An Adaptive Genetic Algorithm for Multi Objective Flexible Manufacturing Systems

Abdulnasser Younes and Hamada Ghenniwa*and Shawki Areibi†
School of Engineering
University of Guelph
Guelph, Ontario, Canada N1G 2W1

Abstract

A large number of combinatorial problems are associated with manufacturing optimization[1]. The use of intelligent techniques in the manufacturing field has therefore been growing in the last decade. This paper presents a Genetic Algorithm solution for the manufacturing systems in general and flexible manufacturing in particular. In our implementation we have combined a Pareto-based approach with an adaptive weighted sum technique for tackling the multi-objective flexible manufacturing systems problem. Experimental results demonstrate that this approach is very effective for handling such complex systems.

1 INTRODUCTION

In recent years, distributed and open architectures are considered appropriate design approaches for systems in many environments, particularly, for flexible manufacturing systems (FMS). These systems consist of multiple-heterogeneous machines (machines robots and/or computers), where each machine is capable of performing a specific set of operations that may overlap with those of the other machines. The ultimate goal for designing these systems might be to maximize the FMS throughput[2]. However, several problems such as part type partitioning, assignment and sequencing must be solved before this goal can be achieved. The main focus of this paper is on modeling and solving the assignment problem of FMS. In FMS literature, the assignment problem, is sometimes dealt with as a flow management problem[3], other attempts are based on reducing the problem to a mathematical programming problem. Most combinatorial optimization problems in manufacturing optimization systems are NP-hard, i.e, there is no polynomial time algorithm that can possibly solve them. Heuristic methods are normally employed for the solution of these problems. A growing number of researchers have adopted the use of meta-heuristic techniques such as Simulated Annealing and Tabu Search for difficult combinatorial problems. Evolutionary computation methods are meta-heuristics that are able to search regions of the solution's space without being trapped in local optima. Multi-objective evolutionary algorithms have been recognized to be particularly suitable for solving flexible manufacturing systems because of their ability to exploit and explore multiple solutions in parallel, and the ability to find an entire set of Pareto-optimal solutions in a single run. This paper focuses on developing a model for a class of FMS at the part machine level and solving the problem using a Genetic Algorithm technique. The rest of the paper is organized as follows. Section 2 introduces the main concepts of flexible manufacturing systems. Section 3 presents a Genetic Algorithm solution for the FMS system. Results are introduced in Section 4. The paper concludes with some comments on how the Genetic Algorithm performed and possible future work.

2 BACKGROUND

Many models for the problem of assigning parts of different types to machines have been developed in the literature [4, 5], [6]. These models assume either parallel machines that are identical in capabilities but may differ in speed [2] or machines that are specialized [7, 6]. The goodness of an assignment is measured in terms of minimizing part transfer and balancing the work-load of the machines. These two objectives are lexicographically ordered, such that the

*The second author is in the Electrical Engineering Department at the University of Western Ontario.

†The work of the third author has been partially supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) operating grant (OGP 43417)

primary objective is minimizing part transfer and the secondary objective is balancing the work-load. The aim is to facilitate the creation of machine cells with minimum part transfer while maximizing the utilization of machines. These objectives are conflicting with each other. While minimizing part transfer tends to favor the assignment of the whole of a part to a single machine, balancing work-load tries to make the work-load distribution even among the machines. Thus satisfying both objectives seems a hard problem to solve [8].

2.1 FMS EXAMPLE

Consider a flexible manufacturing system consisting of three machines, M_1, M_2 and M_3, each of which is characterized by a set of operations, respectively, denoted by $\{O_1, O_2, O_3, O_4\}$, $\{O_3, O_4, O_5\}$, and $\{O_4, O_5\}$, where O_i denotes operation i. This system is needed to process two Part types P_1 and P_2. Part P_1 requires four operations denoted by $\{O_1, O_2, O_3, O_5\}$, while P_2 requires three operations denoted by $\{O_2, O_3, O_5\}$. There are several processing choices for this setting, such as:

- First Choice, for part P_1: $(O_1 \mapsto M_1, O_2 \mapsto M_1, O_3 \mapsto M_1, O_5 \mapsto M_2)$ i.e, assign machine M_1 to process O_1, O_2 and O_3, and assign M_2 to process O_5. For part P_2: $(O_2 \mapsto M_1, O_3 \mapsto M_1, O_5 \mapsto M_2)$ i.e assign machine M_1 to process O_2,O_3 and have M_2 process O_5.
- Second Choice, for part P_1: $(O_1 \mapsto M_1, O_2 \mapsto M_1, O_3 \mapsto M_2, O_5 \mapsto M_3)$ i.e, assign machine M_1 to process O_1, O_2, assign M_2 to process O_3, assign M_3 to process O_5. For part P_2: $(O_2 \mapsto M_1, O_3 \mapsto M_2, O_5 \mapsto M_3)$, i.e, have machine M_1 process O_2, M_2 process O_3 and have M_3 process O_5.

