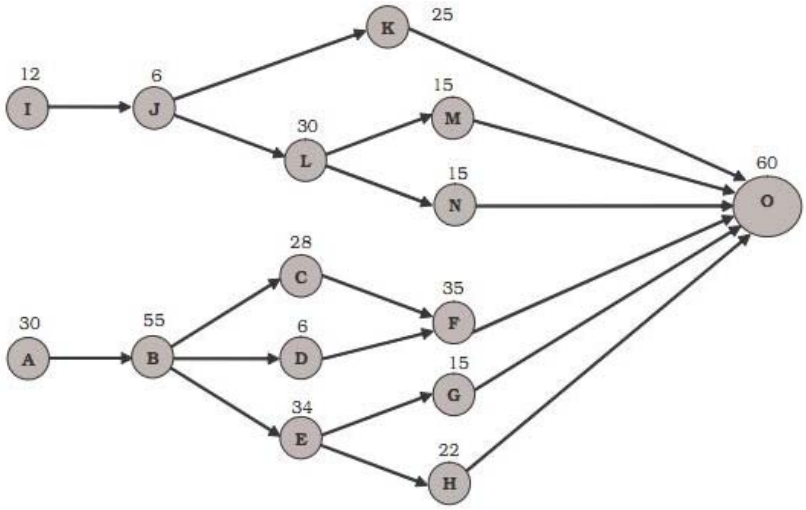
In order to clearly describe the LBP, an example is described in the following. Consider the process diagram (precedence diagram) for the production/assembly of an example product, given in Figure 1.

In this case, the production/assembly of the product under consideration requires $k = 15$ tasks and the time necessary to complete each task (the value depicted above the corresponding operation) is $t(i)$ ($1 \leq i \leq 15$).

The question of LBP is:

- LBP-1: given the *TT*, how to assign the k tasks to n stations (where, generally, $n \leq k$), in order to minimise the number of workstations?
- LBP-2: given the number of workstations (n), how to assign the k tasks to n stations, in order to minimise the cycle time?

Figure 1 Ordered graph of tasks



2.1 Definitions and concepts

Considering that a production/assembly operation is composed of k tasks, and that the time necessary to complete operation i (i = 1, 2, …, k) is $t(i)$, then the time needed to finish a complete unit is given by equation (1):

$$\sum_{i=1}^{k} t_i \tag{1}$$

The TT of an assembly line is given by equation (2):

$$TT = \frac{time\ available\ per\ period}{number\ of\ units\ wanted\ per\ period} \tag{2}$$

The cycle time (c) can be defined as:

1 the time between the output, at the end of the line, of each successive units;

2 the maximum time necessary in each one of the line workstations.

In order to guarantee that the line, after being balanced, is able to produce goods/products according to the requested demand, the cycle time of an assembly line must be $c \leq TT$.

Considering the previous expressions, the theoretical minimal number of workstations necessary for the assembly line, to guarantee the fulfilment of the TT is given by equation (3):

$$n_{\min} = \frac{\sum_{i=1}^{k} t_i}{TT} \tag{3}$$

After the line balancing has been completed, it is possible to compute its efficiency, according to equation (4):

$$e = \frac{\sum_{i=1}^{k} t_i}{n \times TT} \tag{4}$$

2.2 Objectives of line balancing

As a rule, the aim of the flow line balancing is to enable (to the extent possible) a quick and uniform flow of materials through the production line and, at the same time, achieve the maximum use of the human potential, the maximum utilisation of machinery, equipment and tools and expending minimal material or process time.

Therefore, flow line balancing is used to obtain the following results in a production/assembly line:

- minimise the number of workstations
- minimise the cycle time.

2.3 Methods for the balancing of production/assembly lines

There are several methods to make the balancing of a production line, of which the following stand out:

- trial and error
- heuristic, meaning that they are based on logic, and on knowledge
- optimisation methods, based on linear programming and/or dynamic programming
- computational methods that search and evaluate different options to find the best solution, which implement, in most cases, the two previous approaches
- evolutionary computation namely GAs.

Only optimisation methods ensure an optimal balancing, while the other ones return good and viable solutions that approach, to a greater or lesser extent, the optimal solution.

2.4 Software applications for the balancing of production/assembly lines

There are several software applications that offer optimisation solutions for production lines. Among them, are referred in this work the ProBalance and AviX® balance applications.

2.4.1 ProBalance

Proplanner is a software company that offers various programmes related to the production process within the engineering and management areas, being one of these programmes, ProBalance, directed at line balancing (Proplanner, 2013).

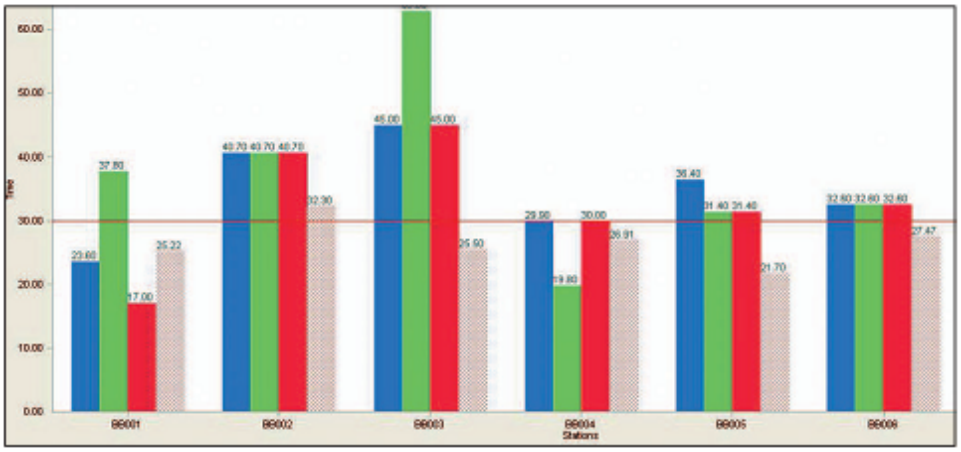
This application allows obtaining different line balancing solutions through the data of the process under analysis. There are two methods of line balancing available in the

ProBlance software. Users can select to minimise the number of workstations (type 1 balancing) or minimise the cycle time of operations (type 2 balancing). Type 1 balancing requires the user to enter the cycle time, from which the application calculates how many workstations are required. The type 2 balancing requires the user to enter the number of workstations available, or desired, and given this value is calculated the cycle time.

