Computing and Informatics, Vol. 25, 2006, 405–420

USING HEURISTIC SEARCH FOR SOLVING SINGLE MACHINE BATCH PROCESSING PROBLEMS

Thanh-Tung Dang, Baltazár Frankovič, Ivana Budinská

Institute of Informatics
Slovak Academy of Sciences
Dúbravská cesta 9
845 07 Bratislava, Slovakia
e-mail: utrrtung@savba.sk, dang.tung@ul.ie

Ben Flood

Center for Telecomunications and Value-Chain-Driven Research
Trinity College Dublin
Dublin, Ireland

Con Sheahan

Enterprise Research Centre
University of Limerick
Limerick, Ireland

Bao-Lam Dang

Department of Design of Machines and Robots
Hanoi University of Technology
Hanoi, Vietnam

Manuscript received 6 June 2005; revised 12 May 2006
Communicated by Imre J. Rudas

Abstract. This paper deals with scheduling for single machine batch processing, specifically the transporting problem with one vehicle. Capacity restrictions of

the machine are considered and the main objective is to find an assignment of jobs to achieve the minimal processing time of all batches, given these capacities. A polynomial algorithm is proposed for solving the case in which the jobs are non-preemptive, non-identical, and are known before the realization of the schedule. The proposed algorithm is implemented and shown to yield better results than alternatives from the literature.

Keywords: Scheduling, batch-processing, optimization, heuristic search

1 INTRODUCTION

In a batch processing machine sets of jobs are processed simultaneously as batches. The processing time of each batch is the maximum processing time of all jobs included in the batch. Each job has a set of weights, and a batch processing machine will have limited capacity.

One application in which batch-processing machines are used is the transportation problem in manufacturing. An automatic guided vehicle (AGV) has to deliver products or packages of materials or components to a number of workstations in different locations, in the shortest time possible. Workstations are located in a ring or on a straight line. A vehicle is a batch processing machine and a job is the delivery of materials from the store to a workstation. The time it takes to complete a batch is independent of the number of jobs in the batch and here is related only to the furthest location. The vehicles have limited size and can carry a limited weight. As a result, each vehicle is only able to deliver combinations of products or packages with total size and weight not exceeding the capacity of the vehicle. Augustine and Seiden [3], and Karuno and Nagamochi [7] describe examples where a vehicle visits a set of places to execute jobs. The limitations of the vehicle are not considered in these cases, since a vehicle, for example a robot, carries out tasks and does not deliver quantities.

There are many other practical applications which use batch-processing machines, for example, testing materials, electrical circuits, and heat-treating ovens. An interesting example is the kiln machine in ceramic production plants, used to produce a range of industrial components such as beads, bridges, pillars, connector blocks, and others (`http://www.ceramicx.com/`). These components are then used in a variety of different products. All of the components are processed in the kiln machine, but each type of component requires different pressure and temperature. Each component type has different dimensions and weight, and the kiln machine has physical limitations, including limited dimension and a total weight restriction. Hence, the components must be grouped into similar batches, so that the total processing time is minimized.

This paper deals with the scheduling problem for a single machine, in general, that can process jobs in batches, where various restrictions such as limited size of

machine, limited carrying capacity, etc., will be considered. The main objective is to find a schedule for assigning non-identical and non-preemptive jobs to single batches so that the total processing time is minimized.

2 RELATED WORK

If there are n jobs to be processed on m machines, then there are $(n!)^m$ possible schedules, implying the problem is NP hard [8]. Much work has been done in batch processing problems similar to the one presented here. [2] show that the problem can be solved in polynomial time when the sequence of the jobs is predetermined. [5] present a polynomial algorithm to solve the scheduling problem for the special case when the weights of each job are equal.

Many papers discuss problems in which the processing time varies for different batches, and is dependent on the jobs used to form the batches. [14] discuss the scheduling problem for a *flowshop* of batch processing machines. The jobs are assumed to be processed in the same order on each machine. That is, if two jobs i and j are processed in two different batches on machine M_k, and i is processed before j, then job i will always be processed not later than job j on every subsequent machine in the flowshop. Such an assumption reduces the complexity of the scheduling problem, as the calculation only needs focus on the machines whose capacity restrictions are *bottlenecks* of the flowshop. Dynamic programming (DP) is used to minimize the completion time heuristically. DP guarantees that the optimal solution will be found. However, this requires the exploration of all possible schedules. As a result, DP has very high complexity and it can only be applied for small problems. When jobs are non-preemptive, i.e. when jobs are processed, they have to be completed as a whole, job's processing cannot be paused or resumed, and jobs can be processed in any order, then applying DP is infeasible for large sets of jobs.

