A DIFFERENTIAL EVOLUTION ALGORITHM FOR SIMPLE ASSEMBLY LINE BALANCING

Andreas C. Nearchou

Department of Business Administration,
University of Patras 26 500 Rio-Patras, Greece,
E-mail: nearchou@upatras.gr, Fax: +30 2610-996.350

Abstract: This paper describes the application of differential evolution algorithm (DEA) to the simple assembly line balancing problem (SALBP). DEA is an evolutionary algorithm similar to a real-coded genetic algorithm for global optimization over continues spaces. The paper is concerned with SALBP type-1 whose objective is to minimize the number of workstations required to manufacture a product in an assembly line within a given fixed cycle time. Extensive experimental work over public benchmarks test problems show the effectiveness of the proposed approach.

Key words: assembly line balancing, differential evolution, meta-heuristics, manufacturing optimization.

1. INTRODUCTION

Production planning and control is associated with a large number of complex optimization problems. Most of these problems are of combinatorial nature and have been proved to be *NP*-complete, i.e., there is no exact algorithm that can solve them in polynomial time unless it is proved that *P=NP*. The combinatorial nature of these problems encourages the use of modern meta-heuristics techniques such as evolutionary algorithms, simulated annealing, tabu-search, etc.

The assembly line balancing problem (ALBP) 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 (Baybars, 1986, Scholl, 1999). The simple assembly line balancing problem (SALBP) is a basic version of the general problem and has captured the research interest for the last four decades. Two formulation types are commonly used with SALBP: (a) SALBP-1 which attempts to minimize the number of stations for a given fixed cycle time, and (b) SALBP-2 which attempts to minimize the cycle time of the line for a given number of stations. The former type is used when a new assembly line has to be installed in the shop floor, while the latter type is used in an existing assembly line when changes in the production process and manufacturing requirements occur.

Any variant of the ALBP is known to be NP-complete combinatorial problem, which implies that the right way to proceed is through the use of heuristic techniques. A large number of exact and heuristic techniques for the SALBP-1 are available in the literature. Exact algorithms are mostly based on the branch and bound method and the dynamic

programming approach. Some of the most powerful branch and bound procedures are the FABLE being the most effective of all (Scholl, 1999). Recently, some researchers turned their attention to meta-heuristics techniques such as tabu-search (Scholl and Voβ, 1996), simulated annealing (Suresh and Sahu, 1994) and genetic algorithms (GAs) (Anderson and Ferris, 1994, Kim, *et al.*,1996). However, in respect to other sequencing and scheduling optimization problems little attention has been paid to the application of GAs to ALBPs.

In this paper the potential of the differential evolution algorithm (DEA) for the solution of the SALBP-1 is investigated. DEA is an evolutionary algorithm very similar to a real-coded genetic algorithm, which has been recently applied with high success to solve various complex numerical optimization problems (Storn and Price, 1997).

2. THE SALBP-1

Following the analysis given in (Baybars, 1986, Scholl, 1999) the SALBP-1 can be stated as follows:

- An assembly line consists of workstations arranged along a conveyor belt or a similar materials handling equipment. Let S={1,…,m} the set of workstations..
- Manufacturing a single product on the assembly line requires the partitioning of the total assembly work into a set of elementary operations called tasks. Let V={1,…n} the set of tasks.
- Each task j (j∈V) is performed on exactly one workstation and requires a deterministic (Johnson, 1988), EUREKA (Hoffmann, 1992), and SALOME-1 (Scholl and Klein, 1997) with the latter processing time t_j. Let t_{sum} the sum of all task times.
- The assembly line is associated with a cycle time denoting the maximum (or average) processing time available for each work cycle. Each station can complete its assigned tasks within the specified cycle time.
- The tasks are partially ordered by precedence relations. That is, precedence constraints between the tasks occur and must not be violated.
- The objective is to minimize the number of workstations subject to the given cycle time and the precedence constraints of the tasks.

Usually, ALBPs are modeled through the use of precedence graphs. Each node in the graph corresponds to a specific task, while an edge joining two nodes represents the precedence relation between the corresponding tasks. Figure 1 illustrates an example of a precedence graph for an 8-tasks ALBP having processing times between 3 and 17 time units. The numbers inside the nodes of the graph correspond to the task labels, and those outside the nodes to the processing times. Therefore, task 1 has a processing time equal to 11 time units, task 2 a processing time equal to 17 time units, etc. The precedence constraints for example, for task 6 defines that, this task must proceed after the completion of tasks 3 and 4 (direct predecessors), and tasks 1 and 2 (indirect predecessors). While, task 6 must be completed before its direct (or indirect) successors, which is task 8.