Two tasks are required to complete the line balancing using ProBalance. First, the user needs to specify the data of the problem (e.g., ID, process time, precedence). After entering the data, the second task is to create a balanced line (manually or generated by ProBalance) based on user input data.

Through algorithms of automatic assignment of tasks and resources, the application generates a solution to the problem introduced by the user. These algorithms operate by optimisation methods. When the application generates an automatic solution, this deals with several restrictions (Proplanner, 2013). When the balancing is done manually by the user, ProBalance displays all violations incurred during the process, again based on the constraints previously presented. Figure 2 presents a chart with an example of the line balancing results interface.

Figure 2 Line balancing chart (see online version for colours)



Source: Proplanner (2013)

2.4.2 AviX® balance

AviX® balance is a flow line balancing application, which helps optimise the production line. With the data from the application it is possible to create and simulate new production lines for products (AviX®, 2013).

The application allows the user to:

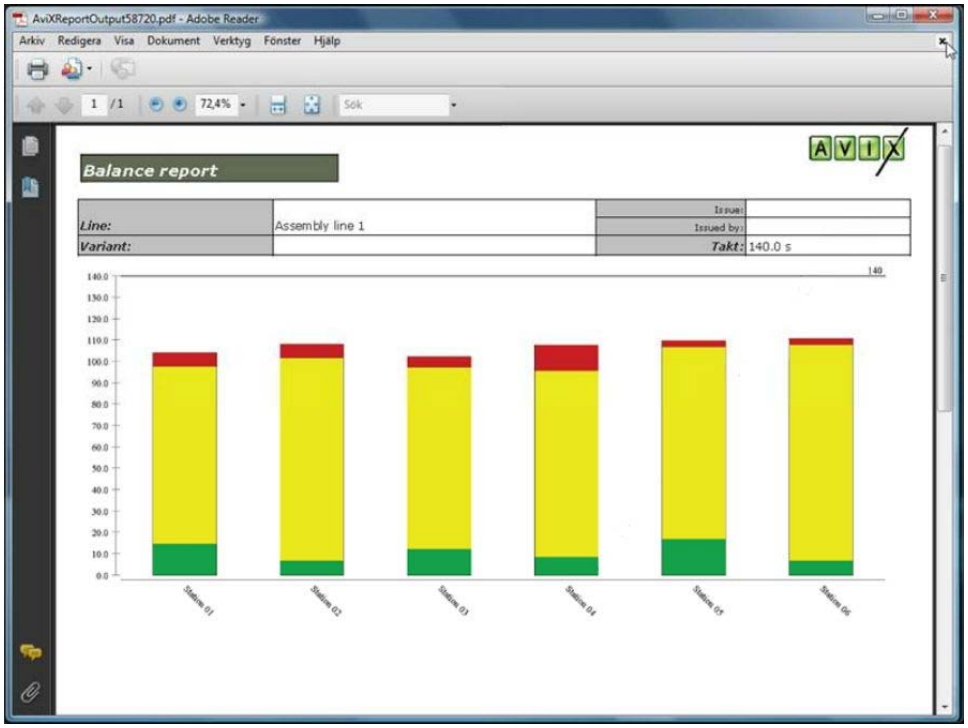
- view the results of balancing directly on the screen
- balancing multiple products simultaneously
- easy optimising production efficiency and increased production
- get layouts lossless

- balance with the control of the working methods, objects, tools and components.

This application allows to manually balancing lines by moving tasks between stations. After each step taken in balancing, the application shows their influence on productivity and efficiency of the production line.

After the line is balanced, it is possible to obtain various reports on the line. Figure 3 shows an example of one of them, where one can see the operation times at the stations.

Figure 3 Line balancing report (see online version for colours)



Source: Avix® (2013)

2.5 Line balancing software applications using GAs

In this subsection are briefly described two works developed for line balancing using GAs.

2.5.1 GA for balancing single product production lines

In this project, developed by Mayerle and Santos, is discussed the problem of balancing simple (single-product) production lines, which consists in obtaining an allocation of a set of tasks to workstations, respecting the precedence relationship of these tasks. Two different situations are considered: reducing the number of jobs, considering the cycle time that meets the demand, or reduce cycle time, given the number of jobs (Mayerle and Santos, 2003).

Although, the authors do not have performed extensive tests with the algorithm, its performance was close to optimum; moreover, they suggest that it is expanded the number of tests performed with the algorithm, including problems that have solutions through other heuristic methods found in the literature.

2.5.2 Balancing of mixed model production lines with hybrid GA

Considering the existing problem of balancing mixed model production lines, which is a production line where there are several products with similar features, which are manufactured according to customer needs, it is proposed by Ying et al. (2009) a mathematical model based on two factors, which are integrated, including the number of workstations and the assembly line efficiency.

Then, it is developed a new hybrid GA to find the optimal solution of the problems. To avoid that the problem converges to a solution prematurely, and improve the optimisation, the GA is combined with a simulator of recognition algorithms (simulated annealing algorithms). The simulation results indicate that, in this way, the hybrid algorithm has a better optimisation and performance efficiency.

3 Genetic algorithms

GAs were developed by Holland (1992), his colleagues, and students, in the 1970s. The purposes of Holland's research were two:

1 to abstract and rigorously explain the adaptive processes of natural systems, and

2 design artificial systems software that retains the important mechanisms of natural system.

