Task Realization's Optimization with Earliness and Tardiness Penalties in Distributed Computation Systems

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

[1] Institute of Engineering, Wrocław University of Technology,
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wbo@ict.pwr.wroc.pl
[2] Institute of Computer Science, University of Wrocław,
Przesmyckiego 20, 51-151 Wrocław, Poland
mwd@ii.uni.wroc.pl

Abstract. There are many service systems (especially in reservation systems, electronic commerce, in tasks synchronized directly with the Internet), where each task has to be executed in some fixed range of time. Exceeding the term is disadvantageous and causes additional penalties. Therefore, it is necessary to establish optimal sequence of tasks (which minimizes penalties) and its starting times. It amounts to some job scheduling problem with earliness and tardiness. Because usually tasks are received in the distributed system (in the web), so to solve the presented problem we propose a parallel coevolutionary algorithm based on the Lamarck evolution and the island model of migration.*

1 Introduction

There are some types of manufacturing systems, called Just In Time (JIT), where costs are connected not only with executing a job too late, but also too early. Such a situation takes place especially when tasks are connected directly with the Web, e.g. routing, agents, similarity classification, etc. This induces formulating many optimization problems with goal functions, where there is a penalty for both tardiness and earliness of a job. The problem of scheduling with earliness and tardiness (total weighted earliness/tardiness problem, $TWET$) is one of the most frequently considered in literature. In this problem each job from a set $J = \{1, 2, ..., n\}$ has to be processed, without interruption, on a machine, which can execute at most one job in every moment. By p_i we represent the execution time of a job $i \in J$, and by e_i and d_i we mean an adequately demanded earliest and latest moment of the finishing processing of a job. If a scheduling of jobs is established and C_i is the moment of finishing of a job i, then we call $E_i = \max\{0, e_i - C_i\}$ an *earliness* and $T_i = \max\{0, C_i - d_i\}$ a *tardiness*. The

* The work was supported by KBN Poland, within the grant No. T11A01624.

P.S. Szczepaniak et al. (Eds.): AWIC 2005, LNAI 3528, pp. 69–75, 2005.

expression $u_iE_i + w_iT$ is a *cost* of execution a job, where u_i and w_i $(i \in J)$ are nonnegative coefficients of a goal function. The problem consists in minimizing a sum of costs of jobs, that is the function $\sum_{i=1}^{n}(u_iE_i + w_iT_i)$. Such a problem is represented by $1||\sum(u_iE_i + w_iT_i)$ in literature and it belongs to a strong NP-hard class (if we assume $u_i = 0$, $i = 1, 2, \ ... \ , n$, we will obtain a strong NP-hard problem $1||\sum w_iT_i$ - Lawler [6] and Lenstra et al. [7]). Baker and Scudder [1] proved, that there can be an idle time in an optimal solution (jobs need not be processed directly one after another), that is $C_{i+1} - p_{i+1} \geq C_i$, $i = 1, 2, \ ... \ , n-1$. Solving the problem amounts to establishing a sequence of jobs and its starting times. Hoogeven and van de Velde [5] proposed an algorithm based on a branch and bound method. Because of exponentially growing computation's time, this algorithm can be used only to solve instances where the number of jobs is not greater than 20. Therefore in practice almost always approximate algorithms are used. The best of them are based on artificial intelligence methods. Calculations are performed in two stages:

1. Determining the scheduling of jobs (with no idle times).
2. Establishing jobs' optimal starting times.

There is an algorithm in the paper of Wan and Yen [9] based on this scheme. To determine scheduling a tabu search algorithm is used.

In this paper we consider a $TWET$ problem additionally assuming that the machine begins execution of jobs in time zero and it works with no idle ($TWET$-*no-idle* problem). We present a parallel genetic algorithm (called coevolutionary), in which a part of a population is replaced with adequately local minima (so-called Lamarck evolution). The property of partitioning a permutation into subsequences (blocks) was used in an algorithm of determining local minima. This method decreases the size of a neighborhood to about 50% in a local optimization algorithm, it improves the solution's values and significantly speeds up computations.

