Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem

Marco Antonio Cruz-Chávez, Fredy Juárez-Pérez, Erika Yesenia Ávila-Melgar, Alina Martínez-Oropeza

CIICAp, Universidad Autónoma del Estado de Morelos
Avenida Universidad 1001. Col. Chamilpa, C.P. 62209. Cuernavaca, Morelos, México.
{mcruz, juarezfredy,erika,alinam}@uaem.mx

Abstract

In this paper, a solution is presented to the unrelated parallel machines problem that minimizes the total weighted completion time ($\sum kC_i$). Simulated annealing is applied to the problem, which is modeled as a Weighted Bipartite Matching Problem. Experimental results with benchmarks are presented, evaluating the efficiency and efficacy of the algorithm. It is then compared with an exact algorithm that solves the pondered model of Integer Linear Programming. The results demonstrate that Simulated Annealing Algorithm has high performance because for all the evaluated instances, it finds the optimum global solution.

1. Introducción

The general problem of a sequence of jobs n on unrelated parallel machines m without interruptions is classified as an NP-complete problem [2].
Researchers have focused on this type of problem because of the hardness it presents when searching for an optimum solution. It is also of interest due to its wide application in the manufacturing industry.
Many investigations have been carried out in order to find algorithms that solve this problem. In [7], the model UPMP, acronym of unrelated parallel machines problem, has been used to solve transport problems, and has been solved with approximation algorithms demonstrating better performance than that obtained with the formulations of integer linear programming. In [8], a search is conducted to optimize the total weighted tardiness. A hybrid ant colony algorithm is proposed which incorporates a genetic transfer operator in a local search, yielding excellent performance by the algorithm. In [9], an efficient method in order to optimize the total weighted tardiness is presented. An optimization approach with ant colony is applied, using paths of artificial pheromone and information from the heuristic. In [10], the setup times are involved to minimize the total weighted number of tardy jobs. This applies an iterative local search algorithm. In [11], the problem is simplified to an assignment problem with the objective of reducing the setup times required in order to exchange production of one type of product for another. A Branch and Bound algorithm is used. In [12], a shifting bottleneck algorithm is presented. With this algorithm, a bottleneck is identified and a scheduling of bottlenecks is built. It is used to elaborate stoppage scheduling that is used in the assignment of jobs between the intervals identified as bottlenecks. In [13], an application is described which focuses on the analysis of an automated on line support process for a computation center. The on line consulting activity of the center is modeled as a group of unrelated Parallel machines. The users have a process of on line help applications, which is an incoming rain of jobs. The automatic index is a type of Natural Language Processing, which is applied to each job in order to estimate the processing time depending on the respective machine. In [14], a rounding algorithm is developed to find a scheduling in unrelated parallel machines; this algorithm works well with models of linear programming, quadratic programming, and convex programming expanded from a scheduling to optimize the makespan.
The present work gives a solution to a pondered model of Integer Linear Programming (ILP) of unrelated parallel machines presented in [3].
The study of problems of unrelated parallel machines is very important because of its frequency in industry. The efficient and effective solution to theoretical models makes a simpler approach to the study of real models that permit the manufacturing industry to increase their degree of competitiveness.
This work presents a solution to the weighted unrelated parallel machines problem using a simulated annealing algorithm that solves the problem as a weighted bipartite matching problem.

The paper is divided into the following sections: Section 1 presents the introduction to the problem and mentions some investigations carried out by the scientific community. Section 2 describes the model of integer linear programming for the SUPM problem. Section 3 explains the model of the bipartite graph that is used in the simulated annealing algorithm in order to find a solution to the problem. Section 4 describes the simulated annealing algorithm. Section 5 reports the solution methodology applied in simulated annealing. Section 6 presents the test scenario. Section 7 shows the experimental results. Finally, in section 8 are the conclusions.

2. Weighted Integer Linear Programming Model

The problem Rm || ΣC_j, [1], is a model of the NP-complete type. It can be formulated as a problem of Integer Lineal Programming (ILP), with a special structure, which makes the problem solvable in polynomial time using a deterministic algorithm. This ILP model presents the problem as a set of machines that should execute a set of jobs. Each job requires one operation in order to be finished. Each machine can carry out any job. The processing time of each job depends on the machine that executes it.