This approach led to important discoveries in both natural and artificial systems (Jonnalagedda, 2010; Li et al., 2006).

GAs are a family of computational models inspired by genetics and natural evolution. These algorithms model a solution to a specific problem in a data structure, called a chromosome, and apply operators that recombine these structures while preserving critical information (Davis, 1992; Goldberg, 1989).

Briefly, the outline of a GA is as follows:

1 *Start*: Generate a random population of *n* suitable solutions (chromosomes). The values for the genes that constitute the chromosome are randomly distributed according to the corresponding parameters.

2 *Fitness*: Select and evaluate the fitness function for each chromosome.

3 *New population*: Create a new population by repeating the following steps:
   a *Selection* – Select the best parent chromosomes according to their fitness. These solutions are copied without changes to the new population (elitism).
   b *Crossover* – Select 60% to 90% of the individuals to be replaced by the crossover of the parents: two random parents are chosen and an arithmetic mean operation is performed to produce one new offspring.

c *Mutation* – Select 0.1% to 5% of the individuals to be replaced by mutation of the parents: one random parent is chosen.

4 *Spontaneous generation* – The remaining individuals are replaced by new randomly generated ones (such as in Step 1).

5 *Loop*: If the stop criteria were achieved then the algorithm stops, else, go to Step 2.

4 GA development

This section describes all aspects of the developed GAs and explains their settings. It also introduces the graphical user interface (GUI) developed to make the connection between the GAs and the problem to analyse, simplifying the data entry, as well as to present the results to the user.

The first step in the development of a GA is the coding of the problem whose resolution is intended in the algorithm. Following is the generation of the initial population, the selection process and the performance of the crossover and mutation operations. Finally, there is also the need to consider the fitness function and the choice of the stopping criterion.

4.1 Coding problem

To simplify the coding of the problem, a number is assigned to each task. For example, to task 'A' it is assigned number '1', to task 'B', number '2', and so forth, until all tasks have a number assigned. This way the problem is codified using permutation encoding. Each gene in the chromosome represents a task. The order of the genes in the chromosome sets the order in which the tasks are executed.

Figure 4 shows an example of a chromosome coded. From this example, it is possible to verify that every number represents a task.

Figure 4 Example chromosome and the corresponding genes (see online version for colours)

2	1	8	3	4	6	7	5	10	9

4.2 Initial population

The first GA works with a fixed population size of 20 individuals and the initial population is randomly generated.

The chromosome is completely filled, starting with assigning the value 1 to the first gene and incrementing the value of each gene in the chromosome to be filled. After the algorithm has completed the chromosome, this will be 'scrambled'. This process is carried out through the exchange of genes, selecting random positions. To make this change, two random positions are selected and the genes corresponding to these positions are exchanged. This process is executed x times, where x equals the number of operations multiplied by 3 (this value was found to be the optimum through a series of experiments), which allows the number of times that the chromosome is shuffled to be directly related to the number of operations of the problem.

4.3 Selection

The selection method is 'selection by tournament'. According to this selection method three chromosomes are randomly selected from the population. From these chromosomes, the one with the highest fitness is selected to be used in the subsequent crossover and mutation operations.

4.4 Crossover and mutation

In this GA, and due to the used problem coding method, the crossover operation is not applied to the chromosomes. In this particular case, this operation would compromise the integrity of the solutions, since it would make the same gene repeated in a chromosome (unless both chromosomes were exactly equal). Therefore, it is only applied the mutation operator to generate a new chromosome.

The mutation operator was tested with three options. In the first option, two random positions are selected, and the corresponding genes are exchanged. For the second option, only one position is randomly selected and the corresponding gene is switched with the consecutive one. After several tests, the results that revealed better performance were the tests applying the third mutation operator option. This third option is described below and is the one that was implemented.

The mutation operator works according to the following: two random positions are selected in the chromosome; the first selected position is the position of the gene that will be moved, the second is the new position of the gene. The genes between the first and second positions, including the gene on the second position will be shifted one position to the right, or left, according to the selected positions, as presented in the example depicted in Figure 5.

Figure 5 Example of a mutation operation (see online version for colours)

Initial Chromosome	7	5	1	4	2	9	10	3	8	6
Final chromosome	7	5	1	2	9	10	3	4	8	6

To improve the algorithm, the mutations are done dynamically. This is performed by counting the number of operations in the problem, and the total number of precedents (the sum of the number of precedence's of all tasks). The number of mutations done in each new generation is set according with this total (corresponds to one quarter of this total).

4.5 Fitness function

While balancing an assembly line, it must be ensured that the precedences between the operations are fulfilled and that the cycle time is guaranteed, which in itself makes the number of necessary stations established. Crossover and mutation may create some infeasible solutions when the GA is used to solve LBP. In these cases, individuals that infringe one or more precedence restrictions might appear in the population. To prevent these situations, a penalty function is created to force the individuals of each generation

to evolve to feasible solutions. Based on this knowledge, was developed a fitness function that evaluates the chromosomes through the verification of compliance with precedence relations and the number of stations that each possible solution requires.

The fitness (or evaluation) function is given by expression (5) and has the purpose, as the name implies, to evaluate each chromosome according to the objective of the problem.

$$Fitness = \frac{Max_{penalty} - penalty}{Max_{penalty}} \times 100 \tag{5}$$

At the beginning of the algorithm, the number of precedences existing in the problem to be solved is counted. This value is used to know the maximum number of possible penalties, and is saved in the variable '$Max_{penalty}$'. Then, it is checked whether the precedences are fulfilled in each gene of the chromosome to which the fitness function is being applied – the penalty variable (variable *penalty*) is incremented for each unrespect precedence (*penalty*:= *penalty* + 1). During this process, is also summed the temporal duration of each task that is being checked, and once the accumulated time exceeds the required *TT*, it is required to add a further workstation. At the end, is checked by how many workstations the analysed solution exceeds the computed theoretical minimum number of stations, and for each extra station the variable penalty is incremented by one (*penalty*:= *penalty* + 1).

