Int J Adv Manuf Technol (2015) 79:1293–1302
DOI 10.1007/s00170-015-6923-7

ORIGINAL ARTICLE

Planning the construction process of a robotic arm using a genetic algorithm

P. T. Zacharia · S. A. Tsirkas · G. Kabouridis · G. I. Giannopoulos

Received: 10 August 2014 / Accepted: 16 February 2015 / Published online: 3 March 2015

Abstract This paper presents an application of the simple assembly line balancing problem of type 2 (SALBP-2). A considerable number of robot components comprising a robotic arm are elaborated through four machine tools. Real data values for the processing times are gathered and precedence constraints are considered for finding the optimum cycle time needed for finishing the robotic arm. A genetic algorithm (GA) is used as the optimization tool for the solution of the SALBP-2, which is an NP-hard problem. The experimental results demonstrate the effectiveness and efficiency of the proposed GA in determining the optimum sequence of the tasks assigned to workstations and providing the optimum cycle time.

Keywords Scheduling · Production · Process planning · Machine tools · Metal parts

1 Introduction

An assembly line is a familiar example from the realm of manufacturing. Flow lines are found in all types of industries, wherever products may be imagined to move along from station to station. Assembly is a process by which subassemblies and components are put together, yielding the finished products. The assembled product takes shape gradually, starting with one part, the remaining parts being attached at the various stations the product visits [1].

P. T. Zacharia · S. A. Tsirkas · G. Kabouridis · G. I. Giannopoulos (✉)
Department of Mechanical Engineering, Technological Educational Institute of Western Greece, Megalou Alexandrou 1, 263 34 Patras, Greece
e-mail: ggiannopoulos@teiwest.gr

Assembly lines are a traditional widely used type of production systems for mass and large-scale production. They consist of a number of workstations arranged along an automated material handling system such as a conveyor belt. Work pieces are moved along the line from station to station, while each station performs a number of repeated operations (tasks) necessary to manufacture a desired final product. These tasks can be performed by machinery, robots, and/or human operators. The order of processing the tasks is restricted by technological and organizational conditions, e.g., mounting a radio device requires having installed a fixture and cables before. These partial orderings of tasks are collected within a precedence graph. Once the part enters a station, a task is then performed on the part and the part is fed to the next operation. Each sub-product unit remains at each station for a fixed work rate called the *cycle time* of the line.

The rapid qualitative and quantitative changes in market demands cause manufacturers to seek the best possible methods for managing their assembly lines so as to produce more sophisticated and more competitive products. To respond to these diverse market needs, modern assembly lines must be highly automated and easily reconfigurable [2]. The simple assembly line balancing problem (SALBP) is a decision problem arising when an assembly line has to be configured or redesigned. The problem consists of determining the optimal partitioning (balancing) of the assembly work among the workstations while optimizing one or more objectives without violating the restrictions imposed on the line. Rashid et al. [3] have recently provided an interesting review concerning research on assembly sequence planning and SALBP that use soft computing approaches. Fathi and Ghobakhloo [4] have efficiently attempted to supplement the previous review. Simultaneously, they have provided up-to-date information regarding the research on SALBP and have proposed a guide to future relevant research.

According to the definition of simple assembly line balance (SALB) problems by Scholl and Becker [5], SALBP-1, SALBP-2, and SALBP-E are the three common subcategories of SALBP. These kinds of problems are of combinatorial nature and belong to the *NP*-hard class of combinatorial optimization problems [6]. Therefore, exact algorithms can hardly be designed to solve large sizes of any variant of SALBP and consequently, the right way to proceed is through the use of heuristics techniques.

SALBP type 1 [7–19] intends to assign tasks to workstations such that the number of stations is minimized for a prespecified cycle time which generally happens when the organization desires to design new assembly lines. SALBP type 2 [20–26] aims to minimize the cycle time, or equivalently, to maximize the production rate for a specific number of stations. This problem is appropriate for the available systems which intend to improve their line efficiency. Finally, SALBP type E [27, 28] targets at maximizing the line efficiency thereby simultaneously minimizing the number of stations and cycle time considering their interrelationship. The solution of SALBP problems may be achieved by utilizing approaches such as GA [7, 20, 21, 28], ant colony optimization [10, 13, 26], particle swarm optimization [14, 15, 18], Petri net [9, 11, 22], tabu search [8], bacterial foraging optimization [19], or other heuristic algorithms [12, 16, 17, 23–25, 27].

This paper introduces a genetic algorithm (GA) [29, 30] for solving the SALBP-2 for the assembly line of a robot. The central focus of this work lies on the solution of the assembly line balancing problem by using an evolutionary algorithm for the test case of a robotic arm construction. Thus, the main body of this paper concentrates on the analysis of the assembly line balancing problem using a GA, whereas the construction of the robotic arm and the resulting processing times are used to demonstrate the applicability of the proposed approach. To the best of our knowledge, no other published work in the literature studies the SALBP-2 for a robotic system. Hence, unfortunately, there is no existing solution algorithm to make comparisons with our method.

