A Concurrent Approach to the Periodic Event Scheduling Problem

Ralf Borndörfer, Niels Lindner, Sarah Roth

Zuse Institute Berlin

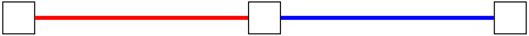
RailNorrköping 2019

June 18, 2019

§1

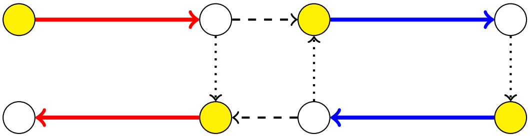
Introduction

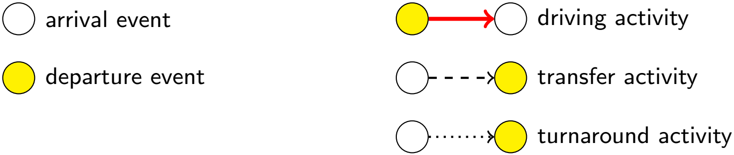
Periodic Timetabling Example



two lines meeting at a common station

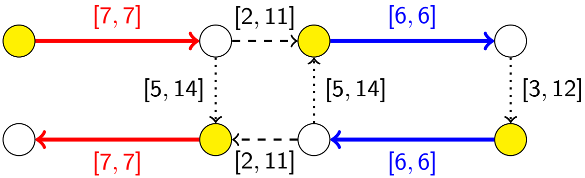
Periodic Timetabling Example

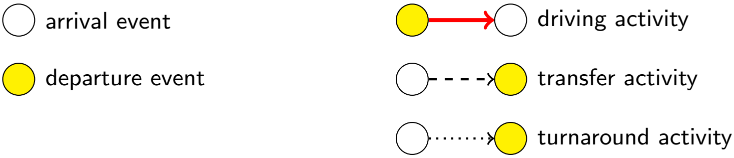




event-activity network model

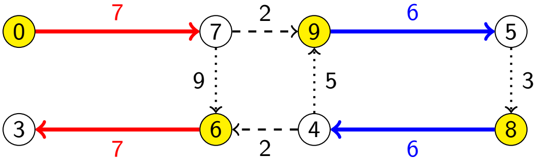
Periodic Timetabling Example





PESP instance (still unweighted), period time $T = 10$

Periodic Timetabling Example



arrival event

departure event

driving activity

transfer activity

turnaround activity

periodic timetable, period time $T = 10$

Periodic Event Scheduling Problem

Serafini and Ukovich (1989)

Given

- an event-activity network $G = (V, E)$,
- a period time $T \in \mathbb{N}$,
- lower bounds $\ell \in \mathbb{Z}^E$, $\ell \geq 0$,
- upper bounds $u \in \mathbb{Z}^E$, $u \geq \ell$,
- weights $w \in \mathbb{R}^E$, $w \geq 0$,

Periodic Event Scheduling Problem

Serafini and Ukovich (1989)

Given

- an event-activity network $G = (V, E)$,
- a period time $T \in \mathbb{N}$,
- lower bounds $\ell \in \mathbb{Z}^E$, $\ell \geq 0$,
- upper bounds $u \in \mathbb{Z}^E$, $u \geq \ell$,
- weights $w \in \mathbb{R}^E$, $w \geq 0$,

the (integer) **periodic event scheduling problem (PESP)** is to find a periodic timetable $\pi \in \{0, 1, \ldots, T-1\}^V$ and a periodic tension $x \in \mathbb{Z}^E$ such that

- $\ell \leq x \leq u$,
- $\sum_{ij \in E} w_{ij} x_{ij}$ is minimal.

Periodic Event Scheduling Problem

Serafini and Ukovich (1989)

Given

- an event-activity network $G = (V, E)$,
- a period time $T \in \mathbb{N}$,
- lower bounds $\ell \in \mathbb{Z}^E$, $\ell \geq 0$,
- upper bounds $u \in \mathbb{Z}^E$, $u \geq \ell$,
- weights $w \in \mathbb{R}^E$, $w \geq 0$,

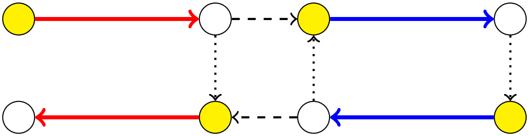
the (integer) **periodic event scheduling problem (PESP)** is to find a periodic timetable $\pi \in \{0, 1, \dots, T-1\}^V$ and a periodic tension $x \in \mathbb{Z}^E$ such that

- $\ell \leq x \leq u$,
- $\sum_{ij \in E} w_{ij} x_{ij}$ is minimal.

Equivalently, one can minimize $\sum_{ij \in E} w_{ij} y_{ij}$, where $y := x - \ell$ denotes the **periodic slack**.

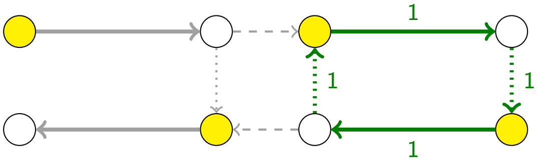
Cycle Periodicity Property

Oriented Cycles



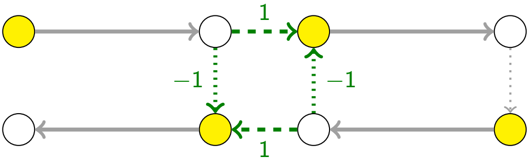
Cycle Periodicity Property

Oriented Cycles



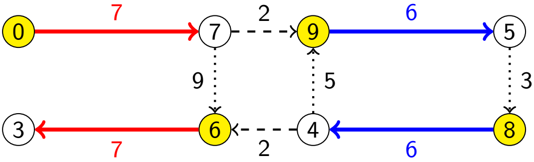
Cycle Periodicity Property

Oriented Cycles



Cycle Periodicity Property

Oriented Cycles



Theorem (Cycle Periodicity Property, Odijk 1994)

Let (G, T, ℓ, u, w) *be a PESP instance. Let* $x \in \mathbb{Z}^E$ *be a vector with* $\ell \leq x \leq u$. *Then the following are equivalent:*

(1) *There exists a periodic timetable* π *compatible to* x.

