Randomized Problem-Relaxation Solving for Over-Constrained Schedules

Patrick Rodler, Erich Teppan, Dietmar Jannach

KR 2021

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT I WIEN GRAZ

Motivation

1

Job Shop Scheduling (JSSP): important NP-hard problem in today's industries

Constraint Programming (CP):
- prominent approach to JSSP
- long and successful history
- state-of-the-art CP solvers can handle large-scale JSSP instances

However:
- modern production regimes are often highly dynamic
- can lead to computationally hard optimization problems on top of JSSP
- typical such problem:

  over-constrained JSSP: set of orders (jobs) exceeds current production capacities wrt. a planning horizon (e.g., a week)

  reasons: seasonal order fluctuations, unforeseen machine breakdowns, incoming high-priority orders, …

  goal: find set of jobs (orders) **of maximal utility** (e.g., revenue) that can be finished in time

  → we call this the **Job Set Optimization Problem (JOP)**

Issue:
- JOP can be solved by CP solvers by suitably adapting the JSSP encoding
- but even most powerful CP solvers struggle with the increased complexity of JOP

Our Solution:
- framework to solve JOP
- based on a randomized computation of solutions to a relaxed version of JOP
- relaxed version of JOP:

  goal: find **⊆-maximal** set of jobs (not of maximal utility) that can be finished in time

  → we call this the **Job Set Maximization Problem (JMP)**

Evaluation: suggested approach consistently outperforms a cutting-edge CP solver for JOP problems from a well-known benchmark dataset

Definitions

2

DEF: (Job Shop Scheduling Problem – JSSP) *[Blazewicz et al. 2007]*

Given: set of machines M, set of jobs J where every job $j \in J$ consists of an ordered set of operations $Ops_j = \{op_1,\dots,op_{kj}\}$ and each $op \in Ops_j$ has a length $l_{op} \in \mathbb{N}$ and has to be executed on a particular machine $m_{op} \in M$

Find: schedule σ which maps every operation op in $AllOps := \bigcup_{j\in J} Ops_j$ to a start time $\sigma(op) \in \mathbb{N}$ on its respective machine m_{op} such that
- (1) each operation of a job may only start after its preceding operation of the same job has been finished
  formally: $\sigma(op_{x+1}) \geq \sigma(op_x) + l_{opx}$ for each pair of successive operations op_x, op_{x+1} in Ops_j for all jobs $j \in J$
- (2) on each machine, a next operation may only start after the current operation has been finished
  formally: for each pair of operations op_i, $op_j \in AllOps$ with $m_{opi} = m_{opj}$, either $\sigma(op_i) \geq \sigma(op_j)+l_{opj}$ or $\sigma(op_j) \geq \sigma(op_i)+l_{opi}$
- (3) completion time is minimized
  formally: among all schedules, σ has minimal $time(\sigma):=\max_{op\in AllOps}(\sigma(op)+l_{op})$

JSSP Decision Version: given a deadline $k \in \mathbb{N}$ as additional input, is there a schedule σ satisfying (1) and (2) and $time(\sigma) \leq k$?

DEF: (Job Set Optimization Problem – JOP)

Given: deadline $k \in \mathbb{N}$, JSSP instance P with job set J, utility function u that assigns a utility $u_j \in \mathbb{N}$ to each job $j \in J$

Find: $\Delta \subseteq J$ such that
- (i) P with the reduced job set $J \setminus \Delta$ has a solution schedule σ with $time(\sigma) \leq k$
- (ii) there is no other such $\Delta' \subseteq J$ that satisfies $\sum_{j\in J\setminus\Delta'} u_j > \sum_{j\in J\setminus\Delta} u_j$

DEF: (Job Set Maximization Problem – JMP)

Given: deadline $k \in \mathbb{N}$, JSSP instance P with job set J

Find: $\Delta \subseteq J$ such that
- (i) P with the reduced job set $J \setminus \Delta$ has a solution schedule σ with $time(\sigma) \leq k$
- (ii) there is no other such $\Delta' \subseteq J$ that satisfies $\Delta' \subset \Delta$