Other heuristic solutions to the scheduling problem are are genetic algorithms (GA) [6], and simulated annealing (SA) [9]. The quality of solutions achieved by these methods depends strongly on the running time of the algorithms. The crucial step in the algorithms based on the evolutional principle is generating neighboring solutions, which takes a lot of time because of the process of recalculating the plan parameters. In the algorithm presented in [9], two randomly chosen jobs from different batches are exchanged with each other to create a new solution, if the machine capacity will not be violated. One of the selected jobs is the longest processing job in its batch. This is called a mutation process. The mutation process stops when the ratio between the improvement of completion time and a 'temperature' parameter, T, which is regularly decreased by a predefined constant in each cycle, is equal to or larger than a predefined constant.

Another approach to the scheduling problem is constraint programming (CP) [4]. The problem is considered as a set of variables (i.e. starting time of each job, assignment of each job to a batch) and set of constraints (i.e. all batch restrictions). The task is to find an assignment to the variables to satisfy all constraints. The main

principle of these methods based on CP is to combine constraint propagation and backtracking search to find a solution.

[17] review methods for solving batch processing machines scheduling problems based on the Lagrangian relaxation (LR) principle. LR is used to decompose the problem into disjoint sub-problems. The schedules are achieved by iteratively solving those sub-problems exactly, but resulting usually in a sub-optimal total solution. Two methods in LR are backward dynamic programming (BDP) and forward dynamic programming (FDP). [16] use backward dynamic programming to find a solution for the batch processing machine scheduling problem with a single machine, assuming that each batch can process maximally b jobs simultaneously, regardless of their capacity. A similar approach is discussed in [12], considering a stochastic batch service problem in which the inputs and outputs of each batch are mutually dependent.

This paper deals with the scheduling problem for one batch-machine, but with some significant differences, namely that jobs are both non-preemptive and non-identical in size or weight. That is, the batch-machine has limited size and weight-carrying capacity. The objective is to find a schedule which minimizes the sum of completion times of all of the batches.

3 PROBLEM FORMULATION

There are n packages $\{P_i | i = 1, \ldots, n\}$. Each package P_i, is represented by a triple (s_i, w_i, t_i), where s_i and w_i are respectively the size and weight of package P_i, and t_i denotes the delivery time from the store to a workstation. The delivery time includes the time required for loading and unloading the jobs. All workstations are treated with the same priority.

Consider a vehicle with size s and carrying capacity c. The objective is to schedule the jobs the vehicle takes, the batches, so that the total delivery time of all jobs is minimized. The vehicle can take new packages only when it finishes the distribution of all the packages included in its current batch. The processing time of each batch is defined by the longest delivery time among all packages in the batch, as a result of the assumption that workstations are located in a ring or a straight line. Let k be the number of batches, $\{B_i | i = 1, \ldots, k\}$ be the set of packages included in batch i, and T_i be the processing time of batch i. The following equations describe the problem constraints.

$$\forall i \neq j \in [1, k] : B_i \cap B_j = 0 \tag{1}$$

$$\bigcup_{i=1,\ldots,k} B_i = \{P_i |\, i = 1, \ldots, n\} \tag{2}$$

$\forall i \in [1, k]$:

$$\sum_{j | P_j \in B_i} s_j \leq S \tag{3}$$

and

$$\sum_{j|P_j \in B_i} w_j \leq C \tag{4}$$

$$T_i = \max_{j|P_j \in B_i} \{t_j\}. \tag{5}$$

The objective is to find an allocation of packages to single batches with a small value of the total processing time of all batches T,

$$T = \sum_{i=1}^{k} T_i \tag{6}$$

Equations (1) and (2) guarantee that each package is only assigned to one batch. Constraints (3) and (4) ensure that the total size and weight of all packages included in each batch do not exceed the capacity restrictions of the vehicle. Equation (5) specifies the processing time of each batch. Equation (6) expresses the total time of the schedule which should be minimized (also known as the *makespan* of the schedule).

