# A POLYNOMIAL ALGORITHM FOR SINGLE MACHINE BATCH PROCESSING

Dang Thanh-Tung, Baltázar Frankovič, Ivana Budinská

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, Bratislava 84507
Slovakia
Email: utrrtung@savba.sk

*Abstract*: **this paper deals with the scheduling problem for single machine batch processing, concretely the transporting problem with one vehicle. All capacity restrictions of the machine are considered and the main objective is to find such an assignment of jobs in order to achieve the minimal processing time of all batches. A polynomial algorithm is proposed for solving cases, in which the jobs are non-preemptive, non-identical, and available before the realization of the schedule. The proposed algorithm is implemented and show better results in comparison with some other published ones.**

## 1. INTRODUCTION

A batch processing machine is a one in which a set of jobs are processed simultaneously as a batch. The processing time of a batch is equal to the longest processing time among all the jobs included in the batch. Each job has a certain size or further associated parameters. Similarly, a machine has also a limited capacity, speed of processing or so on. One of the applications, in which batch-processing machines are used, is the transport system. In the framework of the transport problem, a vehicle has often to deliver products or packages to a number of customers settled in different locations. Let us consider only distances between the distributor and customer's location as an important factor that influences on the realization time. The total time until the vehicle finishes the distribution depends on the furthest location. On the other hand, a vehicle has usually a limited size and carrying capacity. As a result, each vehicle is able to carry only certain products or packages, whose size and weight altogether do not exceed the capacity restrictions. There are also many other practical applications which use batch-processing machines, e.g. testing process of material, electrical circuits, heat-treating evens, etc.

This paper deals with the scheduling problem for a single machine that can process jobs in batches, where various restrictions like limited size of machine, limited carrying capacity, etc., will be considered. The main objective is to find such a schedule for assigning non-identical and non-preemptive jobs to single batches so that the total processing time is minimized.

## 2. RELATED WORK

Many papers dealt with the batch-processing problem as the one presented in this paper. In general, this problem is NP-hard, however [1] proved that the problem is polynomially solvable when the job's sequence is predetermined. Similarly, [3] presented a polynomial algorithm to solve the mentioned problem for a special case, when the job's weight (or size) is equal.

Many papers dealt with varying batch-processing time problem, in which the processing time is dependent on the batches formed. [10] dealt with the scheduling problem for a flowshop of batch-processing machines. The important point in this work is that the jobs are assumed to be processed in the same order in each machine, i.e. any two jobs $i$ and $j$ are processed in two different batches on machine $M_k$, job $i$ is processed before job $j$, then job $i$ will always be processed not later than job $j$ on every other machine. Such an assumption makes the problem solving to be simpler, and the calculation process will focus only on some specific machines, whose capacity restrictions are *bottleneck* of the flowshop. To minimize the completion time, the authors proposed a heuristic algorithm based on dynamic programming (DP). DP is able to guarantee finding the optimal solution; however, it requires exploring all possible configurations. As a result, DP has very high complexity and it could only be applied for instances with small number of machines and jobs. When jobs are non-preemptive and they could be processed in whichever order, then applying DP is unrealizable for large sets of jobs.

Some other authors used genetic algorithm (GA) e.g., [4] or simulated annealing (SA), e.g., [6] to resolve the scheduling problem. The quality of solutions achieved by these methods depends strongly on the time running of these algorithms. The crucial step in those algorithms based on the evolutional principle is generating neighboring solutions, which takes a lot of time because of a process of recalculating plan's parameters. In the algorithm presented by [6], any two jobs from two different batches are exchanged one with another to create a new solution, if the machine's capacity will not be violated. One of the selected jobs is the longest processing job in its batch. The mutation process stops when the ratio between the improvement of completion time and the temperature T, which is regularly decreased by a predefined

constant in each cycle, is equal or larger than the predefined constant.

Another possible way for solving is to use constraint programming (CP) techniques e.g. [2], [5]. The given 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. Since the deadline of jobs is still not considered in this paper and all jobs are available at the time zero, so the precedence among jobs is not detectable. That reason leads to the fact that applying constraint programming is not very practical in this case.