Looking back at the suggested choices we can notice that the first solution is biased towards part transfer objective function (i.e minimize transfer of parts between machines) where the total number of machines involved for P_1 were two machines and the same applies for $P2$. The number of operations performed by the machines are five, two and zero respectively i.e there is no balance in the operations performed by different machines. On the other hand the second choice indicates that the balance criteria is met (M_1 performs 3 operations, M_2 performs 2 operations and M_3 performs 2 operations), but the total number of part transfer has increased to four machines. What needs to be accomplished is minimizing part transfer as a primary objective and balancing the workload as a secondary objective instead of optimizing a single objective at a time.

2.2 MATHEMATICAL FORMULATION

The assignment problem can be formulated in terms of minimizing the part transfer and balancing the machine workload. In order to formulate the problem, the following notations are introduced:
i,l : machine index (i,l = 1,2,3,...,n_m)
j : part index (j = 1,2,3,...,n_p)
$\hat{k}_j$: is processing choice for part j (j = 1,2,3,....,n_p)
k_j : is the number of processing choices of P_j
$n_{ji\hat{k}_j}$: is the number of necessary operations required by Pj on M_i in processing choice $\hat{k}_j$, $\quad 1 \leq \hat{k}_j \leq k_j$
$t_{ji\hat{k}_j}$: is the work-load of machine M_i to process part P_j in processing choice $\hat{k}_j$.

$$x_{ji\hat{k}_j} = \begin{cases} 1 & \text{if } P_j \text{ requires } M_i \text{ in processing choice } \hat{k}_j \\ 0 & \text{otherwise} \end{cases}$$

$$q_{j\hat{k}_j} = \begin{cases} 1 & \text{if processing choice } \hat{k}_j \text{ is selected for part j} \\ 0 & \text{otherwise} \end{cases}$$

Using this notation, then the objective functions are:

1. Minimization of part transfer (by minimizing the number of machines required to process the part):

$$F_1 = min_{\hat{k}_j} \sum_{i=1}^{n_m} q_{j\hat{k}_j} x_{ji\hat{k}_j}, \forall_j \qquad (1)$$

2. Minimization of the number of necessary operations required from each machine over the possible processing choices:

$$F_2 = min_{\hat{k}_j} \sum_{i=1}^{n_m} q_{j\hat{k}_j} x_{ji\hat{k}_j} n_{ji\hat{k}_j}, \forall_j \qquad (2)$$

3. Load Balancing by minimizing the cardinality distance between the workload of any pair of machines:

$$F_3 = min_{\hat{k}_j} \sum_{j=1}^{n_p} q_{j\hat{k}_j} \sum_{i=1}^{n_m} \sum_{l=(i+1)}^{n_m} |x_{ji\hat{k}_j} t_{ji\hat{k}_j} - x_{jl\hat{k}_j} t_{jl\hat{k}_j}| \qquad (3)$$

The overall multi-objective mathematical model of FMS can be formulated as follows:

$$solve for \quad F_1, F_2, F_3$$

s.t.

$$\sum_{\hat{k}_j=1}^{k_j} q_{j\hat{k}_j} = 1$$

$$x_{ji\hat{k}_j} \in \{0,1\};\ q_{j\hat{k}_j} \in \{0,1\};\ n_{ji\hat{k}_j} \geq 1;\ t_{ji\hat{k}_j} \geq 0$$

However utilizing equations 1, 2 and 3 directly as a mathematical programming formulation is too complex. Therefore, for this class of problems, it is more practical to seek a heuristic solution rather than insisting on the optimal. The following section investigates finding such a solution using a Genetic Algorithm heuristic solution.