The rest of the paper is organized as follows: Section 2 considers selected terms and aspects of assembly line balancing of type 2. Section 3 presents the assembly problem tackled in practice for the manufacture of the robot components. Computational results concerning the performance of the GA, while conclusions and directions for future work are pointed out and discussed in Section 5. To the authors' best knowledge, it is the first time that a simple assembly line balancing problem is considered for the manufacturing of a robotic arm via the use of cutting and forming machine tools. For this reason, appropriate experimental data are presented while a suitable genetic algorithm is utilized. The outcome of the proposed paper may have industrial application in the effort to effectively plan the production line of a robotic arm or other similar manufacturing lines associated with metal cutting and forming.

2 Background of assembly line balancing

Each assembly line can be modeled as a sequence of m stations. Manufacturing a single product on the line requires the partitioning of the total work into a set $V=\{1,\ldots,n\}$ of n elementary operations called tasks. Each task j is performed on exactly one station and requires a processing time t_j. At each station, a specific set of tasks, called station load $S_z(z=1,\ldots,m)$ is repeatedly executed. The station time $tS_z = \sum_{j\in S_z} t_j (z = 1, \ldots, m)$ must not exceed a given cycle time c available per work cycle. The tasks are partially ordered by precedence relations defining a directed acyclic graph (DAG) $G=(V,E)$; with V being the set of the nodes denoting the tasks in G and E the set of the edges representing the precedence constraints among these tasks. Besides precedence constraints, the precedence graph also shows independencies of tasks.

The classical problem contains the following main characteristics:

- Mass-production of one homogeneous product
- Given production process
- Paced line with fixed cycle time c
- Deterministic operation times t_j
- The processing sequence of tasks is subject to precedence restrictions
- No assignment restrictions besides the precedence constraints
- Serial line layout with m one-sided stations
- All stations are equally equipped with respect to machines and workers
- Every task must be performed only once in each cycle
- Setups are assumed to be negligible

The SALBP-2 has the objective of minimizing the cycle time c of the assembly line given the number of workstations m.

The notations used in this article are listed below:

n	Number of assembly tasks
m	Number of stations in the line
j	Index of the assembly task j ($j=1, \ldots, n$)
t_j	Processing time for task j
c	Cycle time of the line
S_z	Load of station z ($z=1, \ldots, m$); i.e., the set of the tasks assigned to station z
tS_z	Time required to complete all the tasks assigned to station z
G	A precedence graph for the assembly tasks (denoting the precedence constraints between the assembly tasks)
V	The set of vertices in G
E	The set of edges in G

3 Problem statement

Most robots are designed for repetitive manufacturing work. They handle tasks, which are demanding, hazardous, or tiresome, in a more efficient way than human beings because they are significantly precise. The most common manufacturing robot is the robotic arm. An industrial robot with six joints closely resembles a human arm since it has the equivalent of a shoulder, an elbow, and a wrist. In the present study, the construction process of a metallic robotic arm has been chosen as a test case. This kind of machine has been preferred among others since its production presents complexity, various challenges, and technological importance.

The robotic arm under consideration is generally made from 3-mm thick aluminum sheet, is approximately 600 mm in length and has been designed to meet specific needs of mobility, i.e., six principal degrees of freedom. Furthermore, the developed robot uses a controller to control the operation of arm action and movement. The computer controls the robot by rotating individual step motors connected to each individual joint, allowing sufficient link lengths. Unlike ordinary motors, step motors move in exact. This allows the computer to move the arm very precisely, repeating exactly the same movement over and over again. A simple, scalable control system allows coordinated Cartesian control, and offers expandability for future research. Figure 1 illustrates the prototype that has been constructed for the needs of the present study.

As depicted, the developed robotic arm consists of a waist, a shoulder, an elbow, a wrist, and a gripper. All these robotic parts are coupled with appropriate joints. The metal frame of the robotic arm is composed of several aluminum components which are illustrated and numbered in Fig. 2.

The aluminum structure of the robotic arm takes shape in a simple assembly line which has four workstations, i.e., a milling, a drilling, a lathe, and a bending machine which are depicted in Fig. 3.

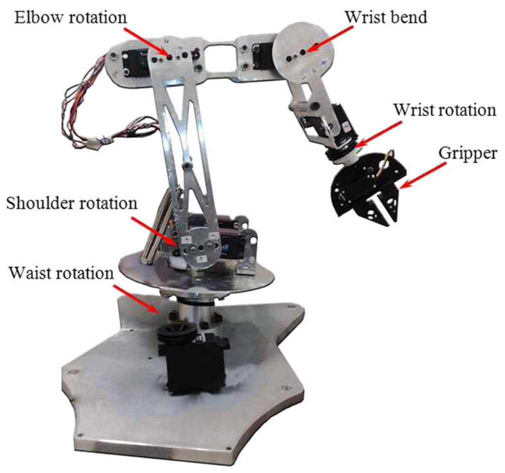


Fig. 1 Manufactured robotic arm

4 The proposed optimization approach

GAs are probabilistic search methods that employ search techniques inspired by Darwin's evolutionary theory which is based on the principles and mechanisms of natural selection and the survival of the fittest. GAs employ a random yet directed, search for finding the globally optimal solution. They have the advantage over the gradient descent techniques that they do not require the derivative of the objective function and the search is not biased toward the locally optimal solution. In contrast to random sampling algorithms, GAs have the ability to direct the search toward relatively promising regions in the problem's search space. In addition, they have been empirically proven very effective in solving a large number of complex combinatorial optimization problems.