(2) *For every incidence vector* $\gamma \in \{-1, 0, 1\}^E$ *of an oriented cycle in* G *holds* $\gamma^t x \equiv 0 \bmod T$.

§2

Solving PESP

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

Modulo Network Simplex (MNS, Nachtigall/Opitz, 2008)

- fast, but local, improving heuristic

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

Modulo Network Simplex (MNS, Nachtigall/Opitz, 2008)

- fast, but local, improving heuristic
- vertices of timetabling polytope $\leftrightarrow$ spanning tree structures

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

Modulo Network Simplex (MNS, Nachtigall/Opitz, 2008)

- fast, but local, improving heuristic
- vertices of timetabling polytope $\leftrightarrow$ spanning tree structures
- various escape strategies

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

Modulo Network Simplex (MNS, Nachtigall/Opitz, 2008)

- fast, but local, improving heuristic
- vertices of timetabling polytope $\leftrightarrow$ spanning tree structures
- various escape strategies

Boolean Satisfiability (SAT, Großmann et al., 2012)

- pseudo-polynomial transformations

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

Modulo Network Simplex (MNS, Nachtigall/Opitz, 2008)

- fast, but local, improving heuristic
- vertices of timetabling polytope $\leftrightarrow$ spanning tree structures
- various escape strategies

Boolean Satisfiability (SAT, Großmann et al., 2012)

- pseudo-polynomial transformations
- feasibility: SAT solver (very fast)

Existing Approaches

Mixed Integer Programming (MIP, Liebchen, 2006)

- global, but slow
- several formulations, weak linear programming relaxations
- cutting planes by e.g. (change-)cycle inequalities

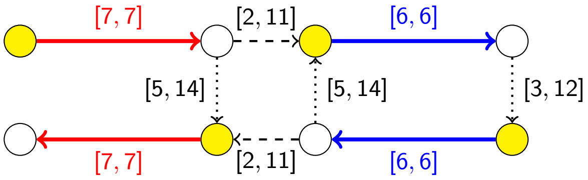
Modulo Network Simplex (MNS, Nachtigall/Opitz, 2008)

- fast, but local, improving heuristic
- vertices of timetabling polytope $\leftrightarrow$ spanning tree structures
- various escape strategies

Boolean Satisfiability (SAT, Großmann et al., 2012)

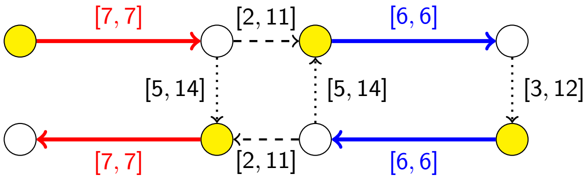
- pseudo-polynomial transformations
- feasibility: SAT solver (very fast)
- optimality: weighted partial MaxSAT solver (very slow)

Network Simplification



Preprocessing

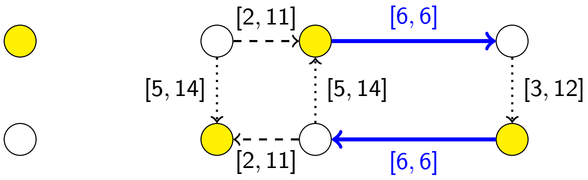
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)

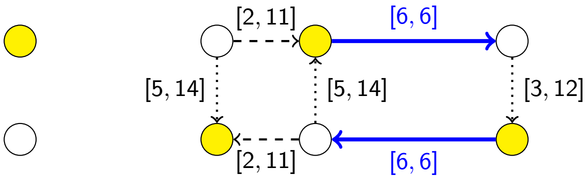
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)

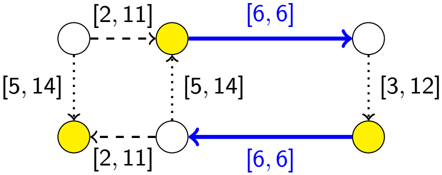
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events

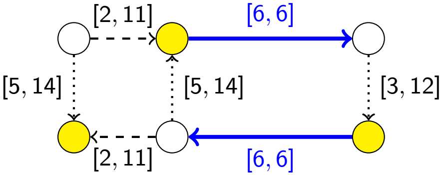
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events

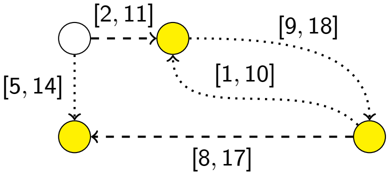
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events
- contract fixed activities (i.e., $\ell_a = u_a$)

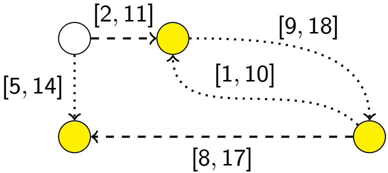
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events
- contract fixed activities (i.e., $\ell_a = u_a$)

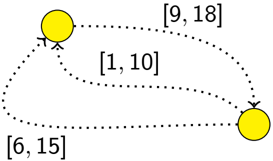
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events
- contract fixed activities (i.e., $\ell_a = u_a$)
- contract events of degree two (inexact)

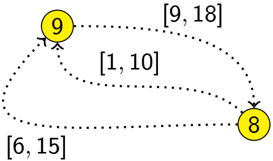
Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events
- contract fixed activities (i.e., $\ell_a = u_a$)
- contract events of degree two (inexact)

Network Simplification



Preprocessing

- remove bridges (i.e., activities that are not part of any cycle)
- remove isolated events
- contract fixed activities (i.e., $\ell_a = u_a$)
- contract events of degree two (inexact)

Ignoring Light Free Activities

Idea (Goerigk/Liebchen, 2017)

Removing free activities does not affect feasibility. The number of arcs, linearly independent cycles, and the objective value decrease.