3 A GA ALGORITHM FOR FMS

Genetic Algorithms **(GA's)** are a class of optimization algorithms that seek improved performance by sampling areas of the parameter space that have a high probability for leading to good solutions [9]. The inherent characteristic of Genetic Algorithms demonstrates why Genetic search may be well suited to multiple-objective optimization problems. The basic feature of Genetic Algorithms is multiple directional and global search through maintaining a population of potential solutions from generation to generation. The population-to-population approach is useful when exploring Pareto solutions. It is important to refine several components such as the encoding method recombination method, fitness assignment, and constraint handling to obtain effective implementations to the given problem. Therefore, when considering how to adapt Genetic Algorithms to multiple-objective optimization problems we have to determine the fitness value of individuals according to multiple objectives. In our implementation we have combined a Pareto-based approach with an adaptive weighted sum technique for tackling the multi-objective flexible manufacturing systems problem.

3.1 GENETIC ALGORITHM MODULES

There are essentially four basic components necessary for the successful implementation of a Genetic Algorithm. At the outset, there must be a code or scheme that allows for a bit string representation of possible solutions to the problem. Next, a suitable function must be devised that allows for a ranking or fitness assessment of any solution. The third component, contains transformation functions that create new individuals from existing solutions in a population. Finally, techniques for selecting parents for mating, and deletion methods to create new generations are required. We will restrict our discussion on the first three modules for flexible manufacturing systems.

As seen in Figure 1, the chromosome representation is defined as a series of operations for all parts involved. Each gene in the chromosome represents the machine type that can possibly process a specific operation. The assignment of machine types to operations is made by randomly generating random numbers within the range of the total number of machines available. In Figure 1, we have a system consisting of 3 parts. Part P_1 requires 3 operations $\{O_2,O_4,O_6\}$ which can be processed by M_2, M_3 and M_1 respectively. The second part requires four operations $\{O_1,O_2,O_5,O_6\}$ which are assigned to Machines M_1 and M_2 and part P_3 requires two operations O_4 and O_6 which are handled by machines M_3 and M_1 respectively. The advantages of this representation scheme are the simplicity and the capability of applying standard operators. However an offspring may not be feasible and thus some special repair heuristics are used to modify the chromosomes to become feasible. Genetic Algorithms

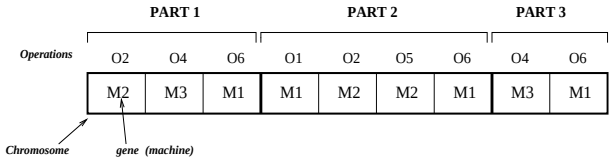

Figure 1: Chromosome Representation

work by assigning a value to each string in the population according to a problem-specific *fitness* function. For the flexible manufacturing problem, the evaluation function measures the goodness of any solution in terms of minimizing part transfer and balancing the workload of the machines.

For any part P_j , **part transfer** is $(\gamma_j - 1)$ with γ_j being the number of different machines assigned to part P_j. Thus, a normalized part transfer function for all the parts can be given as:

$$f_{trans} = \frac{\sum_{j=1}^{parts}(\gamma_j - 1)}{\Gamma}$$

where Γ is an upper bound for the total number of part transfer which is calculated as follows:
$\Gamma = \sum_{j=1}^{parts}(\phi_j - 1)$ and ϕ_j is the number of operations for part P_j.

Machine imbalance can be measured in terms of the deviation, δ_i, of the number of operations performed by machine M_i from the average. Thus a normalized measure of machine imbalance can be given by

$$f_{baln} = (\frac{\sqrt{SSE}}{\sum_{j=1}^{parts}\phi_j}) \times n_m$$

where, n_m is the total number of machines, SSE is the sum of the squared deviations, given by $SSE = \sum_{i=1}^{machines}\delta_i^2$. The fitness assignment strategy of our implementation uses a weighted based fitness function. For an individual χ_u, the score or fitness can be given

by:

$$Score(\chi_u) = (w_{tran} \times (1.0 - f_{trans})) + \frac{w_{baln}}{f_{baln}}$$

where w_{tran} and w_{baln} are the weights assigned to transfer and balance objective functions respectively. Operators in the reproduction module, mimic the biological evolution process, by using unary (mutation type) and higher order (crossover type) transformation to create new individuals. *Mutation* is simply the introduction of a random element, that creates new individuals by a small change in a single individual. When mutation is applied to a bit string, it sweeps down the list of bits, replacing each by a randomly selected bit, if a probability test is passed. On the other hand, *crossover* recombines the genetic material in two parent chromosomes to make two children. Crossover begins by randomly choosing a cut point K where $1 \leq K \leq L$, and L is the string length. The parent strings are both bisected so that the leftmost partition contains K string elements, and the rightmost partition contains $L - K$ elements. The child string is formed by copying the rightmost partition from parent P_T^1 and then the leftmost partition from parent P_T^2. Figure 2 shows an example of applying three types of

