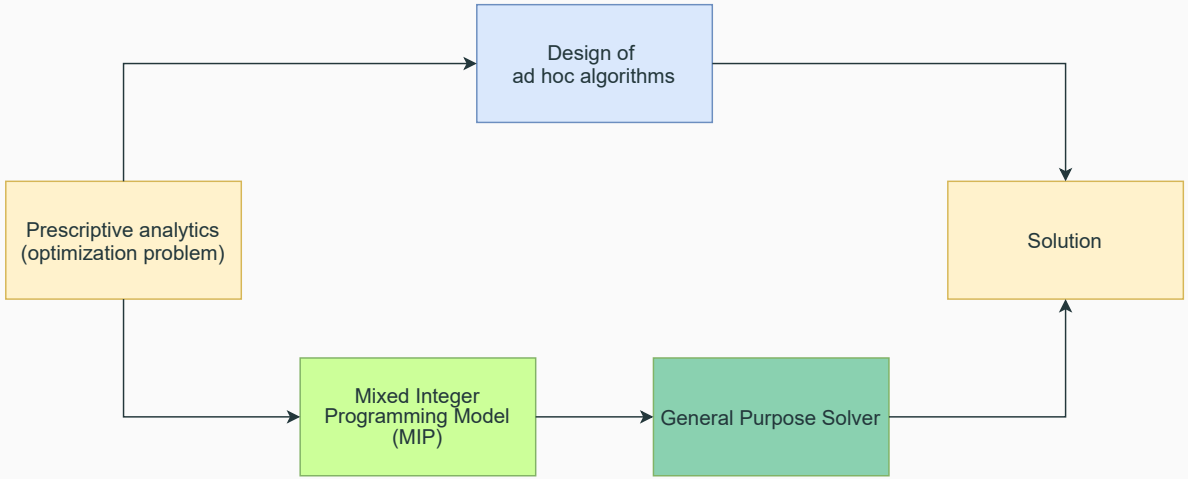
Università
degli Studi
di Milano

Dantzig-Wolfe: from data-driven decomposition to parallel resolution

Alberto Ceselli
Dipartimento di Informatica, Università degli Studi di Milano
Saverio Basso
SUPSI and IDSIA, Lugano

Column Generation Workshop 2023

Solving Prescriptive Analytics Optimization Problems



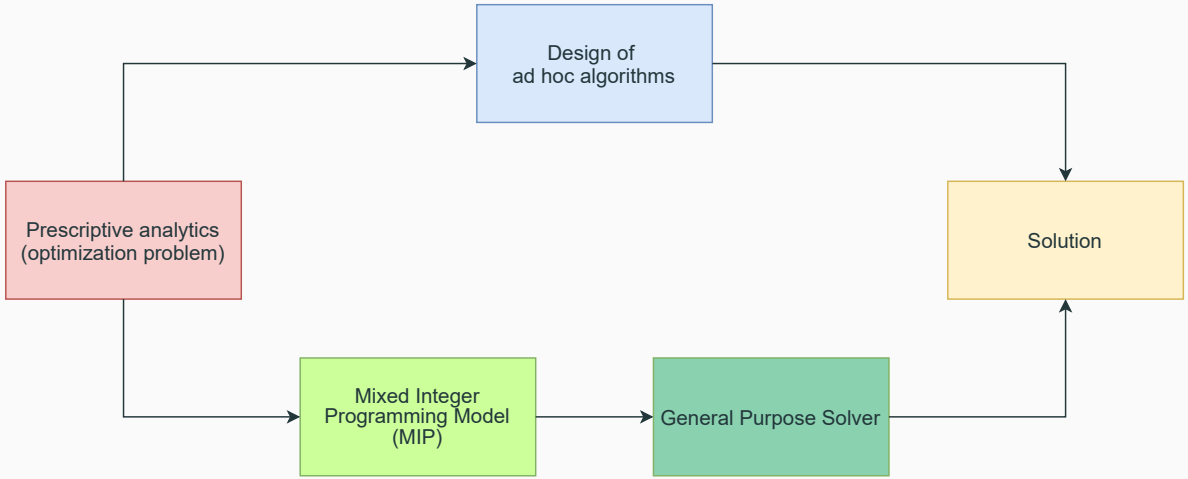
5	10.4	17.2	12	18.3	58	16	11

1	0	0	0	0	0	0	0	1
0	23.3	0	15	17	102	51	0	237
0	5.6	0	0	0	0	0	0	12.5
0	0	0	0	0	6.3	4.3	2.3	12.5
0	0	6.2	0	6.2	0	0	0	12.5
0	0	1	1	0	0	0	0	1
0	0	0	0	0	-1	1	0	1
7.4	0	0	3.1	0	-2.5	0	5.3	15

Use **branch-and-cut**

- problem complexity issues
- data size scalability issues

Solving Prescriptive Analytics Optimization Problems



5	10.4	17.2	12	18.3	58	16	11

1	0	0	0	0	0	0	0	1
0	23.3	0	15	17	102	51	0	237
0	5.6	0	0	0	0	0	0	12.5
0	0	0	0	0	6.3	4.3	2.3	12.5
0	0	6.2	0	6.2	0	0	0	12.5
0	0	1	1	0	0	0	0	1
0	0	0	0	0	-1	1	0	1
7.4	0	0	3.1	0	-2.5	0	5.3	15

1	0	0	0	0	0	0	0
0	5.6	0	0	0	0	0	0
0	0	6.2	0	6.2	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	-1	1	0
0	0	0	0	0	6.3	4.3	2.3
7.4	0	0	3.1	0	-2.5	0	5.3
0	23.3	0	15	17	102	51	0

1. to detect block structures by data driven decomposition methods
2. to exploit massive parallelism for optimization of large scale instances

Part I: Automatic Decomposition of MIPs

Looking for decomposition schemes:

- on the problem formulation (Frangioni et al, 2010 -)
- on the MIP instance (Wang Ralphs, 2013)

Detectors search suitable decomposition patterns of a given MIP instance

- by looking for previously known structures
- by exploiting *static* properties of that instance (Bergner Caprara Ceselli Furini Luebbecke Malaguti Traversi, 2015)
- by *communality* detection (Khaniyev Elhedhli Erenay, 2018)

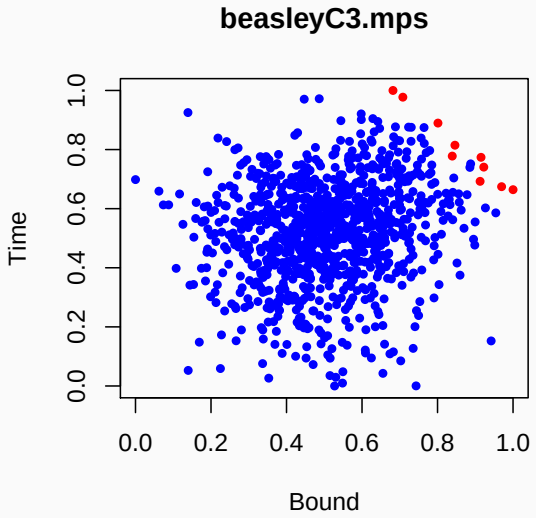
Generic branch-and-price-and-cut solvers:

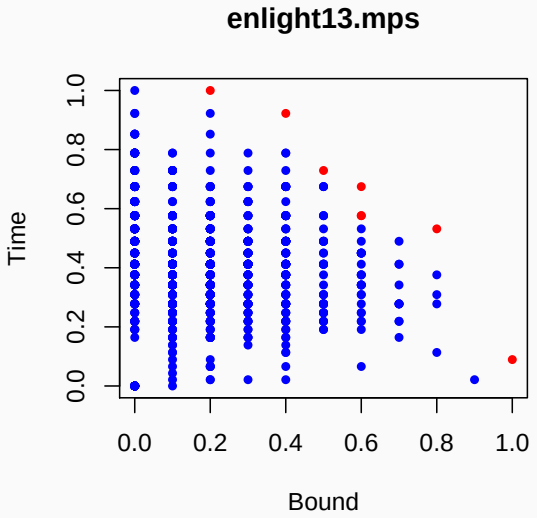
- GCG (Gamrath Luebbecke, 2010 -)
- DIP (Ralphs Galati, 2017 -)
- BapCod (Vanderbeck, 2017 -)
- Coluna.jl (Marquez Nesello Vanderbeck Pessoa Sadykov, 2023)

