Automatic Generation of Effective Composite Dispatching Rules for Large-scale Job Shop Scheduling

N.A.

Abstract

Generating optimized large-scale production plans is an important open problem where even small improvements result in significant savings. Application scenarios in the semiconductor industry comprise thousands of machines and hundred thousands of job operations and are therefore among the most challenging scheduling problems regarding their size. Moreover, acceptable computing times are short as frequent rescheduling is needed in order to react quickly on unpredictable events like machine failures. In this paper we present a novel approach for fast generation of highly effective composite dispatching rules, i.e. heuristics for job sequencing, for makespan optimization in such large-scale job shops. We test our approach on a set of benchmark instances with proven optima that comprise up to 100000 operations to be scheduled on up to 1000 machines and compare the performance to CP Optimizer, a high-performing proprietary constraint solver of IBM. Results show that, despite or because of its simplicity, the proposed approach is clearly superior in most cases.

Introduction

The scheduling of jobs (Blazewicz et al. 2007) is an important task in almost all production systems in order to optimize various objectives such as resource consumption, makespan (time to finish all products), tardiness (lateness of products), or flow time. Driven by the demands of the semiconductor industry, our general aim is the design of practically applicable algorithms for job shop scheduling problems for domains comprising thousands of machines and hundred thousands of job operations.

In real world domains like semiconductor manufacturing, common problem instances for a weekly workload are of the order of 10^4 operations on 10^2 machines in the back-end, i.e. where the products are made ready for shipping, and 10^5 operations on 10^3 machines in the front-end, i.e. where the chips are actually produced. To the best of our knowledge, the size of such large-scale job shop scheduling problem instances go beyond existing benchmarks. For example, the well-known benchmark targeting on makespan optimization from (Demirkol, Mehta, and Uzsoy 1998) comprises up to 50 jobs and 20 machines, which results in 1000 job operations in total. The famous Taillard benchmark (Taillard 1993) contains job shop instances of sizes up to 100 jobs on 20 machines which results in 2000 job operations. When focusing on tardiness optimization, somewhat bigger instances can be found. For example, the benchmark described in (Zhang and Wu 2010) comprises up to 20000 job operations. In this paper we introduce a benchmark comprising instances with up to 100000 operations to be scheduled on up to 1000 machines.

Many approaches have been used to solve scheduling problems. Constraint based approaches (e.g. (Bartk, Salido, and Rossi 2010; Sadeh and Fox 1996)), branch and bound (e.g. (Brucker, Jurisch, and Sievers 1994)), branch and cut (e.g. (Stecco, Cordeau, and Moretti 2008)) or mixed integer programming (e.g. (Ku and Beck 2016)) have a long and successful history. Also meta-heuristic approaches based on tabu and large neighborhood search (e.g. (Watson et al. 2003; Boejko et al. 2017; Danna and Perron 2003)), simulated annealing or genetic algorithms (e.g. (Sadegheih 2006)) have been applied.

One widely employed state-of-the-art technique for dealing with large and complex scheduling problems in nowadays manufacturing environments is the application of dispatching rules (e.g. (Hildebrandt, Goswami, and Freitag 2014; Kaban, Othman, and Rohmah 2012; Panwalkar and Iskander 1977)). Dispatching rules are greedy heuristics for step-wise deciding which is the operation to be processed next by a machine. One big advantage of dispatching rules is that they can be computed typically in linear time.

In this paper we focus on the automatic creation of dispatching rules for the optimization of the makespan (i.e. minimization of time needed for accomplishing all operations) in very large job shops.

The job shop scheduling problem (JSP) is among the most famous $\mathcal{NP}$-hard (Garey and Johnson 1990) combinatorial problems and can be defined as follows:

- Given is a set $M = \{machine_1, \ldots, machine_m\}$ of machines and a set $J = \{job_1, \ldots, job_j\}$ of jobs.
- Each job $j \in J$ consists of a sequence of operations $O_j = \{j_1, \ldots, j_{l_j}\}$ whereby j_{l_j} is the last operation of job j.

  Practically, jobs can be interpreted as products and operations can be interpreted as their production steps. With respect to a job j and its operation j_i, the operation j_{i+1}

is called successor and the operation j_{i-1} is called predecessor.