Ignoring Light Free Activities

Idea (Goerigk/Liebchen, 2017)

Removing free activities does not affect feasibility. The number of arcs, linearly independent cycles, and the objective value decrease.

Ignore r %

1. Sort the free activities in ascending order w.r.t. weight.

Ignoring Light Free Activities

Idea (Goerigk/Liebchen, 2017)

Removing free activities does not affect feasibility. The number of arcs, linearly independent cycles, and the objective value decrease.

Ignore r %

1. Sort the free activities in ascending order w.r.t. weight.
2. Delete the first activities until a certain ratio r % of the total free weight has been removed.

Ignoring Light Free Activities

Idea (Goerigk/Liebchen, 2017)

Removing free activities does not affect feasibility. The number of arcs, linearly independent cycles, and the objective value decrease.

Ignore r %

1. Sort the free activities in ascending order w.r.t. weight.
2. Delete the first activities until a certain ratio r % of the total free weight has been removed.
3. Apply network preprocessing again.

Ignoring Light Free Activities

Idea (Goerigk/Liebchen, 2017)

Removing free activities does not affect feasibility. The number of arcs, linearly independent cycles, and the objective value decrease.

Ignore $r\,\%$

1. Sort the free activities in ascending order w.r.t. weight.
2. Delete the first activities until a certain ratio $r\,\%$ of the total free weight has been removed.
3. Apply network preprocessing again.

Remarks

- Ignore $0\,\%$: original network after preprocessing
- If the total free weight is W, then the decrease in weighted slack is at most $r\,\% \cdot W \cdot (T-1)$.

State of the Art in 2017

Algorithm (Goerigk/Liebchen, 2017)

1. Find an initial solution using constraint programming.

State of the Art in 2017

Algorithm (Goerigk/Liebchen, 2017)

1. Find an initial solution using constraint programming.
2. Ignore 50 %. 15 minutes MNS. 45 minutes MIP.

State of the Art in 2017

Algorithm (Goerigk/Liebchen, 2017)

1. Find an initial solution using constraint programming.
2. Ignore 50 %. 15 minutes MNS. 45 minutes MIP.
3. Ignore 30 %. 15 minutes MNS. 45 minutes MIP.

State of the Art in 2017

Algorithm (Goerigk/Liebchen, 2017)

1. Find an initial solution using constraint programming.
2. Ignore 50 %. 15 minutes MNS. 45 minutes MIP.
3. Ignore 30 %. 15 minutes MNS. 45 minutes MIP.
4. Ignore 18 %. 15 minutes MNS. 45 minutes MIP.

   ⋮

   (for 8 hours in total)

State of the Art in 2017

Algorithm (Goerigk/Liebchen, 2017)

1. Find an initial solution using constraint programming.
2. Ignore 50 %. 15 minutes MNS. 45 minutes MIP.
3. Ignore 30 %. 15 minutes MNS. 45 minutes MIP.
4. Ignore 18 %. 15 minutes MNS. 45 minutes MIP.

   ⋮

   (for 8 hours in total)

Idea

MNS provides solutions fast, and MIP helps MNS out of local optima.

State of the Art in 2017

Algorithm (Goerigk/Liebchen, 2017)

1. Find an initial solution using constraint programming.
2. Ignore 50 %. 15 minutes MNS. 45 minutes MIP.
3. Ignore 30 %. 15 minutes MNS. 45 minutes MIP.
4. Ignore 18 %. 15 minutes MNS. 45 minutes MIP.

   ⋮

   (for 8 hours in total)

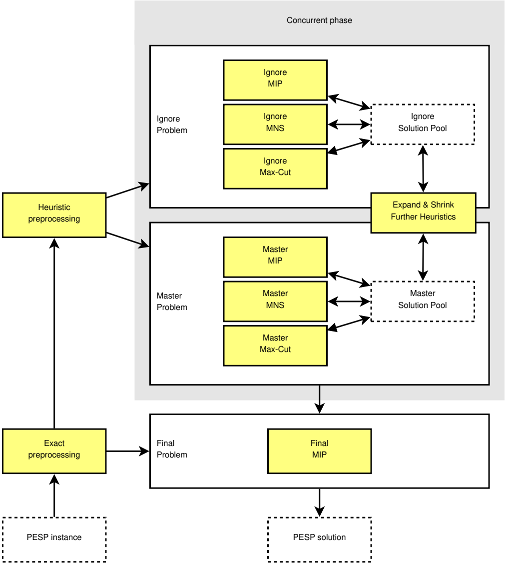
Idea

MNS provides solutions fast, and MIP helps MNS out of local optima.

