Acta Agraria Kaposváriensis (2006) Vol 10 No 1, 185-194
Kaposvári Egyetem, Állattudományi Kar, Kaposvár
University of Kaposvár, Faculty of Animal Science, Kaposvár

Adaptibility and limitations of the enumerating method

Á. Buza
University of Miskolc, Department of Information Engineering, H-3515 Miskolc, Egyetemváros

ABSTRACT

Nowadays there are a lot of scheduling algorithms which are able to solve different scheduling problems. But the efficiency of these methods is doubtful, because the methods are unable to answer the question, how far is the solution from the optimal solution in the case of Makespan. Only the enumeration can answer the above mentioned question. Enumeration is an NP-hard task, but we can improve its efficiency through parallel processing.
(Keywords: scheduling, heuristics, enumeration)

ÖSSZEFOGLALÁS

A leszámláló algoritmus alkalmazhatósága és korlátai

Buza Á.
Miskolci Egyetem, Alkalmazott Informatikai Tanszék, 3515 Miskolc, Egyetemváros

Jelenleg már számos ütemező algoritmus áll rendelkezésünkre ahhoz, hogy a termelésütemezési feladatokat megoldjuk. Arra a kérdésre azonban, hogy az egyes algoritmusokkal kapott eredmények mennyire közelítik meg az optimális ütemterv jósági indexének értékét az átfutási idő vonatkozásában, egyik sem tud érdemi választ adni – néhány kivételesen egyszerű esettől eltekintve. Kizárólag a leszámlálással nyílik mód arra, hogy a fenti kérdésre érdemi választ adjunk. Az univerzális leszámláló algoritmus a gyakorlatban – annak hosszú futási ideje miatt – kevésbé elterjedt. Az említett futási idő redukálására számos lehetőség kínálkozik.
(Kulcsszavak: ütemezés, heurisztikák, leszámlálás)

INTRODUCTION

Typical scheduling tasks

The scheduling as manufacturing task, arises in every manufacturing plant, where different jobs appear and use the same resources. A job means the completing of finite operations, which appear one after another. The manufacturing control has to allocate the machines to different jobs and calculate the starting time of the different operations on different jobs. There could exist complicated constrains, which determine the non-allowed orders. The mutual contact of the tasks and resources is described by intricate relations and these are stored in the production plans. A scheduling task could be formulated as a discreet optimum problem and in the most cases exact solution does not exist.

A scheduling task could be given generally in the following way: There are *n* pieces of job, which must be completed in a manufacturing plant and there are *m* pieces of machine, which are available to finish the jobs. An executive order must be determined, which satisfies the conditions and has an optimum result from the previously given technological and economical aspect.

If the marks are the following:

- **J** the jobs, which have to be completed (J={$J_1,\ldots,J_j,\ldots,J_n$}, $n \in$N),
- **M** the available resources (M={$M_1,\ldots,M_j,\ldots,M_m$}, $m \in$N),

the $O_{i,k}(j)$ means k^{th} operation of the i^{th} job, which has to be completed on the j^{th} resource. In the most scheduling tasks *i*, *j* and *k* have an overriding importance, so it is a key-factor to understand their meaning.

There are different task-types, which differ from each other in the number of possible solutions, the length of the solution time and the formulation of the solution method. The following classes can be distinguished by the formulation of the tasks: (*Erdélyi* 1999)

- Job-shop task:
  In the job-shop task *n* jobs have to be completed on *m* machines. Every job has to be completed on every machine. The length of the $O_{i,k}(j)$ operation equals to p_{ji} (can be 0). The sequences of the jobs on the different machines are given and can be different at different jobs.
- Open-shop task:
  In the open-shop task n jobs have to be completed on the machines. The operations of the differrent jobs have to be done on different machines and every job must be completed on every machine one time (more than once is prohibited). The sequence of the operations belonging to one job is optional, but different operations of one job can not be processed simultaneously.
- Flow-shop task:
  In this task n jobs have to be worked on m machines, but the sequences of the different jobs are identical. This sequence has to be determined at the beginning.
  In the flow-shop task two different possibilities can be distinguished:
  - the sequence of the jobs at the different machines has to correspond at each machines;
  - or the operations of the different jobs can overtake each other in each machine-buffer.