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} kp_{ij}x_{ikj} \tag{1}$$

Sujeto a:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ikj} \qquad j=1,\ldots,n \tag{2}$$

$$\sum_{j=1}^{n} x_{ikj} \le 1 \qquad i=1,\ldots,m,\ k=1,\ldots,n \tag{3}$$

$$x_{ikj} \in \{0,1\} \qquad i=1,\ldots,m,\ k=1,\ldots,n \tag{4}$$

The ILP problem presented in [3] is formulated as follows: The contribution to the objective function in (1) of each job *j* to be carried out, depends on the machine *i* where the job *j* is carried out, as well as on the position *k* where the job is executed in the machine *i* within an orderly set of the jobs the machine *i* executes. The job *j* contributes kp_{ij} to the value of the objective function. The objective function is the optimization of the total weighted completion time (ΣkC_j). In this model, the constraints in (4) indicate that the variables *x* take only values 0 or 1. If $x_{ikj} = 1$, then the job *j* is assigned to the machine *i* in the *k*-th position. If $x_{ikj} = 0$, it is the opposite. Constraints in (2) assure that each job is assigned to only one position *k*. Constraints in (3) assure that each position *k* for each machine *i* takes at most one job *j*. This model can be mapped to the transport problems [4] if the constraints set in (4) are replaced by a set of constraints of non negative values, that is, $x_{ikj} >= 0$. This is important because with this relaxed model one could obtain bounds with exact algorithms for NP-complete problems, like the model *Rm|| Cj*. In the case of the model with the presented structure, the relaxed formulation gives results of 0 or 1 for the *x* variables. Due to this, it is possible to use an exact algorithm like SIMPLEX in order to give a solution to the model, without needing to relax the restrictions in (4).

3. Bipartite Graph Model

A special case of the transport problem is the WBMP (acronym of Weighted Bipartite Matching Problem). The model $Rm \mid\mid \Sigma kC_j$ for the structure defined in the ILP model of section 2, can also be represented as a WBMP [3]. Figure 1 presents an instance of 3 jobs and 2 machines for the problem $Rm \mid\mid \Sigma kC_j$ by means of a bipartite graph. The graph is composed of n jobs and *mn* positions where each machine can process at most n jobs. The job *j* that is assigned the position *ik*, presents a cost kp_{ij}. The objective is to determine the assignment *ik* for each *j* in the bipartite graph with a minimum cost of total time.

The interpretation occurs based on Figure 1 as follows. For the machine $i = 1$, the job $j = 1$, first place (p_{11}), second place ($2p_{11}$) or in third place ($3p_{11}$). For the same machine, the job $j = 2$ can be processed in first place (p_{12}), second place ($2p_{12}$) or in third place ($3p_{12}$). The job $j = 3$ could be processed in first place (p_{13}), second place ($2p_{13}$) or in third place ($3p_{13}$).

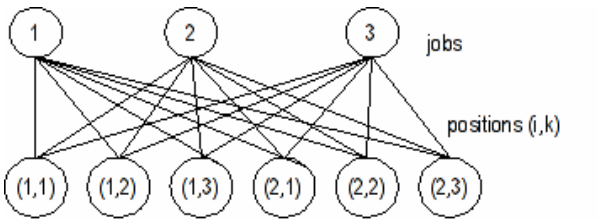

Figure 1. Bipartite Graph for ΣkC_j

For the machine $i = 2$, the job $j = 1$ can be processed in first place (p_{21}), second place ($2p_{21}$) or in third place ($3p_{21}$), and so on.

The WBMP model can be solved with several meta heuristics. The following is the meta heuristic of simulated annealing by which a solution is obtained in this paper.