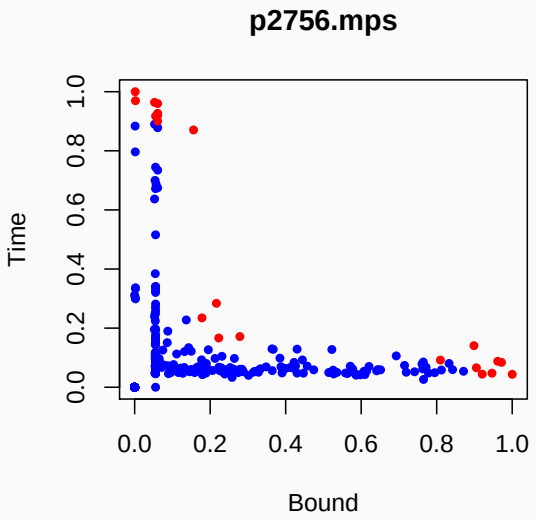
Machine learning to choose a decomposition algorithm (Kruber Luebbecke Parmentier, 2017)

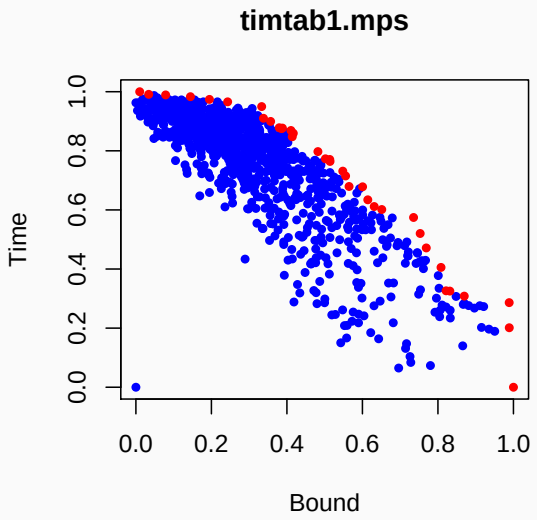
Promising Decompositions

Good decompositions: low running time, tight dual bound (Post-process)









Preliminary investigation

Research target
Investigate the link between static properties of MIP base instances and good decomposition patterns

Dataset A generation:

- 36 MIPLIB instances, generated 1000 random decompositions each
- Computed (117) static features (indep. on instance size)
- Optimized every decomposition, recorded time and bound

Experimental analysis of new decompositions from

- unknown decomp of *known* MIPs: base MIPs are part of Dataset A
- unknown decomp of *unknown* MIPs: new, unseen MIP instances

---

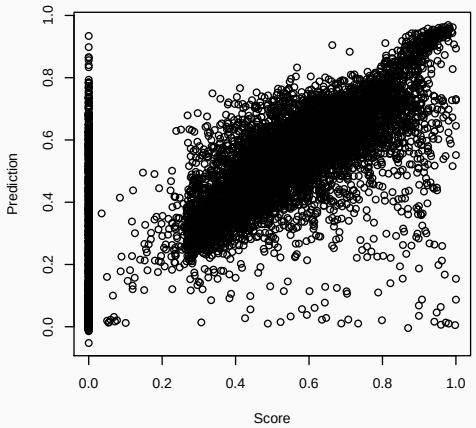
S. Basso, A. Ceselli, A. Tettamanzi "Random Sampling and Machine Learning to Understand Good Decompositions", Annals of Operations Research 284 (2018)

Mixing features: Regression for unknown problems

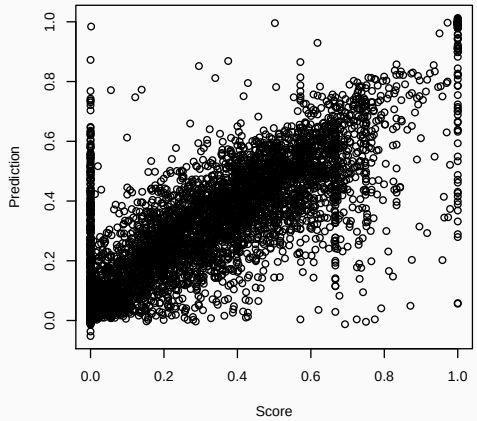
Independent Time and Bound regressors (XGboost). Preliminary results:

Known MIPs

Time

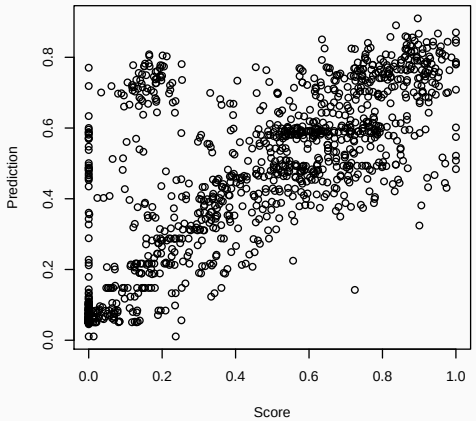


Bound

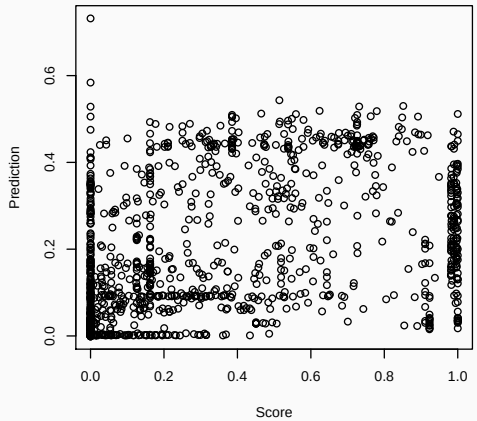


Unknown MIPs

Time



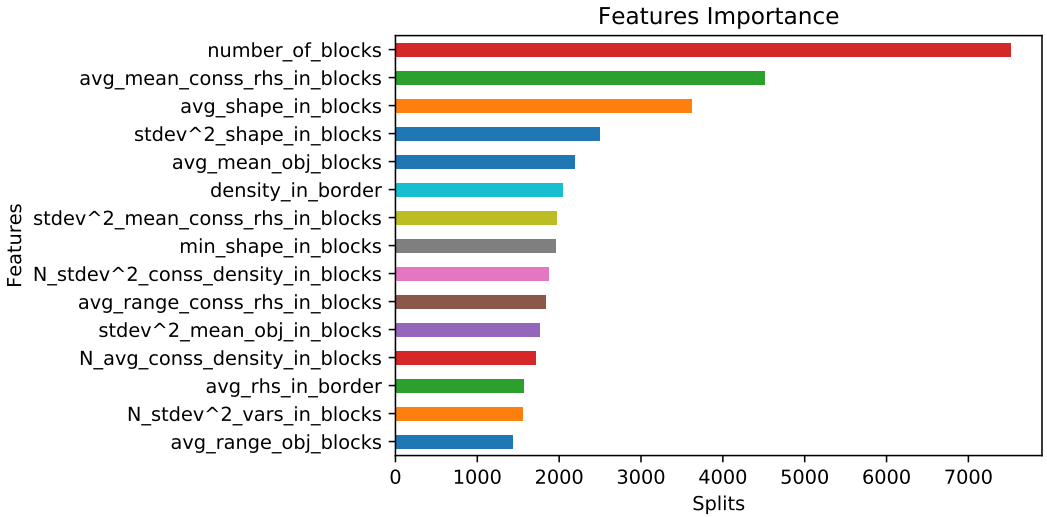
Bound

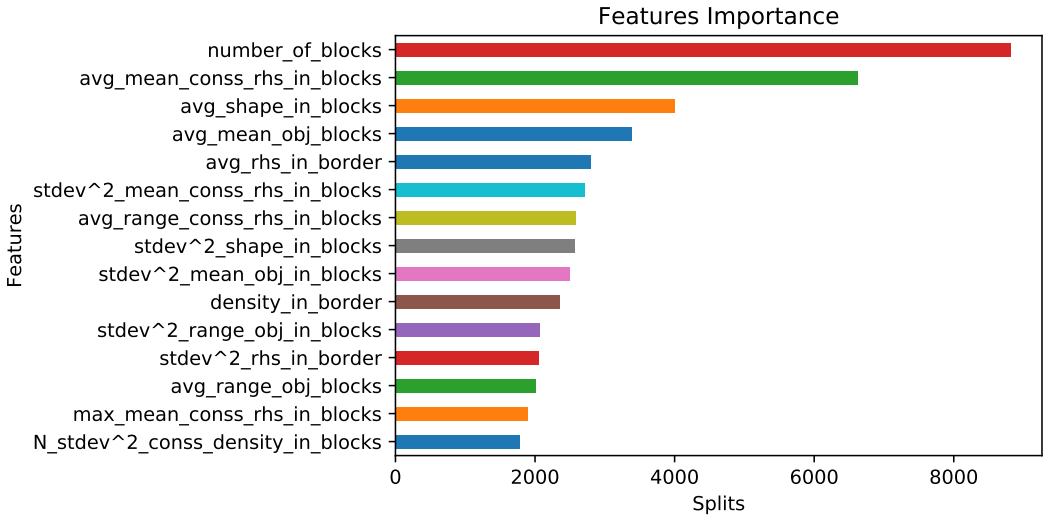


Time prediction always possible, Bound prediction out of reach

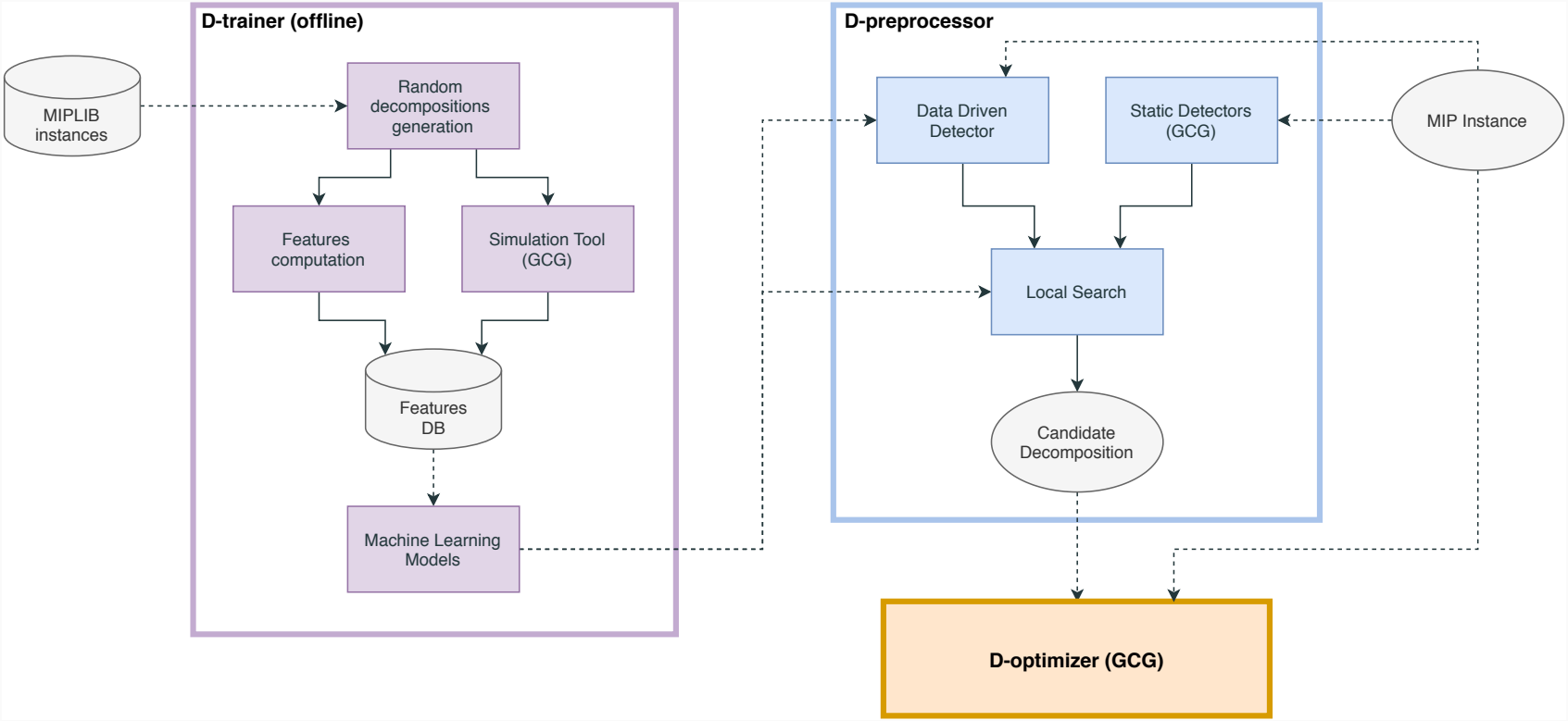
Feature importance

Time (top) and bound (bottom) regressors most important features:





A Data Driven Detection Framework

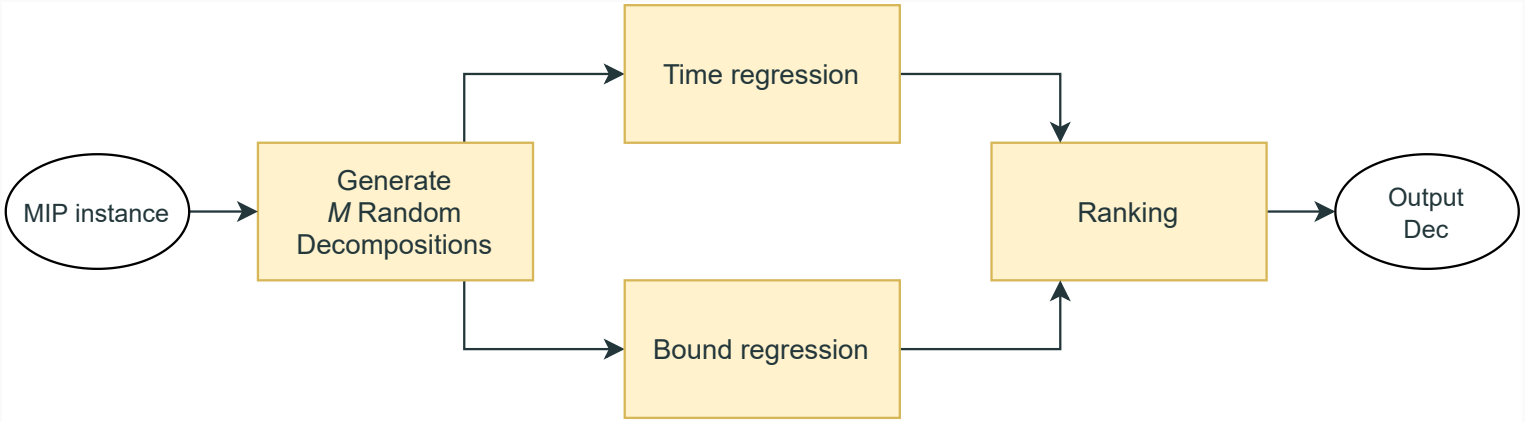


Three main components:

- D-trainer (Dataset A, xgboost regressors for Time and Bound)
- D-preprocessor
- D-optimizer (GCG)

A Data Driven Detector

Research target
Given a MIP instance, generate a suitable decomposition pattern



Ranking function D(i): Percentage of decompositions dominated by decomposition i

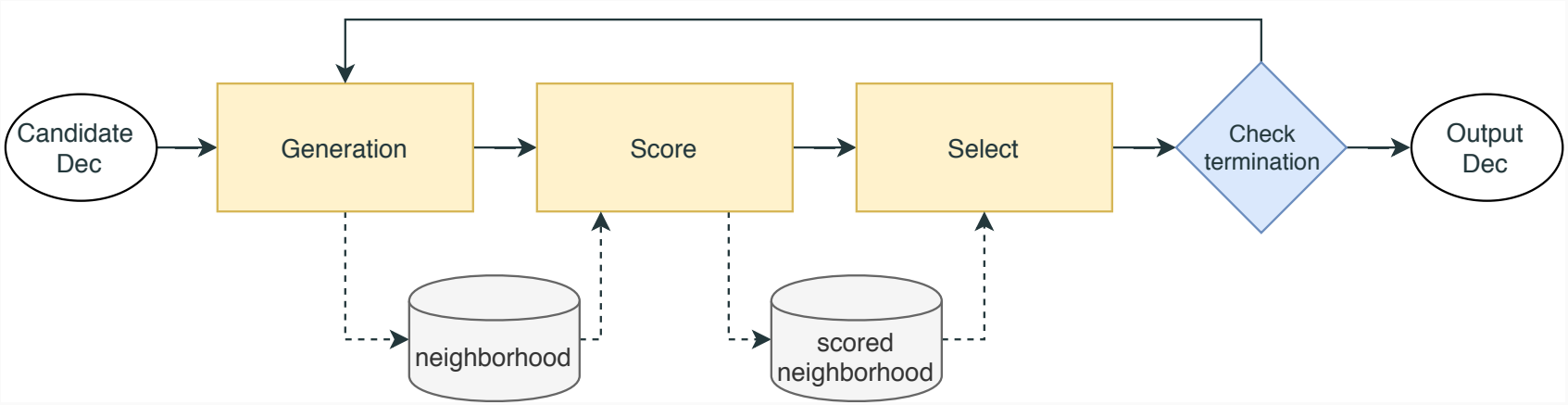
Given decompositions i and j, $i \to j$:

$$Time(i) \geq Time(j) \wedge Bound(i) > Bound(j), \quad \text{or}$$
$$Time(i) > Time(j) \wedge Bound(i) \geq Bound(j)$$