[2] presents a nice review about methods for solving batch-machines scheduling problems based on the Lagrangian relaxation (LR) principle. LR is used to decompose the problem into individual part subproblems with intuitive appeal. The sub-optimal schedules are achieved by iteratively solving those subproblems. Known methods built in the LR are, namely, backward dynamic programming (BDP) and forward dynamic programming (FDP) algorithms, e.g. [9].

This paper deals with the scheduling problem for one batch-machine, but with some significant differences, namely, given jobs are non-preemptive, non-identical size or weight. The batch-machine (concretely it is the vehicle for transporting) has limited size and carrying capacity. The main objective is to find such a schedule which minimizes the sum of completion time of all batches.

## 3. PROBLEM FORMULATION

The problem solving that this paper deals with could be described as follows.

Given $n$ packages $\{P_i\}|_{i=1,\ldots,n}$. Each package $P_i$ is represented by a triple $\{s_i, w_i, t_i\}$, where $s_i$ is size (measured by $m^3$), $w_i$ is weight of package $P_i$ (measured by *kg*), respectively; $t_i$ denotes the time (measured by *minute*) delivery from the producer to the customer, whom this package is assigned for, by using the given vehicle. All customers have the same priority and therefore no package is preferred over another.

Given a vehicle with limited size $S$ and limited carrying capacity $C$. the main objective is to schedule these batches so that the total delivery time is minimized. The vehicle can take new packages, if it finishes distribution of all the packages included in the current batch. The processing time of each batch is defined by the largest time delivery among all packages included in it.

All the above constraints could be described by following equations.

Let $k$ be the number of batches; $\{B_i|_{i=1,..,k}\}$ be a set of packages included in batch $i$ and $T_i$ be a processing time of batch $i$.

$$\forall\, i \neq j \in [1,k]: B_i \cap B_j = \varnothing \qquad (1)$$

$$\bigcup_{i=1,..,k} B_i = \{P_i\}|_{i=1,..,n} \qquad (2)$$

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

$$\sum_{j|P_j \in B_i} s_j \leq S \qquad (3)$$

and

$$\sum_{j|P_j \in B_i} w_j \leq C \qquad (4)$$

$$T_i = \max_{j|P_j \in B_i} \{t_j\} \qquad (5)$$

The main objective is to find such an allocation of packages to single batches in order to minimize the following function:

$$T = \sum_{i=1}^{k} T_i \qquad (6)$$

where $T$ is the total processing time of all batches.

Equations (1) and (2) guarantee that each package could be assigned only 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 vehicle's capacity restrictions. Equation (5) specifies the processing time of each batch; and equation (6) expresses the total time of the schedule, which should be minimized.

## 4. SOLVING BY USING HEURISTIC SEARCH

Due to the fact that all parameters of packages and the vehicle are given at the beginning of the solving process, the optimal solution could be achieved by exploring all the possible schedules. On the other hand, the given packages are equally-priority; the number of potential solutions is theoretically very large and it grows exponentially with the number of packages. That fact leads to a conclusion that when the number of packages is large, using some heuristic search algorithms, which have realizable complexity, is more practical than exploring all space of candidate solutions in order to find the optimal one. The quality of the achieved solution by using heuristic search algorithms depends on the proposed algorithm and it could be verified by applying the proposed algorithms in real applications.

Concerning the definition of the processing time of a batch, one of possible ways to minimize the total processing time is to group packages with close delivery time together to the same batch. Because of the vehicle limited capacities, packages with close delivery time might not be added to the same batch. In order to reduce free space in each batch, before creating next batch, it is useful to verify the possibility of adding the unallocated package to one of the already created batches. If such a possibility exists, this package is added to the batch that has enough free space. On the basis of this discussion, the following algorithm is proposed.

numBatch = number of created batches,

*Algorithm 1 – Filling Empty Space in batches (FES)*
Phase 1: sort packages according to their delivery time → $\{P_i|_{i=1,\ldots,n}\}$ where $\forall i \in [1,n\text{-}1]\ t_i \geq t_{i+1}$.
Phase 2: numBatch = 1
for $i$ =1 to $n$

1. list all already created batches and search for the one, which has enough space for package $P_i$.
2. if a batch with sufficient empty space for $P_i$ does not exist, then create next batch and add the current package to the new batch. numBatch = numBatch +1. Otherwise → step 3.
3. add package $P_i$ to the first found free batch.
4. $i$=$i$+1.

Phase 3: calculate the total time of the created schedule.