4.6 GA stop criteria

In this algorithm, there are two stopping criteria. Being the objective of the GA to obtain a solution that meets the precedence criteria and satisfies the desired number of workstations, the first stopping criteria corresponds to the GA founding a solution – a solution is obtained when there is a chromosome with a value of the fitness function equal to 100%.

The second stopping criterion is based on the number of iterations. In this case, the GA stops when 10 million iterations are performed presenting the results obtained so far.

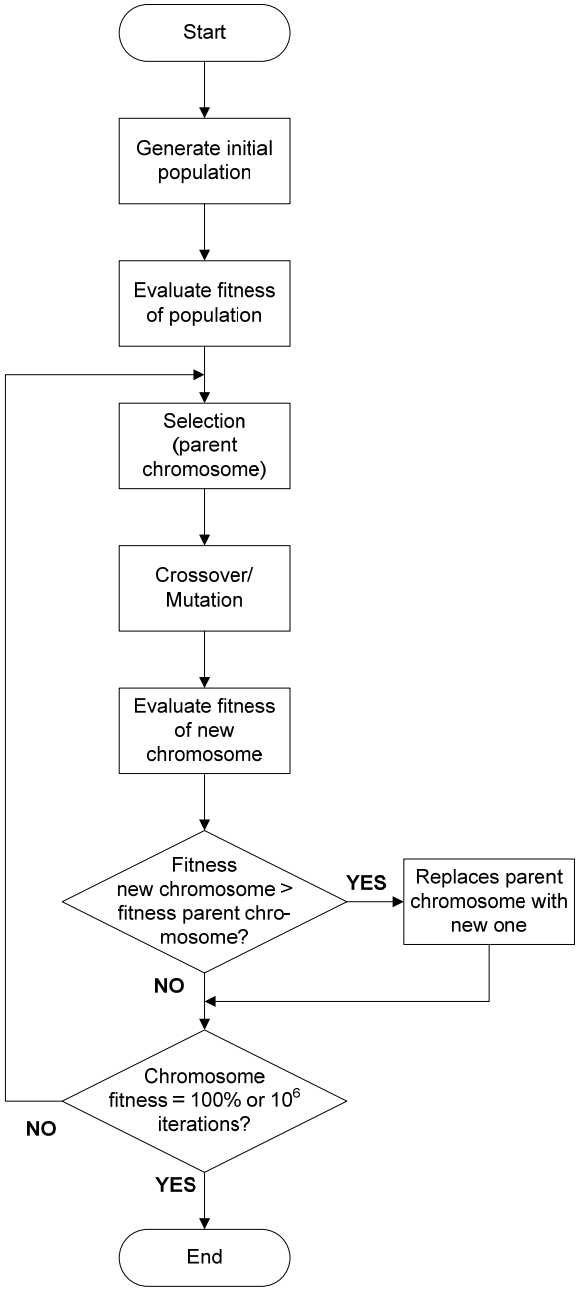
4.7 Flow diagram of the developed GA

Figure 6 presents a flow diagram of the developed GA, being possible to see how the algorithm works. This GA runs until it obtains the number of solutions defined by the user. This way, if it is defined that the software has to find three solutions, this GA will automatically run for three times.

4.8 GA for improvement of the solution

Once implemented the described GA, and after it has been applied to a set of test problems, it was found that the system gave good results but not optimal ones. For this reason, a second GA was developed, which improves the solutions obtained with the first GA. This algorithm aims to decrease the total idle time of the solution and decreasing the standard deviation of the individual workstations idle times.

Figure 6 Flow diagram of the GA



4.8.1 Selection

This second GA only works on the solutions obtained by the first one, which may be called 'feasible' solutions. Consequently, the population of this GA is very small, and limited to the solutions found by the first one.

As this algorithm has a small population, because the population is made up of the solutions obtained in the first algorithm, it is not possible to use a selection method as used in the first algorithm. Then the selection method adopted is simply the random choice of an element of this population.

4.8.2 Mutation

This second GA will randomly select one of the obtained solutions chromosome, and apply a mutation to it. The mutation process implemented in this algorithm is the same as in the main algorithm, with the slight difference that the mutation is carried out only once in each iteration.

4.8.3 Fitness function

The evaluation function of this second GA is divided into two parts. The fitness function of the previous GA is applied after performing the mutation. This is to ensure that the new chromosome, created by the mutation, still complies with all precedences, and that the number of workstations is also achieved. If this proves true, is applied the new evaluation function; if the solution has been compromised, the mutation made is then immediately rejected.

The new solution is then submitted to an evaluation, based on a new fitness function, being calculated the idleness of the corresponding solution and the standard deviation of the idleness of each workstation. Based on those values, it is verified if the solution is better than the original one; in case it is, then this solution chromosome replaces the original chromosome.

Following, a second fitness function is applied that calculates the idle time and the standard deviation of the idle time of the solution, and compares these values of the new solution with the ones of the solution chosen for mutation (parent chromosome). If this new solution contains an idle time value less than the one of the parent chromosome, then the parent chromosome is replaced by the new chromosome. If the new solution has an idle time equal to that of the parent chromosome is then compared the standard deviation of both solutions and, if the new solution has a value lower than that of the parent then this chromosome is replaced with the new solution. If the new solution presents an idle time higher than the one of the parent chromosome, is immediately discarded as it is a worse solution.