The architecture of any GA consists of the following seven basic components:

a. A representation mechanism, i.e., a way of encoding the phenotypes to genotypes.
b. A decoding mechanism, i.e., a way of mapping the phenotypes to actual solutions of the optimization problem under consideration.
c. An evaluation mechanism, i.e., a way of computing the cost-function for each genotype.
d. A way to generate the initial population of the genotypes.
e. Generate new genotypes by applying operators on the entire population.
f. Control parameters
g. A termination condition

The sketch of the proposed GA in pseudo-code is presented in Table 1.

The main steps of the GA are discussed in detail in the following paragraphs:

The representation mechanism: In this work, a real-valued GA was adopted for use, i.e., genotypes are represented by floating-point vectors. Therefore, since actual assembly line balancing problem (ALBP) solutions are represented by strings of integers [6], an appropriate mapping is needed from the genotypic state-level (the real-valued vectors) to the phenotypic level (the actual ALB solutions). To achieve this mapping, a simple yet effective topological ordering scheme has been developed based on the relative priorities impose by the components of a genotype. Assuming an n-task ALBP with precedence relations given by a DAG $G=(V, E)$, the developed encoding scheme consists of generating a topological sort of G from a specific n-dimensional floating-point vector ψ (genotype). Each vector's component ψ_i ($i=1, \ldots, n$) represents the relative priority of task i ($i \in V$). The topological sort is therefore a ranking of all the tasks