4. Simulated Annealing Algorithm

The foundations of the Meta heuristic denominated SA (acronym of Simulated Annealing) were introduced by Kirkpatrick [5]. He proposes a probabilistic method of local search in order to escape to the optimum local.
The name of this method comes from the similarity with the metallurgist process of slow annealing, with which a solid of minimum entropy is obtained. This follows a similar outline in order to work with optimization problems.
Figure 2 presents the SA used in this work. The process of SA requires of the knowledge of the following parameters:

s_o : Initial solution of the problem.
T : Parameter of control of SA.
r : Coefficient of control. $0 < r < 1$.
$MCS > 0$: Markov Chain Size.
C: Function of cost.
N: Neighborhood structure.
FROZEN: Stop criterion. Minimum value of T.

```
function SimulatedAnnealing ( )
Begin
  s:= s_m:= s_o, T = T_0
    Repeat
      cycle := 1 to MCS
          generate new solution, s' = N(s)
          if C(s') < C(s) then s := s'
          else
            d := C(s, s') - C(s, s')
            Random ρ (0 ,1)
                          -d/T
            if ρ ≤ e      then s := s'
          end else
          if s_m > s then s_m := s
      end cycle
      T := r T
    Until FROZEN
  return s_m
End
```

Figure 2. Simulated Annealing Algorithm

In Figure 2, at the beginning the SA with a high value of T accepts almost all the transitions between the solutions. Later T and r gradually diminish so that the acceptance of movement is more and more selective. Finally the only movements accepted are those that improve the current solution. A movement is a neighboring s' of s that is obtained from a neighborhood structure. The cost function C is the objective function of the problem to be solved and it allows the evaluation of the solution quality. The acceptance of a movement (new solution) is controlled by the approach of Metropolis that involves the Boltzmman function. In each cycle of T, Metropolis repeats MCS cycles. For the different values of T, which pass through the cooling sequence, an exploration process is carried out for the best solutions. The stop criterion is considered FROZEN, which is the minimum value to which T arrives and is controlled by r.

5. Methodology of solution

In order to solve $Rm \mid\mid \Sigma k C_j$, the following methodology is proposed based on the bipartite graph (Figure 1). It consists of the following steps:

1. Symbolic representation of bipartite graph.
2. Determine a neighborhood structure that obtained feasible solutions.
3. Tuning of parameters of SA. *T, r, MCL, FROZEN.*
4. Apply the Meta heuristic of SA.
5. Mechanism generator of solutions.

5.1. Symbolic Representation

For the symbolic representation of the bipartite graph, a one-dimensional structure of data is used in order to represent the mn position that has the machines ($i=0,..., m-1$) and the k-th position that a j job ($j=0,..., n-1$) could take inside of the i machine. This structure has n jobs with a schedule on the arrangement of mn size and mn - 1 free positions toward which jobs could be randomly exchanged. One solution to the instance presented in Figure 2 is presented in Figure 3 using the symbolic representation.

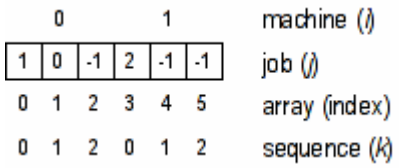

Figure 3. Symbolic Representation of the problem

In Figure 3, the structure represents 2 machines in a series and 3 jobs. The job $j = 1$ is processed at the beginning with $k = 0$, in the first machine $i = 0$, and with a real position of $index = 0$. The job $j = 0$ is processed with k in second place, $k = 1$, in the first machine $i = 0$, and with a real position of $index = 1$ and so on, successively. The positions inside the arrangement that contain $j = -1$ are available positions.

5.2. Mechanism to Generate Solutions in SA

The neighborhood structure is based on the symbolic representation of the positions that the bipartite graph of Figure 1 represents. In this work, a method of local search in a neighborhood is developed, which includes the following steps:

1. Generate initial schedule randomly.
2. Apply direct mapping and indirect mapping.
3. Generate a quick mapping.
4. Change position of a job randomly toward a free position.
5. Randomly exchange of two jobs.

