A Cluster-based Evolutionary Algorithm for the Single Machine Total Weighted Tardiness-scheduling Problem

István Borgulya

University of Pécs, Hungary

In this paper a new evolutionary algorithm is described for the single machine total weighted tardiness problem. The operation of this method can be divided in three stages: a cluster forming and two local search stages. In the first stage it approaches some locally optimal solutions by grouping based on similarity. In the second stage it improves the accuracy of the approximation of the solutions with a local search procedure while periodically generating new solutions. In the third stage the algorithm continues the application of the local search procedure. We tested our algorithm on all the benchmark problems of ORLIB. The algorithm managed to find, within an acceptable time limit, the best-known solution for the problems, or found solutions within 1% of the best-known solutions in 99 % of the tasks.

Keywords: scheduling, deterministic single machine, evolutionary algorithm.

1. Introduction

The single machine total weighted tardiness problem (SMTWTP) is a scheduling problem. In SMTWTP n jobs have to be sequentially processed on a single machine. Each job j has a processing time p_j, a weight w_j, and a due date d_j associated, and the jobs become available for processing at time zero. The tardiness of a job j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j in the current job sequence. The goal is to find a job sequence which minimizes the sum of the weighted tardiness given by $\sum_{i=1,..,n} w_i T_i$.

The SMTWTP is an *NP*-hard scheduling problem. Techniques which can be used to find exact optimal solution are limited to dynamic programming and branch and bound methods. Often, instances with more than 50 jobs cannot be solved to optimality with branch and bound algorithms. Therefore, several heuristic methods have been proposed for their solution [4], [5]. These include simple construction heuristics, local search and metaheuristic methods.

The three standard construction heuristics are the Earliest Due Date (EDD), the Modified Due Date (MDD) and the Apparent Urgency (AU) [7]. The EDD heuristic puts the jobs in non-decreasing order of the due dates d_j. The MDD heuristic puts the jobs in non-decreasing order of the modified due dates mdd_j, given by $mdd_j = \max\{\mathrm{C} + p_j d_j\}$, where C is the sum of the processing times of the already sequenced jobs. Finally the AU heuristic puts the jobs in non-decreasing order of the apparent urgency au_j, given by $au_j = (w_j/p_j)\exp(-(\max\{d_j - c_j, 0\})/(k * ap)$, where ap is the average processing time of the remaining jobs, k is the "look ahead" parameter [6].

Local search algorithms repeatedly replace the current permutation π with a better sequence found in the neighborhood of π. The simplest local search algorithm is the lexicographic search that finds a better or the best solution in a neighborhood. We can improve the performance of the local search by choosing neighborhood structures. The following neighborhood structures are customary to use:

- *interchange*: exchange of the jobs placed at the ith and the jth position, $i \neq j$
- *insert*: removal of the job at the ith position and insertion in the jth position.

More complex neighborhood structures are e.g. general pairwise interchange (GPI), adjacent pairwise interchange (API) (this is a subset of GPI) and dynasearch swaps (*dyna*). E.g. the dynasearch algorithm uses dynamic programming to find the best move (transformation) in a *dyna* witch is composed of a set of independent interchange moves (Two interchanges moves are independent if for the two moves involving position i, j and k, l have that $\min\{i, j\} \geq \max\{k, l\}$, or vice versa [6]). Complex neighborhood structure is also the variable neighborhood descent (VDN). In VDN we concatenate different neighborhood e.g. *interchange* + *insert, insert* + *interchange* and *insert* + *dyna*.

We can improve performance of the local search with multi-start or iterated local search (ILS) techniques. By multi-start the procedure run multiple, starting from different initial solutions, and we select the best sequence as final solution. A far more effective approach is to allow dependent runs by generating the new starting solution from one of the previous local optima. Such an approach, known as ILS. ILS, appears to be a very promising approach for solving the SMTWTP. A variant of the ILS, called iterated dynasearch (ILS + *dyna*) gives one of the best performance results for the SMTWTP [6].

Further methods for the SMTWTP are metaheuristics. We can use e.g. simulated annealing (SA), tabu search (TS), evolutionary algorithms (EA) and ant colony optimization (ACO). These methods are approaching the global optimum on the analogies from physics or biology ([1], [5]).