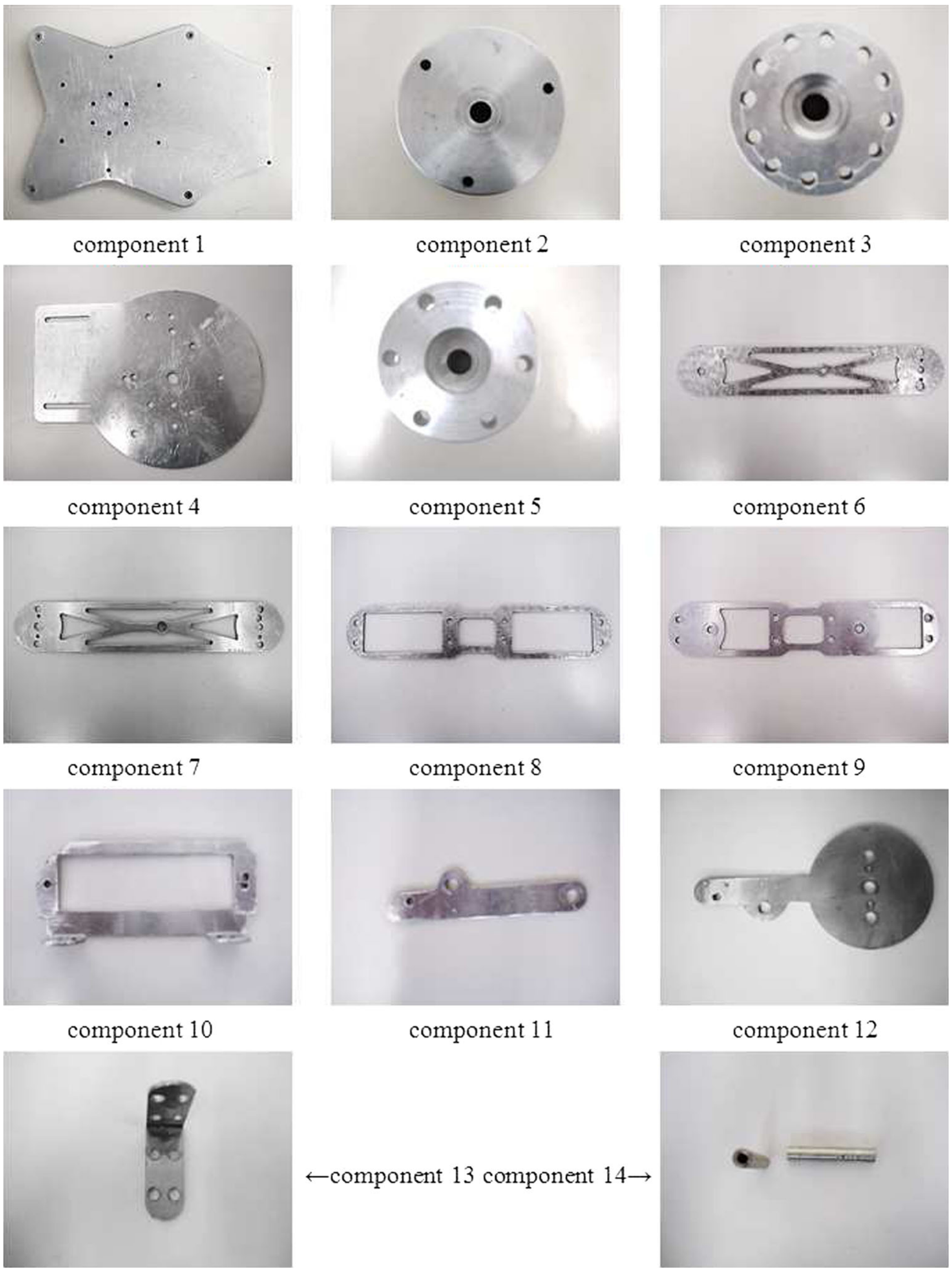


Fig. 2 Numbered components of the robotic aluminum frame

according to their priorities in an appropriate order to meet the precedence constraints.

In each step, the tasks with no predecessors are identified and put in set V'. Then, the task in V' having the highest gene's value in ψ is selected, removed from V', and placed in the next available position of *PS*. The process is repeated until the completion of *PS*.

The decoding mechanism: Once a specific genotype is encoded into a phenotype, a way is needed to assign the tasks in the generated task-sequence into the stations. In this work, a scheme proposed by Kim et al. [31] was adopted; this scheme was found to be superior to other traditional schemes [6] in terms of quality of solutions. This scheme works as follows:

- Step 1: Set c initially equal to the theoretical minimum cycle time, i.e., $c_{\mathrm{th}}=t_{\mathrm{sum}}/m$.
- Step 2: Assign as many tasks as possible into the first *m-1* workstations. Assign all the remaining tasks to the last workstation, *m*.
- Step 3: Calculate the work load W_z for each workstation z (z=1, 2, …, m), and the potential workload PW_z (z=1, 2, …, m-1) as follows: W_z=the station time tS_z (z=1, 2, …, m). $PW_z=tS_z+$ the processing

Fig. 3 Machining workstations

time of the first task assigned to (z+1) station (z=1, 2, …, m-1).

- Step 4: Set $c_{\mathrm{w}}=\max\{W_1, W_2, \ldots, W_m\}$ and $\widetilde{c}=\min\{PW_1, PW_2, \ldots, PW_{m\text{-}1}\}$
- Step 5: If $c_{\mathrm{w}} \succ c$ then go to Step 2, or else return c_{w}

The evaluation mechanism: The evaluation mechanism uses the objective function of the current problem, which has to be minimized. This function is transformed to the fitness function, which is evaluated for all the chromosomes of the population. The value of the fitness function for one chromosome is a reflection of how well this chromosome is adapted to the environment, i.e., the ability of the chromosome to survive and be reproduced in the next generation.

In the problem discussed here, the objective function deals with the cycle time c of the assembly line. The fitness function is the inverse of the objective function:

$$\text{fitness} = \frac{1}{c} \tag{1}$$

Since $c \neq 0$, it is not necessary to take any precaution to avoid infinite values.

The initial population: Usually, the initial population is randomly created in order to uniformly distribute the selected chromosomes (solutions over the search space).

The GA operators: In the proposed GA, *reproduction* is based on the roulette wheel scheme, where the chromosomes that will be copied are selected with rates proportional to their fitness. *Crossover* is a recombination operator and follows the reproduction. The role of this operator is to join together parts of several individuals in order to produce new ones for the next generation. The individuals are randomly selected according to a predefined probability (crossover rate). In the proposed GA, the one-point crossover [32] is applied. *Mutation* is applied in order to inject new genetic material into the population and thereby avoid premature convergence to local minima. The mutation operator is applied changing a random gene (i.e., a floating number) to another one lying in the searching space with a small-predefined probability (mutation rate).

Control parameters: The most important control parameters are the following:

Population size: The population size determines the number of the chromosomes and therefore, how much genetic

Table 1 Utilized GA pseudo-code

```
Genetic algorithm

Input: n tasks; m workstations; processing times $t_j$, precedence relations
  $G=(V,E)$, $p_c$, $p_m$, max_gen.
Output: cycle time c
begin
  Step 1. Initialization:
    Generate randomly an initial population ($P_k$)
    Set generation counter, $k=0$
  Step 2. Evaluation:
    Compute fitness (i) for each $i \in P_k$
  Step 3. New population generation
    Apply reproduction operator in $P_k$ using roulette wheel selection
      scheme.
    Apply one-point crossover to some individuals selected according
      to probability $p_c$.
    Apply random mutation to some individuals selected according
      to probability $p_m$.
    $k=k+1$;
  Step 4. Evaluation:
    Compute fitness (i) for each $i \in P_{k+1}$
  Step 5. Elitist preserving strategy:
    Determine the best solution in $P_{k+1}$;
    Compare the best solution in $P_{k+1}$ with the best solution in $P_k$
      and keep the best one between them as the overall best solution.
  Step 6. if $gen \le$ max_gen then go to Step 3.
  Step 7. Return c;
end;
```