S. Basso, A. Ceselli "Computational evaluation of ranking models in an automatic decomposition framework", Proc. of EURO/ALIO 2018 (2018)

Local Search Algorithms

Research target
Given a base MIP instance, and a decomposition for it, improve it algorithmically



Starting decomposition:

static detectors or data driven detector

Neighbourhood:

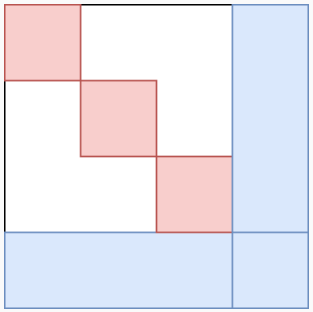
set of decompositions that differ from candidate decomposition of one constraint

---

S. Basso, A. Ceselli "Computational Evaluation of Data Driven Local Search for MIP Decompositions", Proc. of ODS 2019 (2019)

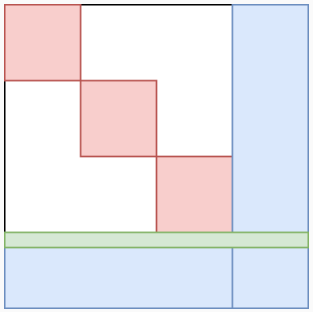
Generation

Generation: Choose one constraint from the border, insert into one block



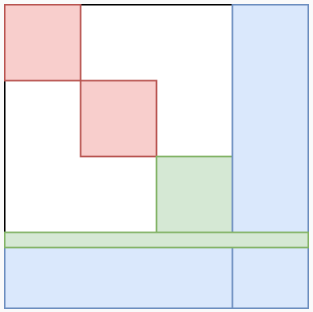
Generation

Generation: Choose one constraint from the border, insert into one block



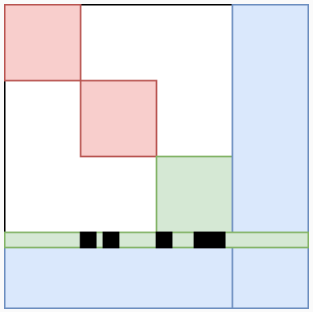
Generation

Generation: Choose one constraint from the border, insert into one block



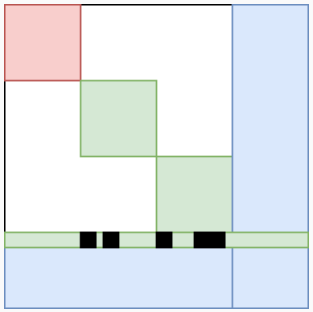
Generation

Generation: Choose one constraint from the border, insert into one block



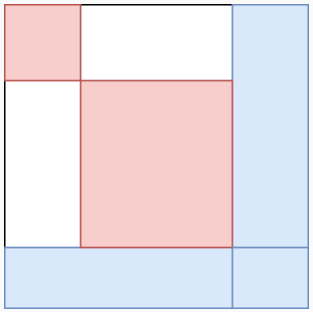
Generation

Generation: Choose one constraint from the border, insert into one block



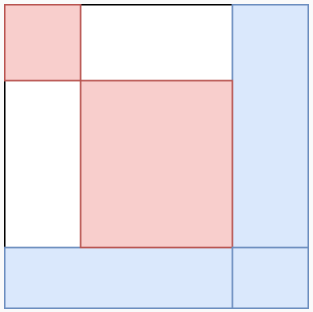
Generation

Generation: Choose one constraint from the border, insert into one block



Generation

Generation: Choose one constraint from the border, insert into one block



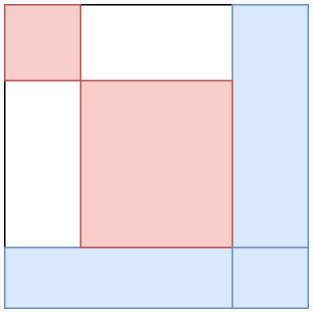
\+ bound cannot worsen

\- time prediction to avoid slow decompositions

Repeat for each constraint (border)/block combination

Generation

Generation: Choose one constraint from the border, insert into one block

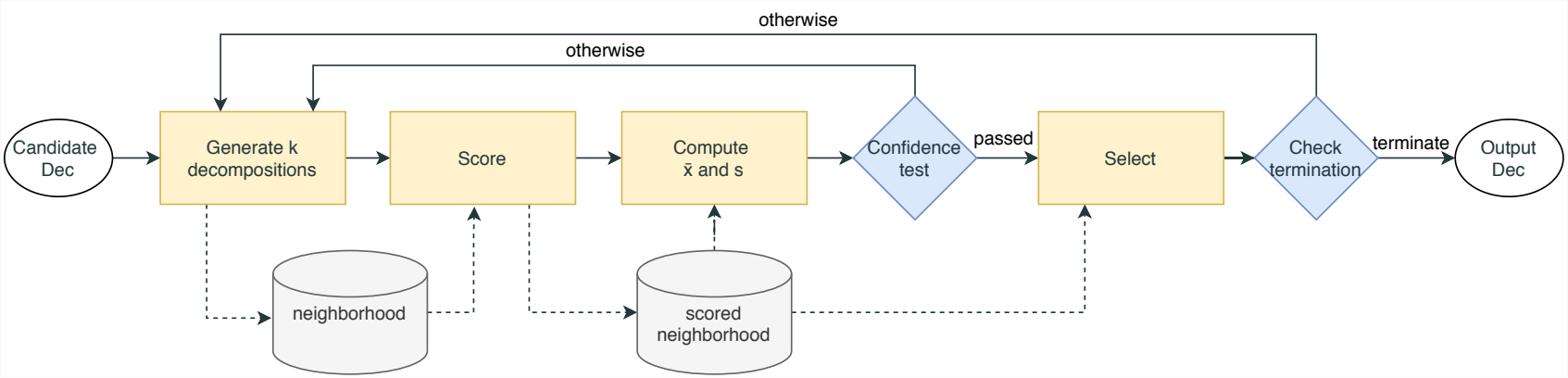


\+ bound cannot worsen

\- time prediction to avoid slow decompositions

Repeat for each constraint (border)/block combination

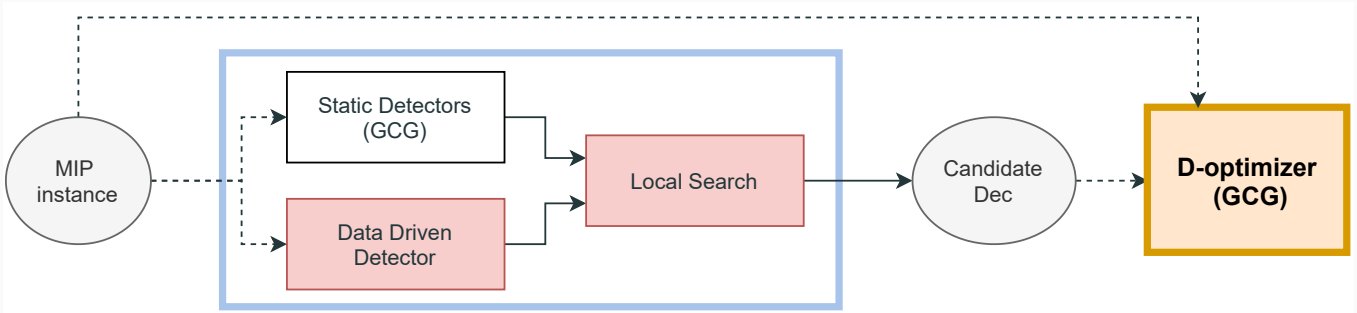
Faster computation: Sample a representative neighbourhood (**25x speedup**, little loss of quality)



Framework testing and Benchmark Configurations

Framework (D-preprocessor) configurations

D-preprocessor experimental setup



with sampling and orthogonal selection, termination: 85% convexification.