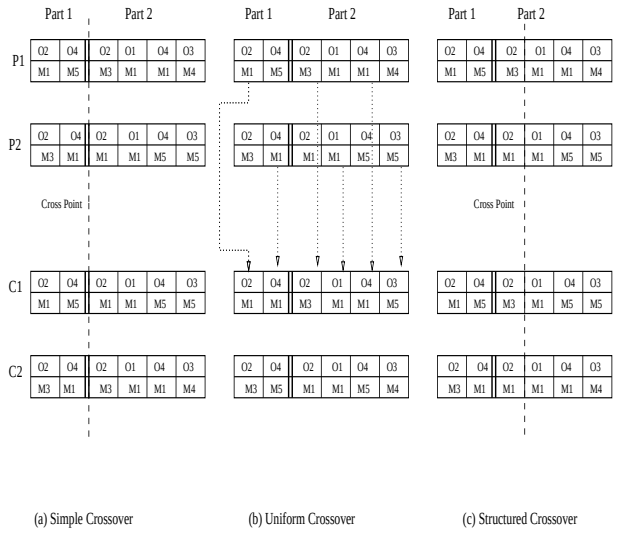

Figure 2: Crossover Operator

crossover operators implemented in this paper. In the simple crossover strategy the cut point(s) are set at the part delimiter such that a complete part is transfered from a parent to a child. In uniform crossover every other gene is received from a different parent as seen in Figure 2b. In the third crossover strategy, the operator begins by randomly choosing a cut point in the string (or multi-points in the chromosome) as described above (see Figure 2c). If all initial solutions are feasible then these crossover strategies lead to complete feasible solutions. Due to the mutation operator, some chromosome may become infeasible (i.e an operation assigned to a machine that cannot handle the operation). In this situation a simple heuristic technique is used to repair the chromosome by assigning the correct machine to the designated operation.

3.2 GA IMPLEMENTATION

In our weighted-sum approach we assign weights to each objective function and combine the weighted objectives into a single objective function as explained in Section 3.1. To fully utilize the power of the Genetic Algorithm we use several approaches: (i) fixed-weight approach, (ii) random-weight approach, and (iii) adaptive weight approach. In the fixed-weight

```
GENETIC ALGORITHM
1. Encode Solution Space
2.(a) set pop_size, max_gen, gen=0;
  (b) set cross_rate, mutate_rate;
3. Initialize Population.
4. For phase ≤ max_phases
    Adapt Weights of Obj Functions
    While max_gen ≥ gen
      Evaluate Fitness
      For (i=1 to pop_size)
        Select (mate1,mate2)
        if (rnd(0,1) ≤ cross_rate)
          child = Crossover(mate1,mate2);
        if (rnd(0,1) ≤ mutate_rate)
          child = Mutation();
        Repair child if necessary
      End For
      Add offsprings to New Generation.
      Save Best δ Solutions
      gen = gen + 1
    End While
5. Return best chromosomes.
```

Figure 3: A Genetic Algorithm for FMS

approach, weights are not changed during an entire evolutionary process. Weights are determined apriori to give selective pressure towards the part transfer objective function. In the random based implementation weights are randomly reset at each step in the selection procedure to give an even chance to all possible combinations. Finally, in the adaptive weight approach, weights are adjusted adaptively based on the current generation to obtain search pressure toward part transfer while balancing the work-load. Figure 3

shows a Genetic Algorithm implementation for flexible manufacturing systems. The algorithm begins with an encoding and initialization phase during which each string in the population is assigned a uniformly distributed random point in the solution space. In the first phase the system assigns a large weight to the first objective function (i.e w_{trans}) and the values of the objective functions f_{trans} and f_{baln} are calculated and set to f_{trans}^{phase1} and f_{baln}^{phase1} respectively. In the next few phases the weights of the objective functions are adjusted adaptively such that the value of f_{trans} is within a $\delta\%$ tolerance of f_{trans}^{phase1}. Each iteration of the genetic algorithm begins by evaluating the fitness of the current generation of strings. A new generation of offspring is created by applying crossover and mutation to pairs of parents who have been selected based on their fitness (the function rnd(0,1) basically returns a random number between 0 and 1). The algorithm terminates after some fixed number of iterations.