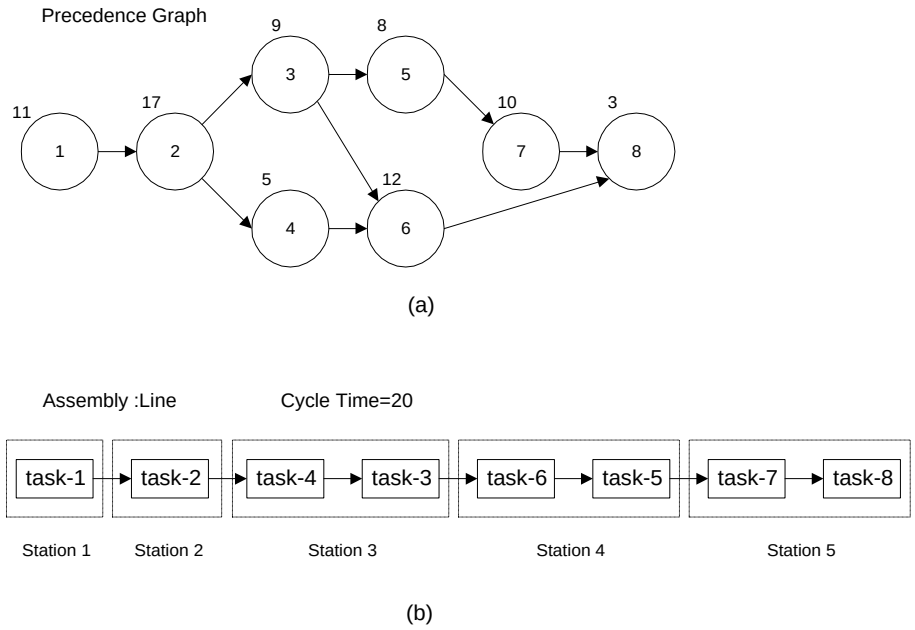

Fig.1 An example of 8-tasks ALBP: (a) the precedence graph. (b) Assembly line corresponding to the feasible line balance solution (1 2 4 3 6 5 7 8).

3. DIFFERENTIAL EVOLUTION FOR THE SALBP-1

Differential evolution (DE) is an evolutionary algorithm very similar to a real-coded genetic algorithm (GA). The differences rely on the way the mechanisms of mutation and crossover are performed over the floating-point vectors (chromosomes). The most distinct feature of DE is that it mutates vectors by adding weighted, random vector differentials to them. As with all evolutionary algorithms, DE starts by generating a population of real-valued *n*-dimensional vectors whose initial parameter values are chosen randomly from within user-defined bounds. This population undergoes evolution in a form of natural selection. In each generation, the operators of selection, mutation, and crossover are applied on the entire population in order to produce new, 'better' populations with higher 'quality' individuals. Actually, in every generation, each vector (chromosome) of the population becomes a *target* vector and crossovers with a *mutant* vector in order to produce a *trial* vector. Each mutant vector is generated, by adding the weighted difference between two randomly selected population vectors to a third vector. The best (that with the lowest cost) between the target and the trial vector survives into the next generation. The evolutionary process terminates after a number of generations and the structure of the best vector found so far is postulated as the definite solution to the actual optimization problem. The basic structure of DE algorithm is shown in figure 2, details about DE and the way it operates can be found in the pioneered work of (Storn and Price, 1997).

```
Procedure Differential_Evolution
begin
  Generate randomly a population Φ of Np
     real-coded vectors;
  Evaluate (Φ);
  while termination condition not satisfied do
  begin
     for j=1 to Np do
     begin
       Select the next target vector Xj from Φ;
       Choose randomly 3 vectors Xa, Xb, Xc
         from Φ;  // a≠b≠c≠j //
       Generate a mutant vector Vj according
         to the relation Vj=Xa+F.(Xb-Xc);
       Crossover the mutant and the target
         vector to generate a trial vector Uj;
       Evaluate Uj;
       if COST(Uj) < COST(Xj) then
        Replace Xj with Uj;
     endfor
     Save best-so-far vector to X*
  endwhile
  Return X*
end
```

Fig. 2: The general body of DE algorithm.

3.1 Encoding mechanism for the SALBP.

The natural coding for sequencing and scheduling problems including SALBP is permutation vectors, i.e., strings with integers. Therefore, a solution to the SALBP (a phenotype) is a sequence of tasks with each task represented by an integer number. When this sequence of tasks does not break the precedence constraints the solution is *feasible*. Then, the tasks are allocated into workstations (according to their order in the solution string) such that the sum of their processing times in each workstation does not exceed the cycle time. This scheme is demonstrated in Fig. 1.(b) for the 8-task SALBP with precedence graph shown in Fig.1(a). A cycle time equal to 20 time units was assumed. A feasible assembly balancing solution for this example is (1 2 4 3 6 5 7 8) which results to five workstations in the assembly line.