In order to generate an initial solution, a job *j* and a machine *i* are selected. The job *j* is located on the selected machine. Figure 4 presents the algorithm that generates the initial solution. A set of jobs *J* and a set of machines *M* are used to make a schedule. A job *j* and a machine *i* are randomly selected from the sets J and M respectively. A position k_i is assigned in the machine where there is still no job assignment. The job *j* is stored in *Array* (Figure 3) in the k_i position. The position k_i is updated, and *j* is eliminated from the set *J*. The procedure is repeated as long as *J* is not an empty set.

```
function initial_solution( )
Begin
  k = {k1=0,...,km-1=0}
  M = {m1 = 0,...,mm=m-1}
  J= {0,...,n-1}
  Repeat
    j = random(j0)
    i = random(0,m-1)
    Array[n* mi + ki] = j
    ki=ki + 1
    J = J - {j}
  Until j0 ≠ ϕ
End
```

Figure 4. Generation of an initial solution

The indirect mapping finds the elements *ikj* starting from a real position inside the neighborhood structure. The real position is obtained from the *index* of access to the *Array*. The functions are:

$$j = \text{Array}(\ index\ ) \quad (5)$$

$$i = \text{integer\_truncate}(index / n\ ) \quad (6)$$

$$k = \text{operator\_ module}\ (index, n) \quad (7)$$

For example, in Figure 3, using (5), for *index* = 3 job 2 in Array results. With the same *index*, using (6), machine 1 is obtained. Using (7), the *k = 0* position is found.

Direct mapping finds the real position inside the neighborhood structure denoted by index using the elements *ikj*. The function is:

$$index = i * n + k \quad (8)$$

Figure 5 presents the quick mapping which generates a schedule in a structure of jobs of the size *2n*. The *Array* corresponds to a schedule (jobs, machines and positions) smaller in size than the neighborhood structure size *mn*. This structure allows a job to be randomly selected and located inside the neighborhood structure based on the mapping functions previously described.

The efficiency in the search for a new solution also increases. It can be seen in Figure 5 that the order of the neighborhood structure is according to the number of machines and the structure of quick mapping for the number of jobs.

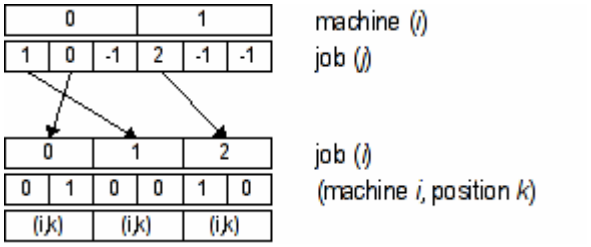

Figure 5. Quick mapping

The movement of a job toward an empty position of *Array* generates a neighbor. The neighborhood structure selects a job randomly using a quick mapping that contains all the schedule jobs, as shown in Figure 6a. Then it carries out the transformation operations described in steps 1 through 5 of the local search, in order to locate the index that targets that job in the neighborhood structure, Figure 6b.

Next a position inside *Array* is selected and an index chosen. This position represents the place toward which the previously selected job moves. Figure 6c shows a free position (-1). In Figure 6d, the movement of the job *j = 0* is shown from position 1 to position 4. Lastly, based on the mapping operations described, the positions of the job *j = 0* are updates, as can be seen in Figure 6e.

The exchange of a pair of jobs in a solution also generates a new neighbor. In Figure 7d the exchange

of jobs between positions 1 and 3 is shown. The selection of the pair of jobs is random.
Lastly, in Figure 7e, it can be seen that it is necessary to update the content of the jobs 0 and 2 to have a correct reference.