DEF: (Minimal Set wrt. a Monotone Predicate Problem – MSMP) *[Marques-Silva et al. 2013]*

Given: set U, monotone predicate p

Find: $X \subseteq U$ such that $p(X) = 1$ and there is no $X' \subset X$ with $p(X') = 1$

(A function $p: 2^U \rightarrow \{0,1\}$ is called a monotone predicate if $p(\emptyset) = 0$ and for all $X,X' \subseteq U$ it holds that $X \subset X' \Rightarrow p(X) \leq p(X')$)

Example

3

Given: JSSP instance
- 3 machines, 3 jobs, each 3 operations
- operation lengths l_{op}:
  jobs 1, 2, 3: (2,2,2), job 4: (3,2,1)
- machine numbers m_{op} of operations:
  jobs 1, 4: (1,2,3), job 2: (2,3,1), job 3: (3,1,2)

Solution: Fig. F1 (a): schedule σ, $time(\sigma) = 9$

Given: JOP instance
- same JSSP instance (see left)
- deadline $k = 6$
- all jobs have equal utility u

Solution:

Fig. F1 (b): schedule σ for JOP solution $\Delta = \{\text{job } 4\}$, $time(\sigma) = 6$

Fig. F1 (c): schedule σ for JMP (but not JOP) solution $\Delta = \{\text{job } 1, \text{job } 3\}$, $time(\sigma) = 6$

Approach

4

Foundation: four observations

Obs1: each JOP solution is a JMP solution (see Fig. F2 (b))
e.g., if every 10th JMP solution is a JOP solution, then generating 20 random JMP solutions yields a JOP solution with 88 % probability

Obs2: JMP has lower complexity than JOP
intuitively: JOP = finding **best** JMP solution

Obs3: JMP is manifestation of MSMP problem for which efficient algorithms exist
$O(|J|)$ calls to an oracle for JSSP decision problems per JMP solution

Obs4: CP solvers typically do not support JMP solving
cannot use CP solver to exploit JMP solving for JOP solving

Idea: draw a random sample in the JMP solution space (which covers all JOP solutions)

Procedure (Outline):
- solve multiple randomly modified JMP instances
- store solution with best utility throughout the process, stop if required solution quality is achieved

Modules:
- CP solver (for solving decision versions of JSSP)
- MSMP unit (for finding JMP solutions)
- random number generator (for generating multiple random solutions)

Rationale: trade one hard optimization (JOP) for multiple easier decisions (JSSP)
- direct CP solver approach: (cf. Fig. F2 (c)) black-box solving of two problems at once: (implicit) subset-minimality + optimal utility of the JOP solution
- proposed approach: (cf. Fig. F2 (a)) disentangle these two problems by
  - (1) extracting (efficient + well-understood) MSMP reasoning from solver
  - (2) using solver only for deciding a polynomial number of JSSP instances (for which state-of-the-art solvers are optimized)

Properties:
- no information exchange between different JMP computations → efficient parallelization
- completeness wrt. JMP/JOP achievable by using suitable random number generator
- no manual adaptation of CP encoding of given JSSP needed
- all modules viewed as black-boxes (cf. Fig. F2 (a))
  → can be realized by most suitable/performant algorithms for given problem
  → approach can profit from latest research advancements in MSMP + JSSP

Evaluation

5

Dataset:
- based on subset of widely used benchmark problems of *[Taillard 1993]*
  10 instances with (50 jobs, 15 machines)
  10 instances with (100 jobs, 20 machines)
- generation of JOP instances:

  given: Taillard JSSP problem instance P, optimal completion time k^* for P

  define: 5 over-constrained problems with deadlines $k := r \cdot k^*$ using $r \in \{0.95, 0.9, 0.85, 0.8, 0.75\}$ (different deadline scenarios for company) uniform job utilities (since no utilities given in Taillard's benchmarks)

  result: 20 (Tailllard instances) x 5 (completion time levels r) = 100 JOP problem instances