We wish to enrich the scope of these heuristic methods with a new algorithm. We have aimed at developing an algorithm, which:

- gives a good approximation within an acceptable time limit for recurrent tasks when using a PC (estimates the optimum within 1% of the best-known solutions); and
- is easy to use with the adjustment of only a few algorithm parameters.

In the rest of the paper, let us review the principle of the new algorithm, the algorithm in detail, and its applications.

2. The principle of the new algorithm

The new method, named CE_ST (Cluster-based Evolutionary algorithm for the Single machine total weighted Tardiness problem) estimates the global optimum by the simultaneous search for a number of local optima. It is a cluster-based evolutionary algorithm, which generates the first estimations of the local optima (permutations) by grouping based on the similarity, then it improves the results by a complex local search procedure. Further on, based on the first estimations, it periodically generates new solutions while correcting the measure of approximation by local search procedures. (The algorithm was developed by the modification of previous optimization algorithms [2], [3]).

The operation of CE_ST can be divided in three stages: cluster forming and two local search stages. During the first stage it estimates some local optima by grouping. For problems where the neighborhood of the global optima is easy to find, it is enough to form only two clusters. But, provided the local optima hardly differ from each other, it is advisable to divide the set of permutations into more clusters. In the second stage it takes turns with a local search procedure to correct all the estimates, meanwhile it periodically generates new estimations of the local optima in the neighborhood of existing ones. Finally, in the third stage the algorithm further refines the estimations. Unlike in the previous stage, this time, the number of estimations is constant.

1. The first stage forms some clusters continuously from randomly generated feasible solutions. The solutions are arranged into t clusters by their similarity and we can then accord a prototype to each cluster. The prototype of a cluster is always set to be the solution with the least objective function value, and from all possible members of the cluster only the prototype is stored. The algorithm takes the prototypes as approximations of the solution.
2. In the second stage accuracy of the prototypes is improved with a local search procedure that takes candidates for the improved optimum from the neighborhood of the prototype.
   After studying various neighborhood structures the one that is the best for search proved

to be the VDN structure, which applies the *interchange+insert* and the *insert+interchange* transformations (moves) with equal probabilities and applies the *insert* structure in both moves at a probability of only 0.25. In selecting the prototype to be corrected, priority is given to the best, smallest objective function value. The algorithm selects the fittest prototype with 0.5% probability or one of the prototypes, with $0.5/t$ probability (where t is the number of prototypes). At certain tasks this neighborhood structure is not enough for the complete run, the algorithm might get "stuck" at one of the local optima. To help the solution, the algorithm generates new prototypes. A new prototype is generated randomly from an already existing one with an *interchange + insert + interchange* neighborhood structure. Thus, new prototypes can be found in the "narrow neighborhood" of the existing ones, and, as new variants, they improve the capability and the speed of the algorithm to approximate the global optimum. Thus, in the second phase new prototypes are periodically generated in the neighborhood of the existing ones until a maximum number of the prototypes is completed.

3. In the third stage the solutions are further improved by the local search procedures employed in the second stage. Unlike in the previous stage, this time, the number of the estimations is constant.

3. New algorithm

3.1. Algorithm as EA

The CE_ST is a population-based procedure. Let us view the estimations (as permutation vectors) as individuals and the generation of new estimations from previous ones as mutations. In this case each stage can be understood as a distinct EA. In each EA a descendent is produced by generations (iterations) from a parent by copy making. We apply mutation, then having matched the descendant with the best cognate (or with the parent itself) we select between the parent and the descendent. We can apply the objective function as fitness function. The algorithm comprises three steady-state EAs which produce one descendent by generations (iterations) and select between the individuals by means of deterministic crowding.

Let us designate the algorithm of the 3 stages as EA1, EA2 and EA3. We describe the functioning of CE_ST by means of these EAs. These are as follows:

EA1: In the first stage it creates clusters.

EA2: In the second stage with a local search it improves the quality of the solutions and periodically, in every mth iteration, increases the number of the individuals (generates new prototypes).

EA3: In the third stage with the local search it improves the quality of the solutions.