NUMBER OF THE POSSIBLE SOLUTIONS

The scheduling tasks have finite but numerous solutions in case of finite *n* and *m*. The mathematical models which are able to describe the complete task belonging to the circle of combinatorial optimization and they are the typical NP hard problems.

The number of the possible solutions depends on the given mathematical models and the type of the scheduling task. The solution space of the previously mentioned models is huge. Among them the flow-shop task includes the fewest possible solutions. In that case the number of the different possible schedules is $n!$. The solution space of the other models is much higher and the following refer only to the flow-shop tasks (non-overtaking case).

METHODS FOR SOLUTION

Lot of methods are able to solve scheduling problems and the number of them grows continuously. The researchers – mathematicians and engineers – guess more and more methods and some of these could be useful in practice, too.

The methods could be classified into four groups: (*Tóth*, 1998)

- Exact methods:
  They are able to give optimal results, but the adaptabilities of them are crucial. The circumstances determine their usage, and these possible circumstances are very lean. One of them is the enumerating method.
- Rule-based methods:
  In the case of these methods different rules are used in the scheduling situations. They are very simple but the efficiency of these rules is doubtful.
- Heuristical methods:
  The heuristical methods keep the signature of their developers. Their performance is much better than the rule-based methods, but the efficiency can not be measured and therefore they can not be estimated impartially.
- Searching algorithms:
  The searching algorithms follow different strategies. Some of them follow different pro-cesses of the nature. Others come from the biology. They work well, but this efficiency can not be measured and for that reason they can not be estimated impartially.

PROBLEMS AND THEIR SOLUTION METHOD

There are a lot of grouping of scheduling algorithms, and they can be classified according to solution methods, usages or their models. One classification can go along the scheduling problems. Then the main classes are: (*Erdélyi*, 1999)

- Problems, where the optimal solution can be found in polynomial time. In this case the scheduling algorithm can follow the well-known attributes of the problem and it is able to find the optimal solution in polynomial time.
- Small problems, where the optimal solution can be found by enumeration within a reasonable time. This method guarantees the optimal solution and this is the main strength of the enumeration.
- NP-hard problems, where the enumeration can not be used in real time and there are not polynomial solutions which guarantee the optimal solution. In these cases the possibilities are:
  - Heuristics: The solution is fast but the result is just quasi-optimal, so the efficiency of these methods is doubtful.
  - Searching algorithms: These methods follow different strategies and try to find the best solution. The main types are the following:
    - Global strategy: The searching method examines the full solution space but it does not look for the global optimal solution.
    - Local strategy: The local searching methods search for local optimal solutions and they hope that the optimal solution is close to the found solution. The main advantage of these methods is the high velocity.

From the aspect of efficiency the enumeration must be underlined, because this method guarantees the optimal solution. Its price is the relative high running time. At a small scheduling problem (number of jobs = 14) the running time is about 2 days on an average computer.

Counterpoints are the local searching methods and heuristics which are very fast but their results are quasi-optima.

One of the exact methods: the enumerating method

A common property of the exact methods is that their schedules are optimal from the aspect of the examined objective function. Using this common character we are able to create a scheduling model which provides the optimal schedule.

The amount of works, when the exact methods work efficiently, is very small, but contrary to this the importance of these methods is very high from – at least – two aspects:

- The exact methods can be built into several heuristical methods,
- The analysis of the exact solutions can explore the deciding reserves and the structure of the possible but non-optimum solutions.

The number of these methods is slight. There are known only a few – in practice useable – methods because of the complexity of the scheduling task.

At present useful exact algorithms:

- SPT method, for 1 machine,
- Johnson algorithm, for 2 machines (in case of Makespan manager index),
- Extended Johnson algorithm, for 3 machines (in case of Makespan manager index and other criteria!),
- *enumeration*.

From the listed methods the simplest is the enumeration, which makes the other algorithms unnecessary among suitable circumstances and if it is adaptable. The main idea is that all possibilities have to be tried in a model which represents the real manufacturing environment, and after having the necessary data, the most suitable schedule is eligible easily.

The algorithm is very simple but its exclusive usage is excluded by the specialty of the scheduling task. The number of possible events grows factorial according to the number of the jobs. That is why there is no way to examine all possibilities neither at relative little job number, nor at few operations belonging to different jobs in case of job-shop. The opportunity of real time decision supporting is marginal.

The examination run-time can not be reduced to an acceptable measurement even by using the modern high capacity computers.