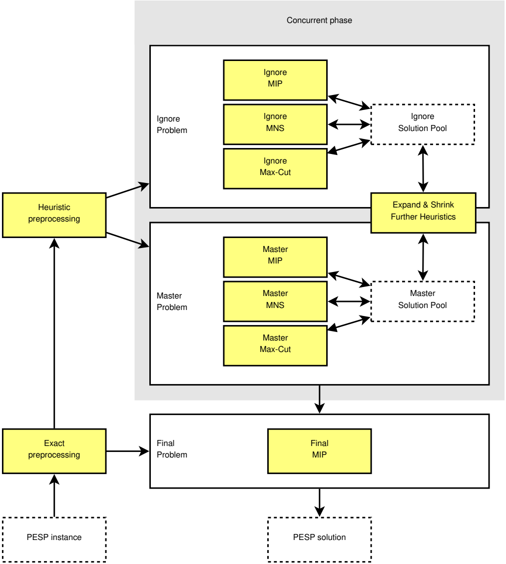
Our Goal

Combine MNS, MIP and other powerful methods to a **concurrent** solver.

Concurrent Solver Architecture

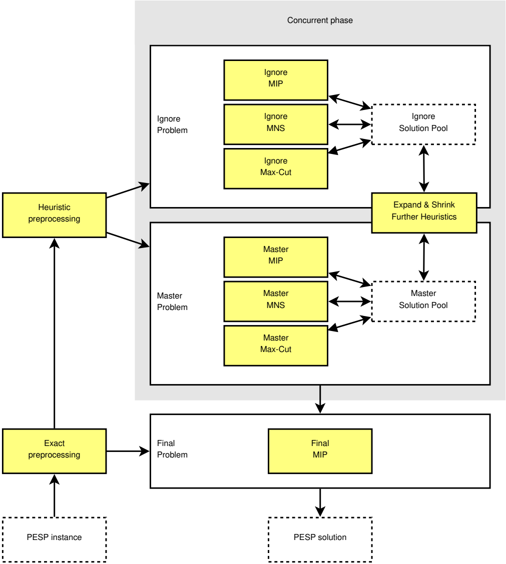


Concurrent Solver Architecture



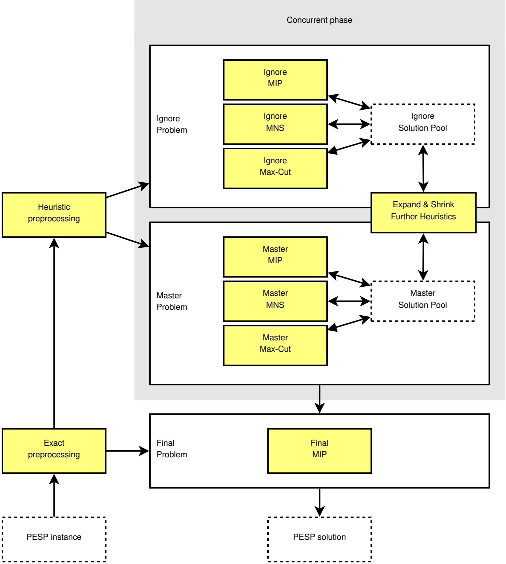
- Two problems at the same time: Master, Ignore

Concurrent Solver Architecture



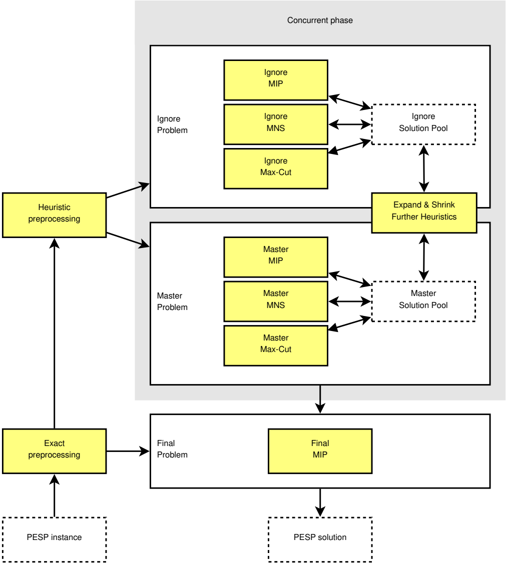
- Two problems at the same time: Master, Ignore
- Master: preprocessed input instance, does not change

Concurrent Solver Architecture



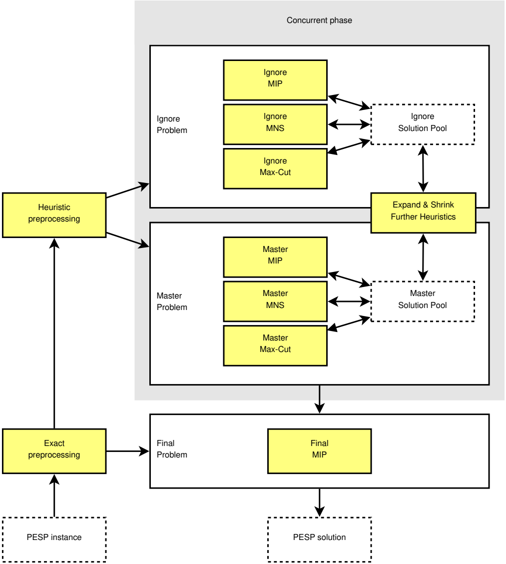
- Two problems at the same time: Master, Ignore
- Master: preprocessed input instance, does not change
- Ignore: ignore light free activities, can change