- C++11 custom library, Boost library, Python 3.6 scripts
- Intel i7-6700K CPU and 32GB RAM
- 30 unknown MIPLIB problems
- D-optimizer: GCG 3.0.1, 5 hours timelimit
- Benchmarks: GCG, SAS-Decomp

Root Node bounds comparison

DDW_{Sample}	SAS_{comm}		GCG		GCG_{Hmetis}		GCG_{Full}	
run 1	**DDW Best**	**10**	DDW Best	6	DDW Best	6	DDW Best	6
	Draw	4	**Draw**	**15**	**Draw**	**14**	**Draw**	**14**
	SAS Best	9	GCG Best	3	GCG Best	4	GCG Best	4
run 2	**DDW Best**	**10**	DDW Best	7	DDW Best	7	DDW Best	7
	Draw	5	**Draw**	**15**	**Draw**	**14**	**Draw**	**14**
	SAS Best	9	GCG Best	3	GCG Best	4	GCG Best	4
run 3	**DDW Best**	**10**	DDW Best	8	DDW Best	7	DDW Best	8
	Draw	5	**Draw**	**14**	**Draw**	**13**	**Draw**	**13**
	SAS Best	9	GCG Best	3	GCG Best	5	GCG Best	4

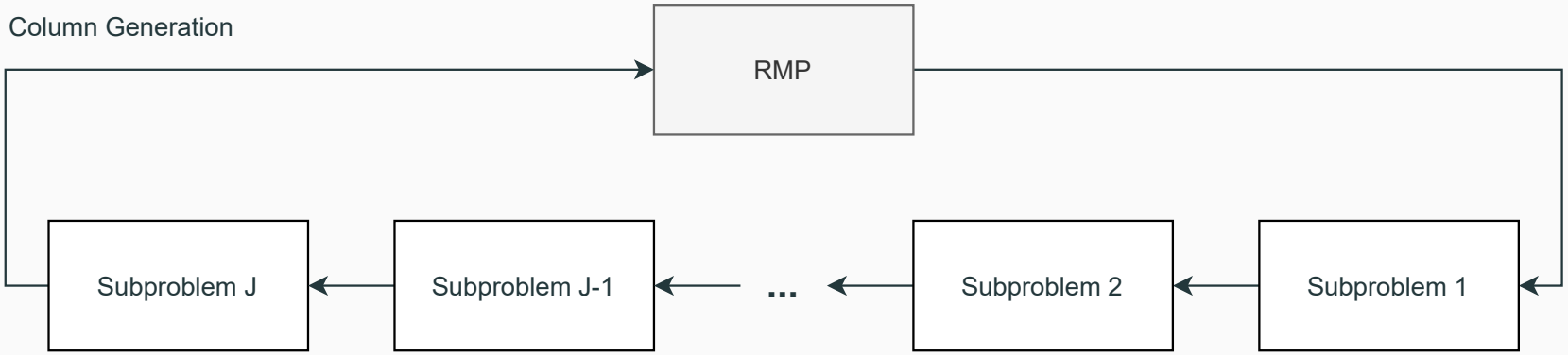
				DDW_{sample}		
	GCG	GCG_{Hmetis}	GCG_{Full}	run 1	run 2	run 3
Time [s]	905.17	416.92	847.79	344.08	347.41	420.92
Solver errors	1	1	0	6	5	5
Timeouts	14	13	14	6	5	7

SAS Decomp slightly faster, 6 timeouts

Part II: Concurrent Column Generation

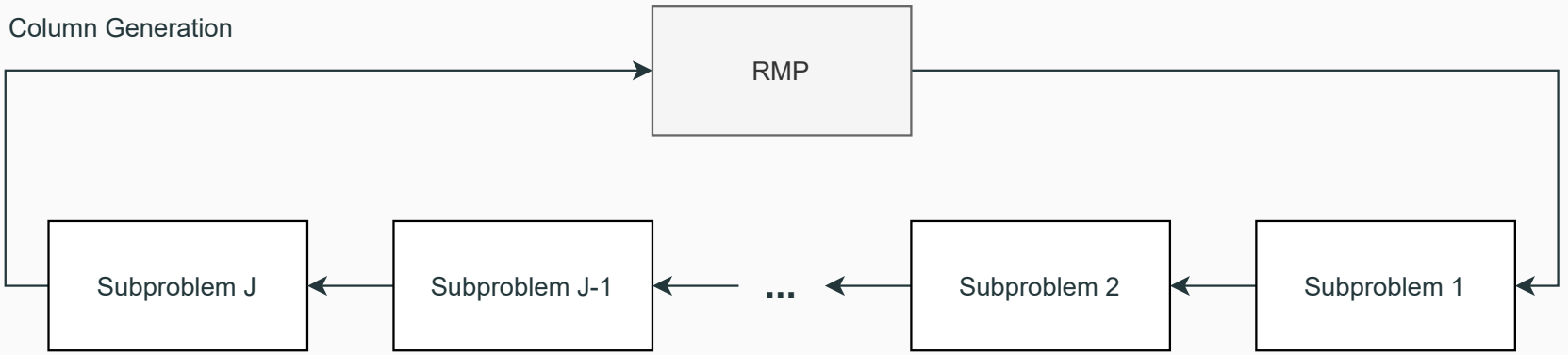
Column Generation (CG)

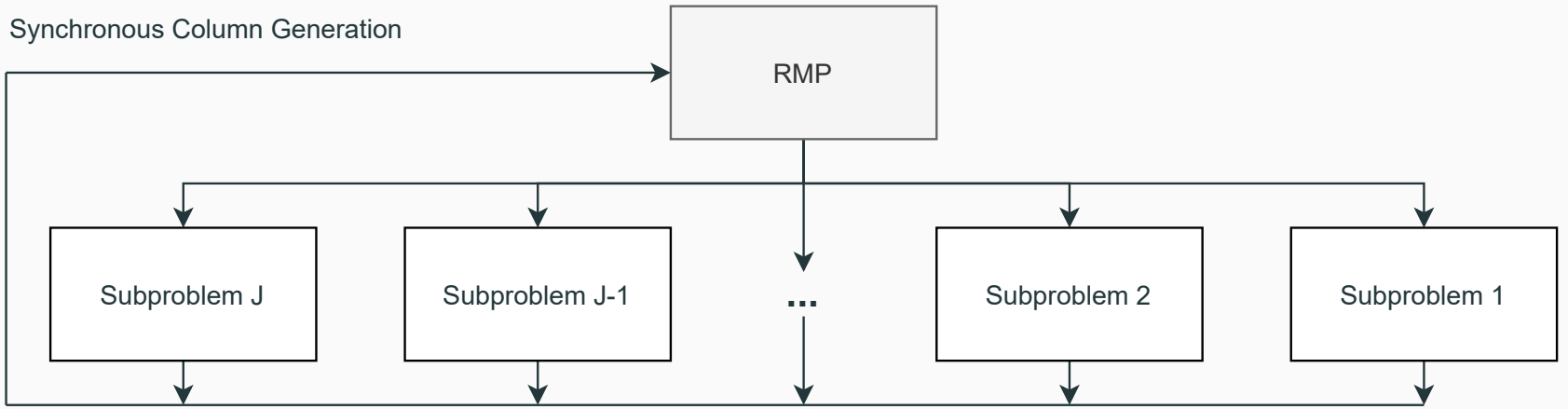
Research target
Given a base MIP instance, and a decomposition for it, solve it as quickly as possible



Column Generation (CG)

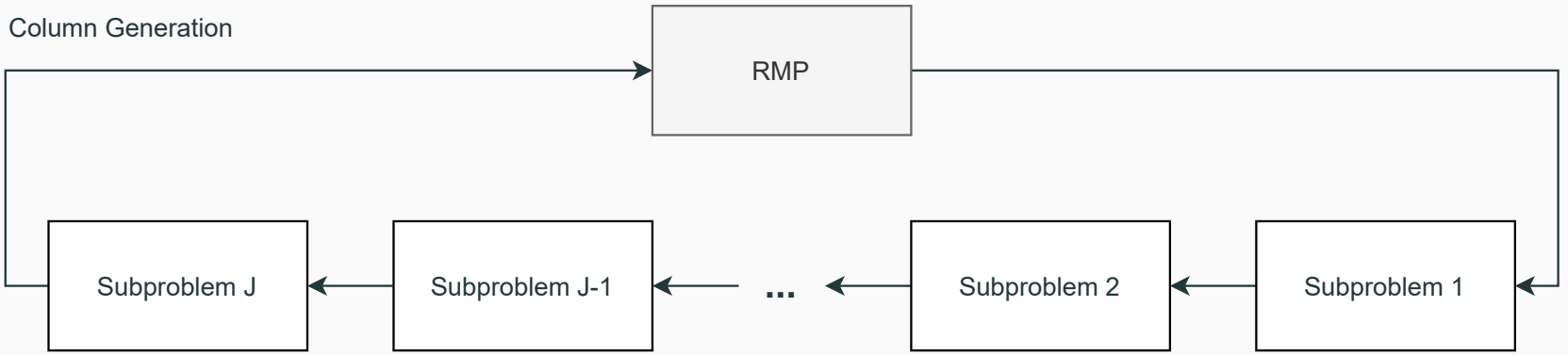
Research target
Given a base MIP instance, and a decomposition for it, solve it as quickly as possible