*Theorem 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 $i$, therefore step 1 requires maximally $i$ (≤n) checking operations. Step 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.

The above explanation leads to conclusion that the proposed FES algorithm has complexity ≅ $O(n^2)$.

Although FES is a polynomial algorithm, however in some cases it might be improved by some simple modifications. In order to better understand, an illustration example is shown.

*Example: Let us consider a situation when the vehicle has limited size S = 5; limited carrying capacity C = 10. Packages are given with following parameters* $P_1$= {2,3,100}; $P_2$ = {2,2,80}; $P_3$ = {3,2,60}; $P_4$ = {3,2,60}; *and* $P_5$ = {4,2,20}. *These parameters express size* ($m^3$), *weight (kg) and delivery time (min) 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 240 min (=100+60+60+20).

On the other hand, 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 (200 min = 100+80+20).

FES misses the better solution, since after phase 1, the order of each package is fixed; packages $P_1$ and $P_2$ are scheduled as first to one batch, and it forces the rest of packages to be assigned each to one different batch due to their size. Plan 2 could be achieved, if packages $\{P_1, P_2\}$ are scheduled after $\{P_3, P_4\}$. This idea could be generalized by a rule: before applying FES, some packages with small size (or small weight) are temporarily removed from the calculation process; after achieving a temporary plan, the left packages are sequentially added to single created batch, if it has enough free space. On the basis of this discussion, the following algorithm is proposed, which is modified from the FES one.

*Algorithm 2: Partial Filling Empty Space in batches (PFES)*
$k$ = 0
Phase 1:

1. sort packages according to their size (or weight),
2. ($n$–$k$) packages with larger size (or weight) are added to one set (called set 1), the rest ones ($k$ packages) to the second set (called set 2),
3. sort packages in each set according to their delivery time.

Phase 2:

1. apply FES for packages within set 1,
2. apply FES for packages within set 2.

Phase 3: calculate the total time of the created schedule.

- If $k$<$n$, then $k$=$k$+1 → return to phase 1. Otherwise, stop.

*Theorem 2*: PFES has complexity ≅ $O(n^3)$.
*Proof*: by using the *quick-sort* algorithm, it is easy to see that phase 1 of PFES has complexity ≅ $O(n*log(n))$. Theorem 1 shows that FES has complexity ≅ $O(n^2)$, 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 ≅ $O(n^2)$, and there are $n$ such cycles, it follows that PFES has complexity ≅ $O(n^3)$.

In general, FES is a special case of PFES when $k$=0, and therefore PFES will always achieve better solution than FES.

The following section describes simulation results of the proposed algorithms in real situations and their comparison with other published algorithms for solving the given problem.

## 5. SIMULATION RESULTS AND COMPARISON

Both the proposed algorithms and other selected ones for comparison have been implemented in Java and verified in a number of experiments. The SA algorithm presented by [6] is selected for comparison. Another selected method is BDP [9], where the original set of packages is divided into two disjoint sets and DP is used to find an optimal schedule in each part. The final solution is achieved by combining the partial schedules. If the number of packages is large, BDP could also be recursively applied in each disjoint part of packages. The numerical results of simulations are shown in Table 1. In this table, the first column shows the number of packages; in next columns, there are the

completion times of the schedules achieved by each corresponding algorithm and the time of running of each algorithm.

In these experiments, parameters of each package are generated randomly with the assumed measures. The number of packages is chosen from 50 up to $10^4$. All experiments are tested in the same PC with processor Pentium 3, 733 MHz. 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 $10^3$ or more, recursively applying BDP is necessary in order to reduce the time of running. Concerning the fact that DP might explore up to (*num_package*)! different configurations, where *num_package* is the number of packages included in each disjoint part, each disjoint part should include maximally 50 packages to make problem being solvable by using DP. SA theoretically runs very long, until the stop condition is matched. In order to compare the quality of solutions from the time point of view, results of SA and BDP are recorded after when 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 to a number of small size subsets. Each subset involves in average from 5 up to 10 packages, depending on the size of the original set. DP is applied to find an optimal schedule for single subset. The achieved schedules will be merged to complete the plan of batches. If the time available for running does not expire, these subsets are sequentially merged and the whole process as described above is reapplied again.