4 RESULTS

The Genetic Algorithm code was developed on a SUN Sparc Ultra II workstation running Solaris 8. The code was written in C and compiled using GNU g++ version 2.95.2. Table 1 shows several benchmarks that have been used for evaluating the performance of the Genetic Algorithm. These benchmarks were randomly generated with different number of machines, parts and operations. In Table 1 the second column gives the total number of machines involved and the possible operations performed by each machine. The third column specifies the number of parts that need to be manufactured and the operation required by each part. The rest of the table gives the maximum operations to be performed and the average operations performed by each machine. Figure 4 presents the convergence rate of the Genetic Algorithm for the two objective functions, namely the part transfer and balance

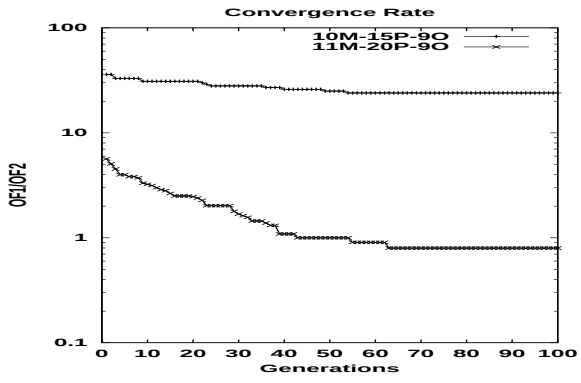

Figure 4: GA Convergence

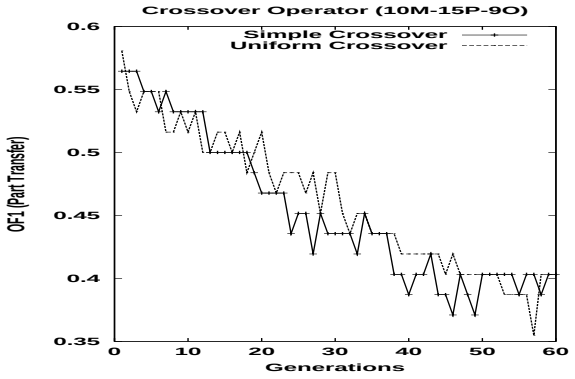

Figure 5: Performance of different Crossover methods

objectives. The figure clearly indicate that after 60 generations there is no improvement in solution quality. Figure 5 presents the performance of two crossover techniques proposed for this paper. The graph clearly indicates that uniform crossover performs better than simple crossover For small benchmarks the quality of solutions obtained from the three crossover operators are quite similar. As the benchmarks increase in size the performance of simple crossover and structured crossover deteriorates. Figure 6 shows the results obtained as a function of the mutation rate. The graphs clearly show that as we increase the mutation rate the two objective functions deteriorate. Mutation rates in the range of 1-5% give the best results. This is obvious since increasing the mutation rate beyond 5% leads to random walks of the solution space and results in unproductive wandering. Figure 7 plots the problem in the criterion space. It is evident from the figure that the search takes place in multiple directions versus a fixed direction as would be the case in a fixed weight approach. Table 2 shows the results obtained when running the Genetic Algorithm by optimizing each objective function separately. The Table is organized as

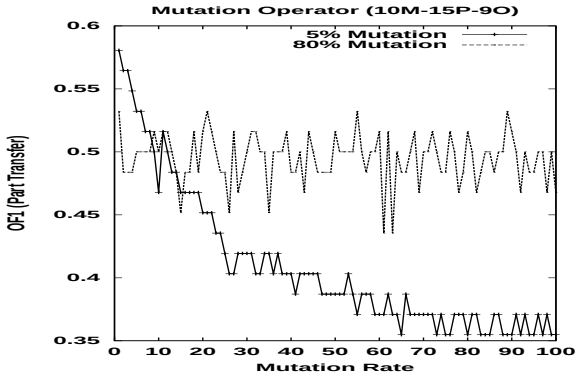

Figure 6: Affect of Mutation Rates

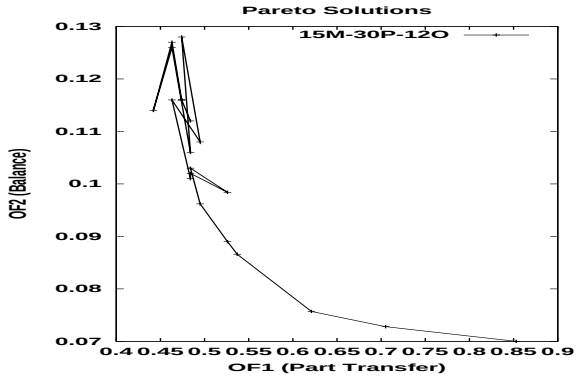

Figure 7: Objectives in Criterion Space

follows: The first column (Prob) indicates the name of the benchmark. The second column (ObjT) specifies the type of objective function being optimized where **PT** stands for part transfer function and **BAL** stands for machine work-load balance. The third and fourth columns specify the objective function values for part transfer and machine work-load balance respectively. The fifth column indicates the total number of machines involved in processing the parts (i.e part transfers involved). Finally, the sixth and seventh columns specify the number of machines involved in processing a specific part and the total number of operations performed by each machine respectively.