Concurrent Solver Architecture



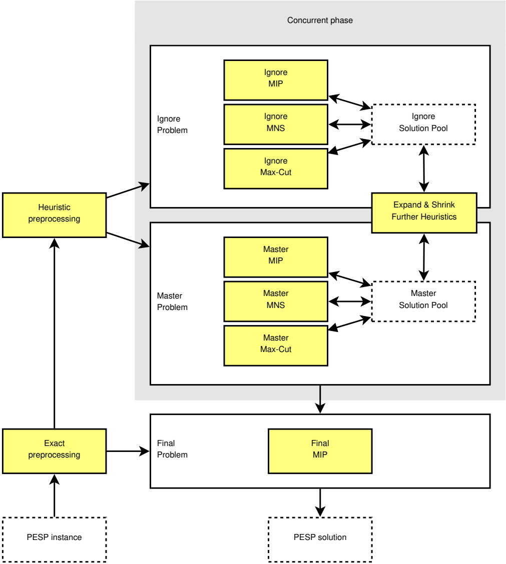
- Two problems at the same time: Master, Ignore
- Master: preprocessed input instance, does not change
- Ignore: ignore light free activities, can change
- Each problem is tackled with MIP, MNS, Max-Cut

Concurrent Solver Architecture



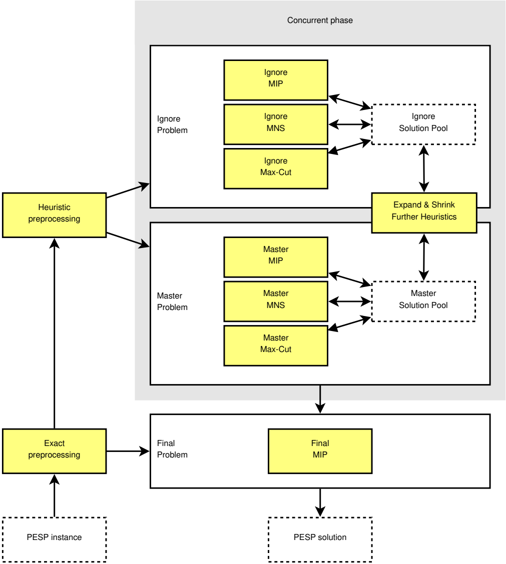
- Two problems at the same time: Master, Ignore
- Master: preprocessed input instance, does not change
- Ignore: ignore light free activities, can change
- Each problem is tackled with MIP, MNS, Max-Cut
- Algorithms run in parallel and talk to a common solution pool

Concurrent Solver Architecture



- Two problems at the same time: Master, Ignore
- Master: preprocessed input instance, does not change
- Ignore: ignore light free activities, can change
- Each problem is tackled with MIP, MNS, Max-Cut
- Algorithms run in parallel and talk to a common solution pool
- Further heuristic: Solve LP for fixed integer variables

Concurrent Solver Architecture



- Two problems at the same time: Master, Ignore
- Master: preprocessed input instance, does not change
- Ignore: ignore light free activities, can change
- Each problem is tackled with MIP, MNS, Max-Cut
- Algorithms run in parallel and talk to a common solution pool
- Further heuristic: Solve LP for fixed integer variables
- Initial solution: SAT solver

Concurrent Solver Features

MIP Features

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule
- single-node and multi-node cuts

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule
- single-node and multi-node cuts
- tabu search

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule
- single-node and multi-node cuts
- tabu search

More Features

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule
- single-node and multi-node cuts
- tabu search

More Features

- maximally improving delay cuts using SCIP as MIP solver

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule
- single-node and multi-node cuts
- tabu search

More Features

- maximally improving delay cuts using SCIP as MIP solver
- SAT and MaxSAT strategies

Concurrent Solver Features

MIP Features

- Solver interface: SCIP, CPLEX
- Model: incidence matrix, cycle matrix, (change-)cycle inequalities, fundamental or minimum undirected cycle basis
- Callbacks: heuristic (change-)cycle separator, SAT propagator

MNS Features

- modulo network simplex implementation with quality-first pivot rule
- single-node and multi-node cuts
- tabu search

More Features

- maximally improving delay cuts using SCIP as MIP solver
- SAT and MaxSAT strategies
- work in progress: divide and conquer

Maximum Cut Heuristic

Delay Cuts

A delay cut is a pair (S, d) consisting of a subset S of the events and a shift $d \in \{1, \ldots, T - 1\}$.

Maximum Cut Heuristic

Delay Cuts

A delay cut is a pair (S, d) consisting of a subset S of the events and a shift $d \in \{1, \dots, T-1\}$.

Improving Delay Cuts

If π is a periodic timetable, then a delay cut (S, d) produces a new timetable $\pi^{(S,d)}$ by setting

$$\pi_i^{(S,d)} := \begin{cases} (\pi_i + d) \bmod T & \text{if } i \in S, \\ \pi_i & \text{otherwise.} \end{cases}$$

Caveat: This timetable might violate some bounds.

Maximum Cut Heuristic

Delay Cuts

A delay cut is a pair (S, d) consisting of a subset S of the events and a shift $d \in \{1, \dots, T-1\}$.

Improving Delay Cuts

If π is a periodic timetable, then a delay cut (S, d) produces a new timetable $\pi^{(S,d)}$ by setting

$$\pi_i^{(S,d)} := \begin{cases} (\pi_i + d) \bmod T & \text{if } i \in S, \\ \pi_i & \text{otherwise.} \end{cases}$$

Caveat: This timetable might violate some bounds.

Theorem (–, 2019)

For fixed d, a maximally improving feasible delay cut (S, d) can be found by solving a maximum cut problem with positive and negative weights.

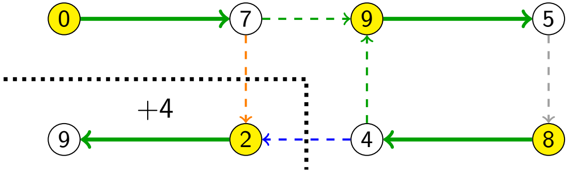
Escaping Local Optima

Examples of Delay Cuts

Escaping Local Optima

Examples of Delay Cuts

- *Modulo network simplex loop* (Nachtigall/Opitz, 1998):
  An exchange move of the modulo network simplex is a delay cut corresponding to the fundamental cut of a spanning tree arc. The delay depends on the co-tree arc.