Numerical testing for practical data sets shows that both the proposed algorithms achieve significantly improvements in comparison with SA and BDP, particularly when the number of packages is large. Both algorithms SA and BDP achieve solutions in the same level of quality as FES and PFES, but after very long time of running. That fact leads to 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 main reason why BDP does not achieve quality solutions as FES and PFES is that the original problem solving is divided to a large number of subproblems. The solution of each subproblem might be optimal, but only for this subset. Merging the achieved partial solutions does not bring a quality complete plan, since the total empty place left within the created batches is too much. The total empty place could be reduced when the number of packages included in each subproblem increases, but it results on exponentially increasing the time of running.

On the other hand, it might be useful to mention that, BDP could be an ideal case for parallel processing, since solving partial subproblems is independent one from another. 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 firstly generated schedule. After certain cycles, a quality of achieved solution does not change as drastically as at the beginning. In general, SA requires some time for searching, until it reaches such a quality solution like the one achieved by applying FES or PFES. On the other hand, SA is an anytime algorithm, which could provide solutions at any time, when the program is stopped. For that reason, SA is suitable for instance, in which the time for solving is not fixed and the program could whenever be interrupted.

Table 1: experiment results of all algorithms

| packages | FES | | PFES | | SA | | BDP | |
|---|---|---|---|---|---|---|---|---|
| $n$ | Completion time of plan (min) | Time running (ms) | Completion time of plan (min) | Time running (ms) | Completion time of plan (min) | Time running (ms) | Completion time of plan (min) | Time running (ms) |
| 50 | 285 | 1 | 285 | 39 | 313 | 136 | 460 | 200 |
| 100 | 583 | 3 | 581 | 112 | 696 | 133 | 1022 | 283 |
| 200 | 973 | 9 | 973 | 168 | 1154 | 216 | 1885 | 466 |
| 300 | 1544 | 18 | 1540 | 181 | 1755 | 548 | 2876 | 978 |
| 500 | 2634 | 25 | 2626 | 184 | 3017 | 1519 | 5048 | 3096 |
| 1000 | 5259 | 86 | 5229 | 872 | 5944 | 3929 | 9970 | 11346 |
| 2000 | 9954 | 96 | 9803 | 4654 | 11240 | 9720 | 19290 | 68529 |
| 5000 | 25626 | 237 | 25623 | 7299 | 29168 | 12584 | 47900 | 399820 |
| $10^4$ | 52941 | 691 | 52741 | 80980 | 59120 | 116050 | 99135 | 2919800 |

Table 2: experiment results of all algorithms after the same time of running

| packages | FES | PFES | SA | BDP | |
|---|---|---|---|---|---|
| $n$ | Completion time of plan (min) | Completion time of plan (min) | Completion time of plan (min) | Completion time of plan (min) | Time running (ms) |
| 50 | 285 | 285 | 345 | 489 | 39 |
| 100 | 583 | 581 | 756 | 1422 | 112 |
| 200 | 973 | 973 | 1357 | 1785 | 168 |
| 300 | 1544 | 1540 | 1965 | 3176 | 181 |
| 500 | 2634 | 2626 | 3214 | 5348 | 184 |
| 1000 | 5259 | 5229 | 6440 | 10950 | 872 |
| 2000 | 9954 | 9803 | 12238 | 22290 | 4654 |
| 5000 | 25626 | 25623 | 31062 | 50404 | 7299 |
| $10^4$ | 52941 | 52741 | 63282 | 11239 | 80980 |

When the number of packages is large (more than 1000), simulation results show that PFES does not achieve significantly better results than FES (i.e. 0.1%, it is not such a high improvement; simulated results with 5000 packages are shown in Figure 1). Moreover, PFES takes much more time than FES (about 10 – 20 times more; comparison of the time running of these algorithms is shown in Figure 2). For that reason, applying FES seems as the most optimal approach for solving the mentioned problem in cases with large number of packages.