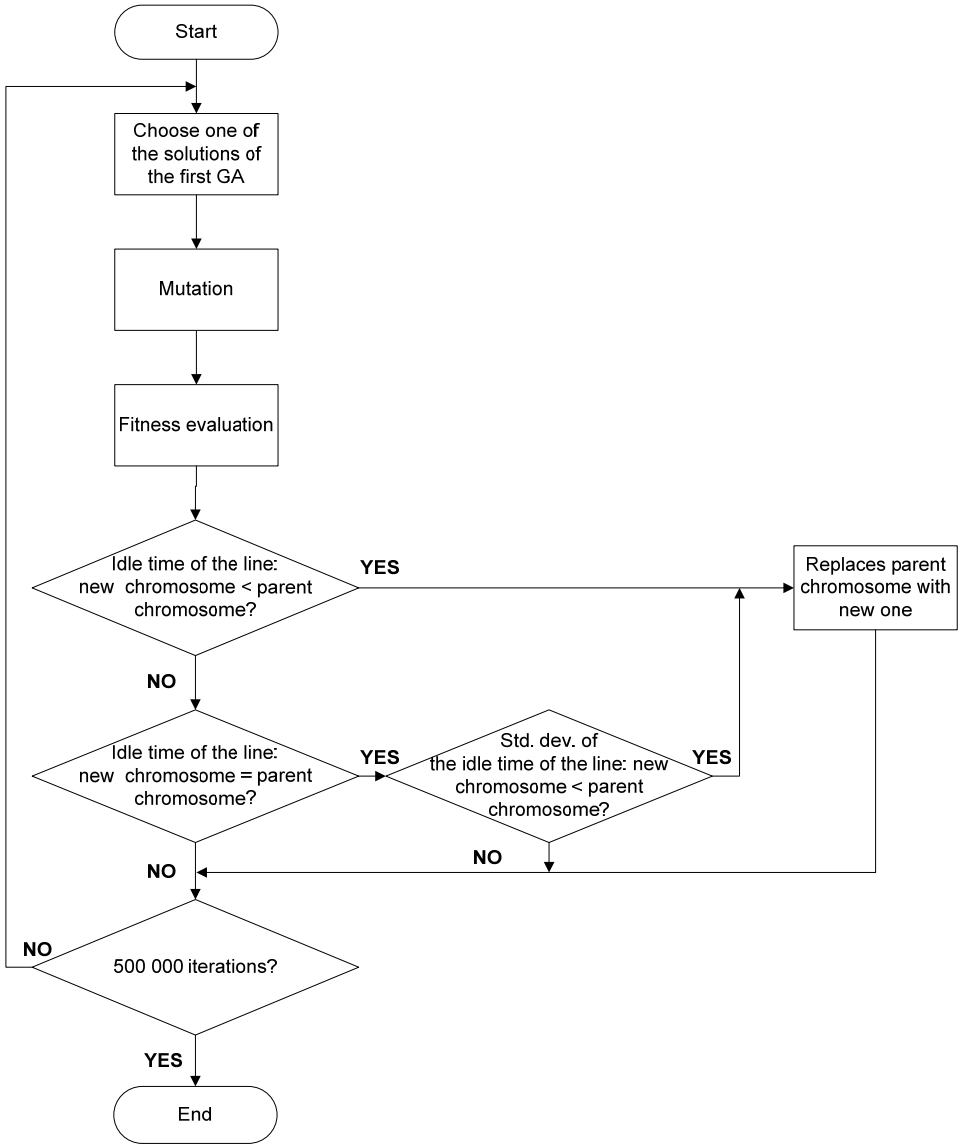
4.8.4 Stopping criteria

The stopping criterion of this algorithm is the number of iterations, i.e., this GA stops when reaching the set value of 500,000 iterations.

4.8.5 Flow diagram of the developed GA

Figure 7 shows the structure of this second algorithm. In this structure, it is possible to see all the stages of the algorithm, which were explained in the previous subsections.

Figure 7 Flow diagram of the GA



4.9 Graphical user interface

The GUI developed for the introduction of the problem and algorithm data, and for the presentation of the results, was developed in the C # programming language. Figure 8 shows the main window of the GUI, that provides the user an easy to use and intuitive interface.

The main window of the GUI allows the user to enter the data of the problem, starting by defining the number of tasks and the time units for the duration of each task to be entered into the table (seconds or minutes).

Afterwards, are automatically added rows to the table, corresponding to the number of tasks (each line corresponds to a task). In each row is possible to assign an ID to the task, input a brief description of it, and insert the corresponding task time and its precedences (the application allows the definition of a maximum of six precedences per task).

After the table is completed, the user has the possibility to choose between two kinds of problems: the first one in which is known the total operating time of the line and the number of desired parts, and a second in which the line has a fixed number of stations and it is desired to get the maximum possible output for that number of stations.

In the case of the first GA exists yet another field (filled by the user) to define how many solutions this should get. This value will be the population of the second algorithm.

Figure 8 Main window of the GUI (see online version for colours)