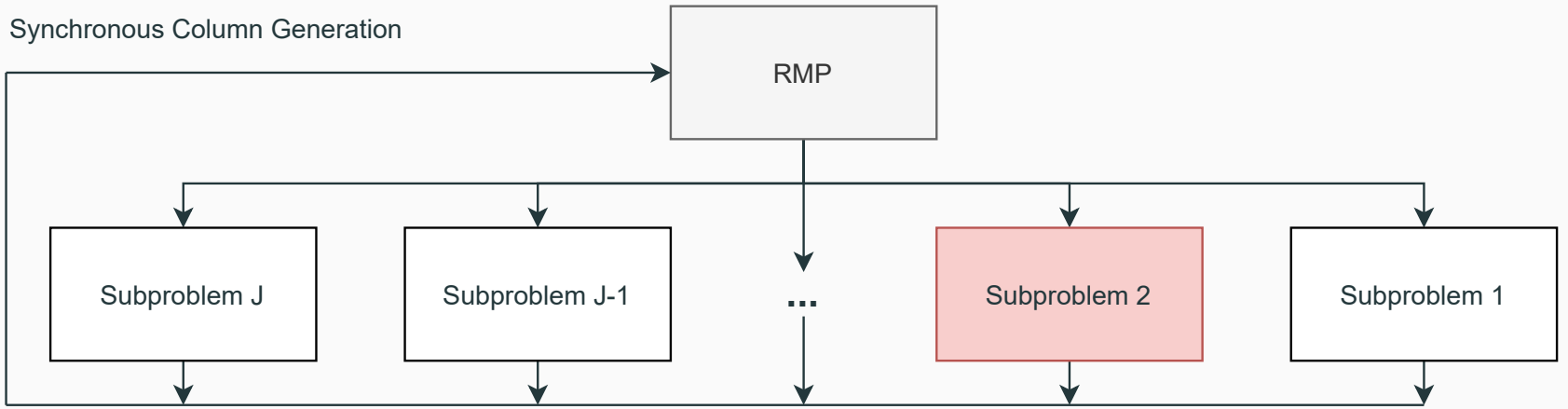




Column Generation (CG)

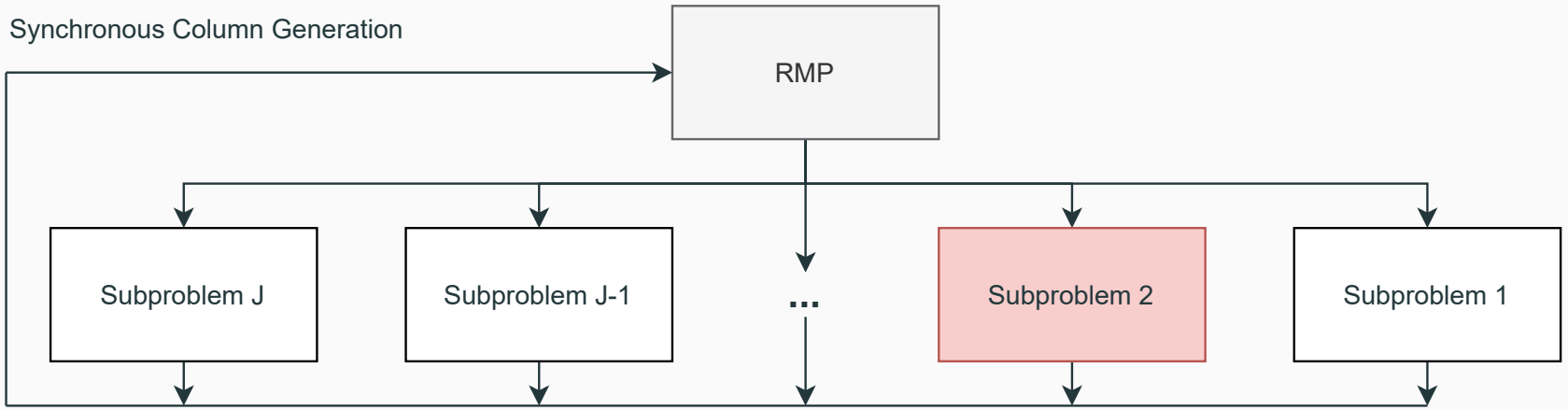
Research target
Given a base MIP instance, and a decomposition for it, solve it as quickly as possible





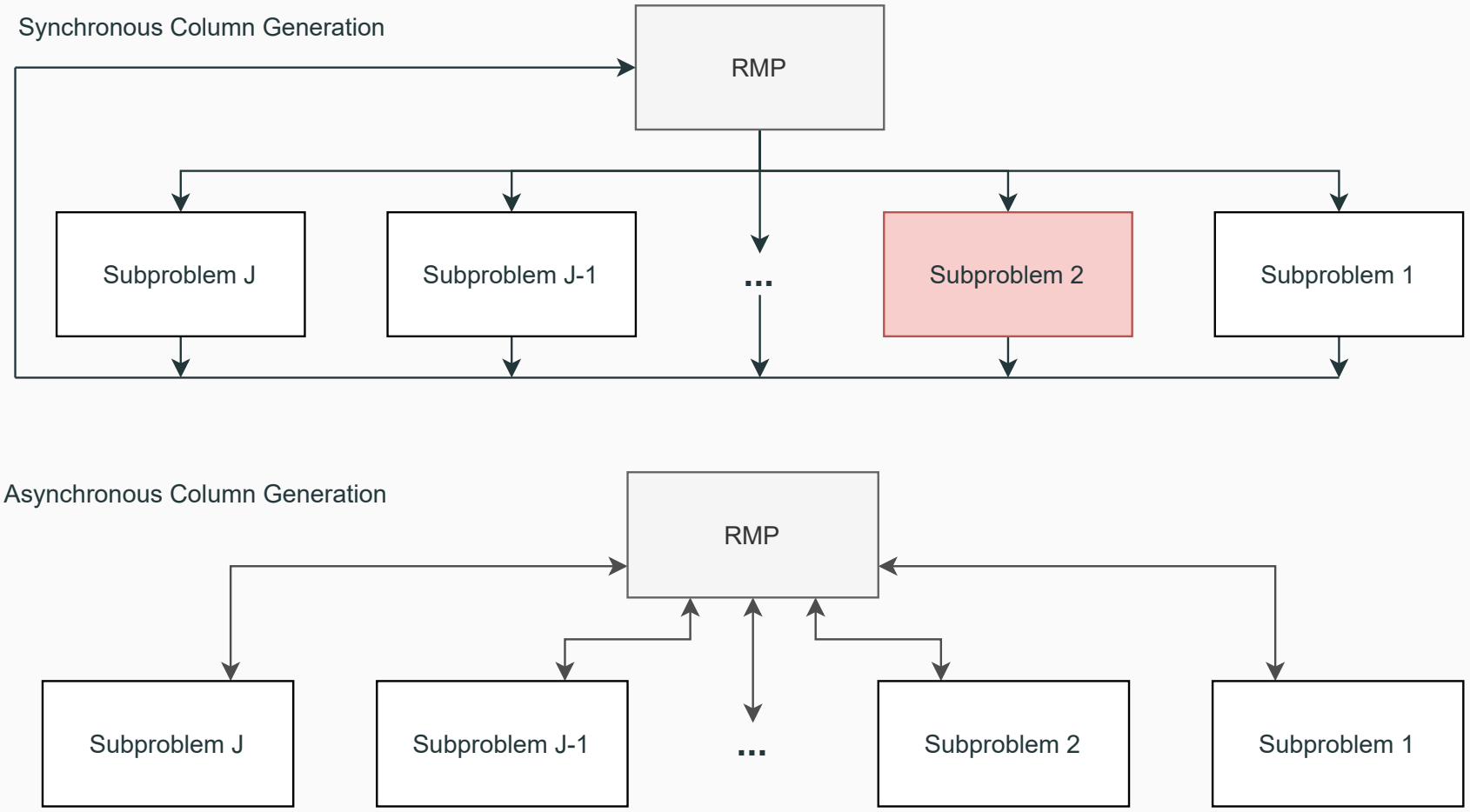
Column Generation (CG)

Research target
Given a base MIP instance, and a decomposition for it, solve it as quickly as possible