We would notice from the table that whenever we optimize the first objective function (i.e part transfer) the less the number of machines involved (i.e sixth column) and the excessive imbalance in the operations performed by the machines. Table 3 shows the results obtained when running the Genetic Algorithm by optimizing both objective functions together. It is clear from the results that both techniques are very competitive. As the size of the benchmarks increase the performance of the adaptive weight technique surpasses that for the fixed weight approach. For example in the 10 machine example the number of part transfer is reduced by two and in the last benchmark (15M30P12O) the systems achieves a reduction in part transfer and a better balance when using the adaptive approach over that based on fixed weights. We should notice that the largest benchmark used in this paper involves fifteen machines, thirty parts and twelve operations. Currently we are seeking real life benchmarks that involve a large number of parts and operations to quantify the computation ability of the Genetic Algorithm on such large benchmarks. We anticipate that the computational complexity will depend entirely on the population and generation size used to solve the problem. Running a Genetic Algorithm entails setting a number of parameter values. As the benchmarks increase in size we have to make sure that our algorithm has the capability of adaptively tuning the parameters to achieve robustness and good performance.

5 CONCLUSIONS

This paper presented a Genetic Algorithm solution for flexible manufacturing systems. The use of evolutionary computation methods for manufacturing optimization is expanding. The amount of work indicates that evolutionary computation methods have established themselves as a useful optimization technique in the manufacturing field, despite the fact that their theoretical foundation are still debated. Results obtained indicate that our Genetic Algorithm implementation achieves excellent results with respect to part transfer and balancing the work among the machines. Future work should compare this Genetic Algorithm implementation with other heuristic search approaches (possibly a memtic algorithm that combines a Genetic Algorithm with local search techniques) and the feasibility of integrating Genetic Algorithms with Multi Agent Systems. We also seek to include sequencing constraints and tools costs in the objective function in our future implementation.

References

[1] C. Dimopoulos and A. Zalzala, "Recent Developments in Evolutionary Computation for Manufacturing Optimization: Problems, Solutions, and Comparisons", *IEEE Transactions on Evolutionary Computation*, vol. 4, n. 2, pp. 93–113, 2000.

[2] J. Blazewicz, K. Ecker, E. Pesch and G. Schmidt, *Scheduling Computer and Manufacturing Processes, 2nd Ed*, Springer, Berlin, New York, 2001.

[3] J. Chen and S. Ho, "Multi-Objective Evolutionary Optimization of Flexible Manufacturing Systems", in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1260–1268, San Francisco, California, 2001, Morgan Koffman.

[4] K. Baker, *Introduction to Sequencing and Scheduling*, Kluwer Academic Publishers, Boston, 1974.

[5] P. Brucker, *Scheduling Algorithms*, Springer, Berlin, New York, 2000.

[6] S. French, *Sequencing and Scheduling*, ellis Horwood Limited, England, 1982.

[7] E. Coffman, *Computers and Job Scheduling*, John Wiley, New York, 1976.

[8] H. Ghenniwa and M. Kamel, "Minimizing Part Transfer and Balancing the Workload in Assigning Parts to Machines Having Overlapping Capabilities", *Journal of Flexible Manufacturing Systems*, 1998.

[9] V. Venkatasubramanian and I.P. Androulakis, "A Genetic Algorithm Framework for Process Design and Optimization", *Computers Chemical Engineering*, vol. 15, n. 4, pp. 217–228, 1991.

