Hybridize optimization Algorithms for the Single Machine Total Tardiness Problem

اعداد : م. د. شاكر ناجي

م.م. هيثم غني احمد

م.م. اسماعيل خليل علي

كلية بغداد للعلوم الاقتصادية الجامعة

Abstract

Various optimization heuristics are investigated and applied in a number of areas in the field of single machine scheduling problems. We present efficient heuristic optimization algorithms (Genetic Algorithm and Simulated Annealing) for single machine scheduling problems with and without release times. The increasingly important issue of parallelization is considered with an example implementation being provided in the case of single machine problem is shown. The results show that these algorithms were able to produce high quality optimization, especially for w_jT_j.

Keywords:

Single Machine, Minimizing Tardiness, Heuristic Optimization, Simulated Annealing, Genetic Algorithms

Software engineering Dept. Baghdad College of Economic Science University

Introduction

The study of numerical optimization heuristics and their application of single machine scheduling problem is an important consideration in a world where computer processing power is continually increasing and distributed computing systems are becoming more and more prevalent. Parallel implementations of such heuristics can dramatically improve performance and enhance the overall effectiveness of a wide range of machine scheduling. In this paper a broad introduction to the area of single machine scheduling problem using optimization heuristics is given.

The two algorithms discussed here are the Simulated Annealing (SA) and Genetic Algorithm (GA). Both of these algorithms have different characteristics, which means that one may be more useful than the other for particular applications. These algorithms are used for solving problems where a deterministic approach may be prohibitively complex and time consuming. The types of problems being solved are usually characterized by an *objective* (or *cost*) function, the aim being to minimize this function. They do this by searching only a small part of the solution space and attempting to avoid solutions which do not have the characteristics of a good solution. Defining the characteristics of a good solution is not always easy and is perhaps the most important task when using one of the algorithms presented here.

In (section 2) of this paper a detailed description of each of the algorithms is given. Of particular interest is the genetic algorithm which has been implemented in parallel for a number of different applications. In (section 3) a very brief description of the single machine scheduling problem is given with particular attention being given to the characteristics of this problem which make them relatively easy to perform under strictly limited computation time. Results of computational tests to evaluate the performance of these heuristic algorithms are reported in (section 4).

1. Optimization Heuristics

In this section two optimization techniques, simulated annealing and the genetic algorithm, are introduced.

1.1. Simulated Annealing

Simulated Annealing is a combinatorial optimization technique first introduced in 1983 by Kirkpatrick, Gelatt and Vecchi [7]. They used the technique for deciding the optimal placement of components on an integrated circuit (IC) chip. The number of variables in a problem of this type can be enormous, making determination of an optimal solution impossible. Simulated annealing scans a small area of the solution space in the search for the global minimum.

1.1.1. The Metropolis Algorithm

Simulated annealing utilizes a process known as the Metropolis Algorithm which is based on the equations governing the movement of particles in a gas or liquid between different energy states. Equation (1) below describes the probability of a particle moving between two energy levels, E_1 and E_2,

$$P(E) = \exp^{\frac{-\Delta E}{kT}} \tag{1}$$

where $\Delta E = E_2 - E_1$, k is Boltzmann’s constant and T is the temperature. The Metropolis Algorithm uses in equation (1) to make a decision as to whether or not a transition between different energy levels will be accepted [9]. The Metropolis Algorithm can be summarized by the following equation (2),

$$P(E)=\begin{cases}1 & if \quad \Delta E\geq 0\\ \exp\frac{-\Delta E}{T} & if \quad \Delta E\subset 0\end{cases} \qquad (2)$$

Consider, now, if the evaluation of the cost function for the problem being solved is equivalent to the energy in (2). A transition which decreases the cost (an increase in energy) will always be accepted. However, the Metropolis Algorithm is structured so that a transition to a solution with a higher cost (lower energy) is accepted, with a probability that decreases as the temperature increases. This gives the algorithm the ability to move away from regions of local minima. This is not the case for the so called “iterative improvement” techniques which only move in the direction of decreasing cost. That is, a transition is only accepted if $\Delta E > 0$.