Sections 2 contains a description of a genetic algorithm and local optimization algorithm. Subsequent sections describe computer simulations of sequential and parallel algorithms.

2 Evolutionary Algorithm Method

The evolutionary algorithm is a search procedure, based on the process of natural evolution following the principles of natural selection, crossover and survival. The method has been proposed and developed by Holland [4]. In the beginning, a population of individuals (solutions of the problem, for example permutations) is created. Each individual is evaluated according to the fitness function. Individuals with higher evaluations (more fitted, with a smaller goal function value) are selected to generate a new generation of this population. So, there are three essential steps of the genetic algorithm: (1) selection – choosing some subset of individuals, so-called parents, (2) crossover – combining parts from pairs of parents to generate new ones, (3) mutation – transformation that creates a new

individual by small changes applied to an existing one taken from the population, (4) succession – determining of the next population. New individuals created by crossover or mutation replace all or a part of the old population. The process of evaluating fitness and creating a new population generation is repeated until a termination criterion is achieved.

Let P_0 be an initial population, k – number of iteration of the algorithm, P_k – population. Let P'_k be a set of parents – subset of the most fitted individuals of the population P_k. By the mechanism of crossover, the algorithm generates a set of offsprings P''_k from the set P'_k. Next, some of the individuals from the set P''_k are mutated. *Succession* operation consists in reducing P''_k to the size $|P_0|$, forming the new population P_{k+1}. The algorithm stops after a fixed number of iterations. Of course the complexity of the algorithm depends on the number of iterations and the size of the population.

All operations in a coevolutionary algorithm (selection, crossover and succession) are executed locally, on some subsets of the current population called *islands*. It is a strongly decentralized model of an evolutionary algorithm. There are independent evolution processes on each of the islands, and communication takes place sporadically. Exchanging individuals between islands secures diversity of populations and it prevents fast imitating of an individual with a local minimum as its goal function. Additionally, the so-called Lamarck model of evolution is applied to intensify the optimization process. In each generation some part of the population is replaced by "their" local minima. From the current population some subset is drawn. Each individual of this subset is a starting solution for the local optimization algorithm. Therefore on each island a hybrid algorithm is applied, in which an evolutionary algorithm is used to determine the starting solutions for the local search algorithm.

Evolutionary algorithm
Number of iteration k :=0; $P_0 \leftarrow$ initial population;
repeat
$P'_k \leftarrow$Selection(P_k); {*Selection of parents*}
$P''_k \leftarrow$Crossover(P'_k); {*Generating an offspring*}
$P''_k \leftarrow$Mutation(P''_k);
$A \leftarrow$RandomSubSet(P''_k); {*Subpopulation*}
$P''_k \leftarrow P''_k \cup$LocalMinimumSet($A$);
$P_{k+1} \leftarrow$Succession(P_k, P''_k) {*A new population*}
$k := k + 1$;

until some termination condition is satisfied;

The *RandomSubSet* procedure determines a random subset A of elements of the current population P''_k, and the *LocalMinimumSet* procedure calculates a subset of local minima.

Because in the coevolutionary algorithm subpopulation develops on the islands independently, so calculations can be executed in the distributed web.

Parallel coevolutionary algorithm
The parallel algorithms based on the island model divide the population into a few subpopulations. Each of them is assigned to a different processor which performs a sequential genetic algorithm based on its own subpopulation. The crossover involves only individuals within the same population. Occasionally, the processor exchanges individuals through a migration operator. The main determinants of this model are: (1) size of the subpopulations, (2) topology of the connection network, (3) number of individuals to be exchanged, (4) frequency of exchanging.

The island model is characterized by a significant reduction of the communication time, compared to previous models. Shared memory is not required, so this model quite flexible.