Prob	Mach	Part	Oper	Ops/Mach	
				MAX	$\overline{x}$
3M1P5O	3 (1,2,3,4);(3,4,5);(4,5)	1 (1,2,3,5)	5	4	2.3
3M2P5O	3 (1,2,3,4);(3,4,5);(4,5)	2 (1,2,3,5);(2,3,5)	5	4	3
5M2P7O	5 (1,2,7);(1,2);(2,3) (2,4,5),(1,4,6)	2 (7);(1,2,3,4,5,6)	7	3	2.1
5M5P6O	5 (2,3,4,5);(2,3,4,6); (1,2,3,4);(1,2,4,6);(1,2,5)	5 (1,2,3,4,5);(1,2,3,4,6) (1,2,5);(2,3,4);(2,3,4,6)	6	3	3.2
5M10P8O	5 (1,2,3);(4,5,6);(2,3,7) (5,6);(7,8)	10 (1);(4);(8);(2,3,4,5) (2,4,6,8);(4,6,7,8) (2,3,5,6,7);(1,2,4,5,7) (1,4,8);(3,5,6)	8	3	3.2
6M6P10O	6 (1,2,3);(4,5);(1,2,3,4,5) (6,7,8);(9,10);(6,7,8,9,10)	6 (1,2,3,4,5);(1,2,3) (4,5);(6,7,8,9,10) (6,7,8);(9,10)	10	5	3.5
7M7P7O	7 (7);(6);(5);(4);(3);(2);(1)	7 (1);(2);(3);(4);(5);(6);(7)	7	1	1
10M15P9O	10 (1,2,3);(1,4,6);(5) (1,2,3,4,5);(1,7,8,9) (1,2,7,8,9);(1,2,3,4,7,8,9) (5,7,8);(1,3,5);(2,4,6)	15 (1,2,3);(1,2,4);(1,2,5) (1,3,6,7);(1,4,7,8) (2,3,5,8,9);(4,5,6,7,9) (1,2,4,5,9);(1,3,5,7,9) (2,4,6,7,8,9);(1,2,3,4,5,6) (2,4,5,6,7,8);(1,2,3,4,5,6,7) (2,3,4,5,6,7,8) (1,2,3,4,5,6,7,8)	9	7	4.2
11M20P9O	11 (1,2);(3,4);(5,6);(7,8) (1,2,3);(3,4,5);(5,6,7) (7,8,9);(1,2,3,4) (5,6,7,8);(1,2,3,4,9)	20 (1,2,3);(1,2,4);(1,2,5) (3,6,7);(4,7,8);(5,8,9) (6,7,9);(4,5,9);(6,8,9) (5,7,9);(7,8,9);(3,4,5) (4,5,6);(6,7,8);(2,4,6) (1,3,5);(1,3,7);(1,3,9) (2,5,7,9);(1,3,5,7)	9	5	2.9
15M30P12O	15 (1,2);(3,4);(5,6);(7,8) (1,2,3);(3,4,5);(5,6,7) (7,8,9);(1,2,3,4) (5,6,7,8);(1,2,3,4,9) (9,10);(8,11,12);(8,9,10,11) (7,8,9,10,11,12)	30 (1,2,3);(1,2,4);(1,2,5) (3,6,7);(4,7,8);(5,8,9) (6,7,9);(4,5,9);(6,8,9) (5,7,9);(7,8,9);(3,4,5) (4,5,6);(6,7,8);(2,4,6) (1,3,5);(1,3,7);(1,3,9) (2,5,7,9);(1,3,5,7) (1,2,9,10);(2,4,8,11) (3,5,9,11);(4,6,10,12) (5,7,8,9,10);(2,3,8,9,10,11) (2,3,8,10,11,12);(1,2,3,8,10,11,12) (2,4,6,8,10,11,12);(1,2,3,4,5,6,7,8) (5,6,7,8,9,10,11,12)	12	6	3.2