1.1.2. The Simulated Annealing Algorithm

As its name suggest the simulated annealing algorithm simulates the annealing process. Annealing is the process by which a metal, initially at a high temperature, is slowly cooled in such a manner that the molecules in the metal are able to move towards a state of least energy. A metal which is in a state of least energy usually has a crystalline structure i.e. the molecules are very ordered. In the case of a combinatorial problem, the current solution is analogous to the structure of the molecules at any stage of the annealing process.

The simulated annealing algorithm consists of a number of components.

- There first must exist some measure for evaluating the “goodness” of a particular configuration (or solution). This is called the *cost function* or the *objective function*.

- A *cooling schedule* must be determined. A cooling schedule defines the initial temperature, the way in which the temperature decreases at each iteration, and when the annealing should cease. Many complex mathematical models have been devised in consideration of the cooling schedule, however a simple model will usually suffice.

- There also must be a set of rules which state how a particular solution is changed in the search for a better solution.

The following is an outline of the simulated annealing algorithm [7]:
1. Generate an initial solution to the problem (usually random).
2. Calculate the *cost* of the initial solution.
3. Set the initial temperature $T = T^{(0)}$.
4. For temperature, T, do many times:

- Generate a new solution which involves modifying the current solution in some manner.
- Calculate the cost of the modified solution.
- Determine the difference in cost between the current solution and the proposed solution.
- Consult the Metropolis Algorithm to decide if the proposed solution should be accepted.
- If the proposed solution is accepted, the required changes are made to the current solution.

5. If the stopping criterion is satisfied the algorithm ceases with the current solution, otherwise the temperature is decreased and the algorithm returns to Step 4.

1.1.3. The Cooling Schedule

As mentioned above, the cooling schedule has three main purposes.

1. It defines the initial temperature. This temperature is chosen to be high enough so that all proposed transitions are accepted by the Metropolis Algorithm.

2. The cooling schedule also describes how the temperature is reduced. Although there are other methods, two possibilities are presented here.

   (a) An exponential decay in the temperature:

   $$T^{(k+1)} = \alpha \times T^{(k)} \quad , \text{ where } 0 < \alpha < 1 \qquad (3)$$

   Usually $\alpha \approx 0.9$ but can be as high as 0.99.

   (b) Linear decay, here the overall temperature range is divided into a number of intervals, say K.

   $$T^{(k+1)} = \frac{K-k}{K} \times T^{(0)} \quad , \text{ where } k = 1, \ldots\ldots, K \qquad (4)$$

3. Finally, the cooling schedule indicates when the annealing process should stop. This is usually referred to as the *stopping criterion*. In the case where a linear decay is used the algorithm can be run for its K iterations, provided K is not too large. For the case where exponential decay is used, the process usually ceases when the number of accepted transitions at a particular temperature is very small (≈ 0).

1.2. Genetic Algorithms

The genetic algorithm uses concepts such as selection, breeding and mutation which are borrowed from Darwin's theory of evolution. The genetic algorithm was originally designed for solving problems which have a solution which can be represented as a binary vector. As will be seen in this paper, a similar strategy can be used for problems with a solution represented as a vector of integers. Problems solved in this manner, where the solution is not represented in a binary format, are sometimes termed *evolution programs*. However, throughout this paper we refer to our algorithm more generally, as a genetic algorithm.

Unlike simulated annealing which uses a cost function, a genetic algorithm usually has associated with it a *fitness* function. Typically the fitness function evaluates to a figure in the range zero to one (ideally the optimum solution will have the lowest fitness, one if possible). A good fitness function will give an accurate indication of how “close” a feasible solution is to the optimal solution.

A genetic algorithm works on a *pool* of solutions (sometimes referred to as a *gene pool*, where each *gene* is a feasible solution to the problem being solved). The solution pool is iteratively updated in an “evolutionary” manner. From a given pool, a number of pairs of parents are chosen for mating. In the mating process, a pair of parents will produce a pair of children. The children then undergo some mutation process, after which a selection is made determining which genes will survive to the next generation.

The following is an outline of the genetic algorithm [13]:

1. [Start] Generate random population of *n* chromosomes (suitable solutions for the problem).
2. [Fitness] Evaluate the fitness *f(x)* of each chromosome *x* in the population.
3. [New population] Create a new population by repeating following steps until the new population is complete:
    - [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

- [Crossover] With a crossover probability cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
- [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
- [Accepting] Place new offspring in the new population

4. [Replace] Use new generated population for a further run of the algorithm.
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population.
6. [Loop] Go to step 2.