- Each operation o has an operation length $length_o \in \mathbb{N}$.
- Each operation o is assigned to a machine $machine_o \in M$ by which it is processed.
- A (consistent and complete) schedule consists of a starting time $start_o$ for each operation o such that:
  - An operation's successor starts after the operation has been finished,
    i.e. with respect to a job j and the operations j_i and j_{i+1}:
    * $start_{j_{i+1}} \geq start_{j_i} + length_{j_i}$
  - Operations processed by the same machine are non-overlapping,
    i.e. with respect to two operations $o1 \neq o2$ with $machine_{o1} = machine_{o2}$:
    * $start_{o1} \neq start_{o2}$
    * $start_{o1} < start_{o2} \rightarrow start_{o1} + length_{o1} \leq start_{o2}$
- Makespan, i.e. the time period needed for processing all operations, is minimized. I.e.:
  - $\max_{j \in J, o \in O_j} \{start_o + length_o\} \rightarrow min$

The JSP is a special case of scheduling in manufacturing lines, in particular wafer fab scheduling in the semiconductor domain. The flexible JSP is a direct generalization of the JSP. In the flexible JSP for an operation type there can be many machines. Yet, for the flexible JSP it still holds that a machine can only perform one operation type. Allowing also that a machine can perform various operation types, as it is possible to change the setup of machines, leads to the flexible JSP with sequence dependent setup times. Wafer fab scheduling introduces further concepts and constraints. For example, in wafer fabs it is common to have different types of machines. 'Single' machines only allow to perform one operation at a time. In (flexible) JSPs there are only 'single' machines. In wafer fabs there are also other types of machines, for example, batch machines that allow to perform a set of operations of the same type at the same time.

From the theoretical point of view, wafer fab scheduling is not more complex than the classic JSP. To see this, it is convenient to look at the decision problem versions of the optimization problems. The corresponding decision problems are about answering the question whether a solution below a certain makespan is existing. Having an oracle in $\mathcal{NP}$ to guess a solution it is obviously possible to check correctness of the solution in polynomial time. Thus, the corresponding decision problems are both in $\mathcal{NP}$[1]. Consequently, coping with the complexity of the JSP is the key for coping with large-scale wafer fab scheduling.

In the following, we present a novel approach for fully automatic and fast generation of composite dispatching rules. Composite dispatching rules are based on underlying basic dispatching rules, like for example 'shortest-job first', and thus can be seen as hyper-heuristics (Burke et al. 2013).

After discussing the architecture of the proposed approach for automatically creating effective dispatching rules and consecutively generating schedules for very large job shops, we report on the design and the results of a large-scale job shop experiment. In this experiment we compare the performance of our prototype system to IBM's well known CP Optimizer. The novel benchmark set incorporates job shop problem instances that have proven optimal makespans and comprise up to 100000 operations to be scheduled on up to 1000 machines. Furthermore, they differ in the average number of operations per job, which has a big impact on the practical hardness of the problem instances.

Generation of Composite Dispatching Rules

Our approach, named Random Composite Dispatching Rules (RCDR), builds on event-based simulation in combination with composite dispatching rules. The composite rules constitute weighted combinations of basic dispatching rules, whereas each basic dispatching rule is implemented as a function returning a priority value between 0 and 1. The weights in RCDR are integers between 0 and 10 and are randomly guessed. To put it more formally: The composite priority value v_c produced by composite dispatching rule c that is based on basic rues $b_1, ..., b_n$ equals to $w_1 * v_{b_1} + w_2 * v_{b_2} + ... + w_n * v_{b_n}$. The weights $w_1, ..., w_n$ are random integers in $\{0, ..., 10\}$ and the basic priority values $v_{b_1}, ..., v_{b_n}$ are in $[0; 1]$.

Once a composite dispatching rule has been created, an event-based simulation engine is used to apply the rule on a given problem instance and produce a corresponding schedule. This simple process, i.e. (1) produce a random composite rule by guessing the weights for the underlying basic dispatching rules and (2) produce a schedule by using the rule within a simulation, is repeated over and over again until a stopping criterion is reached or a timeout has occurred.

The event-based simulation engine implements a simulation model in which each machine possesses one queue. Initially, only the starting operations, i.e. those without predecessors, are enqueued. After finishing an operation on a machine, the successor operation is enqueued on its machine. As soon as a machine becomes idle and its queue is not empty the next operation to be run is selected among the operations in the queue. This selection is based on a priority value calculated by a composite dispatching rule.