Escaping Local Optima

Examples of Delay Cuts

- *Modulo network simplex loop* (Nachtigall/Opitz, 1998):
  An exchange move of the modulo network simplex is a delay cut corresponding to the fundamental cut of a spanning tree arc. The delay depends on the co-tree arc.
- *Single-node cuts* (Nachtigall/Opitz, 1998): Delay cuts with $|S| = 1$.

Escaping Local Optima

Examples of Delay Cuts

- *Modulo network simplex loop* (Nachtigall/Opitz, 1998):
  An exchange move of the modulo network simplex is a delay cut corresponding to the fundamental cut of a spanning tree arc. The delay depends on the co-tree arc.
- *Single-node cuts* (Nachtigall/Opitz, 1998): Delay cuts with $|S| = 1$.
- *Waiting edge cuts* (Goerigk/Schöbel, 2012):
  Delay cuts with $|S| = 2$, the vertices of S are connected by an edge with small span $u - \ell$.

Escaping Local Optima

Examples of Delay Cuts

- *Modulo network simplex loop* (Nachtigall/Opitz, 1998):
  An exchange move of the modulo network simplex is a delay cut corresponding to the fundamental cut of a spanning tree arc. The delay depends on the co-tree arc.
- *Single-node cuts* (Nachtigall/Opitz, 1998): Delay cuts with $|S| = 1$.
- *Waiting edge cuts* (Goerigk/Schöbel, 2012):
  Delay cuts with $|S| = 2$, the vertices of S are connected by an edge with small span $u - \ell$.
- *Multi-node cuts* (Goerigk/Schöbel, 2012):
  Delay cuts obtained by a greedy procedure.

Escaping Local Optima

Examples of Delay Cuts

- *Modulo network simplex loop* (Nachtigall/Opitz, 1998):
  An exchange move of the modulo network simplex is a delay cut corresponding to the fundamental cut of a spanning tree arc. The delay depends on the co-tree arc.
- *Single-node cuts* (Nachtigall/Opitz, 1998): Delay cuts with $|S| = 1$.
- *Waiting edge cuts* (Goerigk/Schöbel, 2012):
  Delay cuts with $|S| = 2$, the vertices of S are connected by an edge with small span $u - \ell$.
- *Multi-node cuts* (Goerigk/Schöbel, 2012):
  Delay cuts obtained by a greedy procedure.

Corollary

Delay cuts are "more global": If a periodic timetable cannot be improved by a delay cut, then it cannot be improved by any the above strategies.

§3

Benchmarks

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$
- cyclomatic number μ from 2 722 to 9 371

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$
- cyclomatic number μ from 2 722 to 9 371
- no instance solved to proven optimality

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$
- cyclomatic number μ from 2 722 to 9 371
- no instance solved to proven optimality
- biggest instance is part of MIPLIB 2017

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$
- cyclomatic number μ from 2 722 to 9 371
- no instance solved to proven optimality
- biggest instance is part of MIPLIB 2017

Short History: Computing Power vs Algorithmic Power

- 2004: U-Bahn Berlin ($\mu = 184$), 0.5 s, 4% gap, CPLEX + cycle basis

Hard PESP Instances

PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$
- cyclomatic number μ from 2 722 to 9 371
- no instance solved to proven optimality
- biggest instance is part of MIPLIB 2017

Short History: Computing Power vs Algorithmic Power

- 2004: U-Bahn Berlin ($\mu = 184$), 0.5 s, 4% gap, CPLEX + cycle basis
- 2008: timtab2 ($\mu = 294$), 22 h, optimal, CPLEX + user cuts

Hard PESP Instances

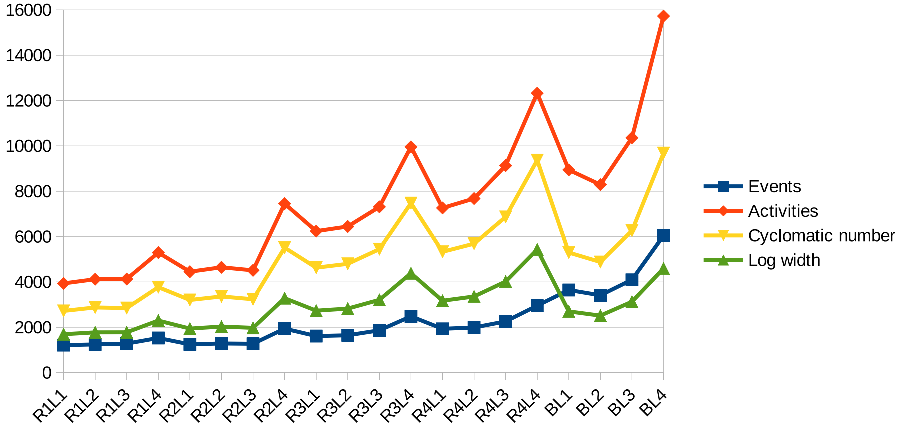
PESPlib

- `num.math.uni-goettingen.de/~m.goerigk/pesplib`
- est. 2012 by Goerigk
- 16 railway instances, 4 bus instances, period time $T = 60$
- cyclomatic number μ from 2 722 to 9 371
- no instance solved to proven optimality
- biggest instance is part of MIPLIB 2017