material is available during the genetic search. It has to be mentioned that a small population size covers a small area of the search space, which means that it may not be a representative sample of the solutions. So, a small population size decreases the possibility of finding a global optimum. On the other hand, a large population size significantly increases the CPU time. The population size depends on the nature and the complexity of the current problem. In this work, the proposed algorithm was tested for various population sizes. Finally, the selected population size is equal to 100 for the case of 4 workstations and 30 tasks.

Crossover rate: The crossover rate (p_c) determines the frequency with which the crossover operator is applied to the chromosomes of the population, so that a new population is generated. The higher the crossover rate is, the more individuals are introduced in the new population. The crossover rate is usually in the range between 0.6 and 1.0 and in our case, it is selected equal to 0.8 after a considerable number of trials.

Mutation rate: The mutation rate (p_m) determines the probability that a gene's value in a chromosome would be changed. Mutation introduces new areas of the unexplored search space. However, the mutation rate should not be too high, because it increases the randomness in the search. The mutation rate is usually less than 0.4 and it was selected equal to 0.1 in our case.

Elitism: Elitism is the selection strategy that guarantees the survival of the best chromosome of the population to the next generation. This is achieved by comparing the best chromosome of the current generation with the best one of the previous generation and preserving the best of the two chromosomes.

Termination conditions: There is no mathematical proof of convergence or any guarantee that the genetic algorithm will find the global optimum. In addition, it is not clear which is the best way to terminate the algorithm. In many cases, a maximum number of iterations (generations) is defined in advance (*max_gen*). However, the predetermination of the maximum number of generations implies that the duration of the genetic search is fixed, regardless of the search success. Moreover, it is difficult to determine beforehand the number of generations needed to find near-optimum solutions. Thus, an assessment of the quality level of the genetic algorithm should be made online.

In our approach, the condition to be satisfied so that the evolution is aborted is the iteration of the same solution for a predefined number of generations. So, the algorithm terminates by defining in advance the number of iterations (generations) for which the same chromosome constantly appears as the optimum one. It should be mentioned that the best solution appears for several iterations before a fortuitous crossover or mutation produces a better solution. For this reason, the maximum number of iterations should be large enough; otherwise, a misleading result may arise.

5 Computational study

For the manufacturing of the robotic arm, 14 components should be machined in four machine tools (milling, lathe, drilling, and bending machine); 4 parts for component no. 13 and 10 parts for component no. 14. Table 2 summarizes all the details to the 57 total tasks. The aim is to find the optimum sequence of tasks assigned to stations so that the minimum total time for executing all tasks is achieved. The precedence constraints are presented in Fig. 4.

Considering the total time for each of the 57 tasks given in Table 2 as well as the aforementioned precedence relations, the

Table 2 Detailed table of execution times for each robot component

No. of tasks	No. of component	Type of task	Execution time (s)	Component placement time (s)	Cutting tool placement time (s)	Zero points time (s)	Total time (s)
1	1	milling	2937	103	20	45	3105
2	1	drilling	55	2	17	–	74
3	2	milling	624	103	20	45	792
4	2	drilling	34	2	17	–	53
5	3	milling	658	103	20	45	826
6	3	drilling	22	2	17	–	41
7	4	milling	695	103	20	45	863
8	4	drilling	48	2	17	–	67
9	5	milling	355	103	20	45	523
10	5	drilling	26	2	17	–	45
11	6	milling	681	103	20	45	849
12	6	drilling	24	2	17	–	43
13	7	milling	884	103	20	45	1052
14	7	drilling	59	2	17	–	18
15	8	milling	617	103	20	45	785
16	8	drilling	40	2	17	–	59
17	9	milling	638	103	20	45	806
18	9	drilling	36	2	17	–	55
19	10	milling	376	103	20	45	544
20	10	drilling	6	2	17	–	25
21	10	bending	4	37	–	–	41
22	11	milling	228	103	20	45	396
23	11	drilling	7	2	17	–	26
24	12	milling	217	103	20	45	385
25	12	drilling	22	2	17		41
26	13 (1st out of 4 items)	milling	783	103	20	45	951
27	13	drilling	44	2	17	–	252
28	13	bending	4	37	–	–	41
29–37	13 (3 items)						
38	14 (1st out of 10 items)	lathing	1396	38	135	–	1569
39	14	drilling	46	2	17	–	65
40–57	14 (9 items)						