Basic Dispatching Rules

Composite dispatching rules in our system basically constitute weighted combinations of 12 basic rules. These basic dispatching rules are:

1. shortest processing time (SPT): Prefer the operation with the shortest processing time.
2. longest processing time (LPT): Prefer the operation with the longest processing time.
3. shortest job first (SJF): Prefer the operation of which the job possesses the shortest total processing time.

[1] We can easily define an optimization procedure on top of the decision problems that calculates the optimal makespan by applying binary search that has only logarithmic complexity.

4. longest job first (LJF): Prefer the operation of which the job possesses the longest total processing time.
5. least total work remaining (LTWR): Prefer the operation for which the sum of the operation length and successor operations' lengths is the least.
6. most total work remaining (MTWR): Prefer the operation for which the sum of the operation length and successor operations' lengths is the most.
7. relative least total work remaining (RLTWR): As LTWR but normalized by total job length.
8. relative most total work remaining (RMTWR): As MTWR but normalized by total job length.
9. shortest waiting time (SWT): Prefer the operation that was enqueued as the last.
10. longest waiting time (LWT): Prefer the operation that was enqueued as the first.
11. urgency next (UN): Prefer the operation of which the successor operation's machine becomes idle as the first (based on the current queues).
12. urgency any (UA): Prefer the operation of which one of the successor operation's machine becomes idle as the first (based on the current queues).

SPT, LPT, SJF, LJF, LTWR, MTWR, SWT and LWT are well known (see, e.g. (K. Kaban, Othman, and Rohmah 2012)). RLTWR and RMTWR are variants of LTWR and MTWR respectively. In RLTWR and RMTWR the total work remaining (TWR) is divided by the total job length. Hence, TWR is interpreted as the share of the remaining processing time of the operation's job relative to the total processing time of the job.

Also UN and UA are rule variants based on the well known WINQ rule (Conway 1964; 1965). WINQ selects the operation of which the next successor operation will go on to the machine that has currently the least workload in the queue. The idea is to prevent idle times because of empty queues. UN follows the same idea but also takes into account the currently running operation of the successor's machine. Thus, with respect to some operation, UN calculates the nearest time point when the successor operation's machine can become idle.

UA is a generalization of UN calculating the earliest time point when one of the successor machines can become idle. I.e. whereas UN only considers the machine of the direct successor operation, UA takes all machines of the direct and indirect successor operations into account.

In order to purposefully combine the basic dispatching rules, every dispatching rule is implemented by a function that returns normalized priority values between 0 and 1. The semantic is: the higher the priority value the earlier an operation is processed by a machine.

Experimental Setup

In our experiment we compare the approach described in the last section to IBM's CP Optimizer[2], which is a proprietary constraint solver especially dedicated to scheduling problems and currently one of the best systems available in this field (Laborie et al. 2018). The maximal allowed solving time before a timeout occurred was set to 1000 seconds. For real life applications in semiconductor manufacturing, this is roughly the time period in which it must be possible to come up with a new schedule in order to be able to react quickly on unpredictable events like new high priority product orders or quite frequent machine failures and breakdowns.

Since our prototype realizing the approach presented in the last section is implemented in Java, we also used the CP Optimizer Java library for connecting to the native C++ routines of CP Optimizer. Hereby, we reused the model and code for job shop scheduling proposed by IBM (i.e. included in the examples of the solving suite).

The experiments were run on an Intel Core i7 quad-core system with hyperthreading, i.e. 8 logic cores, and 16 GByte of RAM[3]. Consequently, we allowed for both competing systems 8 worker threads.

Problem Instances

The problem instances used for the experiments described herein are patterned on real world production scheduling problems, in particular in terms of size. Moreover, the instances have known optimal makespans[4].

The creation process is as follows: First produce an optimal solution without (idle) holes for (1) a given number of job operations to be scheduled, (2) the number of machines and (3) the desired optimal makespan. This is done by randomly partitioning the machines' time continuum into the predefined number of partitions. Each partition corresponds to the processing period of one operation. Consequently, the optimal makespan, average operation length (avg(length)), number of operations (#ops) and number of machines (#machines) relate conforming to $makespan = \frac{\#ops \times avg(opLength)}{\#machines}$. Based on such a partitioning, successor relations are randomly generated. Each partition has maximally one successor and/or predecessor such that the successor's starting time is greater than the predecessors finishing time.