If the purpose is only to minimize the number of batches and not T, the total processing time, then the problem is a *classical minimum bin packing* problem. The *minimum bin packing* problem considers only one parameter, the package size. It is an NP-hard problem, but can be approximated within 1.5, i.e. there is an approximation scheme that guarantees that

$$numBatches_{sol}/numBatches_{opt} \leq 1.5,$$

where $numBatches_{sol}$ and $numBatches_{opt}$ are the numbers of batches achieved by using the approximation scheme, and the optimal number of batches needed to process all jobs, respectively [13]. In problems with low correlation between the parameters of the jobs, independency between size, weight, and processing time, algorithms for solving the minimum bin packing problem would require many modifications in order to solve the problem considered here.

4 SOLVING BY USING HEURISTIC SEARCH

The jobs discussed here are packages to be delivered. All parameters describing the packages and the vehicle are assumed to be known at the beginning of the solving process. Therefore, the optimal solution could be found by exploring all the possible schedules. The given packages are of equal priority, however, and the number of potential solutions is theoretically very large, growing exponentially with the number of packages. Heuristic search algorithms must be used for practically sized problems. The quality of the solution achieved by using heuristic search algorithms depends on the proposed algorithm and the length of time the heuristic algorithm is run for.

Many methods for solving the minimum bin packing problem are based on the principle that packages are sorted according to their size and, subsequently, packages are inserted one by one to each batch, not violating the capacity constraints. This process continues until no packages remain unassigned. In order to minimize the number of batches, methods typically try to insert as many packages to each batch as possible. A solution with minimal amount of batches does not, however, automatically guarantee minimal total processing time, the main objective of this paper.

One way to reduce the total processing time is to group packages with close delivery time together to the same batch. Because capacities of the vehicle, packages with close delivery time are not necessarily added to the same batch. Denote the number of batches created by numBatch. In order to reduce empty space in each batch, it is useful, before creating a new batch, to determine whether the unallocated package can be added to a previously created batch. This is the basis for our proposed algorithm.

4.1 Algorithm 1 – Filling Empty Spaces in Batches (FES)

Phase 1

Sort packages according to their delivery time into $\{P_i | i = 1, \ldots, n\}$. such that $t_i \geq t_{i+1}$, for $i = 1, \ldots, n$.

Phase 2

Let $numBatch = 1$;
For $i = 1, \ldots, n1$.

1. Search for a batch which has enough space for package P_i;
2. If no such batch exists, then create a new batch and add the current package to the new batch and increment numBatch by 1. If a batch with enough space for P_i exists, then go to step 3;
3. Add package P_i to the first empty batch found;

End For

Phase 3

Calculate the total time of the created schedule.

Claim 1. The FES algorithm has complexity $O(n^2)$.

Proof. Phase 1 has complexity $O(n * log(n))$ by using the quick-sort algorithm. In Phase 2, since the number of batches is less than or maximally equal to i, step 1 requires maximally i ($\leq n$) checking operations. Steps 2 and 3 require only one operation. As a result, the cycle of Phase 2 requires maximally n^2 operations. In phase 3, the processing time of each batch is calculated in order to get the total time of the schedule. Since the number of batches is maximally n, then this phase needs

maximally $O(n)$ operations. Hence, the proposed FES algorithm has complexity $O(n^2)$.

Although FES is a polynomial algorithm, in some cases it might be improved by some simple modifications. For example, consider the situation in which the vehicle has limited size $S = 5$, and limited carrying capacity $C = 10$. Packages are given with the following parameters $P_1 = \{2, 3, 10\}$; $P_2 = \{2, 2, 8\}$; $P_3 = \{3, 2, 6\}$; $P_4 = \{3, 2, 6\}$; and $P_5 = \{4, 2, 2\}$. These parameters express size, weight and delivery time of each package, respectively. By applying FES the following plan will be achieved. Plan 1: batch 1: $\{P_1, P_2\}$, batch 2: $\{P_3\}$, batch 3: $\{P_4\}$, and batch 4: $\{P_5\}$. The total time of this plan is $24(= 10 + 6 + 6 + 2)$. However, it is easy to recognize that Plan 2: { batch 1: $\{P_1, P_3\}$, batch 2: $\{P_2, P_4\}$ and batch 3: $\{P_5\}\}$ has better processing time $(20 = 10 + 8 + 2)$.