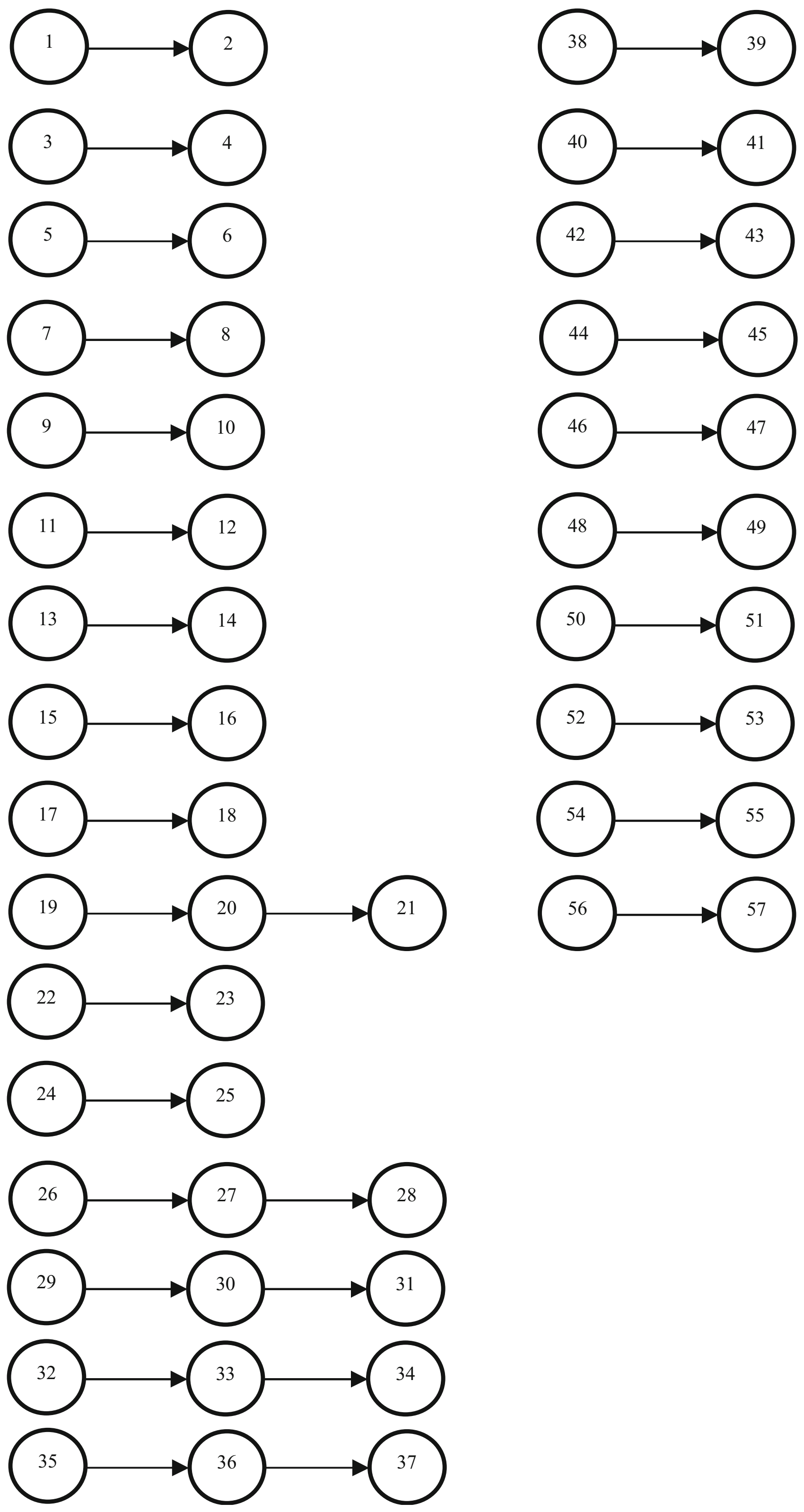
minimum time yielded by the proposed genetic algorithm is equal to 8209 s. The station load for each workstation is provided by the GA best solution and concerns the tasks assigned to each workstation in a specific order and is subject to the aforementioned constraints. The resulting task sequence for each workstation corresponding to c=8209 is as follows:

- Workstation1: 11–22–48–12–29–5–15–6–50–3–24
- Workstation2: 25–9–10–44–45–52– 38–39–51–16–46–13–14
- Workstation3: 19–20–21–17–7–8–47–1–2–26–27–28–35–36–37–49–53
- Workstation4: 42–43–56–57–32–54–55–33–34–30–31–18 –4–40–41–23

In particular, Workstation1 starts with task11 and after finishing, it continues with task 22 and so on until it finishes with task 24. Similarly, Workstation2 starts with task 25, then accomplishes task 9 and finishes with task 14. The robotic arm is manufactured when all tasks assigned to each workstation have been accomplished. According to the aforementioned assignment, the station time for each workstation is:

- Workstation1: 8206 s

Fig. 4 Precedence constraints



- Workstation2: 8208 s
- Workstation3: 8209 s
- Workstation4: 8207 s

which is close to the cycle time.