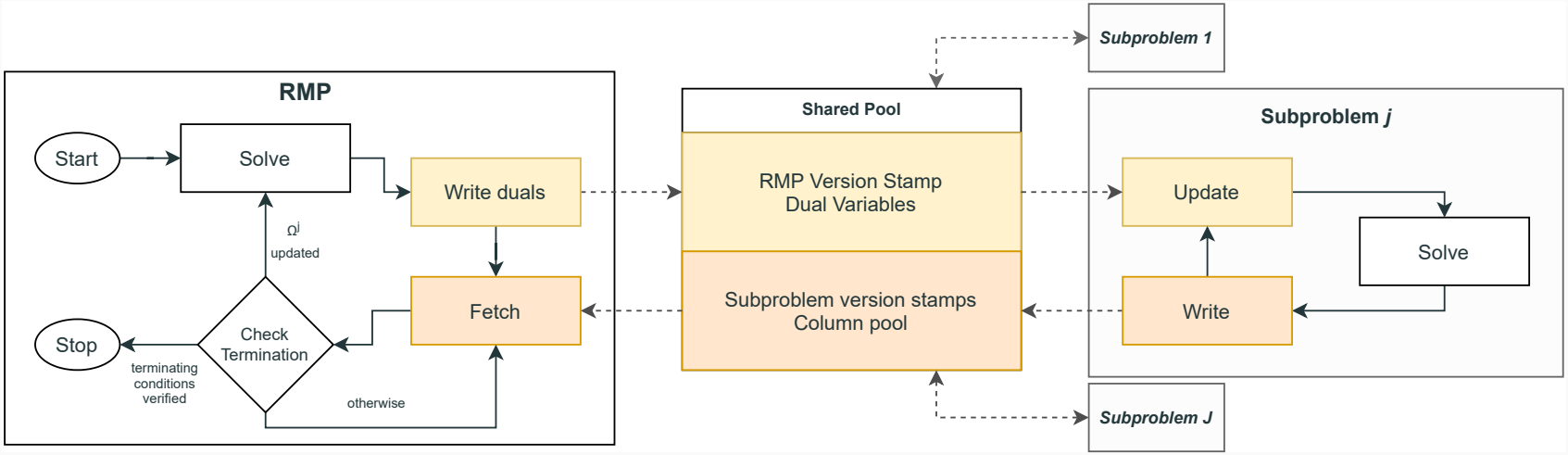


Column Generation (CG)

Research target
Given a base MIP instance, and a decomposition for it, solve it as quickly as possible



Asynchronous Column Generation (ACG)

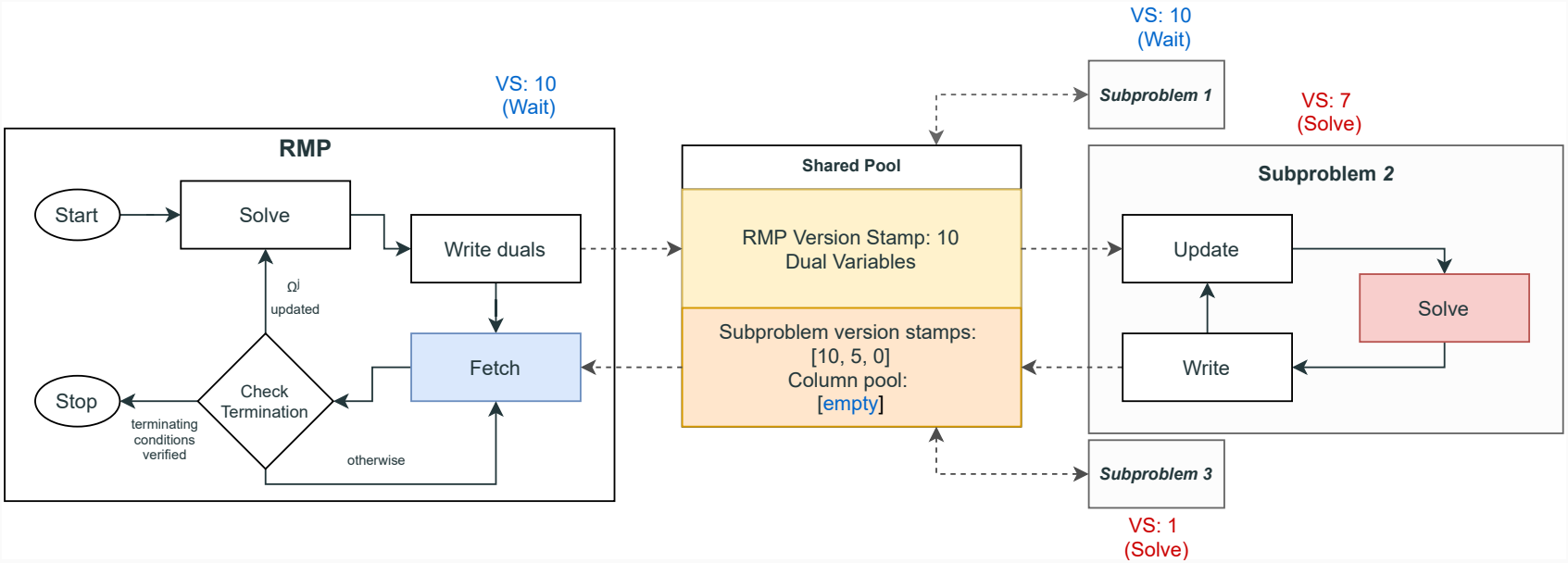


- RMP and subproblem tasks run in parallel
- Communication through shared pool (exclusive access)
- RMP version stamps: Tasks might work with different sets of dual variables
- Generic implementation: tasks solved are solved with branch-and-cut