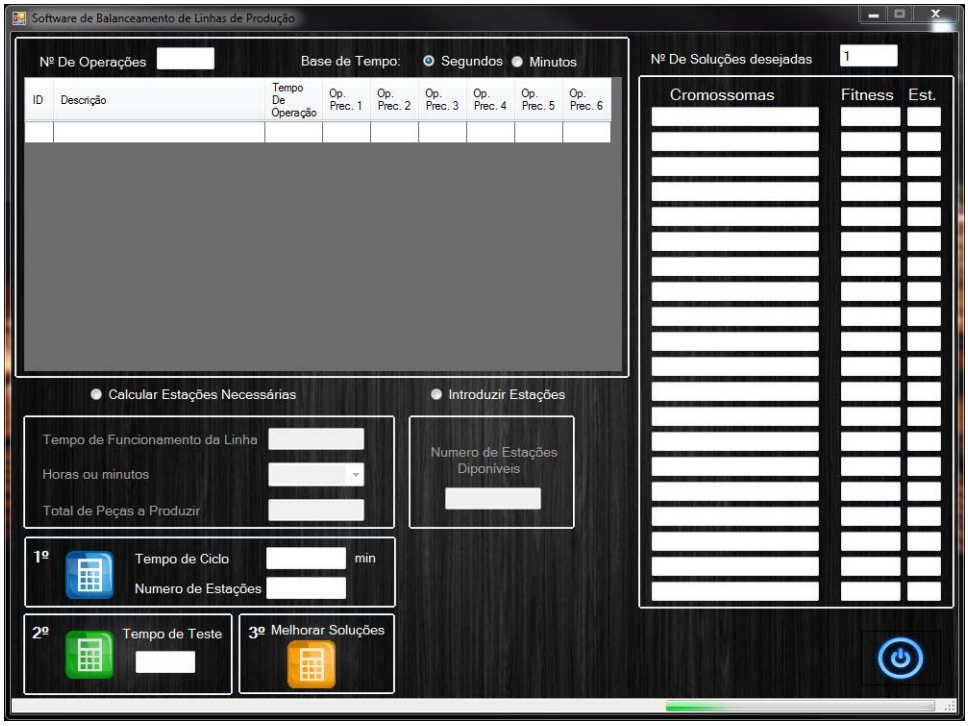


Figure 9 depicts the GUI after running the first GA, in which case the algorithm stopped after 10 million iterations, having obtained two solutions of the five desired by the user (they are marked in blue in Figure 9). In this case, the user can run the algorithm again until the remaining desired solutions are obtained.

After finishing the first GA run is possible to analyse each solution and each chromosome of the population. For this purpose, it is just needed to press a mouse button on the chromosome to be analysed. This opens a new window (Figure 10) that allows the analysis of the solution according to several aspects.

Figure 9 Application GUI after running the first AG on an example problem (see online version for colours)

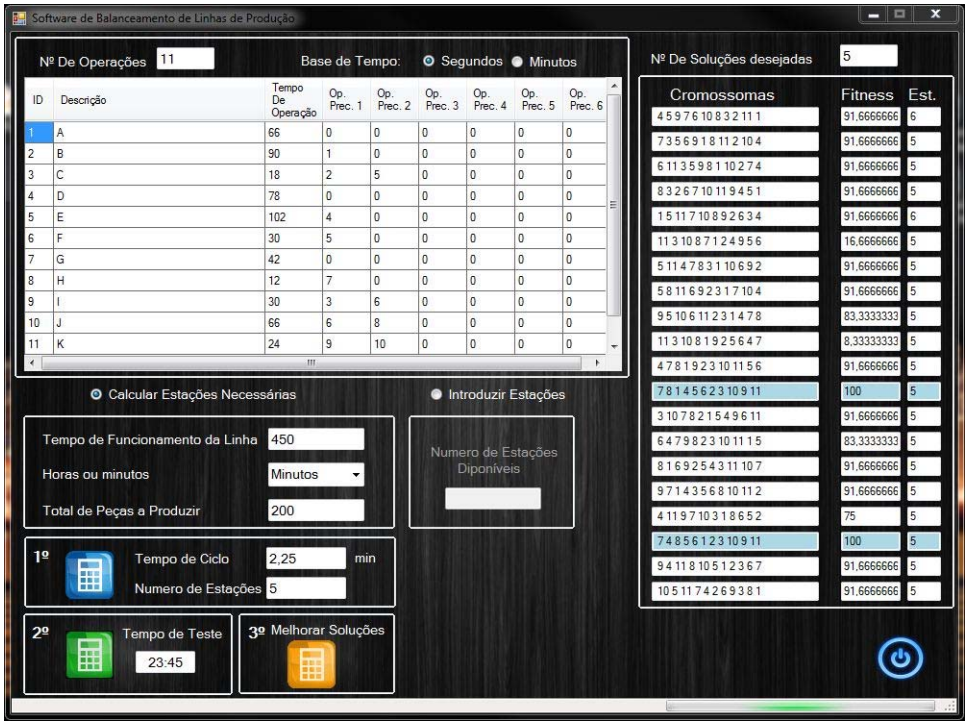


Figure 10 Application GUI result analysis window (see online version for colours)

After analysing the results of the first GA, and returning to the main application window, it is possible to launch the second GA for improving the results obtained by the first one. As previously mentioned, the initial population of this second GA are the solutions of the algorithm implemented in the previous step. This algorithm performs 500,000 iterations and can be run as many times as the user desires. Figure 11 shows the improved solution of the problem under consideration, after the execution of the second AG.

Figure 11 Analysis of the results of the second GA (see online version for colours)

5 Simulation results

To test the developed application, the GA was applied to a LBP-1 type example problem, in order to be possible to compare the results of the obtained solution using this application, and the solutions obtained by other authors, using distinct approaches.

The problem in question is an assembly line for a washing machine water pump, for which there is an estimated demand of 200 units per day. The assembly line is considered to work 480 minutes a day (Aguiar et al., 2007).