2. Problem Formulation and Complexity

The single machine total weighted tardiness problem [10] can be stated as follows: A set of *n* jobs has to be scheduled on a single machine. The machine can only process one job at a time and the execution of a job cannot be interrupted. Each job *j* becomes available at time zero, and machine idle time and job preemption are prohibited. Requires a predefined processing time p_j , has a positive weight w_j and a due date d_j. A schedule is constructed by sequencing the jobs in a certain order such that the completion time C_j of each job can be computed. If the completion time exceeds a job's due date, the objective function gets increased by a tardiness penalty w_jT_j , where $T_j = \max\{0, C_j - d_j\}$.The optimization goal is to find a processing order which minimizes the value of the sum :

$$\sum_{j=1}^{n} w_jT_j \tag{5}$$

The problem $\sum_{j=1}^{n} w_jT_j$ is known to be strongly *NP*-hard [8]. As far as single machine problems with unequal release times are concerned, only the *NP*-hardness of the total tardiness problem has been shown directly [12].

Several enumerative methods have been presented for $\sum_{j=1}^{n} w_j T_j$, such as Branch & Bound algorithms[11]. Although these exact approaches have been constantly improved, they are not able to solve problem instances with more than 40 jobs. The only existing Branch & Bound algorithm for $|rj| \sum_{j=1}^{n} w_j T_j$ is the one introduced by Akturk and Ozdemir [1], which has been applied to problems up to 20 jobs. This indicates that for large, industrial-sized scheduling problems only heuristic optimization techniques are to be considered.

A review and comparison of such methods in the context of $\sum_{j=1}^{n} w_j T_j$ is provided by Crauwels [6]. Neighborhood-based approaches, in particular Tabu Search, as well as Genetic Algorithms turned out to be effective methods for weighted tardiness optimization. During the past few years, several enhanced variants of these methods have been developed [5][2][4]. Anyway, the applied techniques are very sophisticated and tailored to the specific properties of $\sum_{j=1}^{n} w_j T_j$. On the other hand, literature on heuristic approaches to $|rj| \sum_{j=1}^{n} w_j T_j$ is very sparse. As a consequence, we decided to rely on existing methods for $\sum_{j=1}^{n} w_j T_j$ and to adapt them in order to handle arbitrary release times. We further modified the methods for maximum efficiency at a reasonable level of solution quality.

3. Experimental Results

In this section, we present computational results for the single machine scheduling problems considered in this paper. Since efficiency is our main concern, we impose strict time limits for all optimization runs and analyze the resulting solution quality.

Our experiments regarding $\sum_{j=1}^{n} w_j T_j$ are based on a set of well known benchmark problems taken from the OR-library [3]. Unfortunately, the OR-library instances are limited to 100 jobs. However, 200-job problems generated according to the same scheme have been used in [2].

The Genetic Algorithm we use for our experiments can be described as a Standard Genetic Algorithm which has been modified in order to fit better into the problem environment. The most important aspects concerning the genetic algorithm are summarized in the following:

Solution encoding

We adopt a permutation based representation for the sake of compatibility with the local search methods. Additionally, it is the most natural encoding for sequencing and scheduling problems.

Initialization

The initial population is basically initialized at random. However, we additionally insert one solution generated by the heuristic. By this, we possibly introduce relevant building blocks for high quality solutions into the population.

Crossover and mutation

We use the Order Crossover (OX) as defined by Syswerda [13], and an insert based mutation operator.

Hybridization

In order to further improve solution quality, we hybridize the (GA) with local search algorithm (SA). A strict best improvement method based on the local search is used to re-optimize solutions after the crossover.

All tests were run with $n = 100$ and for $n = 200$ and the results are averaged over 20 independent runs. The parameters for each method are summarized in Table 1 and 2.

Table 1: Parameters for the Simulated Annealing

T_{max}	*10000*
T_0	*0.1*
α	*0.995*

Table 2: Parameters for the Genetic Algorithm

Population Size n	*(100 or 200)*
Selection	*Roulette Wheel*
Crossover	*OX*
Crossover Rate	*0.7*
Mutation Rate	*0.05*

Table 3 and 4 summarize the experimental results. The Δ-values represent the relative percentage deviation from the best known solutions.