Short History: Computing Power vs Algorithmic Power

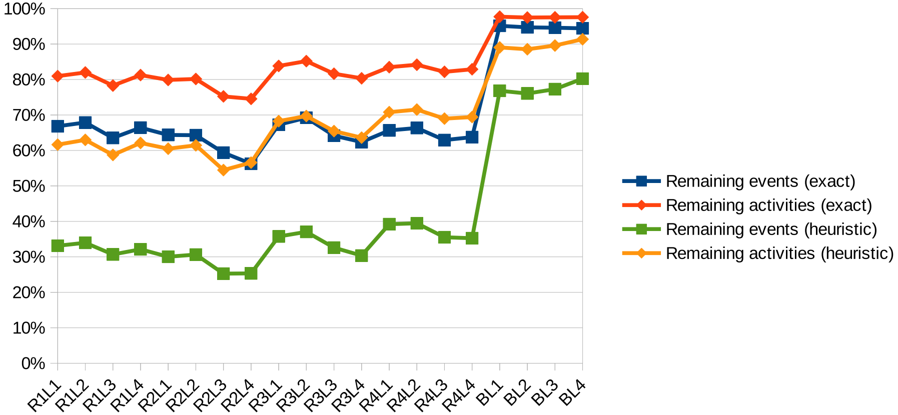
- 2004: U-Bahn Berlin ($\mu = 184$), 0.5 s, 4% gap, CPLEX + cycle basis
- 2008: timtab2 ($\mu = 294$), 22 h, optimal, CPLEX + user cuts
- 2016: timtab2 ($\mu = 294$), 1.78 h, optimal, ParaXpress @ 6 144 cores

PESPlib: Difficulty



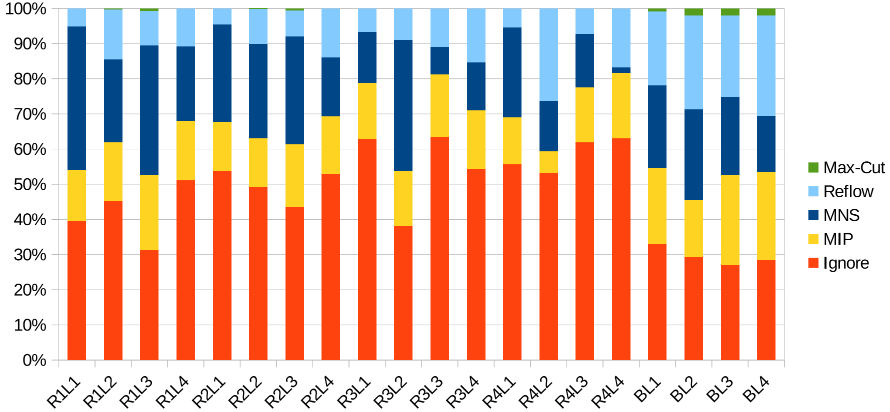
Log width: $\log_{10}$ of combinations of values for the integer variables

PESPlib: Preprocessing



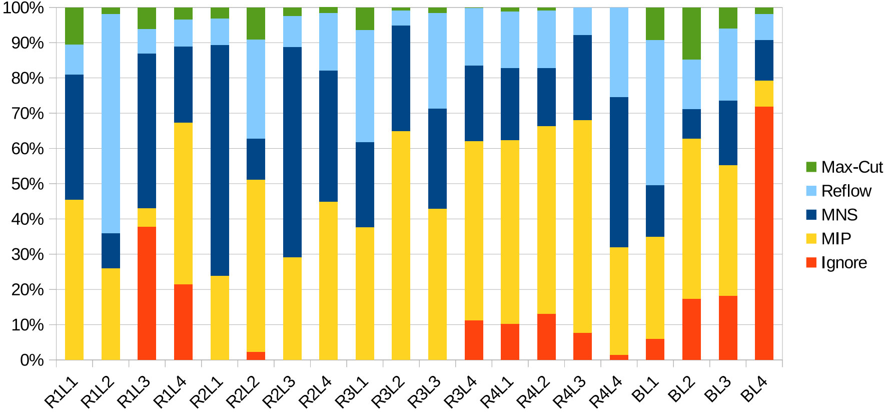
Exact preprocessing: remove bridges & isolated events, contract fixed arcs
Heuristic preprocessing: exact preprocessing, contract events of degree 2

PESPlib: Objective Value Improvement by Algorithm



Round 1: 20 minutes
best of 10

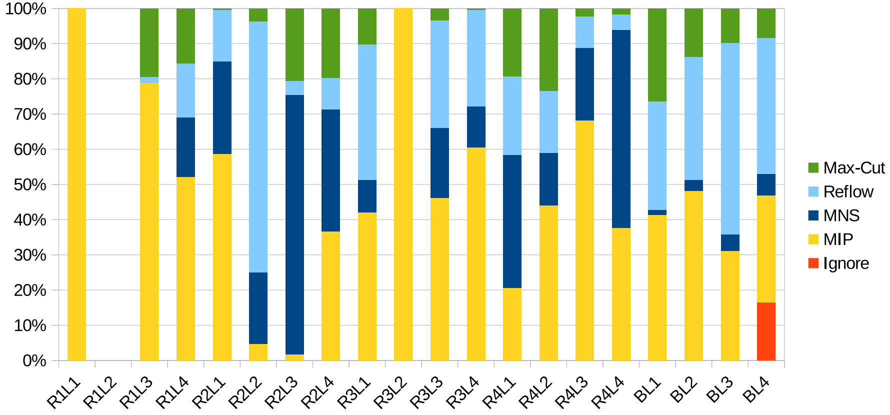
PESPlib: Objective Value Improvement by Algorithm