3.2. Steps of CE_ST

Introduced notations

Let us introduce the following notations:

- The same population and individuals are used with all EAs. Let p_1 , $p_2, \ldots, p_t$ be the prototypes, that are estimations of the global optimum. Let the population of the itth generation be denoted by P(it), and ith individual by p_i. The fitness function is identical to the Z objective function.
- Measure of the similarity between the two permutations x and z is given by $H(x,z) = 1/(1+d(x,z))$ where d(x,z) is the Hamming distance between the permutations.
- Let us designate the random number generator as Rnd (uniform distribution on (0,1)).
- Let as denote the local search procedure *Nhs*(q). The procedure applies the *interchange + insert* and the *insert + interchange* transformations (moves) with equal probabilities for a q permutation. In both moves the *insert* structure is applied only at a probability of 0.25.
- Let us denote *Newind* the procedure which increases the number of the individuals.
- Let us denote the pairwise interchange procedure by *Swap*(q). The procedure randomly chooses two positions in the solution q and swaps their contents.

Parameters

Five parameters affect the run of the algorithm: *tmax*, *t*, *itt*, *kn*, *m* and *itend*. Their roles one-by-one:

- *tmax* – maximal number of the local minima to be found.
- *t* – number of the local minima in the first stage to be found.
- *itt* – a parameter of the local search. If the number of iterations (*it*) reaches *itt*, the local search (stage 2) begins.
- *m* – a parameter of the local search. The size of the population is expanded only at each *m*th iteration.
- *timelimit* – parameter for the stopping condition. The procedure is finished if the running time (in CPU seconds) exceeds the *timelimit*.

```
Procedure CE_ST(tmax, t, itt, m, timelimit, opt, optp).
* EA1 ********************************
it:=0, glob:=1, kn = 100.      /* Initial values.
Let $p_i \in \Pi(n)$ (i=1,...,t), P(it)← {$p_1$, ..., $p_t$}.
Compute Z($p_1$), ..., Z($p_t$).
Repeat
    i:=[t*Rnd]+1, q:=$p_i$.   /* New descendent
    For j:=1 to 4 do Swap(q) od.  /* Mutation
    Compute Z(q).
    Let H(q,$p_z$)=$\max_j$H(q,$p_j$);j,z∈{1,2,...,t}.
    If Z(q)<Z($p_z$) then $p_z$:=q fi.  /* Selection
    it:=it+1, P(it)←P(it-1).
until itt<it.

*EA2**********************************
Repeat
    Repeat
        If 0.5<Rnd then i:=glob
            else i:=[t*Rnd]+1 fi.
        q:=$p_i$                 /* New descendent
        Nhs(q).             /* Mutation
        Compute Z(q).
        If Z(q)<Z($p_i$) then $p_i$:=q fi./* Selection
        it:=it+1, P(it)←P(it-1).
        If  mod(it,m)=0 then Newind fi.
    until mod(it,kn)≠0.
    Let Z($p_i$)=$\min_j$Z($p_j$),i,j∈{1,2,...,t}, glob:=i.
    opt=Z(glob): optp=$p_{glob}$
    If "running time" > timelimit then exit fi.
until t < tmax.

* EA3*********************************
Repeat
    Repeat
        i:=[t*Rnd]+1
        q:=$p_i$.               /* New descendent
        Nhs(q).             /* Mutation
        Compute Z(q).
        If Z(q)<Z($p_i$) then $p_i$:=q fi./* Selection
        it:=it+1, P(it) ← P(it-1).
    until mod(it,kn)≠0.
    Let Z($p_i$)=$\min_j$Z($p_j$), i,j∈{1,2,...,t}, glob:=i.
    opt=Z(glob): optp=$p_{glob}$
until "running time" < timelimit.
*************************************
exit
end
```

3.3. Details of Implementation

When describing the algorithm, some heuristic solutions and the shortcuts of the computations were not described. Let us see them now one-by-one.

Initial population

In our implementation the initial population is generated by the EDD heuristic. First we generate a permutation, which is the solution of the EDD, and second, we generate all individual of the initial population from this permutation with 4 *interchange* moves. With the help of this initial population the algorithm can converge faster.

Heuristic in the neighborhood structure

We use the rule of the EDD heuristic also in the neighborhood structure. The first move in the *Nhs* procedure uses the positions i, j. Our algorithm looks for a random i, j position at the probability of 0.9 so as $d_j < d_i$ is true. It tries to find an appropriate i, j position only 10 times, and if it does not succeed, this heuristic rule is skipped.

Speeding-up the computation

Calculating the values of the objective function values faster can speed up the algorithm. Reduction of the computing time is based on the possibility that, by SMTWTP, local search changes only part of the total weighted tardiness.