3.2 Decoding mechanism.

DE works with floating-point vectors. This means that an appropriate mapping is needed from the genotypic state-level (chromosomes) to the phenotypic level (actual assembly balancing solutions). Random-keys proposed by (Bean, 1994) is the only published representation mechanism used with floating-point chromosomes on combinatorial optimization problems. We found this scheme performing rather poor in the context of a DEA and for this reason we implemented a new more effective representation scheme. This scheme called the *Sub-Range* coding scheme works as in the following:

- For a SALBP with *n* tasks, divide the range [1…*n*] into *n* equal sub-ranges and save the upper bounds of each of the sub-ranges in the array SR=$[1/n, 2/n, 3/n, \ldots, n/n]^T$ (SR stands for Sub-Ranges).
- Build the phenotype of a specific chromosome according to the sub-range in which each specific gene's value belongs to.
- Check and repair if needed the phenotype so that not to contain redundant task labels.

For example, assuming a 5-task SALBP let the genotype, g = (0.23, 0.82, 0.03, 0.47, 0.62). Then the array *SR* has the form $SR = [0.2, 0.4, 0.6, 0.8, 1.0]^T$. The 1st gene in g has the value 0.23, i.e., lies in the 2nd sub-range (0.2<0.23≤0.4), and thus the corresponding phenotype becomes (2 _ _ _ _). The 2nd gene (=0.82) lies in the 5th sub-range (0.8<0.82≤1.0), and the phenotype becomes (2 5 _ _ _), and so on. Finally, the phenotype p corresponding to chromosome g is p=(2 5 1 3 4). Which means that the order the tasks are to be executed is, task 2, followed by task 5, followed by task 1, etc.

3.3 Repairing the phenotype so that to satisfy the precedence constraints.

When the order of the tasks in the generated phenotype does not break the precedence constraints (see section 2), then this sequence corresponds to a *feasible* solution. Otherwise, the solution is infeasible and must be repaired. To repair infeasible solutions we use the following simple yet effective procedure: Note that, as we experimentally observed, it is not necessary to repair all the members in the entire population but only a small portion. In particular, when 5% of the repaired individuals replace their infeasible original structures is enough.

```
Procedure Repair_Phenotype
begin
  for i=1 to n-1 do begin
     A=p[i];
     for j=i+1 to n do begin
       B=p[[j];
       if task-B is a direct or indirect predecessor
           of tsk-A then
         Swap(p[i], p[j]); Swap(g[i],g[j]); A=B;
       endif
     endfor
  endfor
end
```

3.4 Local improvement

Two schemes were used to improve the performance of the DEA: (a) A local search heuristic is applied on the population best solution. This heuristic works as follows: Find the tasks with no predecessors. Insert these tasks to the head of the task-sequence. Generate and check all the possible permutations of these tasks. Always keep track for the feasibility of the solution. The current best is only replaced when a more robust solution has been found. (b) If the diversity of the population becomes too low, then regenerate randomly a portion of its entire individuals..

4. COMPUTATIONAL RESULTS

To demonstrate the effectiveness of the proposed DEA we present computational results obtained on a set of public benchmark test problems found in the literature. The benchmarks are available through the Web at the location http://www.assembly-line-balancing.de/ Here we report the results obtained over the data set proposed by (Talbot *et al.*, 1986). This data set contains 64 test instances varying from 7 tasks to 111 tasks.

Table 1 Experimental results on Talbot *et al.* data set.

Problem	n	CT	**m***	**mDE**	**tcpu**
Mertens	7	6	6	5	0.0
		7	5	4	0.0
		8	5	3	0.0
		10	3	2	0.0
		15	2	6	0.0
		18	2	5	0.0
Bowman	8	20	5	5	0.0
Mansoor	11	48	4	3	0.0
		62	3	2	2.2
		94	2	2	0.0
Jaeschke	9	6	8	8	0.0
		7	7	7	0.0
		8	6	6	0.0
		10	4	4	0.0
		18	3	3	0.0
Jackson	11	7	8	8	0.0
		9	6	6	0.0
		10	5	5	0.6
		13	4	4	0.0
		14	4	4	0.0
		21	3	3	0.0
Mitchell	21	14	8	8	3.3
		15	8	8	0.0
		21	5	5	0.5
		26	5	5	0.0
		35	3	3	6.3
		39	3	3	0.0

In the experiments we used as objective the minimization of the line efficiency $E = {}^{t_{sum}}\!/_{m \times CT}$, where *m* is the number of workstations and *CT* the cycle time. The DEA was run with the following values of the control parameters: population size=10 individuals, crossover rate=0.4, mutation scale factor F=0.3, maximum number of generations=100×*n* (*n*=number of tasks).

Table 1 displays the results obtained by the DEA on the above data sets. The table provides the following information: the problem name, the number of tasks (n), the cycle time (CT), the optimal number of workstation (m*), the generated by the DEA number of workstations (mDE) and the actual CPU time spent in sec. All the experiments were carried out on a Pentium II-MMX PC running at 333MHz.