Theoretically all possibilities must be examined, but fortunately in practice the events which are not allowed because of different technological rules are left out of consideration and its magnitude is invaluable. Hereby the number of allowed schedules could be much fewer.

On the one hand this conception makes the solution easier because of the reduced run-time. On the other hand it excludes the general solution of the original task, because of the prescriptions, which are heuristical and can be different from factory to factory and can change from task to task. These simplifications themselves can be used only in particular industrial cases and they are so task specific that the results coming from the solution will not be useable at another task – with other prescriptions of course.

Tests must examine that the solution space after the simplifications would be so limited that the task could be solved in real time – with the enumerating method, of course.

Remark

This study is the second part of a complex analysis and it depends on the result of the first study. (*Buza*, 2004) Summarizing the first part, it describes the limits on principle and their exemption of parallel processing. Since then a complex system had been developed on which the written advantages and disadvantages can be experienced. It can provide measuring results which can prove the reason for existence of the method in scheduling.

REQUIREMENTS

The main goal of the research was to develop a new client-server system which can solve scheduling problems parallel. The model was the – well known – flow-shop model and the objective function was Makespan.

The following requirements had had to meet:

- Task-decomposition: the problem has to be broken down to smaller segments which can be divided among the clients.
  - This requirement was kept with limitations, which means that the size of the segments is determined by the user. The part-tasks* must be smaller than this size.
- Size of the part-tasks: the above mentioned task-decomposition has to be done properly, the part-tasks can be neither too small nor too big.
  - The size of the part-tasks depends on the clients' hardware environment. But it can be estimated satisfactorily.
- Communication: the communication structure can be either a strong or a weak point. It must be planned carefully because the order of the communication depends on the structure.
  - The communication is a key-factor, during parallel processing the server-client communication could be so high that it reduces the efficiency of utilization of the resources.
- Security: the security means not the vulnerability of the computers but the errors which have to be treated.
  - The treatment of the mentioned errors can be solved, so the calculated results are in safety in case of an error. Furthermore, "dead" clients do not block the partitioned part-task, it will be divided again among the "living" clients.
- Transparency: the client process must run in background, the user's input must have done in real-time.
  - This is a hard requirement, it can be kept by the size of the part-tasks. But it is hard to predict the type of the computer on which the part-task will be done. That is the reason why it can be a good solution if an average computer is predicted and calibrated the size of the part-task on it.

TECHNICAL DETAILS AND RESULTS

Realisation of distribution

As above mentioned, the problem will be divided into smaller part-tasks. The theoretical description was the theme of the previous study. The practical side of it is following now.

* Part-task is a tightened problem. The user has to determine the size of these part-tasks (*p*), and the clients check only the solution space of this small task instead of the complete problem (*n*). The running time is much lower, but the number of these part-tasks is relative high ($\frac{n!}{p!}$ at the flow-shop model).

Let the number of jobs be 5 (n=5), and let the size of part-tasks be 3 (p=3). The complete solution space consists of 120 different schedules – in the flow-shop model. Using the number of part-tasks describing formula ($\frac{n!}{p!}$), number of part-tasks is 20 in this specific case. These part-tasks must be examined. Schedules of complete problem can be seen on *Table 1*.

Table 1

Schedules of the complete problem

Sequential number of schedules (1)	Schedules (2)
1	1, 2, 3, 4, 5
2	1, 2, 3, 5, 4
3	1, 2, 4, 3, 5
4	1, 2, 4, 5, 3
…	…
119	5, 4, 3, 1, 2
120	5, 4, 3, 2, 1

1. táblázat: A teljes probléma lehetséges ütemtervei

Az ütemtervek sorszáma (1), Ütemtervek (2)

The part-task distribution can be followed on *Table 2*.