Local Search Method
The local search (LS) method is a metaheuristic approach designed to find a near-optimal solution of combinatorial optimization problems. The basic version starts from an *initial solution* x^0. The elementary step of the method performs, for a given solution x^i, a search through the *neighborhood* $N(x^i)$ of x^i. The neighborhood $N(x^i)$ is defined by a move performed from x^i. The move transforms a solution into another solution. The aim of this elementary search is to find in $N(x^i)$ a solution $x^{i+1}$with the lowest cost functions. Then the search repeats from the best found solution, as a new starting one (see Glover and Laguna [3]).

Local search algorithm

Select a starting point: x; $x_{best} := x$;
repeat
  Select a point $y \in N(x)$ according to a given criterion
  based on the value of the goal function $F(y)$;
  $x := y$;
  if $F(y) < F(x_{best})$ **then** $x_{best} := y$;
until some termination condition is satisfied;

A fundamental element of the algorithm, which has a crucial influence on the quality and time of computation, is the neighborhood. The neighborhood is generated by the insert moves in the best local search algorithms with the permutation representation of the solution. Let k and l ($k \neq l$) be a pair of positions in a permutation. Insert move (*i-move*) i_l^k, consists in removing the element from the position k and next inserting it in the position l.

For the *TWET-no-idle* problem, each schedule of jobs can be represented by permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of the set of jobs J. Let $S(n)$ denote the set of all such permutations. The total cost $\pi \in S(n)$ is $F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)})$, where $C_{\pi(i)}$ is completed time of the job $\pi(i)$, $C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$. The job $\pi(i)$ is considered *early* if it is completed before its earliest moment of finishing ($C_{\pi(i)} < e_{\pi(i)}$), *on time* if $e_{\pi(i)} \leq C_{\pi(i)} \leq d_{\pi(i)}$, and *tardy* if the job is completed after its due date (i.e. $C_{\pi(i)} > d_{\pi(i)}$).

Each permutation $\pi \in S(n)$ is decomposed into subpermutations (subsequences of jobs) $\Omega = [B_1, B_2, \ldots, B_v]$ called *blocks* in π, each of them contains the jobs, where:

1. $B_i = (\pi(a_i), \pi(a_i+1), \ldots, \pi(b_i-1), \pi(b_i))$, and
   $b_i = a_{i-1}+1, \;\; 1 \le i \le v, \;\; a_0 = 0, \; b_v = n.$
2. All the jobs $j \in B_i$ satisfy the following condition:
   $e_j > C_{\pi(b_i)}$, or (C1)
   $e_j \le C_{\pi(b_{i-1})} + p_j$ and $d_j \ge C_{\pi(b_i)}$, or (C2)
   $d_j < C_{\pi(b_{i-1})} + p_j$. (C3)
3. B_i are maximal subsequences of π in which all the jobs satisfy either Condition $C1$ or Condition $C2$ or Condition $C3$.

By definition, there exist three types of blocks implied by either C1 or C2 or C3. To distinguish them, we will use the *E-block*, *O-block* and *T-block* notions respectively. For any block B in a partition Ω of permutation $\pi \in S(n)$, let

$$F_B(\pi) = \sum_{i \in B} (u_i E_i + w_i T_i).$$

Therefore, the value of a goal function

$$F(\pi) = \sum_{i=1}^{n} (u_i E_i + w_i T_i) = \sum_{B \in \Omega} F_B(\pi).$$

If B is a T-block, then every job inside is early. Therefore, an optimal sequence of the jobs within B of the permutation π (that is minimizing $F_B(\pi)$) can be obtained, using the well-known Weighted Shortest Processing Time ($WSPT$) rule, proposed by Smith [8]. The $WSPT$ rule creates an optimal sequence of jobs in the non-increasing order of the ratios w_j/p_j. Similarly, if B is an E-block, than an optimal sequence of the jobs within can be obtained, using the Weighted Longest Processing Time ($WLPT$) rule which creates a sequence of jobs in the non-decreasing order of the ratios u_j/p_j.

Partition Ω of the permutation π is *ordered*, if there are jobs in the $WSPT$ sequence in any T-block, and there are jobs in the $WLPT$ sequence in any E-block.

