CHEMICAL ENGINEERING TRANSACTIONS Volume 21, 2010
Editor J. J. Klemeš, H. L. Lam, P. S. Varbanov
 ISBN 978-88-95608-05-1 ISSN 1974-9791
DOI: 10.3303/CET1021157

S-graph Based Scheduling Method for Parallel Architecture

J. Smidla, Istvan Heckl*
Department of Computer Science and Systems Technology, FIT, University of Pannonia, Egyetem u. 10, 8200 Veszprém, Hungary
heckl@dcs.uni-pannon.hu

The S-graph framework introduced by Sanmarti et al. (1998), originally developed for single processor, is proved to be an effective tool for batch process scheduling. The basic algorithm of the framework follows branch-and-bound strategy that provides an opportunity for parallelization. Nowadays, multi-core processors are available even in personal computers, thus, parallel algorithms become more important and useful in practice.
Two types of parallel scheduling algorithms are introduced here. In the first one, each processor considers the entire search space, while in the second one, some processors perform a look-ahead strategy for sharpening the bound.
The proposed algorithms are effective in practice, for some examples the so called super-linearity occurs, i.e., for n processor system, the running time can be less than the n-th fraction of the time required by a single processor.

1. Introduction

In present work, multipurpose batch plants are considered with non-intermediate storage policy (NIS). The recipe specifies the ordering of the tasks for the related product. The schedule of an equipment unit determines the tasks to be performed by that unit, the processing order of these tasks, and the timing information, as well. The scheduling problem aims to find the optimal schedule, usually in terms of maximizing the profit or minimizing the time required to perform the schedule (i.e., the makespan).
One of the earliest representation tools for scheduling problems is the State-Task Network (STN) introduced by Kondili et al. (1988). STN is a directed graph with two types of distinctive nodes. The materials are the states represented by circles, and the operations are the tasks represented by rectangles. When a state is consumed or produced by an operation an arc is drawn between them. Pantelides (1994) extended the STN to Resource-Task Network (RTN), which complements the STN with resource representation.

Please cite this article as: Smidla J., and Heckl I., (2010), S-graph based parallel algorithm to the scheduling of multipurpose batch plants, Chemical Engineering Transactions, 21, 937-942 DOI: DOI: 10.3303/CET1021157