Table 2

Part-tasks of the complete solution space

Sequential number of part-task (1)	First schedule in the part-task (2)	Last schedule in the part-task (3)	Number of schedules in the part-task (4)
1	1:1,2,3,4,5	6:1,2,5,4,3	3!=6
2	7:1,3,2,4,5	12:1,3,5,4,2	3!=6
3	13:1,4,2,3,5	18:1,4,5,3,2	3!=6
4	19:1,5,2,3,4	24:1,5,4,3,2	3!=6
5	25:2,1,3,4,5	30:2,1,5,4,3	3!=6
…	…	…	…
18	103:5,2,1,3,4	108:5,2,4,3,1	3!=6
19	109:5,3,1,2,4	114:5,3,4,2,1	3!=6
20	115:5,4,1,2,3	120:5,4,3,2,1	3!=6
Sum:	-	-	20·3!=120

2. táblázat: A teljes megoldási tér részfeladatai

A részfeladat sorszáma (1), A részfeladat első ütemterve (2), A részfeladat utolsó ütemterve (3), Az ütemtervek száma a részfeladatban (4)

After the part-task distribution only the number of the given part-task must have been transferred to the client. It will be enough to identify the specific part-task. Then the client can check the required part-task, calculate the performance index of it and send the best schedule and its performance index (Makespan) to the server application.

With help of the above mentioned method any scheduling problem can be distributed for parallel processing.

The main parameters of the examined tasks can be seen on *Table 3*.

Table 3

The main factors of the problem

Number of jobs (*n*) (1)	14
Number of machines[†] (*m*) (2)	10
Size of part-tasks (*p*) (3)	10
Solution space of the complete problem (*n!*) (4)	87 178 291 200
Solution space of a part-task (*p!*) (5)	3 628 800
Number of part-tasks $\left(\frac{n!}{p!}\right)$ (6)	24 024

3. táblázat: A feladat jellemzői

A munkák száma (1), A gépek száma (2), A részfeladatok mérete (3), A teljes feladat megoldási terének számossága (4), A részfeladatok számossága (5), A részfeladatok darabszáma (6)

Communication conception

The communication conception is one of the most critical elements of the parallel processing. *Figure 1* shows our communication conception with one client. The process of communication is the same by multiple clients, too.

The following steps can be distinguished during the perfect communication:

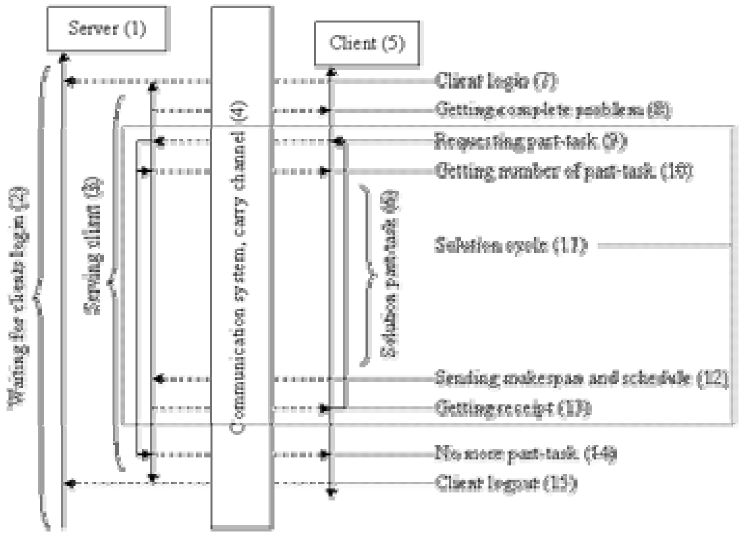
1. The server process is running permanently and waiting for login of the different clients. If the connection is successful, a new thread is created, which communicates with this client alone. Of course, the server is able to receive the new clients.
2. The client-thread – at the server side – sends the complete problem (including the number of machines (*m*) and jobs (*n*), the matrix of machining times and the size of part-tasks (*p*), which is determined previously) to the client. The client would be able to solve the complete problem but we have an other purpose.
3. The client requests the number of part-task (*l*), which has to be solved ($1 \leq l \leq \frac{n!}{p!}$, *l* – number of part-task, *n* – number of jobs, *p* – size of part-task).
4. The client thread on the server side
5. sends the number of the next part-task waiting for solution (*l*). (→5.)
6. disconnects if the full problem is solved. (→8.)
7. The client gets the number of the following part-task and checks it.

[†] It must be mentioned, that number of schedules is independent of number of machines.

8. When the client finishes the analysis, it sends the results (makespan and schedule) of the schedule possessing the best performance index to the server.
9. The server gets the results and sends a receipt to the client about the successful communication. (→3.)
10. The communication can be disconnected, because the solution space of the complete problem is examined. The server sends a disconnection message to the client.
11. The client gets the message and destroys himself.
12. The client thread detects this and the connection is over.
13. The server keeps on running.