We applied two different procedures for generating random successor relations based on a pre-calculated solution:

1. For each operation op (in random order) define as successor suc a random operation such that
   - suc is not on the same machine as op.
   - suc starts later than op ends.
   - suc is not yet a successor of another operation.
   - If no such suc exists op has no successor.
2. For each operation op (in random order) define as successor suc an operation such that
   - suc is not on the same machine as op.

[2]https://www.ibm.com/analytics/cplex-cp-optimizer

[3]In the experiments the maximum measured memory usage was not more than 7 GByte for CP Optimizer and much less for the proposed approach.

[4]Benchmark instances and statistics can be downloaded at 'https://www.dropbox.com/s/unfo14rk2vvux70/icaps19.zip?dl=0'.

#MA	#OPS	NUM	CPO	RCDR	IMPROVEMENT
100	10000	1	892912	826139	7 %
100	10000	2	921526	751939	18%
100	10000	3	885992	986994	-11%
100	10000	avg	900143	855024	5%
1000	10000	1	600000	600000	0%
1000	10000	2	600000	600000	0%
1000	10000	3	600000	600000	0%
1000	10000	avg	600000	600000	0%
100	100000	1	1165907	1019080	13%
100	100000	2	1201578	1013878	16%
100	100000	3	1163840	1017899	13%
100	100000	avg	1177108	1016952	14%
1000	100000	1	813628	600000	26%
1000	100000	2	821096	686276	16%
1000	100000	3	842639	927927	-10%
1000	100000	avg	825788	738068	11%

Table 1: Results for long-jobs instances (min = 600000)

#MA	#OPS	NUM	CPO	RCDR	IMPROVEMENT
100	10000	1	600000	600610	-0.1%
100	10000	2	600000	600534	-0.0%
100	10000	3	600001	600008	-0.0%
100	10000	avg	600000	600384	-0.0%
1000	10000	1	643434	687569	-7%
1000	10000	2	657198	750951	-14%
1000	10000	3	651906	723618	-11%
1000	10000	avg	650846	720713	-11%
100	100000	1	-	600088	-
100	100000	2	-	600079	-
100	100000	3	-	600074	-
100	100000	avg	-	600080	-
1000	100000	1	659662	601945	9%
1000	100000	2	695079	603270	13%
1000	100000	3	689905	603854	12%
1000	100000	avg	681549	603023	12%

Table 2: Results for short-jobs instances (min = 600000)

- suc starts later than op ends.
- suc is not yet a successor of another operation and
- the time between op ends and suc starts is minimal.
- In case that there are multiple possible successors, a random one is chosen.
- If no such suc exists op has no successor.

The two different generating approaches result in benchmark instances that are different in nature: (1) produces many jobs consisting of a small number of operations. We refer to this set of instances as the 'short-jobs' benchmark instances. In contrary, (2) produces fewer jobs but with a larger number of operations per job. We refer to this set of instances as the 'long-jobs' benchmark instances

Furthermore, the problem instances differ in the number of machines, i.e. 100 machines or 1000 machines, and the total number of operations, i.e. 10000 or 100000 operations to be scheduled. These numbers are inspired by real wafer fab sizes and workloads of our industrial partner Infineon Technologies. In the frontend of the fab, that is where the chips are actually produced, you typically can find 1000 machines or above and weekly workloads between 100000 and 200000 operations to be scheduled on those machines. In the backend, that is where the chips are cut, packaged and made ready for shipping, it is common to have roughly 100 machines and weekly workloads comprising between 10000 and 15000 operations.

For all combinations of problem characteristics (i.e. long vs short jobs, 100 vs 1000 machines and 10000 vs 100000 total operations) there are three problem instances. This makes up a total of 2x2x2x3=24 benchmark instances. All benchmark instances have a minimal makespan of 600000. This approximately constitutes one week in seconds, which is a common planning horizon in semiconductor manufacturing.

Results

Tables 1 and 2 depict the achieved makespans and averages for CP Optimizer (CPO) and our Random Composite Dispatching Rule (RCDR) approach for all tested problem instances and classes.

For two problem classes CPO and RCDR perform more or less equally well. This is (1) long-jobs instances with 1000 machines 10000, which is one of the easiest problem classes as there are only 10 operations per machine on average, and (2) short-jobs instances with 100 machines and 10000 operations.

For one problem class CPO performed better than RCDR, which is short-jobs instances with 1000 machines and 10000 operations. Obviously, this class is harder than its long-jobs counterpart, as there are less operation dependencies and hence it is much less constrained.