Round 2: 60 minutes

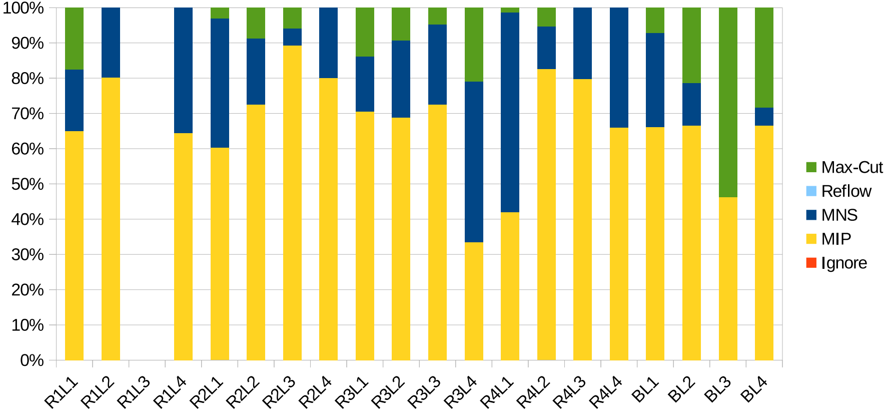
best of 10

PESPlib: Objective Value Improvement by Algorithm



Round 3: 4 hours
best of 1

PESPlib: Objective Value Improvement by Algorithm



Round 4: 8 hours
best of 1, no ignore problem

PESPlib: Primal Results

Instance	SAT start	Exp. 1 20 min	Exp. 2 1 h	Exp. 3 4 h	Exp. 4 8 h	Improvement
R1L1	74 234 870	30 861 021	30 501 068	30 493 800	30 463 638	1.03%
R1L2	72 731 210	30 891 284	30 516 991	30 516 991	30 507 180	3.71%
R1L3	71 682 438	30 348 596	29 335 021	29 319 593	29 319 593	3.26%
R1L4	67 395 169	27 635 070	26 738 840	26 690 573	26 516 727	2.96%
R2L1	97 230 766	42 863 646	42 598 548	42 463 738	42 422 038	0.19%
R2L2	95 898 935	42 024 414	41 149 768	40 876 575	40 642 186	2.15%
R2L3	93 800 082	39 054 513	38 924 083	38 881 659	38 558 371	3.47%
R2L4	84 605 216	33 256 602	32 707 981	32 548 415	32 483 894	1.75%
R3L1	92 939 173	44 216 552	43 521 250	43 460 397	43 271 824	2.53%
R3L2	91 336 260	45 829 180	45 442 171	45 401 718	45 220 083	1.80%
R3L3	89 741 119	42 112 858	41 103 062	41 005 379	40 849 585	4.63%
R3L4	74 142 083	34 589 170	34 018 560	33 454 773	33 335 852	3.91%
R4L1	98 276 297	50 638 727	49 970 330	49 582 677	49 426 919	4.30%
R4L2	101 135 698	50 514 805	49 379 256	49 018 380	48 764 793	1.64%
R4L3	96 629 751	46 406 365	45 656 395	45 530 113	45 493 081	0.85%
R4L4	80 446 905	40 706 349	38 884 544	38 695 188	38 381 922	1.17%
BL1	15 367 998	7 299 228	6 394 914	6 375 778	6 333 641	14.27%
BL2	16 046 736	7 378 468	6 837 447	6 819 856	6 799 331	16.51%
BL3	14 850 854	7 512 685	7 065 270	7 011 324	6 999 313	10.57%
BL4	15 618 608	7 997 783	7 330 393	6 738 582	6 562 147	10.84%
		10 better	18 better	20 better	20 better	

PESPlib: Dual Results

Instance	Dual bound	PESPlib improvement	Optimality gap
R1L1	19 878 200	17.64%	34.75%
R1L2	19 414 800	290.22%	36.36%
R1L3	18 786 300	189.09%	35.93%
R1L4	16 822 200	167.11%	36.56%
R2L1	25 082 000	163.82%	40.88%
R2L2	24 867 400	220.09%	38.81%
R2L3	23 152 300	181.49%	39.96%
R2L4	18 941 500	263.07%	41.69%
R3L1	25 077 800	217.16%	42.05%
R3L2	25 272 600	240.02%	44.11%
R3L3	21 642 500	226.52%	47.02%
R3L4	16 479 500	193.04%	50.57%
R4L1	27 243 900	170.03%	44.88%
R4L2	26 368 200	230.63%	45.93%
R4L3	22 701 400	203.62%	50.10%
R4L4	15 840 600	207.75%	58.73%
BL1	3 668 148	148.26%	42.08%
BL2	3 943 811	127.93%	42.00%
BL3	3 571 976	196.31%	48.97%
BL4	3 131 491	211.81%	52.28%

8 h, 6 threads

PESPlib: Conclusions

- Half of the instances could be improved within only 20 minutes.

PESPlib: Conclusions

- Half of the instances could be improved within only 20 minutes.
- Concurrency pays off: The speed-up compared to the sequential method of Goerigk/Liebchen is bigger than the number of threads.

PESPlib: Conclusions

- Half of the instances could be improved within only 20 minutes.
- Concurrency pays off: The speed-up compared to the sequential method of Goerigk/Liebchen is bigger than the number of threads.
- Solving to proven optimality currently seems to be out of reach: Given the relatively small primal improvements, there is a lot to do on the dual side.

A Concurrent Approach to the Periodic Event Scheduling Problem

Ralf Borndörfer, Niels Lindner, Sarah Roth

Zuse Institute Berlin

Berlin Mathematics Research Center MATH+

ZIB

RailNorrköping 2019

June 18, 2019