Figure 5 shows the evolution of the genetic algorithm. Although it is not clear from Fig. 5, the best cycle time (equal

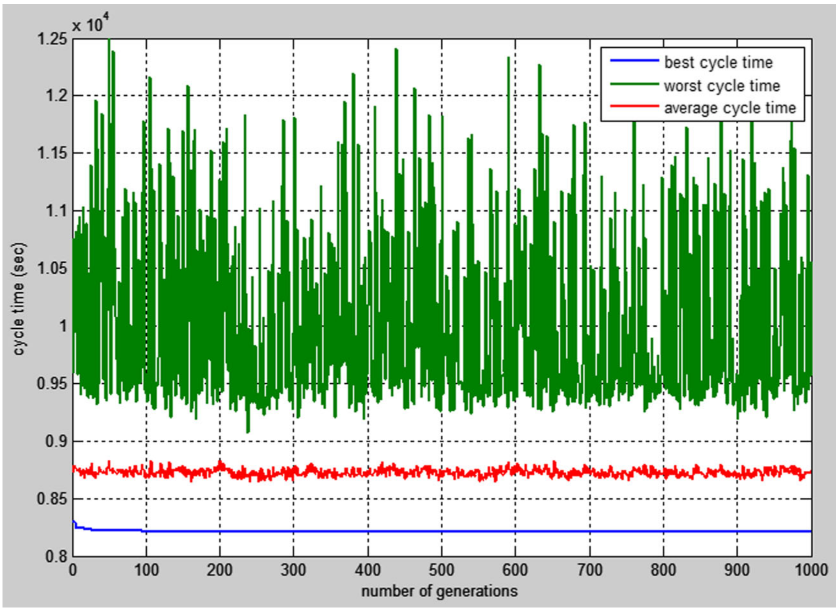


Fig. 5 The evolution of the GA

to 8209 s) is achieved in the 923rd generation. For sake of clarity, Fig. 6 zooms in the convergence point.

6 Conclusions

This work deals with the assembly line balancing problem of type 2 in a real problem of constructing a robotic arm. The robot components are formed using a number of machine tools and the processing times as well as the precedence relations are taken into account for the solution of the problem. The experimental results show that the genetic algorithm exhibited good performance in finding the optimum cycle time for the accomplishment of the construction of robotic arm. Considering future research, some perspectives are discussed below:

a. Applying heuristics in order to minimize the idle time, i.e., maximize the line efficiency.
b. Considering the operator skills problem, where various task processing times are considered for the execution of a certain task depending on which worker executes the task.
c. Taking into account that real-world production systems are often afflicted with uncertainty, variability, and imprecision, fuzzy processing times could be applied.

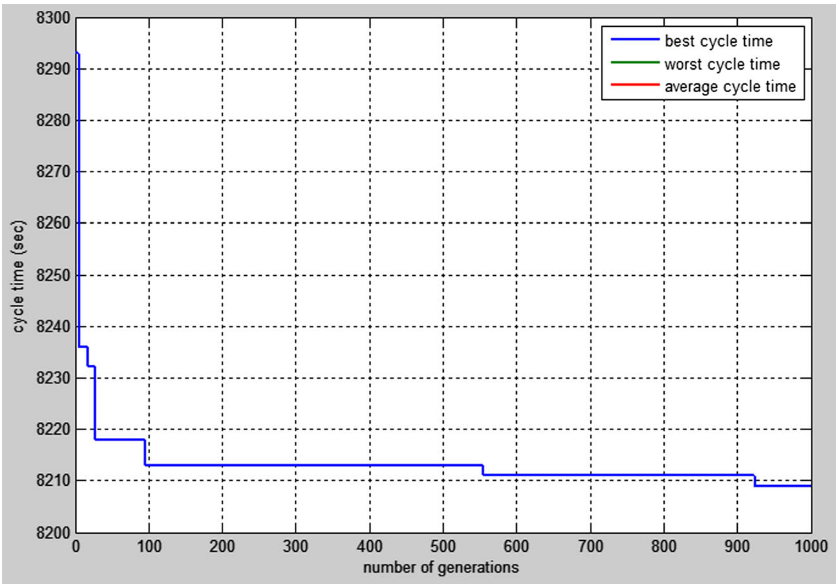


Fig. 6 Zoom in the convergence point

d. Applying a U-shaped assembly line layout, where stations work at two segments of the line facing each other simultaneously.

Acknowledgments This research has been co-financed by the European Union (European Social Fund—ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF)—Research Funding Program: ARCHIMEDES III Investing in knowledge society through the European Social Fund. The authors wish to thank pro-graduate students L.K. Kiafas and L.S. Vasilakis for their help in the construction of the robotic arm.