FES misses the better solution, since after Phase 1, the order of each package is fixed. Packages P_1 and P_2 are scheduled to one batch, forcing the rest of packages to be assigned each to a different batch due to their size. Plan 2 can be identified, if packages $\{P_1, P_2\}$ are scheduled after packages $\{P_3, P_4\}$. This idea can be generalized by the following rule. Before applying FES, some packages with small size (or small weight) are temporarily removed from the calculation process, and after achieving a temporary plan, the remaining packages are sequentially added to a single created batch if there is enough empty space. On this basis, the following modified version of the FES algorithm is proposed. □

4.2 Algorithm 2 – Partial Filling of Empty Spaces in Batches (PFES)

Let $k = 0$.

Phase 1

1. Sort packages according to their size (or weight);
2. Add $(n-k)$ packages with large size (or weight) to one set, called set 1. Add the remaining k packages to the second set, called set 2;
3. Sort the packages in each set according to their delivery time.

Phase 2

1. Apply the FES algorithm to the packages in set 1;
2. Apply the FES algorithm to the packages in set 2.

Phase 3

1. Calculate the total time of the created schedule;
2. If $k < n$, then let $k = k + 1$ and return to Phase 1. If $k = n$, stop.

Claim 2. PFES has complexity $\cong O(n^3)$.

Proof. By using the *quick-sort* algorithm, it can be seen that Phase 1 of PFES has complexity $\cong O(n * \log(n))$. Theorem 1 shows that FES has complexity $\cong O(n^2)$. As a result, Phase 2 requires maximally n^2 operations. Since the number of batches is maximally n, Phase 3 needs maximally n operations. Finally, each cycle of PFES has complexity $\cong O(n^2)$, and there are n such cycles. It follows that PFES has complexity $\cong O(n^3)$.

The FES algorithm is a special case of PFES, where $k = 0$. The PFES algorithm will always achieve at least as good a solution as the FES algorithm. The following section describes simulation results of the proposed algorithms, and compares the performance of the algorithms with other published algorithms. □

5 SIMULATION RESULTS AND COMPARISON

All of the algorithms used here are implemented in Java and have been verified in a number of experiments. The SA algorithm, presented in [9], is selected for comparison. The other method used for comparison is the BDP algorithm, presented in [12]. In the BDP algorithm, the original set of packages is divided into two disjoint sets, and DP is used to find an optimal schedule in each set. The final solution is achieved by combining the partial schedules. If the number of packages is large, BDP can also be recursively applied.

The numerical results of the simulations are shown in Table 1. The first column shows the number of packages. The next columns show the completion times of the schedules achieved by the four algorithms implemented and the running time of each algorithm, measured by milliseconds. The completion time of each schedule is measured by the same units, the delivery time of each package.

The package sizes and weights are generated randomly from a uniform distribution on the interval [1, 30]. The delivery time of each package is generated randomly from a uniform distribution on the interval [1, 50]. The vehicle considered is limited to 50 size units and 80 weight units. The number of packages is in the range 50 up to 104. All experiments are tested on the same PC with a 733 MHz, Pentium 3 processor.

The simulation results show that when the number of packages n is small (up to 100), BDP is realizable for two disjoint sets. However, when the number increases to higher than 103 or more, recursive application of BDP is necessary in order to reduce the running time. The DP algorithm can explore up to *numPackage* different configurations, where *(numPackage)!* is the number of packages included in each disjoint part. Thus each disjoint part is constrained to include maximally 50 packages to make solution time practical.

The SA algorithm theoretically can run for a very long time before the stop condition is matched. In order to compare the quality of solutions with respect to running time, the results of SA and BDP are recorded after FES and PFES have finished. These results are shown in Table 2.

In order to get a complete schedule at any time, BDP is implemented as follows. At the beginning, the original set of packages is divided into a number of small subsets. Each subset involves from 5 to 10 packages on average, depending on the size of the original set. DP is applied to find an optimal schedule for each single subset. The achieved schedules are merged to complete the total schedule. If the time available for running does not expire, these subsets are sequentially merged and the whole process applied again.