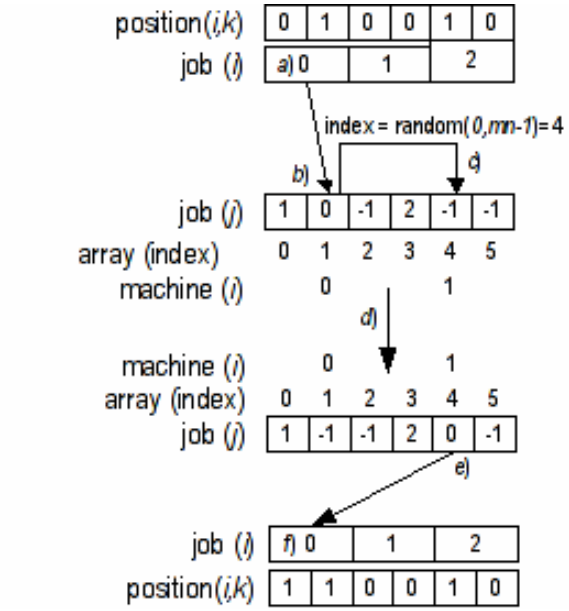

Figure 6. Operation that moves a job from position

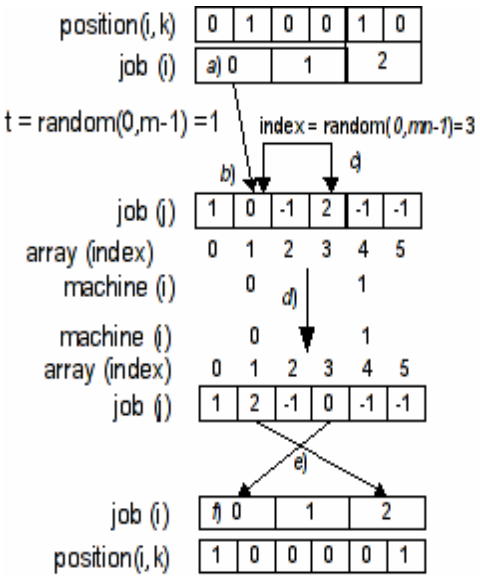

Figure 7. Operation swap

6. Scenario of Tests

Five test problems were used, these benchmarks were proposed in [15]. Each benchmark contains 60 jobs and 4 parallel unrelated machines. Information on setup times is not taken into account, given that the model used in this work does not use these.
In order to prove the efficiency and efficacy of the SA algorithm, an exact algorithm (SIMPLEX) was developed which solves instances with a maximum of 60 jobs with 4 machines. With this algorithm the global optimum solution was obtained. After obtaining the solution, the proposed SA algorithm was used to search for the optimum based on 30 tests for each one of the five benchmarks. The PC used in order to carry out the tests was a PC Intel Celeron E1400 of 2.0 GHz, 2 Gb Memory with S.O. Scientific Linux 4.7 and GCC compiler 3.4.3.
SA, because it is a recurrent local search algorithm, requires a very efficient local search so that it can be efficient. The temporary function of the local search presented in section 5 is $t(n) = 6mn+8m+11$, and the asymptotic complexity is $O(mn)$, where m is the number of machines and n the number of jobs.

7. Experimental results

For all the tests carried out, the values for the parameters entered in the RS algorithm were tuning. $T = 25$, $r = 0.985$, $FROZEN = 0{,}985$ and MCS is six times the neighborhood size. According to [6], the tuning of r was in the range of ($0.8 <= r <= 0.99$).

Benchmark	Opt.	Best	Worst	σ	RE Average
60on4Rp50Rs50_1	62646	62646	62659	3.8201	0.0073
60on4Rp50Rs50_2	62185	62185	62246	13.5190	0.0190
60on4Rp50Rs50_3	62637	62637	62698	14.0426	0.0170
60on4Rp50Rs50_4	62973	62973	63036	11.9474	0.0188
60on4Rp50Rs50_5	63032	63032	63075	12.2476	0.0160

Table 1. Efficacy of the SA Algorithm

In Table 1 it is observed that for all five benchmarks the SA always finds the optimum, showing acceptable standard deviations and Relative Errors under 0.02% for the average of 30 tests for each problem. The best result obtained by the SA algorithm was for the benchmark 60on4Rp50Rs50_1, and the worst result obtained of the 30 tests was 62659 with a standard deviation of 3.8201 and an average relative error of 0.0073%.
Table 2 shows the efficiency results. For the first benchmark, the SA algorithm shows that its best result

of thirty tests reduces the execution time by 96.02% vs. SIMPLEX. For the best results with regard to benchmarks 2, 3, 4 and 5, it is observed that SA reduces the execution time by arriving at the optimum solution 96% more quickly than SIMPLEX.