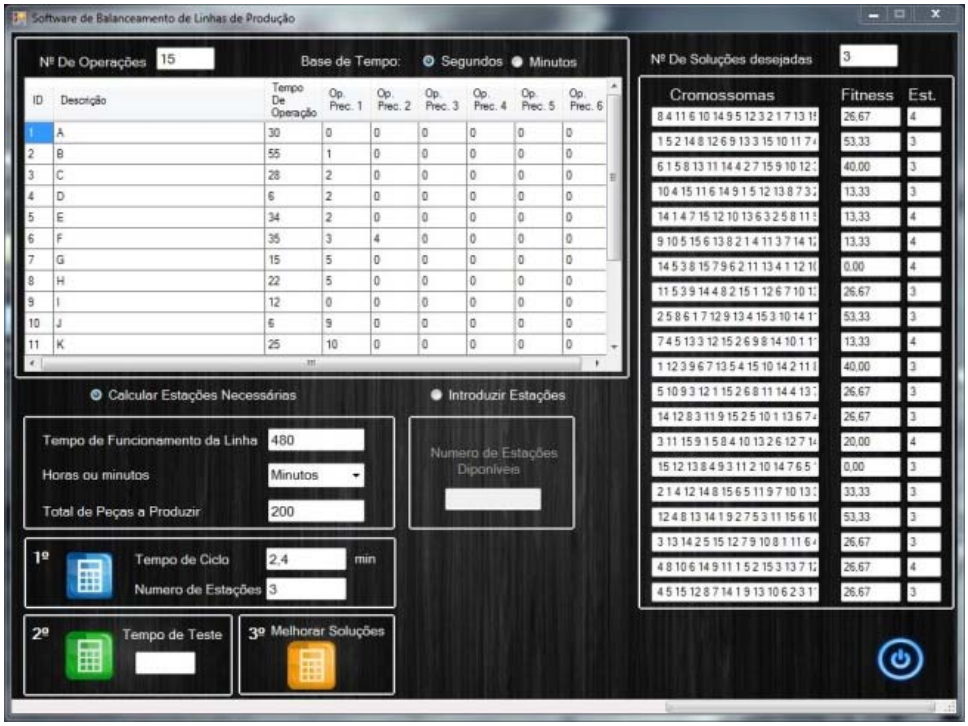
Table 1 presents the necessary tasks to assemble the washing machine water pumps, including the times needed for each task and the required precedence relations.

To solve this problem, the data presented in Table 1 was inserted in the developed software application, and the algorithm was run. Figure 12 presents the input of the parameters in the developed GUI. In the left upper part of this figure is a table where the data corresponding to the assembly line is introduced. Following, is introduced the uptime of the line and the number of units that are desired to produce in that time interval.

Table 1 Operations, time duration and precedences for the assembly of washing machines water pumps

Operations	*Time (seconds)*	*Precedents*
A	30	-
B	55	A
C	28	B
D	6	B
E	34	B
F	35	C,D
G	15	E
H	22	E
I	12	-
J	6	I
K	25	J
L	30	J
M	15	L
N	15	L
O	60	K, M, N, F, G, H
Total time	388 (s) ≈ 6,47 (min)	

Figure 12 Graphical interface showing the input of the parameters for the example problem (see online version for colours)

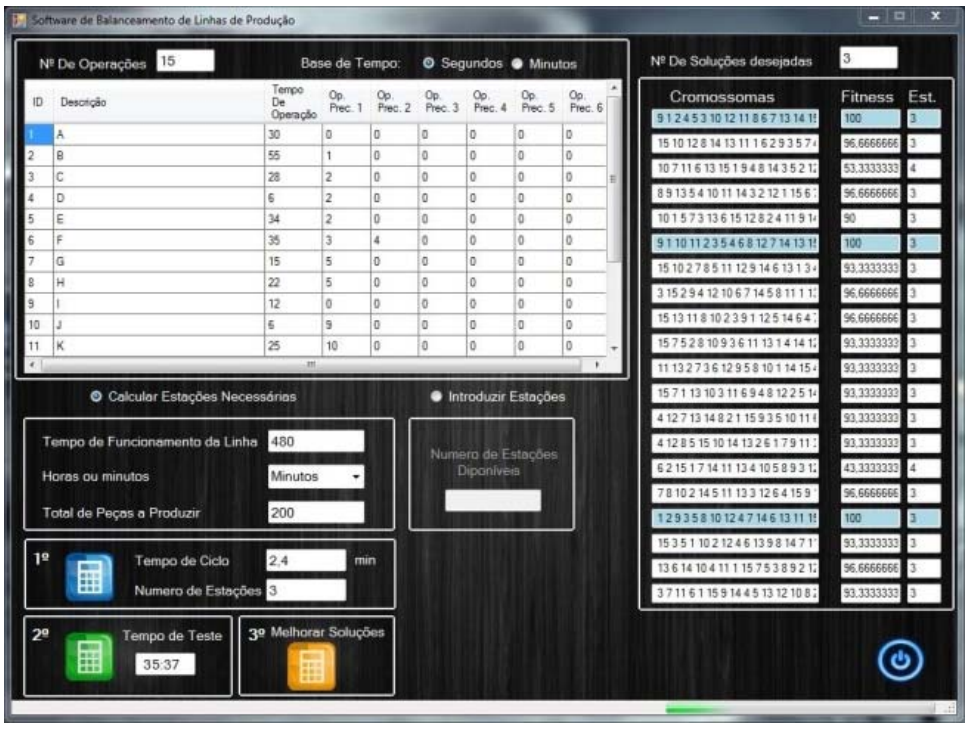


When the application begins to compute the solution, the GA starts by generating the initial population and calculates the fitness of each chromosome, according to the first two blocks presented in the flow diagram depicted in Figure 6.

After that, the GA enters a loop (also depicted in the flow diagram of Figure 6) to find the number of solutions to the problem defined by the user (in the case of this example, the number of desired solutions is three, as can be seen on the top right corner of Figure 12).

In the particular case of the problem under consideration, the algorithm runs for 35 minutes and 37 seconds to find the three solutions. The solutions are showed in Figure 13, being the corresponding chromosomes highlighted in blue.

Figure 13 Balancing line solutions for the example problem (see online version for colours)



After the first GA finds the 'feasible' solutions, the second GA is executed to improve these results. In this example, the software is able to obtain two better solutions, being presented only one of them in Figure 14.

The application also presents for the proposed solution, its efficiency, the delay of the balancing and total line idle time, the idleness and the standard deviation of the idleness of each workstation, as can also be seen in Figure 14.

In the solution window, presented in Figure 14, it is also depicted a graphic, which represents the busy time in each workstation (blue part of the bar) and the idleness (green part of the bar).