	FES		**PFES**		**SA**		**BDP**	
n	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time
50	285	1	285	39	313	136	460	200
100	583	3	581	112	696	133	1 022	283
200	973	9	973	168	1 154	216	1 885	466
300	1 544	18	1 540	181	1 755	548	2 876	978
500	2 634	25	2 626	184	3 017	1 519	5 048	3 096
1 000	5 259	86	5 229	872	5 944	3 929	9 970	11 346
2 000	9 954	96	9 803	4 654	11 240	9 720	19 290	68 529
5 000	25 626	237	25 623	7 299	29 168	12 584	47 900	399 820
10^4	52 941	691	52 741	80 980	59 120	116 050	99 135	2 919 800

Table 1. Experiment results of all selected algorithms

Numerical testing for practical data sets shows that both proposed algorithms achieve significant improvements over SA and BDP, particularly when the number of packages is large. Both the SA and BDP algorithms achieve solutions of similar value to FES and PFES, but after running for a much longer time. This leads to a conclusion that FES and PFES are appropriate algorithms for solving large-scale cases, when the number of packages is large and the time for running is restricted.

The reason that BDP does not achieve solutions of the same quality as FES and PFES is that the original problem is divided into a large number of sub-problems. The solution of each sub-problem may be optimal, but merging the achieved partial solutions may not yield a good total solution. This is because the empty slice left within the created batches is too large. The total empty slice can be reduced when the number of packages included in each sub-problem increases, but it results in exponentially increasing the running time.

BDP could be parallel processed, since it involves solving independent sub-problems. However, parallel processing of BDP does not reduce the complexity of this approach; it only reduces the time of running by solving many independent tasks simultaneously.

The quality of solution achieved by SA depends on the first generated schedule. As the number of cycles increases, the average improvement in the solution per cycle decreases. In the simulations, SA required a lot of time to search before it achieved

	FES	**PFES**	**SA**	**BDP**	
n	Plan compl. time	Plan compl. time	Plan compl. time	Plan compl. time	Running Time (ms)
50	285	285	345	489	39
100	583	581	756	1 422	117
200	973	973	1 357	1 785	168
300	1 544	1 540	1 965	3 176	181
500	2 634	2 626	3 214	5 348	184
1 000	5 259	5 229	6 440	10 950	872
2 000	9 954	9 803	12 238	22 290	4 654
5 000	25 626	25 623	31 062	50 404	7 299
10^4	52 941	52 741	63 282	11 239	80 980

Table 2. Experiment results of all selected algorithms after the same time of running

solutions as good as those found by FES or PFES. On the other hand, SA is an anytime algorithm, able to provide a solution whenever the program stops. For this reason, SA is suitable for instances in which the time allotted to find a schedule is not fixed in advance.

When the number of packages is large, more than 1 000, the simulation results (see Tables 3, 4 and 5) show that PFES does not achieve significantly better results than FES. PFES achieved solution with about 0.1 % better quality. This improvement is not much, given the amount of extra time required to run PFES. In general, PFES takes much more time than FES, about 10–20 times more, to complete its search. For illustration, both algorithms were tested with 5000 packages and the quality of the achieved solutions is compared and shown in Figure 1. Figure 2 shows a comparison of the running time of these algorithms. More numerical results of experiments with large number of packages are shown in Tables 3, 4 and 5. Applying FES seems to be the most suitable approach to use when solving cases with large numbers of packages

	FES		**PFES**		**SA**		**BDP**	
n	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time
1 000	5 259	86	5 229	872	5 944	3 929	9 970	11 346
1 000	5 296	35	5 289	970	6 084	4 206	10 265	11 427
1 000	5 100	28	5 078	917	5 709	4 134	10 097	10 838
1 000	5 010	94	4 990	807	5 692	3 819	9 648	10 635