Benchmark	SIMPLEX	SA	Opt.	Opt.
	CPU sec.	CPU sec.	best	% best
60on4Rp50Rs50_1	78.02	3.13	62646	96.02
60on4Rp50Rs50_2	80.09	2.99	62185	96.26
60on4Rp50Rs50_3	81.05	3.12	62637	96.15
60on4Rp50Rs50_4	78.41	3.08	62973	96.07
60on4Rp50Rs50_5	90.0	2.97	63032	96.70

Table 2. Comparative efficiency of SA vs. SIMPLEX

8. Conclusions

The quick mapping on the algorithm of SA causes that the search of the optimum on the solutions space to be more efficient than the SIMPLEX method in the five analyzed benchmark. When comparing the efficiency in the cases where SA obtains the global optimum, the SA is observed that this is for up of the 96% with regard to SIMPLEX. The efficacy of SA is 100% for the proven benchmark, because it found the optimum solution for all the benchmarks.

8. Referencias

[1] R.L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math.,pages 287–326, 1979.

[2] A.M. Garey and D.S. Johnson, Computers and intractability: A Guide to the theory of NP-completeness, Freeman, San Francisco, 1979.

[3] M. Pinedo, Scheduling Theory, Algorithms and Systems, Prentice Hall, ISBN:0130281387, USA., Aug. p. 586, 2001.

[4] F.S. Hillier, G.J. Lieberman, Introduction to Operation Research, 8th ed.,Mc Graw Hill , ISBN: 0073017795, 2008.

[5] S. Kirkpatrick, C. Gelatt, and M. Vecchi, Optimization by Simulated Annealing. Science Vol. 220(4598): pp. 671-680, 1983.

[6] R, David, R. Sixto and M. Jacinto. Simulación, alfaomega grupo editor., p. 388, 2009.

[7] S. Koranne, Formulating SoC Test Scheduling as a Network Transportation Problem, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, december 2002.

[8] H. Zhou, Z. Li and X. Wu, Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 132-136, August 18 - 21, Jinan, China, 2007.

[9] L. Mönch, Heuristics to Minimize Total Weighted Tardiness of Jobs on Unrelated Parallel Machines, 4th IEEE Conference on Automation Science and Engineering Key Bridge Marriott, Washington DC, USA, pp. 572-577, August 23-26, ISBN: 978-1-4244-2022-3, 2008.

[10] C. Chun.-Lung, An Iterated Local Search for Unrelated Parallel Machines Problem with Unequal Ready Times, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 2044-2047, Qingdao, China September, ISBN: 978-1-4244-2502-0, 2008.

[11] C. Pessan, J.L. Bouquard, E. Néron, An Unrelated Parallel Machines Model For Production Resetting Optimization, International Conference on Service Systems and Service Management IEEE, Vol 2, ISBN: 1-4244-0450-9, pp. 1178-1182, 2007.

[12] C. Chun.-Lung, C. Chuen-Lung, A Heuristic Method for a Flexible Flow Line with Unrelated Parallel Machines Problem, Conference on Robotics, Automation and Mechatronics, pp. 1-4, IEEE, 2006.

[13] K. Anastasova, M. Dror, Intelligent Scheduler for Processing Help Requests on Unrelated Parallel Machines in a Computer Support Administration System, Conference on Systems, Man, and Cybernetics, IEEE, pp. 372-377, ISBN: 0-7803-4778-1, 1998.

[14] V.S. Anil Kumar, M. V. Marathe, Approximation Algorithms for Scheduling on Multiple Machines, Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), pp. 254-263, ISBN: 0-7695-2468-0/05, 2005.

[15] J. P. Arnaut, R. Musa, G. Rabadi, Ant Colony Optimization Algorithm to Parallel Machine Scheduling Problem with Setups, 4th IEEE Conference on Automation Science and Engineering, Washington DC, USA, August 23-26, ISBN: 978-1-4244-2023-0, pp. 578-582, 2008.