Figure 14 Optimum result for the example problem under analysis (see online version for colours)

6 Analysis of the results

In the sequel are compared the solutions obtained with the GA, and the solutions obtained through the use of heuristics (Aguiar et al., 2007). To solve this same problem, Aguiar et al. (2007) used an heuristic based on the assignment of tasks to stations considering their order in the assembly process – that means that the tasks without predecessors are assigned first to a station, then the tasks with only one predecessors, and so forth, until all tasks have been assigned to stations. Furthermore, after completing this process, Aguiar et al. (2007) performed a second improvement round in which they redistributed the tasks to stations in order to make all stations have approximate values for the idle time. The comparison of the solutions obtained with the two methods is depicted in Table 2.

To better compare the results, it is calculated the idleness of the solution obtained using heuristics and this value is compared with the results obtained using the GA. The idleness is the percentage of inactivity in the assembly line. It can be obtained using the following expression.

$$\%\ \text{idleness} = \frac{\sum \text{idleness of each workstation}}{\text{number of workstation} \times TT} \tag{6}$$

Table 2 compares the balancing of the line when both methods are used. This table presents, for each workstation, the tasks, the total time, the idleness, and the idleness of the full assembly line. The solution obtained using the GA has an idleness equal of the solution obtained through the application of heuristics. That means that the solution presented by the GA is a valid solution to this problem.

Table 2 Comparison of the results obtained through the GA with the ones obtained using heuristics

Workstation		*1*	*2*	*3*
GA solution	Tasks performed in workstation	9 1 2 5	3 10 4 11 12 6	7 13 8 14 15
	Tasks performed in workstation	I A B E	C J D K L F	G M H N O
	Sum of the time of the workstation tasks	131	130	127
	Workstation idle time	13	14	17
	Idleness	1,27%		
Heuristics solution	Tasks performed in workstation	1 9 2 10 11	12 3 4 13 14 6	5 7 8 15
	Tasks performed in workstation	A I B J K	L C D M N F	E G H O
	Sum of the time of the workstation tasks	128	129	131
	Workstation idle time	16	15	13
	Idleness	1,27%		

Although both solutions present the same idleness, analysing the table it is possible to realise that the solutions presented by both methods are different.

Concerning the comparison between the two software applications presented in Section 2 and the tool described in this work, it is possible to conclude that one of the main distinctions is that both commercial applications allow the balancing to be made manually by the user, in which case the software provides charts depicting the load of the workstations. This functionality is not implemented in the tool described here. Concerning the ProBlance software, and similarly to the GA application developed by the authors, users can select to perform a LBP-1or LBP-2 type balancing.

7 Conclusions and perspectives for future developments

When defining the layout for flow or production lines, it is necessary to perform the line balancing. In this paper was presented an application that was developed to solve LBPs based on GA. This application, developed in the C# programming language, includes a user interface.

To test the developed application, it was applied to an example problem, in order to make possible a comparison of the obtained results with those obtained using heuristic methods. For the particular case under analysis described in this paper, it was verified that the solution found with the GA presents an efficiency and idle time for the line equal to the solution obtained by the application of heuristic methods, while taking 35 minutes of run time. However, it should be mentioned that the GA found more than one feasible solution to the problem and in a lower amount of time than the one that was needed for finding the solution manually using the heuristic. Furthermore, the developed application is not error prone, as is the case with manual methods, particularly when the user is faced with complex line balancing that must consider several operations and various precedence relations among them.

The application presented in this work is part of a work envisioned to develop a tool for line balancing based on GA. The objective is to reach an application able to cope with

LBP-1or LBP-2 type balancing problems and that reaches a solution faster than the commercial products already available.

Given the results obtained with this application, the authors intend to improve the GA in order to allow the user to dynamically adjust the cycle time or the number of workstations to improve the results for more complex problems.

References

Aguiar, G.F. et al. (2007) 'Simulações de Arranjos Físicos por Produto e Balanceamento de Linha de Produção: O Estudo de um Caso Real no Ensino para Estudantes de Engenharia', *Cobenge 2007 – XXXV Congresso Brasileiro de Educação em Engenharia*, Curitiba/PR, Brazil, 10–13 September, pp.2P01-1–2P01-15.

AviX® (2013) *AviX® Balance* [online] http://www.avix.eu/en/our-products/avix-balance.html (accessed May 2013).

Davis, L. (1992) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Professional, Reading, MA, ISBN: 978-0201157673.

Holland, J. (1992) *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, ISBN: 978-0262581110.

Jonnalagedda, B.M. (2010) 'Heuristic procedure for mixed model assembly line balancing problem', *2010 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Macao, China, 7–10 December, pp.552–556.

Li, G., Yao, Z., Yuan, Q. and Fang, M. (2006) 'Optimization methods of the product assembly line system', *ITIC 2006. International Technology and Innovation Conference*, Hangzhou, China, 6–7 November, pp.19–22.

Mayerle, S.F. and Santos, R.N. (2003) 'Algoritmo genético para o balanceamento de linhas de produção', *XXIII Encontro Nac. de Eng. de Produção*, Ouro Preto, MG, Brazil, 21–24 October, pp.1–8.

Proplanner (2013) *ProBalance* [online] http://www.proplanner.com/index.cfm?nodeID=25083 (accessed May 2013).

Scholl, A. and Voß, S. (1996) 'Simple assembly line balancing – heuristic approaches', *Journal of Heuristics*, Vol. 2, No. 3, pp.217–244.

Ying, B., Hongshun, Z. and Liao, Z. (2009) 'Mixed-model assembly line balancing using the hybrid genetic algorithm', *ICMTMA '09. International Conference on Measuring Technology and Mechatronics Automation*, Zhangjiajie, Hunan, China, 11–12 April, pp.242–245.