2. S-graph representation

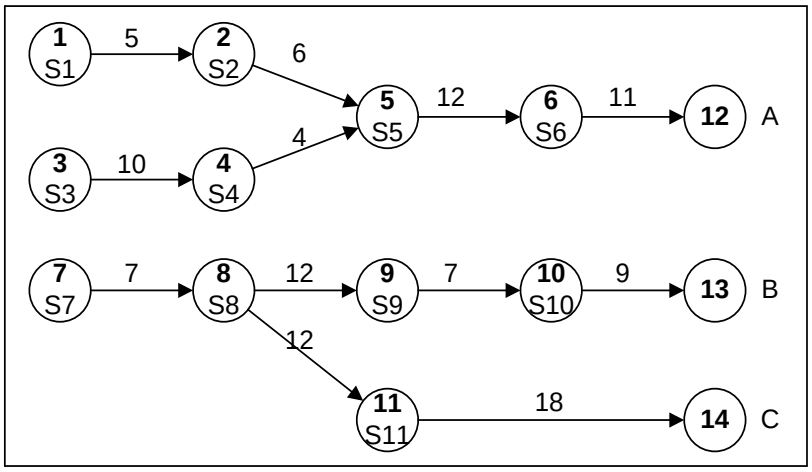

Figure 1: S-graph representation of two recipes

The S-graph framework introduced by Sanmarti et al. (1998), and futher developed and described by Sanmarti et al. (2001), Friedler (2010), is based on a new problem formulation. A directed acyclic graph has been used for the problem representation and in the solver algorithm. In this graph, unique nodes belong to tasks and products. Formally, the S-graph is given by a triplet (N, A_1, A_2), where N is a finite set of nodes, A_1 and A_2 are disjoint set of arcs. A_1 contains the recipe-arcs, and A_2 contains the schedule-arcs, that are inserted to the graph by the optimization algorithm according to the storage-policy. While the recipe-arcs denotes the order of tasks of one product, the schedule-arcs represents the scheduling of different equipment units. Moreover, a nonnegative value $c(i, j)$ denotes the weight of arc from node i to j. This weight means that the task j starts at least $c(i, j)$ time unit later than task i. For example, in Figure 1, $c(1, 2) = 5$, i.e., task 2 starts at least 5 time units after the start of task 1. Figure 1 contains only recipe-arcs termed recipe-graph. In general, several equipment units are able to perform a certain task, however, in the solution exactly one equipment unit is assigned to each task. Formally, set S_i contains the equipment units which can be assigned to task i in the recipe. The scheduling algorithm assigns an equipment unit to each task from a given set of equipment units. This graph is termed schedule-graph.

When the intermediate product can be freely stored in a predetermined storage, the schedule arcs are represented in the following manner: If equipment unit e performs task i and consequently task j then the schedule arc is inserted from task i to task j with weight p_{ij}, where p_{ij} is the processing time of equipment unit e on task i. Equipment unit e can start task j immediately after task i is finished because the intermediate product can be unloaded and stored in a designated storage.

Since in NIS policy there is no intermediate storage, the intermediate product must be stored in the actual equipment unit until the product is loaded into the subsequent equipment unit. In Figure 2, equipment unit E1 performs task 1, after it continues the work with task 8. The schedule-arc starts at task 2 because E1 can only perform the next

task, when the result of task 1 has been loaded into the equipment unit which performs task 2. If the insertion of new scheduling arc created a cycle in the graph then it would represent an infeasible scheduling. Thus, this graph has to be dropped.

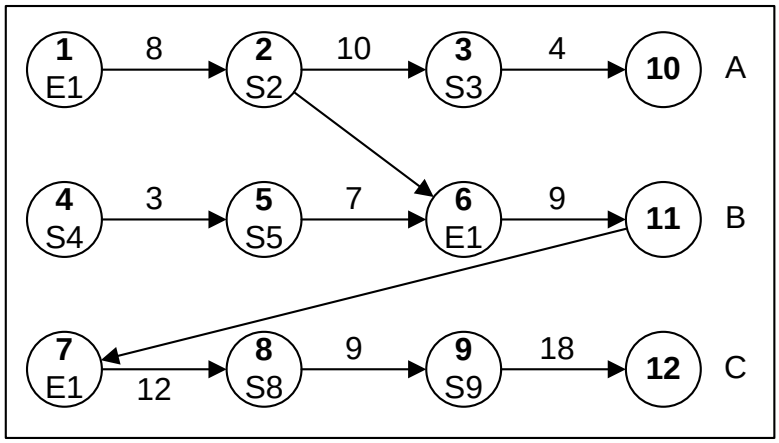

Figure 2: S-graph representation of task sequence 1-6-7 for equipment unit E1 with NIS policy

The scheduling algorithm for makespan minimization is based on the branch-and-bound (B&B) framework. At the beginning, variable *current_best* is infinite. Each node of the B&B tree corresponds to a subproblem, i.e., an S-graph defining a partial schedule. In a branching step, the children of the current problem are generated by extending the partial schedule of this problem. The algorithm selects an equipment unit (*e*) and the next possible task of unit *e* is specified. The last task of equipment unit *e* is different in each child problem.
The bounding function calculates the longest path in the graph or solves an LP model. If the graph corresponding to a node of the B&B tree contains a cycle or its bound is not better than *current_best* are dropped. When a feasible solution is founded, its value updates the *current_best* variable. Finally, the optimal schedule is obtained.

3. Parallel branch-and-bound

Parallel branch-and-bound algorithms are classified into three groups in the literature (Gendron and Crainic, 1994). In group 1, parallelization is performed inside of a branch-and-bound node. Algorithms in group 2 build the branch-and-bound tree in parallel: each processor works on the same tree but on different sub-problems. Finally, algorithms in group 3 generate several branch-and-bound trees, where the processors apply different methods. The two proposed parallel scheduling algorithms for makespan minimization belong to group 2.
Algorithm 1. At the beginning of the optimization, processor 1 generates the children of the root problem. These problems and their descendant problems are assigned to processor 1, thus, only initially processor 1 is expected to perform operations on them.