Table 3. Simulation results with 1 000 packages

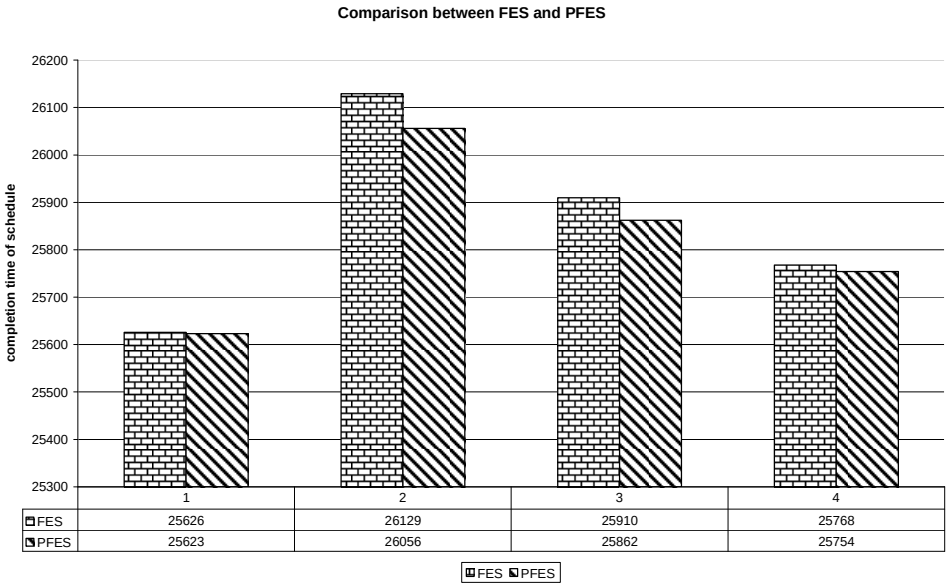


Fig. 1. Comparison of plan completion time between FES and PFES with 5 000 packages

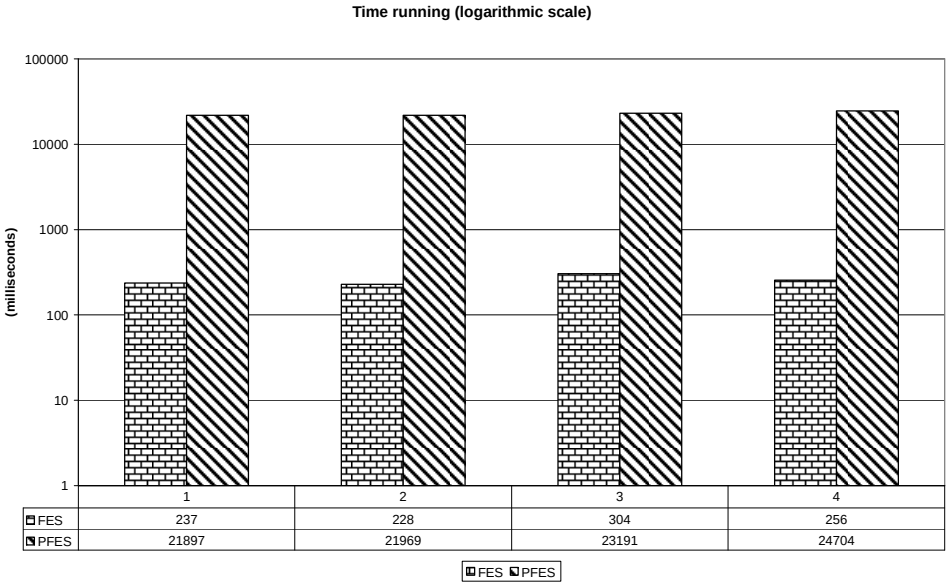


Fig. 2. Comparison of the running time between FES and PFES with 5 000 packages

	FES		PFES		SA		BDP	
n	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time
2 000	9 954	96	9 803	4 654	19 290	9 720	11 240	68 529
2 000	10 412	77	10 361	4 994	19 480	10 144	11 563	56 806
2 000	10 355	57	10 299	4 985	19 642	8 723	11 446	54 158
2 000	10 329	83	19 397	5 913	19 663	11 076	11 633	62 643

Table 4. Simulation results with 2 000 packages

	FES		PFES		SA		BDP	
n	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time	Plan compl. time	Running Time
5 000	25 626	236	25 613	7 299	29 168	12 584	47 900	399 820
5 000	26 129	228	28 956	7 323	29 639	12 272	49 227	399 636
5 000	25 910	304	25 862	7 730	29 641	11 606	48 374	432 436
5 000	25 768	256	25 754	8 235	29 898	11 492	48 509	417 261

Table 5. Simulation results with 5 000 packages

6 DISCUSSION AND FUTURE WORK

A very important area in industry where batch processing machines are used is in ceramics manufacturing (`http://www.ceramicx.com`). Many different components are processed by using the same machine, for example the kiln machine, as mentioned in Section 1. Another typical example of using batch processing machine is for the distribution of milk or fresh food. Small vehicles are usually used to distribute milk or fresh food in order to save fuel consumption.