Figure 1

Communication structure



1. ábra: A kommunikációs struktúra felépítése

Szerver alkalmazás (1), Várakozás a kliens bejelentkezésére (2), A kliens kiszolgálása (3), Kommunikációs közeg, csatorna (4), Kliens alkalmazás (5), A részfeladat megoldása (6), A kliens bejelentkezése (7), A teljes feladat paramétereinek fogadása (8), Részfeladat igénylése (9), A részfeladat sorszámának fogadása (10), Feladat megoldási ciklus (11), Az legrövidebb átfutási idő és a hozzá tartozó ütemterv átküldése (12), Nyugta fogadása (13), A részfeladatok mindegyike megoldásra került (14), A kliens kijelentkezése (15)

The communication is lean, but it is enough to the solution and the safe functioning. There are not any redundant data, which could cause bottle-neck. The complete problem is transmitted so the further communication is very simple.

It means that the clients must have a specific knowledge about the problem and performance indices but this is essential for the efficient communication. If something changes the problem radically, it is enough to change the clients on the server side because the instances on the client side are downloaded – and refreshed – automatically. So there is no need for any service activities, everything is automatic.

The developer environment: JAVA

In the beginning we planned a system, which does not require any specific knowledge and installation from the users. That is why we decided beside Java. The user does not have to install a complex system, just start his internet browser and – with the help of applet programming – write the given URL into his browser. Our application starts automatically and the analysis is beginning immediately.

There are some problems if the user has an own firewall but this problem is attached to every programming language, not only Java.

So the server and client application have been written in Java and they are really easy to use, according to one of our objectives.

Aspects of the usage

It was mentioned above, that load of the processor can be critical. That is why two possibilities must be distinguished:

- The user is sitting at his computer and working on it. The client is running in the background and does not block the work of the user. In this case the user has to decide how he uses the unnecessary processor time (idle time). The user has to set the priority of the client application.
- There is a specific application for this case, and it can be downloaded and started from the Internet. It connects to the server and begins to solve some part-tasks.
- If the computer is running and nobody uses it, then it can be loaded much better than in the previous. The answer-times could be much larger but it is unimportant. In this case there is an independent client process which detects the idle-time and runs automatically. This application works as a screen-saver, so its realization does not demand extra processor capacity.

CONCLUSIONS AND FURTHER DEVELOPMENTS

Conclusions

Parallel processing is an outstanding tool for improving enumeration efficiency and for reducing running time. There are different limitations but these can be treated by smart distribution. The written applications are able to process parallel and the running time can be reduced significantly. They collect experimental results, which are essential to understand the inner system of the problem.

The communication structure works perfectly and it filters the redundant data-movements. It makes the communication safe and efficient.

Further developments

The main directions of the developments are the systematic use of the applications and the collection of experimental results. The main goal is to develop a brand new scheduling method, which keeps the main advantages of the applications – the shorter running time and the ability to find the optimal solution in the complete solution space in the case of Makespan objective function.

ACKNOWLEDGEMENTS

The researches, presented in this article, were realised in a specific project, which members are academic and industrial partners (Project leader: Dr. Monostori László, MTA SZTAKI). This project is sponsored by the Hungarian Government, too. The author would like to render thanks to the financial and scientific support of the project-members.

REFERENCES

Buza, Á. (2004). A párhuzamos processzálás alkalmazhatóságának vizsgálata a leszámlálás ütemező algoritmusa esetén. Doktoranduszok Fóruma, 2004. 11. 08. Miskolc

Tóth, T. (1998). Tervezési elvek, modellek és módszerek a számítógéppel integrált gyártásban. Miskolci Egyetemi Kiadó : Miskolc, 150-187.

Erdélyi, F. (1999). Számítógépes gyártásirányítás. Miskolci Egyetemi Kiadó : Miskolc, 87-123.

Corresponding author (*Levelezési cím*):

Buza Ákos
University of Miskolc, Faculty of Mechanical Engineering
Department of Information Engineering
H-3515 Miskolc, Egyetemváros
Miskolci Egyetem, Gépészmérnöki Kar,
Alkalmazott Informatikai Tanszék,
3515 Miskolc, Egyetemváros
Tel.: 36-46-565-111/19-52
e-mail: iitbuza1@uni-miskolc.hu