After each application of the *Nhs* procedure only one or two parts of the series is changed, all the other positions of the actual permutation remain unchanged. So only those partial sums of the total weighted tardiness that were changed after applying the *Nhs* have to be recalculated. Naturally, this shortcut can be used only if we keep the c_k, $w_k^* t_k$ $(k = 1, \ldots, n)$ vectors calculated earlier and we can use them in the calculation of the new function value. To speed up the calculation, the algorithm is storing the c_k, $w_k^* t_k$ $(k = 1, \ldots, n)$ vectors for each individual.

4. Test Results

Test problems

We have tested the CE_ST with the benchmark set of the randomly generated instances, available via ORLIB at `http://www.ms.ic.ac.uk/info.html`. The benchmark set comprises instances with 40, 50, and 100 jobs. In each three sets there are 125 instances. For the 40 and 50 job instances optimal solutions are known, while for the 100 job instances only the best-known solutions are given.

Parameter selection

To achieve a quick and accurate solution we need appropriate parameter values. Studying some of the more complex problems of the benchmark set we analyzed how parameter values were affecting the convergence, finding of the global optimum and the speed of the calculation.

So we analyzed the number of prototypes, which should be used in CE_ST to achieve a good trade-off between the probability of finding best-known solutions and the necessary run-time to do so. In general, the best behavior has been obtained with a number of prototypes between 5 and 30. For a too small number of prototypes, some optimal solutions are found rather fast, but with increasing the run-time often CE_ST with a larger number of prototypes the achieves higher probabilities of finding optimal solutions. So for test problems the number of prototypes of 30 was found appropriate $(tmax = 30)$.

As a next step, we analyzed the influence of the second stage on the run-time and on the convergence. We concluded that both the speed of convergence and the quality of the result are improved if we start the 2nd stage with a lower initial number of clusters and broaden this number by adding a new element to the population every time after some thousand iterations. But, in this situation, it is also true that a larger number of clusters achieves higher probabilities of finding optimal solutions. The speed of convergence is influenced by the frequency of broadening of the population, too. We found that the convergence is more successful if the population is broadened relatively slowly, after some thousands of iterations, than if we broaden it more quickly, after each 100-200 iterations. We concluded that an initial number of clusters of 10 $(t = 10)$ and a broadening at every 5000 iterations $(m = 5000)$ was observed to be optimal on the benchmark sets.

Finally, we studied the number of iterations of the first stage. We found the first stage to be necessary, because forming the first clusters is beneficial on the accuracy of the results, it usually decreases the scatter of the result. However, 1-2 hundreds of iterations are enough for forming the clusters (this is the most time consuming part of the run of the algorithm).

The following parameters were used at the test problems: $t = 10$, $tmax = 30$, $m = 5000$, $itt = 100$. Run time was set to 6, 12 and 80 seconds for the 40, 50 and 100 job instances, respectively.

Comparative results

As for comparison, we chose the multi-start version of the TS, SA (the number of starts is 5) and the iterated dynasearch (ILS-*dyna*) of Crauwels, Potts and Van Wasserhove (1998). For the comparison [5] and [6] included data of sufficient details, some of which we used. (We couldn't include the ACO method [1] in the range of methods chosen for comparison, because data including sufficient details were not available.)

The comparison was encumbered by the use of various programming languages, operating systems and computers. Only one appropriate aspect of comparison could be found, namely the average relative percentage deviation of the solution from the best known solution, so our table of comparison (Table 1) is based on the results of comparable accuracies. (Our CE_ST program run using a 1.5 GHz Pentium IV HP

Vectra with 128 MB RAM, and the program was written in Visual Basic and run under Windows 2000 professional. The TS and ILS-dynasearch programs were written in *C* and run using an HP 9000-G50 computer).

	N	ARPD	MRPD	NB	ACT	
					G50	Vectra
$SA_{multi\text{-}s}$	40	0.01	0.81	121	2.61	
TS	40	0.06	6.70	115	1.35	
$TS_{multi\text{-}s}$	40	0.00	0.33	118	1.32	
CE_ST	40	0.010	0.37	124		1.30
ILS-*dyna*	40	0.015	1.85	123	0.20	
$SA_{multi\text{-}s}$	50	0.09	11.20	115	6.11	
TS	50	0.01	0.2	111	2.99	
$TS_{multi\text{-}s}$	50	0.01	0.28	113	2.95	
CE_ST	50	0.016	0.48	118		3.13
ILS-*dyna*	50	0.002	0.17	125	0.64	
$SA_{multi\text{-}s}$	100	0.12	4.78	59	40.5	
TS	100	0.06	4.78	96	38.3	
$TS_{multi\text{-}s}$	100	0.04	4.39	103	37.6	
CE_ST	100	0.044	2.28	80		28.04
ILS-*dyna*	100	0.04	1.83	82	2.00	
		0.007	0.41	107	2.80	

Table 1. Comparative results.