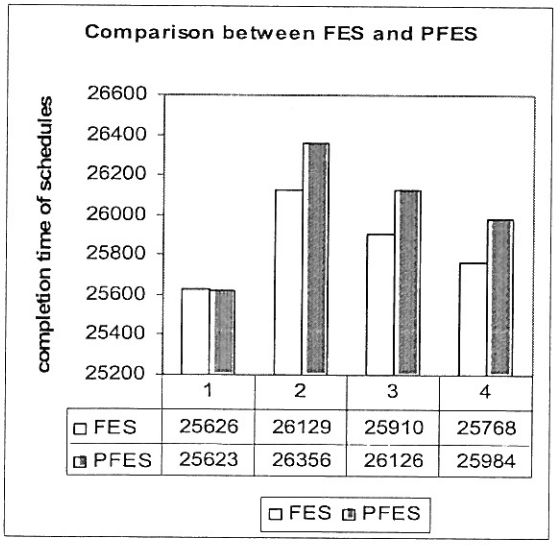


Figure 1: comparison plan completion time (measured by *minute*) between FES and PFES with 5000 packages

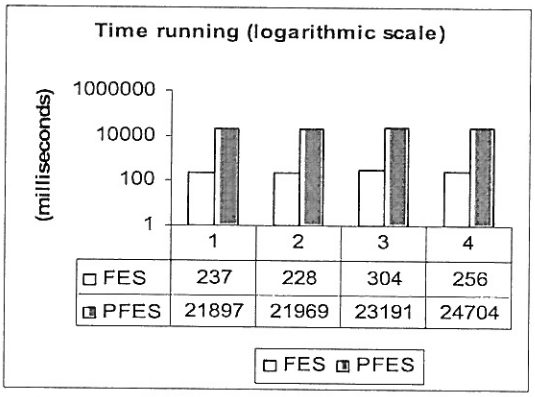


Figure 2: comparison the time running between FES and PFES with 5000 packages

## 6. CONCLUSIONS

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 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. However, in this paper, some simplifications are assumed, namely, all the jobs are available at the time zero and their deadline is still omitted. Considering these constraints in the scheduling problem could be the objective of our future research. The goal of next research is to schedule the batches with minimal completion time, in assumptions that each job is associated with specific deadline or it's arriving is not specified in advance.

## 7. ACKNOWLEDGMENT

This paper is partially supported by APVT and VEGA grant agencies under grants No APVT 51 011602 and VEGA 2/1101/21.

## REFERENCES

1. Albers S. and Brucker P. (1993): The complexity of one-machine batching problems. *Discrete Applied mathematics* Vol. 47, p. 87 – 107.
2. Baptiste P., LePape C., and Nuijten W. (2001): Constraint-Based Scheduling, *Kluwer Academic Publishers, Boston Hardbound,* ISBN 0-7923-7408-8.
3. Coffman Jr. E.G., Yannakakis M., Magazine M.J. and Santos C. (1990): Batch sizing and sequencing on a single machine. *Annals of operation research* Vol. 26, p. 135 -147.
4. Hossien Cheraghi S., Vishwaram Vishwaram, and Krishna K. Krishnan (2003): Scheduling a single batch-processing machine with disagreeable ready times and due dates, *International Journal of Industrial Engineering, Vol.* 10, *Issue* 2.
5. Lavelle D. and Sheahan C. (2002): Scheduling a Multi-Stage Batch Manufacturing Process Via Constraint Logic Programming. *International Conference on Responsive Manufacturing (ICRM) 26-28th June, 2002, University of Gaziantep, Turkey.*
6. Melouk S., Damodaran P. and Chang P. Y. (2004): Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *Int. Journal of Production Economics*, 87, p. 141 – 147.
7. Mete Soner H. and Nizar Touzi (2002): Stochastic Target Problems, Dynamic Programming, and Viscosity Solutions. *SIAM Journal on Control and Optimization, Vol.* 41, *Num.* 2, p. 404-424.
8. Industrial Engineering Applications and Practice: User's Encyclopedia. In CD, ISBN: 0-9654599-0-X
9. Papadaki K. and Powel W. B. (2002): A monotone adaptive dynamic programming algorithm for a stochastic batch service problem. *European Journal of Operation Research*, 142 (1), p. 108-127.
10. Sung C. S. and Choung Y. I. (2000): Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Production Research,* 120, p. 559-574.
11. Sutton R. and Barto A (1998): Reinforcement learning. *The MIT press, Cambridge, Massachusetts.*
12. Wang J., Luh P. B., Zhao X. and Wang J. (1997): An Optimization-based algorithm for job-shop scheduling. *A Journal of Indian Academy of Sciences, a Special Issue on Competitive Manufacturing Systems*, Vol. 22, Part 2, p. 241-256.