For the other five problem classes RCDR performed better than CPO. In particular, for short-jobs instances with 100 machines and 100000 operations CPO was not able to come up with a solution before a timeout occurred, whereas RCDR performed very well on these problem instances.

Summarizing, it can be stated that the proposed RCDR approach, despite its simplicity, performs extremely well on large-scale instances with stringent time requirements and produced better results than CPO in the majority of test cases. However, looking at the size of the problem instances, the performance of CPO is remarkable.

Conclusions

In this paper we propose a novel approach for scheduling in very large job shops like found in semiconductor manufacturing. The proposed approach is based on random and therefore fast generation of composite dispatching rules which are applied within event-based simulations in order to produce the schedules. We compare the performance of our approach to IBM's well known CP Optimizer suite on a set of real-world sized problem instances comprising up to 1000 machines and 100000 operations. Under realistic settings like found at our industrial partners Infineon Technologies the proposed approach outperforms CP Optimizer, observing that the power of CP Optimizer is noteworthy.

References

Bartk, R.; Salido, M.; and Rossi, F. 2010. New trends in constraint satisfaction, planning, and scheduling: a survey. *The Knowledge Engineering Review* 25(3):249–279.

Blazewicz, J.; Ecker, K.; Pesch, E.; Schmidt, G.; and Weglarz, J. 2007. *Handbook on Scheduling: Models and Methods for Advanced Planning (International Handbooks on Information Systems)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Boejko, W.; Gnatowski, A.; Pempera, J.; and Wodecki, M. 2017. Parallel tabu search for the cyclic job shop scheduling problem. *Computers and Industrial Engineering* 113:512 – 524.

Brucker, P.; Jurisch, B.; and Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49(1):107 – 127.

Burke, E. K.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; and Qu, R. 2013. Hyper-heuristics: a survey of the state of the art. *J. of the Operational Research Society* 64(12):1695–1724.

Conway, R. W. 1964. An experimental investigation of priority assignment in a job shop. *RM-3789-PR*.

Conway, R. W. 1965. Priority dispatching and work-in-process inventory in a job shop. *Journal of Industrial Engineering* 16:228–237.

Danna, E., and Perron, L. 2003. Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In Rossi, F., ed., *Principles and Practice of Constraint Programming – CP 2003*, 817–821. Springer Berlin Heidelberg.

Demirkol, E.; Mehta, S.; and Uzsoy, R. 1998. Benchmarks for shop scheduling problems. *European Journal of Operational Research* 109(1):137 – 141.

Garey, M. R., and Johnson, D. S. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Hildebrandt, T.; Goswami, D.; and Freitag, M. 2014. Large-scale simulation-based optimization of semiconductor dispatching rules. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, 2580–2590. Piscataway, NJ, USA: IEEE Press.

K. Kaban, A.; Othman, Z.; and Rohmah, D. 2012. Comparison of dispatching rules in job-shop scheduling problems using simulation: A case study. *Int. J. of Simulation Modelling* 11:129–140.

Kaban, A. K.; Othman, Z.; and Rohmah, D. S. 2012. Comparison of dispatching rules in job-shop scheduling problem using simulation: a case study. *Int. Journal of Simulation Modelling* 11(3):129–140.

Ku, W.-Y., and Beck, J. C. 2016. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers and Operations Research* 73:165 – 173.

Laborie, P.; Rogerie, J.; Shaw, P.; and Vilím, P. 2018. Ibm ilog cp optimizer for scheduling. *Constraints* 23(2):210–250.

Panwalkar, S. S., and Iskander, W. 1977. A survey of scheduling rules. *Oper. Res.* 25(1):45–61.

Sadegheih, A. 2006. Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on ga performance. *Applied Mathematical Modelling* 30(2):147 – 154.

Sadeh, N. M., and Fox, M. S. 1996. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence* 86:1–41.

Stecco, G.; Cordeau, J.-F.; and Moretti, E. 2008. A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Comput. Oper. Res.* 35(8):2635–2655.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64(2):278 – 285. Project Management anf Scheduling.

Watson, J.-P.; Beck, J. C.; Howe, A. E.; and Whitley, L. D. 2003. Problem difficulty for tabu search in job-shop scheduling. *Artif. Intell.* 143(2):189–217.

Zhang, R., and Wu, C. 2010. A hybrid approach to large-scale job shop scheduling. *Applied Intelligence* 32(1):47–59.