Discussions with practitioners of some of these companies have highlighted important facts that should be incorporated into the solution. Food distribution problems differ from the AGV case studied in this paper in some important ways. The first difference is that the time to travel between places is not constant. The time variable ti associated with each package Pi is a continuous variable with some, often predictable, distribution. The distribution depends strongly on both the routes taken between the producer and the customers and on the time of day the route is traveled. The second major difference in food distribution is that the loading and unloading time is variable, and, there is a set-up time between batches that is needed for preparing packages and loading them into vehicles. These variables are to be incorporated into our future research. Unlike the AGV case, in the case of food distribution, orders are usually repeated periodically. As a result, many scheduling problems are almost perfectly repeated. The periods of the orders generally depend on some important factors like time, unit price of products, and some external events, for example, changes of delivery conditions, fuel cost, etc. Identification of

these periods allows manufacturers to predict the short-term future demands, and helps manufacturers to create robust production plans. Using successful schedules for similar previous cases, and including only minor modifications, can save considerable time when generating a schedule. Case-based reasoning (CBR, [1]), is suited to these types of problems.

Some useful comments about the kiln machine in the ceramics company were provided, due to particular characteristics of that machine. Setting-up in the kiln machine takes a considerable time, as a result of that the kiln machine has to be warmed up to specific temperature. Cooling down the temperature of the kiln machine also takes a certain time. Additional factors related to job's execution like input temperature and pressure must be considered in the scheduling solution. Batches consisting of jobs that will be processed at the same temperature and pressure should be processed together to save time setting up and unloading the kiln machine.

In some circumstances, in order to accelerate distribution services vehicles with small capacity can be used. However, fuel consumption increases considerably because vehicles may have to travel longer routes. Choosing appropriate vehicle capacity to save the total fuel costs. This problem will be incorporated into our future research.

7 SUMMARIZATION

This paper deals with the scheduling problem for single batch-processing machine, where the machine capacity restrictions are considered. Two algorithms have been presented for solving instances in which jobs (packages for transporting) are non-preemptive and not identical in size or weight. Both the proposed algorithms are polynomial and applicable for solving large-scale problems. Experimental results show the significant improvements of the proposed algorithms over some other published ones. Some simplifications are assumed. For example, all jobs are available at the time zero and their deadline is still omitted. Considering these constraints in the scheduling problem is part of the objective of our future research. The next step of our research is to schedule the batches with minimal completion time, assuming that each job is associated with specific deadline.

Acknowledgement

This work has been partially funded by Slovak national agencies Vega, under project No. 2/4148/25 and APVV under contract No. 51-024604 and 51-011602. It has also been partially supported by the Irish Research Council for Science, Engineering and Technology (IRCSET).

REFERENCES

[1] Aamodt, A.—Plaza, E.: Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. Artificial Intelligence Communications, IOS Press, Vol. 7, 1994, No. 1, pp. 39–59.

[2] Albers, S.— Brucker, P.: The Complexity of One-Machine Batching Problems. Discrete Applied Mathematics Vol. 47, 1993, pp. 87–107.

[3] Augustine, J. E.—Seiden, S. S.: Linear Time Approximation Schemes for Vehicle Scheduling. Theoretical Computer Science, Vol. 324, 2004, Vol. 2–3, pp. 147–160.

[4] Baptiste, P.—LePape, C.— Nuijten, W.: Constraint-Based Scheduling. Kluwer Academic Publishers, Boston Hardbound, ISBN 0-7923-7408-8, 2001.

[5] Coffman, Jr. E. G.—Yannakakis, M.— Magazine, M. J.—Santos, C.: Batch Sizing and Sequencing on a Single Machine. Annals of operation research, Vol. 26, 1990, pp. 135–147.

[6] Cheraghi, S. H.—Vishwaram, V.—Krishnan, K. K.: Scheduling a Single Batch-Processing Machine with Disagreeable Ready Times and Due Dates. International Journal of Industrial Engineering, Vol. 10, 2003, No. 2.