If no problem is assigned to a processor, e.g., processor 2 (which in the case initially) then this processor requests problems to be solved. If there are processors that have unsolved problems, then one of them assigns an unsolved problem to the idle processor. If all of the processors become idle, the algorithm terminates.
The key point of this approach is the common *current_best* value: if a processor finds a solution, it updates the *current_best*, and the other processors prune nodes according to the updated *current_best*.
Algorithm 2 takes an effort to improve the efficacy of searching of the optimal solution with a modification of algorithm 1: some processors consider a supposed upper bound, LO, which is lower than the actual *current_best*. If the bound of a node is not better than the *current_best*, it is pruned. If the bound of a node is between the *current_best* and LO, the algorithm keeps it in the list of open problems (i.e., it is neither pruned nor its child problems are generated). Finally, if the bound is lower than LO then the child problems are generated. When there is no solution with objective value better than LO, the LO value is increased such that the new value of LO will be the average of its original value and *current_best*, and the child problems in the open problem list are reexamined. When a new solution, better then LO is founded, the algorithm deletes the above-mentioned child problems from the list of open problems.

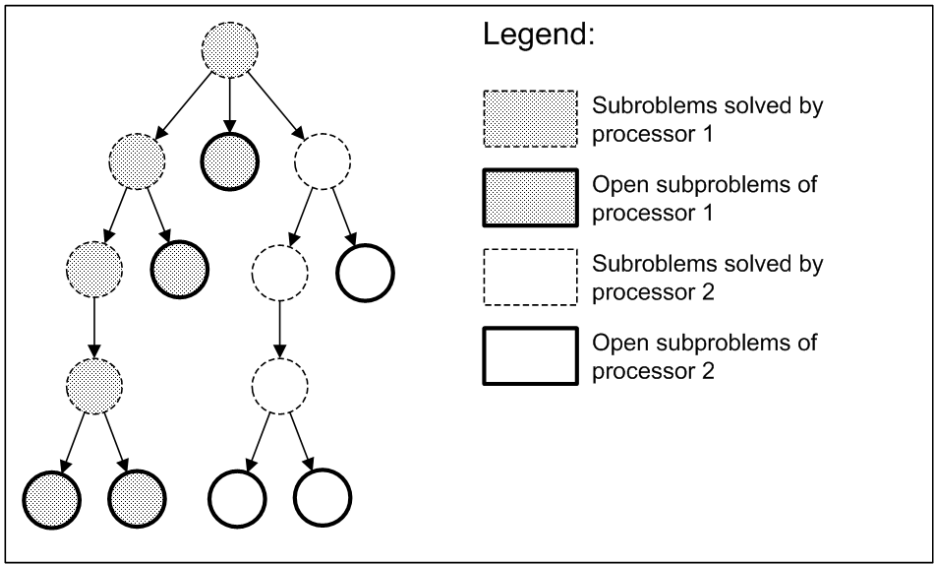

Figure 3: The processors build the B&B tree in parallel

4. Illustrative example

The first scheduling method has been tested on a problem introduced in Voudouris and Grossmann (1996). Table 1 shows the recipe of the scheduling problem.

Table 1: Recipe of literature example by Voudouris and Grossmann (1996)

	Product A		*Product B*		*Product C*		*Product D*	
Task	Eq.	Time [h]	Eq.	Time [h]	Eq.	Time [h]	Eq.	Time [h]
1	E1	8	E1	7	E2	6	E2	4
2	E4	5	E3	3	E4	9	E3	6
3	E5	3	E5	4	E5	3	E5	4

Our computations have been performed on a server with four Xeon E5400 2.83 GHz processors and 1 GB RAM. The testing environment was a 64 bit Gentoo Linux. Table 2 contains ten test cases, and shows the batch numbers in different tests, and the number of processors used. Each test has been performed 10 times and the table shows the average running times.