Table 1: Benchmarks used as test cases

Prob	ObjT	Obj1	Obj2	T-M	P-P-M	O-P-M
3M1P5O	PT	0.3333	1.00	2	2	2,2,0
	BAL	0.6666	0.5773	3	3	2,1,1
3M2P5O	PT	0.4000	1.2909	4	2,2	4,2,1
	BAL	0.8000	0.5773	6	3,3	3,2,2
5M2P7O	PT	0.4000	1.0954	4	1,3	1,0,1,2,3
	BAL	0.6000	0.6324	5	1,4	1,1,1,2,2
5M5P6O	PT	0.1333	1.4142	7	2,2,1,1,1	2,4,6,3,5
	BAL	0.7333	0.0001	16	3,5,2,3,3	4,4,4,4,4
5M10P8O	PT	0.4761	2.4083	20	1,1,1,2,3,2,2,3,3,2	4,10,8,4,5
	BAL	0.7142	0.7745	25	1,1,1,3,4,2,3,5,3,2	6,7,5,7,6
6M6P10O	PT	0.000	3.9157	6	1,1,1,1,1,1	0,0,10,3,0,7
	BAL	0.5714	0.5773	14	3,2,2,3,2,2	3,3,4,4,3,3
7M7P7O	PT	-	2.4494	7	1,1,1,1,1,1,1	7,0,0,0,0,0,0
	BAL	-	0.0	7	1,1,1,1,1,1,1	1,1,1,1,1,1,1
10M15P9O	PT	0.3387	11.157	36	1,1,1,3,1,2,3,3,2,3, 3,3,3,3,4	0,3,0,35,0,11,21,0,1,6
	BAL	0.8225	0.9486	66	3,2,3,4,4,3,5,4,4,5, 5,5,6,6,7	8,9,7,8,7,8,8,8,7,7,8
11M20P9O	PT	0.3809	6.1864	37	1,1,2,2,2,3,2,2,2, 2,1,2,3,1,2,2,2,1,2,2	0,0,0,0,3,3,9,7,7,13,20
	BAL	0.9047	0.7977	58	2,2,2,3,3,3,3,3,2, 3,3,3,3,3,3,3,3,3,4,4	5,6,5,5,6,6,6,6,5,6,6
15M30P12O	PT	0.5052	6.6282	78	1,2,2,2,2,2,2,3,2,3, 2,1,2,2,2,2,2,2,4,2, 2,3,3,3,2,3,5,6,5,4	1,0,0,2,16,14,16,6,11 12,13,1,1,15,17
	BAL	0.853	1.88	111	3,3,2,2,3,3,3,3,3,3, 3,3,3,3,3,3,2,3,4,4, 4,4,4,4,4,4,,9,6,7,6	6,9,7,6,12,10,10,8,8 8,10,7,6,11,7

Table 2: Comparisons Based on Optimizing a Single Objective Function

Prob	ObjT	Obj1	Obj2	T-M	P-P-M	O-P-M
3M1P5O	FWA	0.333	1.000	2	2	2,2,0
	AWA	0.333	1.000	2	2	2,2,0
3M2P5O	FWA	0.800	0.577	6	3,3	3,2,2
	AWA	0.800	0.577	6	3,3	3,2,2
5M2P7O	FWA	0.400	0.894	4	1,3	1,0,2,2,2
	AWA	0.400	0.894	4	1,3	1,0,2,2,2
5M5P6O	FWA	0.733	0.0	16	3,5,2,3,3	4,4,4,4,4
	AWA	0.733	0.0	16	3,5,2,3,3	4,4,4,4,4
5M10P8O	FWA	0.619	0.774	23	1,1,1,3,4,2,2,4,3,2	6,7,6,7,5
	AWA	0.619	0.774	23	1,1,1,3,4,2,2,4,3,2	6,7,5,7,6
6M6P10O	FWA	0.142	0.577	8	2,1,1,2,1,1	3,4,3,3,4,3
	AWA	0.142	1.290	7	2,1,1,1,1,1	3,2,5,3,2,5
7M7P7O	FWA	-	2.449	7	1,1,1,1,1,1,1	7,0,0,0,0,0,0
	AWA	-	2.449	7	1,1,1,1,1,1,1	7,0,0,0,0,0,0
10M15P9O	FWA	0.370	4.868	38	2,2,2,3,2,2,3,3 2,2,2,3,3,4,3	1,9,2,18,5,10,12,4,9,7
	AWA	0.338	6.640	36	1,2,2,3,2,2,3,2 2,2,3,3,3,3,3	2,6,0,23,5,15,8,4,3,11
11M20P9O	FWA	0.380	3.965	36	1,1,2,2,2,2,2,2,2,2,1 2,2,1,2,2,2,1,3,2	1,1,2,0,5,5,9,10,9,10,10
	AWA	0.380	3.030	36	1,1,2,2,2,2,2,2,2,2,1 1,2,1,2,2,2,1,3,3	1,0,4,3,8,8,7,8,7,7,9
15M30P12O	FWA	0.453	7.04	73	1,1,2,2,2,2,2,2,2,2, 2,1,2,2,2,2,2,3,3,3, 3,4,3,3,3,2,4,3,4,4	2,0,0,1,18,15,18,5 9,6,13,0,5,20,13
	AWA	0.410	6.36	69	1,1,2,2,2,2,2,2,2,2, 1,1,2,2,2,2,2,2,3,3, 2,3,3,3,3,2,5,3,4,3	2,0,1,0,19,17,16,9 7,7,14,5,4,16,8

Table 3: Comparisons Based on Optimizing a Combined Objective Function