[7] Karuno, Y.—Nagamochi, H.: 2-Approximation Algorithms for the Multi-Vehicle Scheduling Problem on a Path with Release and Handling Times. Discrete Applied Mathematics, Vol. 129, 2003, No. 2–3, pp. 433–447.

[8] Kasin, O.—Mason, S. J.: Scheduling Batch Processing Machines in Complex Job Shops. Proceedings of the 2001 Winter Simulation Conference, WSC '01.

[9] Melouk, S.—Damodaran, P.—Chang, P. Y.: Minimizing Makespan for Single Machine Batch Processing with Non-Identical Job Sizes Using Simulated Annealing. Int. Journal of Production Economics, Vol. 87, 2004, pp. 141–147.

[10] Soner, H. M.—Touzi, N.: Stochastic Target Problems, Dynamic Programming, and Viscosity Solutions. SIAM Journal on Control and Optimization, Vol. 41, 2002, No. 2, pp. 404–424. Industrial Engineering Applications and Practice: User's Encyclopedia. In CD, ISBN: 0-9654599-0-X.

[11] Papadaki, K.—Powel, W. B.: Exploiting Structure in Adaptive Dynamic Programming Algorithms for a Stochastic Batch Service Problem. European Journal of Operation Research, Vol. 142, 2002, No. 1, pp. 108–127.

[12] Simchi-Levi, D.: New Worst Case Results for the Bin Packing Problem. Naval Res. Logistics 41, 1994, pp. 579–585.

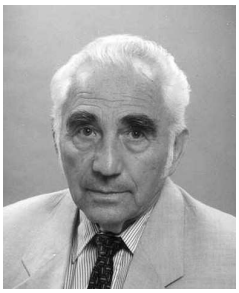
[13] Sung, C. S.—Choung, Y. I.: Minimizing Makespan on a Single Burn-In Oven in Semiconductor Manufacturing. European Journal of Production Research, Vol. 120, 2000, pp. 559–574.

[14] Sutton, R.—Barto, A.: Reinforcement Learning. The MIT Press, Cambridge, Massachusetts, 1998.

[15] Wagelmans, A. P. M.—Gerodimos, A. E. (2000): Improved Dynamic Programs for Some Batching Problems Involving the Maximum Lateness Criterion. Operations Research Letters 27, pp. 109–118.

[16] Wang, J.—Luh, P. B.—Zhao, X.—Wang, J.: An Optimization-Based Algorithm for Job-Shop Scheduling. A Journal of Indian Academy of Sciences, A Special Issue on Competitive Manufacturing Systems, Vol. 22, 1997, No. 2, pp. 241–256.

Thanh-Tung Dang Ing. (M. Sc.) 1997; Ph. D. 2003 in Institute of Informatics, SAS. His research interests include MAS, planning and scheduling, reasoning and knowledge management. His current project is using MAS for intelligent manufacturing and enterprise modeling. He is currently employed in a postdoctoral position in the Enterprise Research Centre, based in the University of Limerick.



Baltazár Frankovič is a member of IFAC-IFIP TC Neumann Society Budapest. He is a member of the Central European Academy of Sciences and Arts. He acts as a member of the Editorial Board of several journals, e.g. Computing and Informatics. His current research interests include modeling and simulation of FMS, adaptive and learning control in DEDS, MAS.

Ivana Budinská graduated from Slovak Technical University, Faculty of Electrical Engineering in 1987, and received the Ph. D. degree from Institute of Informatics, SAS, in 2003. Her research interests are in DEDS, control theory and MAS.

Ben Flood B. Sc. 2002 in applied mathematics and computing from the University of Limerick. Ph. D. in statistics 2006. His research interests are in statistical decision theory in moderately complex systems. He is currently employed in a postdoctoral position in the Centre for Telecommunications Value-Chain Research (CTVR) based in Trinity College Dublin, funded by Science Foundation Ireland (SFI).

Con Sheahan received his Ph. D. degree in 1996. His research interests are in the domain of manufacturing resource optimization, including operations research, discrete mathematics and complexity theory.

Bao-Lam Dang received a M. Sc. degree on automation in 1998 at Slovak University of Technology, Bratislava, Slovakia and a M. Sc. degree on mechanics of machines in 2004 at Hanoi University of Technology, Viet Nam. His research interests are in kinematics, dynamics, control and simulation of robotic systems.