Settings:
- Java proof-of-concept implementation of proposed approach (cf. Fig. F2 (a))
  CP solver: IBM's CP Optimizer *(https://www.ibm.com/analytics/cplex-cp-optimizer)*
  MSMP algorithm: Inverse QuickXplain *[Shchekotykhin et al. 2014]*
- Baseline: CP Optimizer that solves direct encoding of JOP (cf. Fig. F2 (c))
  direct encoding = adapted JSSP encoding such that
  - (1) deadline constraint for job j ("delay for j is not allowed") is active if an associated variable $v_j \in \{0,1\}$ is set to 1
  - (2) optimization criterion = maximize the sum over v_j for $j \in J$
  - (3) calling CP Optimizer given this encoding leads to computation of a JOP solution

Experiments:
- two timeouts: 1h, 2h (idea: allow for frequent intra-day recalculations / re-scheduling to react quickly to dynamics in industrial scenarios)
- 8 worker threads (for both the proposed and the baseline approach)

Results: (cf. Fig. F3)
- for all timeouts + all JOP instances:
  proposed approach yields consistently better schedules with more finished jobs than the direct CP encoding
- improvements:
  (50 jobs, 15 machines) instances: avg 8 %, up to 15 % more jobs
  (100 jobs, 20 machines) instances: avg 5 %, up to 13 % more jobs
  always better results within 1h than direct CP encoding in 2h
  order of magnitude fewer internal solution steps than direct CP encoding

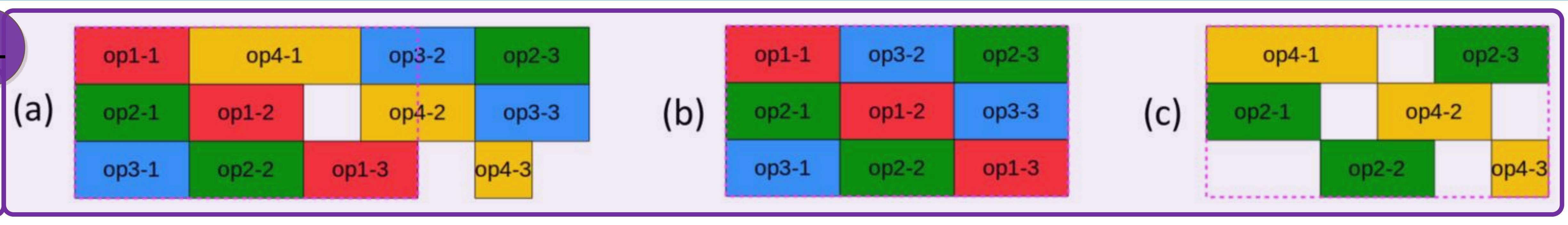
References

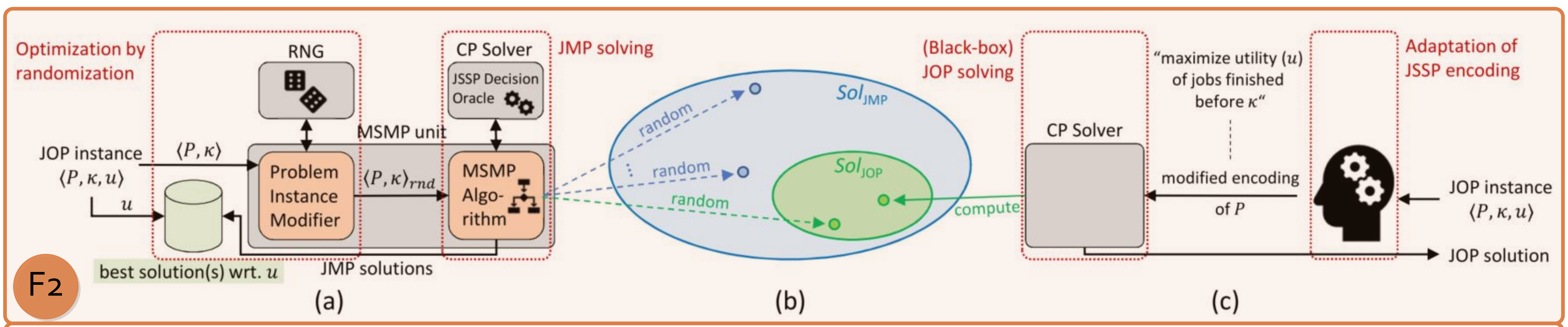
- *Blazewicz, Ecker, Pesch, Schmidt, Weglarz, 2007.* Handbook on Scheduling: Models and Methods for Advanced Planning. Springer.
- *Marques-Silva, Janota, Belov, 2013.* Minimal sets over monotone predicates in Boolean formulae. In: CAV. 592–607 .
- *Shchekotykhin, Friedrich, Rodler, Fleiss, 2014.* Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation. In: ECAI. 813–818.
- *Taillard, 1993.* Benchmarks for basic scheduling problems. EJOR 64(2):278–285.