For comparison purposes, we added results achieved by the *Problem Space Genetic Algorithm*, one of the best available approaches for single machine scheduling problems, as reported in [2]. It has to be remarked that these results were obtained under different conditions (no time restrictions).

According to Table 3, the Genetic Algorithm produces the most stable and robust results for $n = 100$. The hybridization algorithm produces a high Δ-m*ax* value, although its average percentage deviation is still relatively low. The (GA) cannot maintain its result quality for the 200-job problems but still obtains the lowest Δ-*max* value. The hybridization algorithm produces the lowest mean deviation. However, it had severe problems with one of the problem instances yielding a Δ-*max* of 4.02 %. In general, the (GA) seems to produce the most stable overall result quality. Its population-based exploration of the solution space is obviously less prone to be trapped in local optima as is the case for the neighborhood-based methods.

Table 3: Results for the 100-job instances Method

Method	Δ- *avg*	Δ- *median*	Δ- *max*
GA	*0. 20 %*	*0.01 %*	*3.80 %*
SA	*0.15 %*	*0.00 %*	*3.65 %*
Hybridization	*0.04 %*	*0.00 %*	*0.61 %*

Table 4: Results for the 200-job instances

Method	Δ- *avg*	Δ- *median*	Δ- *max*
GA	*0.36 %*	*0.15 %*	*4.82 %*
SA	*0.35 %*	*0.10 %*	*4.70 %*
Hybridization	*0.31 %*	*0.06 %*	*4.02 %*

Since there are no publicly available benchmark instances for $|rj|\sum_{j=1}^{n} w_jT_j$, we generated a set of problems according to the scheme reported in [1]. Processing times p_j and weights w_j were randomly sampled from the uniform distributions. The release times follow a uniform distribution between 0 to $\alpha\sum_{j=1}^{n} p_j$, where $\alpha \in \{0.0,0.5,1.0,1.5\}$. Due dates were computed using a randomly determined slack time d_j- (r_j+p_j) ranging from 0 to $\beta\sum_{j=1}^{n} p_j$, where $\beta \in \{0.05,0.25,0.5\}$. Using all possible combinations of processing time and weight ranges and all possible pairs of values for α and β.Tables 5 and 6 show the average improvement for each pair of α and β. Due to our first experiments, unequal release times add a considerable degree of difficulty. The experimental setup is basically the same as for the equal release time problems. The parameters for the GA were not changed since we did not observe quality improvements. It can be observed that the highest improvement ratios are obtained for the problems with scattered release times and loose due dates. However, only minor improvements are possible for instances with $\alpha = 0$ which correspond to the weighted tardiness problem with equal release dates $\sum_{j=1}^{n} w_jT_j$. (SA) shows the best overall behavior for both n = 100 and n = 200, whereas the Genetic Algorithm cannot keep up with the neighborhood-based methods for $\alpha > 0$. Obviously the structure of the solution space becomes different under arbitrary release date. The only small advantage of the (SA) algorithm over the hill climber emphasizes this assumption.

Table 5: Results for the 100-job instances with unequal release times

α	β	GA		SA		Hybridization	
		$\sum_{j=1}^{n} w_j T_j.$	Improve %	$\sum_{j=1}^{n} w_j T_j.$	Improve %	$\sum_{j=1}^{n} w_j T_j.$	Improve %
0.0	0.05	1682922	0.22 %	1682924	0.22 %	1682586	0.24 %
0.0	0.25	1433105	1.03 %	1433107	1.03 %	1432383	1.08 %
0.0	0.50	616795	1.48 %	616670	1.50 %	615417	1.70 %
0.5	0.05	572459	0.46 %	555551	3.40 %	554975	3.50 %
0.5	0.25	350793	7.35 %	281770	25.58 %	281694	25.60 %
0.5	0.50	60628	8.30 %	60625	8.30 %	60595	8.35 %
1.0	0.05	39841	4.29 %	31479	24.38 %	31220	25.00 %
1.0	0.25	906	54.31 %	630	68.20 %	553	72.10 %
1.0	0.50	0	100.00 %	0	100.00 %	0	100.00 %
1.5	0.05	10075	19.65 %	9316	25.70 %	9291	25.90 %
1.5	0.25	3	99.71 %	3	99.71 %	0	100.00 %
1.5	0.50	0	100.00 %	0	100.00 %	0	100.00 %