Theorem 1. [2] Let Ω be an ordered partition of a permutation $\pi \in S(n)$ to blocks. If $\beta \in S(n)$ and $F(\beta) < F(\pi)$, so at least one job of some block of π was moved before the first or after the last job of this block in permutation β.

Note that Theorem 1 provides the necessary condition to obtain a permutation β from π such, that $F(\beta) < F(\pi)$.

Let $\Omega = [B_1, B_2, \ldots, B_v]$ be an ordered partition of the permutation $\pi \in S(n)$ to blocks. If a job $\pi(j) \in B_i$ ($B_i \in \Omega$), therefore moves which can improving goal function value consists in reordering a job $\pi(j)$ before the first or after the last job of this block. Let N_j^{bf} and N_j^{af} be sets of such moves ($N_j^{bf} = \emptyset$ for $j \in B_1$ and $N_j^{af} = \emptyset$ for $j \in B_v$). Therefore, the neighborhood of the permutation $\pi \in S(n)$,

$$N(\pi) = \bigcup_{j=1}^{n} N_j^{bf} \cup \bigcup_{j=1}^{n} N_j^{af}. \quad (1)$$

As computational experiments show, the neighborhood defined in (1) has a half smaller size than the neighborhood of all the insert moves.

3 Computational Experiments

We have tested the proposed algorithm on a set of randomly generated problems on a Sun Enterprise 4x400MHz using Ada95 language. For each job i, an integer processing time p_i was generated from the uniform distribution [1, 100] and integer weights u_i and w_i were generated from the uniform distribution [1, 10]. Let $P = \sum_{i=1}^{n} p_i$. Distributions of earliness e_i and deadline d_i depend on P and two additional parameters L and R, which take on values from 0.2 to 1.0 in increments of 0.2. An integer deadline d_i was generated from the uniform distribution $[P(L - R/2), P(L + R/2)]$. Earliness e_i was generated as an integer from the uniform distribution $[0, d_i]$. Obtained solutions was compared to the well-known Earliest Due Date (EDD) constructive algorithm.

Table 1. Percentage relative deviation of parallel coevolutionary algorithm's solutions compared to the solutions of the EDD constructive algorithm

n	number of processors		
	1	2	4
40	-87.46%	-86.80%	-87.74%
50	-87.03%	-86.47%	-85.91%
100	-87.60%	-83.16%	-83.18%

The computational results can be found in Table 1. The number of iterations was counted as a sum of iterations on processors, and was permanently set to 500. For example, 4-processor implementations make 125 iterations on each of the 4 processors, so we can obtain comparable costs of computations. As we can see, parallel versions of the algorithm keep the quality of the obtained solutions, working in a shorter time. Because of small cost of the communication, the speedup parameter of the parallel algorithms is almost linear.

4 Conclusions

We have discussed a new approach to the single scheduling problem based on the parallel asynchronous coevolutionary algorithm. Compared to the sequential algorithm, parallelization shortes computation's time, keeping the quality of the obtained solutions.

References

1. Baker K.R., Scudder G.D.: Sequencing with earliness and tardiness penalties: a review, Operations Research 38 (1990), 22-36.
2. Bożejko W., Grabowski J., Wodecki M.: A block approach for single machine total weighted tardiness problem. Tabu search algorithm. (to appear)
3. Glover F., Laguna M.: Tabu Search, Kluwer, 1997.
4. Holland J.H., Adaptation in natural and artificial systems: Ann Arbor, University of Michigan Press, 1975.
5. Hoogeveen J.A., Van de Velde S.L.: A branch and bound algorithm for single-machine earliness-tardiness scheduling with idle time, INFORMS Journal on Computing 8 (1996), 402-412.
6. Lawler E.L.: A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness, Annals of Discrete Mathematics 1 (1977), 331-342.
7. Lenstra J.J., Rinnoy KanA.H.G., Brucker P.: Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977), 343-362.
8. Smith W.E.:Various Optimizers for Single-Stage Production, Naval Research Logist Quartely 3(1956), 59-66.
9. Wan G., Yen B.P.C.: Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties, European Journal of Operational Research 142 (2002), 271-281.