T(p) denotes the running time with *p* processors for a given problem. The speedup is denoted by Equation (1) and the efficiency by Equation (2):

$$S(p) = T(1) / T(p) \tag{1}$$

$$E(p) = S(p) / p \tag{2}$$

Table 2: Running times and speedups with various numbers of processors

Batch numbers				1 processor	2 processors		4 processors	
A	B	C	D	time [s]	time [s]	speedup	time [s]	speedup
5	5	5	4	7.68	2.34	3.27	3.15	2.43
5	5	5	5	31.80	8.96	3.55	7.57	4.20
6	5	5	5	19.72	4.85	4.06	4.43	4.45
6	6	5	5	7.16	2.82	2.54	3.88	1.85
6	6	6	5	136.76	36.59	3.74	26.64	5.13
6	6	6	6	592.70	170.47	3.48	119.55	4.96
7	6	6	6	319.66	78.22	4.09	59.26	5.39
7	7	6	6	60.29	14.40	4.19	11.54	5.22
7	7	7	6	2 545.28	652.38	3.90	467.97	5.44
7	7	7	7	11 308.72	3 188.77	3.55	2 268.88	4.98

At some of these examples, the so-called super-linearity is observable. The speedup is greater than the number of processors used, i.e., E(p) > 1. There are two different explanations for this phenomenon. The first reason is that each processor uses its own cache memory, thus, the multi-processor program uses more cache memory than the sequential version. The second reason is that multiple processors may perform less overall work than one single processor because in the former case more subproblem can be pruned.

5. Conclusions

The formerly developed S-graph framework has been extended for multiple processors. It consists of a graph based problem representation and a branch-and-bound algorithm. The original scheduling algorithm has made parallel to exploit the benefits of the parallel computer architectures.

The implemented algorithm can be used to solve large-scale problems more effectively than the single processor. Occasionally, the proposed parallelization method leads to super-linearity. The test results have demonstrated the importance of the new parallel algorithm.

References

Friedler, F., 2010, Process integration, modelling and optimisation for energy saving and pollution reduction, Applied Thermal Engineering, doi: 10.1016/ j.applthermaleng.2010.04.030.

Gendron, B. and Crainic, G. T., 1994, Parallel Branch-And-Bound Algorithms: Survey and Synthesis, Operations Research 42, 1042-1066.

Grossmann, I. E. and Voudouris, V. T., 1996, MILP Model for Scheduling and Design of a Special Class of Multipurpose Batch Plants, Computers & Chemical Engineering 20, 1335-1360.

Kondili, C. E., Pantelides, C. C. and Sargent, H. R., 1988, A General Algorithm for Scheduling of Batch Operations, Proc. 3rd Intl. Symp. On Process Systems Engineering 17, 62-75.

Pantelides, C. C., 1994, Unified frameworks for optimal process planning and scheduling In: Rippin, D.W.T. and Hale, J., Editors, 1994. Proc. Second Conf. on Foundations of Computer Aided Operations, CACHE Publications, pp. 253–274.

Pinto, M. J. and Grossmann, E. I., 1995, A Continuous Time Mixed Integer Linear Programming Model for Short Term Scheduling of Multistage Batck Plants, Industrial & Engineering Chemistry Research 34, 3037-3051.

Puigjaner, L., 1999, Handling the increasing complexity of detailed batch process simulation and optimization, Computers and Chemical Engineering, 23S, S929-S943.

Sanmarti, E., Friedler, F. and Puigjaner, L., 1998, Combinatorial technique for short term scheduling of multipurpose batch plants based on schedule-graph representation, Computers and Chemical Engineering 22, S847-S850.

Sanmarti, E., Holczinger, T., Puigjainer, L. and Friedler, F., 2002, Combinatorial Framework for Effective Scheduling of Multipurpose Batch Plants, AIChE Journal 48(11), 2557-2570.

Shah, N., 1998, Single-and Multisite Planning and Scheduling: Current Status and Future Challenges, Foundations of Computer-Aided Process Operations 94, 75-90.