Table 6: Results for the 200-job instances with unequal release times

α	β	GA		SA		Hybridization	
		$\sum_{j=1}^{n} w_j T_j$.	Improve %	$\sum_{j=1}^{n} w_j T_j$.	Improve %	$\sum_{j=1}^{n} w_j T$ j.	Improve %
0.0	0.05	6835082	0.12 %	6835100	0.12 %	6834397	0.13 %
0.0	0.25	5126121	0.26 %	5126123	0.26 %	5125093	0.28 %
0.0	0.50	2789613	0.96 %	2787641	1.03 %	2785951	1.09 %
0.5	0.05	2410618	0.17 %	2385746	1.20 %	2382848	1.32 %
0.5	0.25	1195804	1.09 %	924871	23.50 %	920035	23.90 %
0.5	0.50	402520	1.48 %	216617	46.98 %	204202	50.02 %
1.0	0.05	28.199	1.83 %	21113	26.50 %	20966	27.01 %
1.0	0.25	16	79.34 %	0	100.00 %	0	100.00 %
1.0	0.50	27	95.83 %	0	100.00 %	0	100.00 %
1.5	0.05	2993	61.23 %	1351	82.50 %	1127	85.40 %
1.5	0.25	43	93.14 %	31	95.18 %	26	95.90 %
1.5	0.50	0	100.00 %	0	100.00 %	0	100.00 %

CONCLUSION AND OUTLOOK

The purpose of this research was to develop and evaluate methodologies for solving the single machine scheduling problem to minimize total tardiness. Since the complexity of this problem is at least NP-Hard, this paper developed and implemented effective heuristics that would provide good solutions within a reasonable time.

We presented efficient heuristic optimization methods (Genetic Algorithm and Simulated Annealing) for single machine scheduling problems with and without release times. The algorithms were able to produce high quality results, especially for $w_j T_j$.

Computational results show that the proposed heuristics can provide good solutions with the average relative deviation, it is shown that the combined GA/SA heuristic delivers good performance in obtaining the near optimal job sequence inside a family. Furthermore, the two promising heuristics, may be used in a variety of situations.

Reference

1. Akturk, M. S. and Ozdemir, D. (2000). “An exact approach to minimizing total weighted tardiness with release dates”. *IIE Transactions*, 32:1091-1101.

2. Avci, S., Akturk, M. S., and Storer, R. H. (2003).”A problem space algorithm for single machine weighted tardiness problems”. *IIE Transactions*, 35:479-486.

3. Beasley, J. (1990).”Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society”*, 41(11):1069-1072.

4. Bilge, U., Kurtulan, M., and Kirac, F. (2006).”A tabu search algorithm for the single machine total weighted tardiness problem”. *European Journal of Operational Research*. In Press.

5. Congram, R. K., Potts, C. N., and Van de Velde, S. (2002).”An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem”. *INFORMS Journal on Computing*, 14(1):52-67.

6. Crauwels, H. A. J., Potts, C. N., and Van Wassenhove, L. N. (1998). “Local search heuristics for the single machine total weighted tardiness scheduling problem”. *INFORMS Journal on Computing*,10(3):341-350.

7. Kirkpatrick, S., Gelatt, C. D., JR., and Vecchi, M. P. (1983). “Optimization by simulated annealing ”.Science, 220(4598):671-680,.

8. Lawler, E. L. (1977). A pseudo polynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Operations Research*, 1:331-342.

9. Metropolis, N., Rosenblunth, A. W., Rosenblunth, M. N., Teller, A. H., and Teller, E. (1953).” Equations of state calculations by fast computing machines”. Journal of Chemical Physics, 21(6):1087–1092,.

10. Pinedo, M. (2002).” *Scheduling: Theory, Algorithms ,and Systems”*. Prentice Hall, 2nd edition.

11. Potts, C. N. and Van Wassenhove, L. N. (1985). “A branch and bound algorithm for the total weighted tardiness problem”. *Operations Research*, 33(2):363-377.

12. Rinnooy Kan, A. H. G. (1976).” *Machine scheduling Problems: Classification, complexity and computations”*. Nijhoff, The Hague.

13. Syswerda, G. (1991).”Schedule optimization using genetic algorithms”. In Davis, L., editor, *Handbook of Genetic Algorithms*, pages 332-349. Van Nostrand Reinhold, New York.