Table 1 (continue)

Problem	n	CT	m*	mDE	tcpu
Heskiaoff	28	138	8	8	7.9
		205	5	5	0.0
		216	5	5	0.0
		256	4	4	0.0
		324	4	4	0.0
		342	3	3	2.9
Sawyer	30	25	14	14	1.1
		27	13	13	1.1
		30	12	12	3.3
		36	10	10	1.1
		41	8	8	5.0
		54	7	7	0.0
		75	5	5	1.1
Kilbridge	45	57	10	10	0.0
		79	7	7	0.0
		92	6	6	0.0
		110	6	6	0.0
		138	4	4	0.6
		184	3	3	1.1
Tonge	70	176	21	22	6.6
		364	10	10	8.2
		410	9	9	2.7
		468	8	8	0.0
		527	7	7	1.1
Arcus1	83	5048	16	16	1.1
		5853	14	14	0.5
		6842	12	12	0.0
		7571	11	11	0.0
		8412	10	10	0.0
		8898	9	9	1.7
		10816	8	8	0.0
Arcus2	111	5755	27	28	6.0
		8847	18	19	6.4
		10027	16	16	0.8
		10743	15	15	11.3
		11378	14	14	7.3
		17067	9	9	39.5

A summary of these results is displayed in Table 2. In particular, the table illustrates: (a) the number of the instances for which the optimum solution was found (**nopt**), (b) the average relative deviation from the existing optimum in percentage (**av.rel%**), (c) the maximum relative deviation from the existing optimum in percentage (**max.rel%**), and (d) the average CPU-time in seconds (**av.cpu**).

Table 2. A summary of the results obtained by DEA

nopt	av.rel%	max.rel%	av.cpu
61	0.2	5.6	2.0

As one can see from Table 2 the DEA was able to find the optimum solution in 61 out the total 64 test instances which is a result as good as the one produce by the well known branch and bound method EUREKA (Hoffmann 1992). DEA is quite fast, it needed 2 sec of processing time in average to reach the optimum. It is underlined that the plethora of the exact and heuristic algorithms used to solve the SALBP (including the famous FABLE, EURECA, SALOME-1) are in fact problem-depended and have no (or at least a limited) other known applicability. From the other side, the proposed DEA can be used with only minor changes to solve many other sequencing and scheduling problems that follows the permutation property.

5. CONCLUSIONS

The use of a differential evolution (DE) algorithm to solve the SALBP-1 was investigated in this paper. Computational experiments were carried out over public benchmark problems. The results obtained are very promising reporting a high quality performance for the DE.

ACKNOWLEDGMENTS

This work is integrated to I-PROMS NOE.

REFERENCES

Anderson E.J. and Ferris M.C. (1994). Genetic algorithms for combinatorial optimization: the assembly line balancing problem. ORSA Journal on Computing, **6**, 161-173.

Baybars I. (1986), A survey of exact algorithms for the simple assembly line balancing problem, Management Science, **32**, 909-932.

Bean J. (1994), Genetics and random keys for sequencing and optimization, ORSA Journal on Computing, **6** (2), 154-160.

Chiang W.-C. (1998), The application of a tabu search metaheuristic to the assembly line balancing problem, Annals of Operations Research, **77**, 209-227.

Erel E. and Sarin S. (1998), A survey of the assembly line balancing procedures, Production Planning & Control, **9 (**5), 414-434.

Hoffmann T.R. (1992), EUREKA: A hybrid system for assembly line balancing, Management Science, **38**, 39-47. Johnson R.V. (1988),

Optimally balancing large assembly lines with "FABLE", Management Science, **34**, 240-253.
Kim Y.K., Kim Y.JU. and Kim Y..(1996), Genetic algorithms for assembly line balancing with various objectives, Computers and Industrial Engineering, **30** (3), 397-409.
Scholl A. (1999), Balancing and sequencing of assembly lines. Physica-Verlag publ., Germany.
Scholl A. and Klein R. (1997), SALOME: A biderectional branch and bound procedure for assembly line balancing, INFORMS Journal on Computing, **9**, 319-334.
Scholl A. and Voβ S. (1996), Simple assembly line balancing – Heuristic approaches, Journal of Heuristics, **2**, 217-244.
Storn R. and Price K. (1997). Differential Evolution –A simple and efficient heuristic for global optimization over continues spaces, Journal Global Optimization, **11**, 241-354.
Suresh G. and Sahu S. (1994), Stochastic assembly line balancing using simulated annealing, Int. Journal of Production Research, **32**, 1801-1810.
Talbot F.B., Patterson J.H., Gehrlein W.V. (1986), A comparative evaluation of heuristic line balancing techniques, Management Science, **32**, 430-454.