Figures

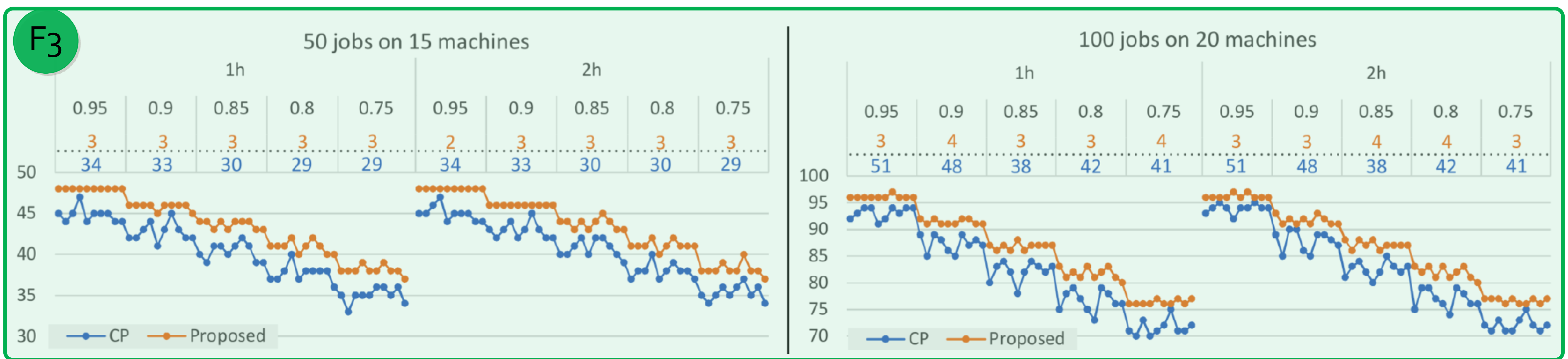
F1

Example Illustration:
- m-th row = machine m,
- horizontal axis = time line
- "opj-i" = i-th operation of job j
- dashed line = deadline k=6





(a) Proposed Framework for JOP based on decomposing JOP into subproblems (i) JMP solving and (ii) optimization via randomization. **(b) Illustration of Solution Spaces** for JOP and JMP. **(c) A Direct CP Approach** to JOP. Remarks: Both approaches require a CP encoding of the JSSP P. Gray rectangles denote black-boxes and can be realized by different suitable algorithms. "RNG" = random number generator



Evaluation Results: Each data point indicates the number of accomplished jobs (y-axis) of the CP approach (blue) versus the proposed approach (orange) per benchmark problem instance, for different time-outs (1h, 2h) and different values (0.95, . . . , 0.75) of r (x-axis). The orange / blue numbers along the x-axis indicate the average number (per value of r) of (within-timeout) generated JMP solutions (proposed approach) / intermediate solutions towards JOP (CP approach), each of which improved the current best solution (w.r.t. utility).