We compared average results: for all the four algorithms we ran each problem (or others ran each problem) 10 times, and the average results are shown. We compared performance of the various algorithms on the basis of the following statistics: the average relative percentage deviation of the solution from the best known solution (ARPD); the maximum relative percentage deviation of the solution from the best known solution (MRPD); the number of the best known solution values found out of 125 (NB); the average computation time in second (ACT) on a HP 9000-G50 (G50) or on a HP Vectra VL800 (Vectra).

Reviewing the results we can conclude that the ILS-dynasearch algorithm yielded the best results. It reached the specified accuracy in the shortest time, and in more cases it was the one that found the best known solutions.

According to the statistics the results of the CE_ST are mostly between the results of the multi-start TS and the ILS-dynasearch. If we focus on the details of the run, we see that the CE_ST managed to find within the acceptable time limit the best-known solution for the problems, or found solutions within 1% of the best-known solutions in 99% of the tasks. This result can be specified further: the CE_ST found solutions within 0.1% of the best-known solutions in 90% of the tasks.

Considering that the TS and ILS-dynasearch are the most effective methods of SMTWTP, we can conclude that the CE_ST belongs to the best available methods, too.

5. Summary

We can conclude that the CE_ST was successfully tested with different kinds of SMTWTP. The method solves the usual test problems at a very good accuracy and performance. Comparing the results with other heuristic methods, we can conclude that the CE_ST belongs to the best methods of this problem scope. The build-up of the method, especially the possibility of broadening the 2nd stage population, allows quicker convergence in comparison to former methods. In this field our method has possibilities comparable to the ones of the ILS technique.

We still did not use up the possibilities of the method: for example, there is a possibility of speeding up the convergence. Changing to the *C* programming language instead of the current one, or realizing further speed-up possibilities would make computation time even shorter, which would allow improved accuracy, too.

Acknowledgments

The Hungarian Research Foundation OTKA T 030861 supported the study.

References

[1] BESTEN M., STÜTZLE T., DORIGO M., Ant Colony Optimization for the Total Weighted Tardiness Problem. In: Schoenauer M. et al. editors, *Parallel Problem Solving from Nature — PPSN VI. Lecture Notes in Computer Science* 1917, Springer-Verlag Berlin 2000, pp. 611–620.

[2] BORGULYA I., Constrained optimization using a clustering algorithm. *Central European Journal of Operations Research.* **8** (1) 2000. pp. 13–34.

[3] BORGULYA I., A Cluster-based Method for the Quadratic Assignment Problem. *XXV. Magyar Operációkutatási Konferencia*, Debrecen, 2001.

[4] BRUCKER P., HURINK J., Complex Sequencing problems and local search heuristics. In: Osman IH, Kelly JP. editors, Metaheuristics: *Theory and Applications*, Kluwer Academic Publishers, Boston, 1986. pp. 151–166.

[5] CRAUWELS HAJ., POTTS CN., VAN WASSENHOVE., Local search heuristics for the single machine total weighted tardiness- scheduling problem. *INFORMS Journal on Computing* **10** 1998. pp. 341–350.

[6] CONGRAM RK., POTTS CN., VAN DE VELDE, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *Technical report*. Faculty of Mathematical Studies, University of Southampton, December (1998).

Received: June, 2002
Accepted: September, 2002

Contact address:

Istvań Borgulya
University of Pécs
Rákóczi ut 80
7621 Pécs, Hungary
e-mail: borgulya@ktk.pte.hu

ISTVÁN BORGULYA is associate professor in the Department of Business Informatics, University of Pécs, Hungary. He received the Ph.D. degree in computer and law from the University of Pécs too. His research interests include modeling in law with the methods of artificial intelligence, fuzzy methods in the decision support, and optimization with evolutionary algorithms.