References

1. Rekiek B, De Lit P, Pellichero F, L’Eglise T, Fouda P, Falkenauer E, Delchambre A (2001) A multiple objective grouping genetic algorithm for assembly line design. Int J Adv Manuf Technol 12(5–6): 467–485
2. Nearchou AC (2007) Balancing large assembly lines by a new heuristic based on differential evolution method. Int J Adv Manuf Technol 34(9–10):1016–1029
3. Rashid MFF, Hutabarat W, Tiwari A (2012) A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. Int J Adv Manuf Technol 59(1–4):335–349
4. Fathi M, Ghobakhloo M (2014) A technical comment on “a review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches”. Int J Adv Manuf Technol 71(9–12):2033–2042
5. Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 168(3):666–693
6. Scholl A (1999) Balancing and sequencing of assembly lines, 2nd edn. Physica, Heidelberg
7. Goncalves JF, Almeida JRD (2002) A hybrid genetic algorithm for assembly line balancing. J Heuristics 8(6):629–642
8. Lapierre SD, Ruiz A, Soriano P (2006) Balancing assembly lines with tabu search. Eur J Oper Res 168(3):826–837
9. Kilincci O, Bayhan GM (2006) A Petri net approach for simple assembly line balancing problems. Int J Adv Manuf Technol 30(11–12):1165–1173
10. Zhang ZQ, Cheng WM, Tang LS, Zhong B (2007) Ant algorithm with summation rules for assembly line balancing problem. In: International conference on management science and engineering, ICMSE’07 (14th), 4421875, pp 369–374
11. Kilincci O, Bayhan GM (2008) A *P*-invariant-based algorithm for simple assembly line balancing problem of type-1. Int J Adv Manuf Technol 37(3–4):400–409
12. Yeh DH, Kao HH (2009) A new bidirectional heuristic for the assembly line balancing problem. Comput Ind Eng 57(4):1155–1160
13. Sulaiman MNI, Choo YH, Chong KE (2011) Ant colony optimization with look forward ant in solving assembly line balancing problem. Data Min Optim 5976514:115–121
14. Dou J, Li J, Lv Q (2011) A hybrid particle swarm algorithm for assembly line balancing problem of type 1. In: International conference on mechatronics and automation, 5986373, pp 1664–1669
15. Dou J, Su C, Li J (2011) A discrete particle swarm optimization algorithm for assembly line balancing problem of type 1. In: 3rd international conference on measuring technology and mechatronics automation, pp 44–47
16. Fathi M, Jahan A, Ariffin MKA, Ismail N (2011) A new heuristic method based on CPM in SALBP. J Ind Eng Int 7(13):1–11
17. Ariffin MKA, Fathi M, Ismail N (2012) A new heuristic method to solve straight assembly line balancing problem. Pertanika J Sci Technol 20(2):355–369
18. Dou J, Li J, Su C (2013) A novel feasible task sequence-oriented discrete particle swarm algorithm for simple assembly line balancing problem of type 1. Int J Adv Manuf Technol 69(9–12):2445–2457
19. Atasagun Y, Kara Y (2014) Bacterial foraging optimization algorithm for assembly line balancing. Neural Comput Applic 25:237–250
20. Zhang R, Chen D, Wang Y, Yang Z, Wang X (2007) Study on line balancing problem based on improved genetic algorithms. In: International conference on wireless communications, networking and mobile computing, WiCOM 2007, 4340283, pp 2033–2036
21. Gu L, Hennequin S, Sava A, Xie X (2007) Assembly line balancing problems solved by estimation of distribution. In: Proceedings of the 3rd IEEE international conference on automation science and engineering, IEEE CASE 2007, pp 123–127
22. Kilincci O (2010) A Petri net-based heuristic for simple assembly line balancing problem of type 2. Int J Adv Manuf Technol 46(1–4):329–338
23. Blum C (2011) Iterative beam search for simple assembly line balancing with a fixed number of work stations. SORT Stat Oper Res Trans 35(2):145–164
24. Tang Q, Lu S, Li M, Floudas CA (2011) Novel cellular automata algorithm for assembly line balancing problem of type-2. In: 6th international conference on pervasive computing and applications, 6106542, pp 422–428
25. Avikal S, Mishra PK, Jain R (2012) A model for assembly line balancing problems. In: Students conference on engineering and systems, 6199117, pp 1–4
26. Zheng Q, Li M, Li Y, Tang Q (2013) Station ant colony optimization for the type 2 assembly line balancing problem. Int J Adv Manuf Technol 66(9–12):1859–1870
27. Wei N-C, Chao I-M (2011) A solution procedure for type E simple assembly line balancing problem. Comput Ind Eng 61(3):824–830
28. Zacharia PT, Nearchou AC (2013) A meta-heuristic algorithm for the fuzzy assembly line balancing type-E problem. Comput Oper Res 40(12):3033–3044
29. Holland JH (1975) Adaption in natural and artificial systems. The University of Michigan Press, Ann Harbor
30. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
31. Kim YK, Kim YJ, Kim Y (1996) Genetic algorithms for assembly line balancing with various objectives. Comput Ind Eng 30(3):397–409
32. Michalewicz Z (1996) Genetic algorithms + data structures = evolution program, 3rd edn. Springer, Berlin