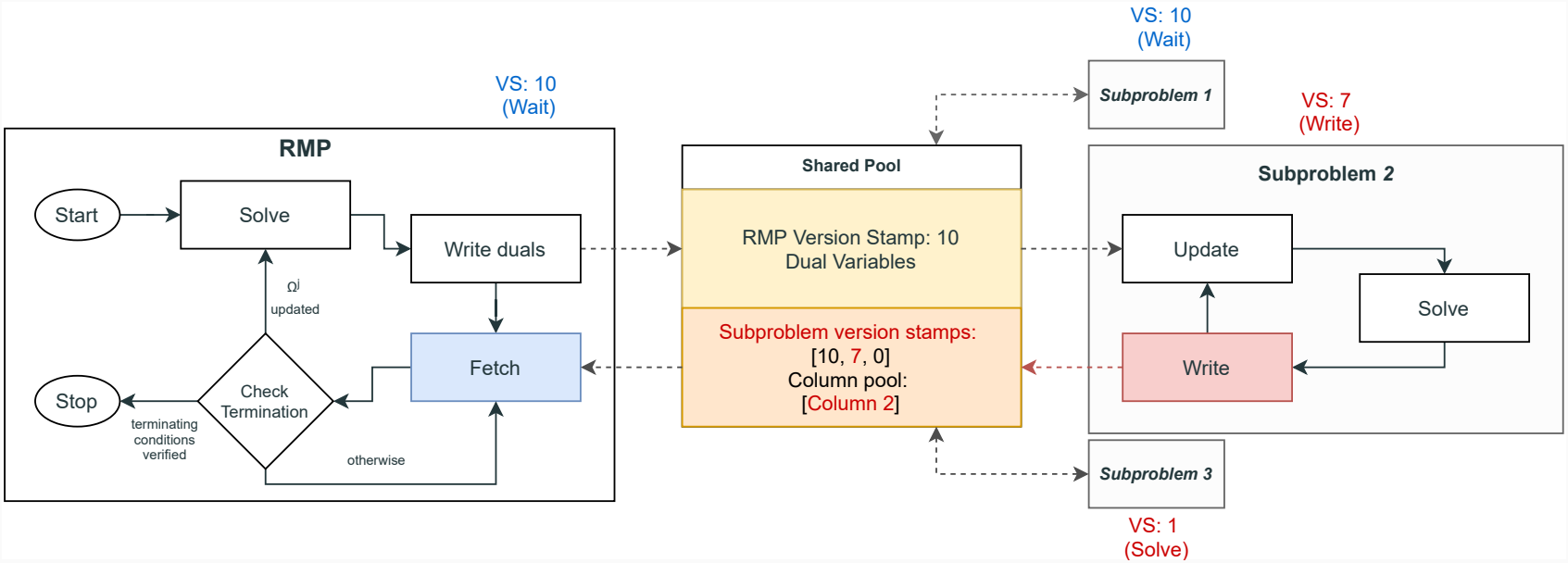
---

S. Basso, A. Ceselli "Asynchronous Column Generation", Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX) (2017)

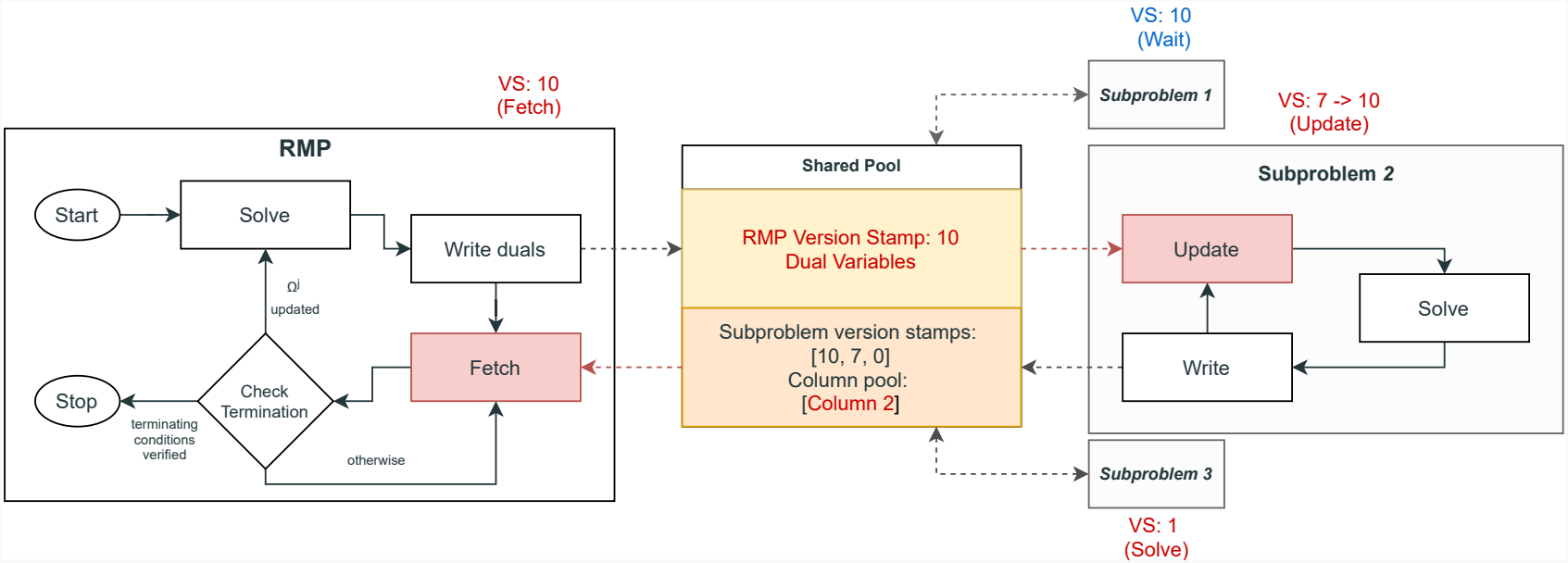
ACG example and terminating conditions



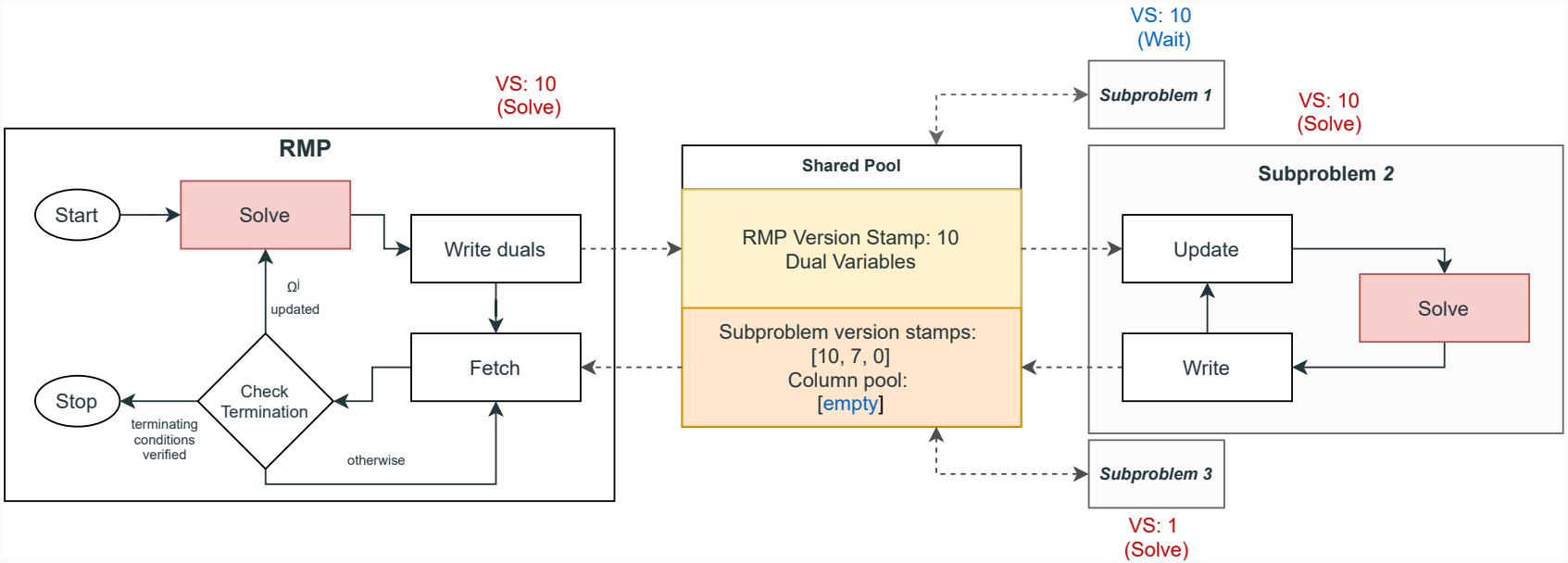
ACG example and terminating conditions



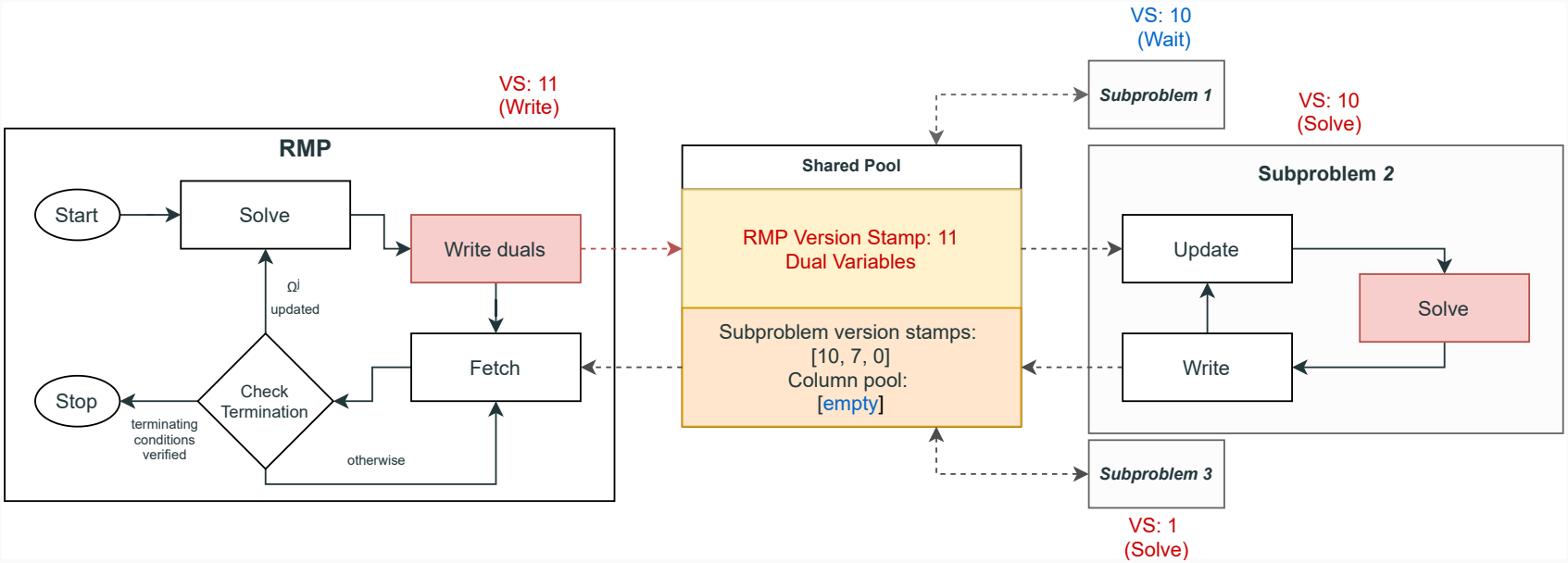
ACG example and terminating conditions



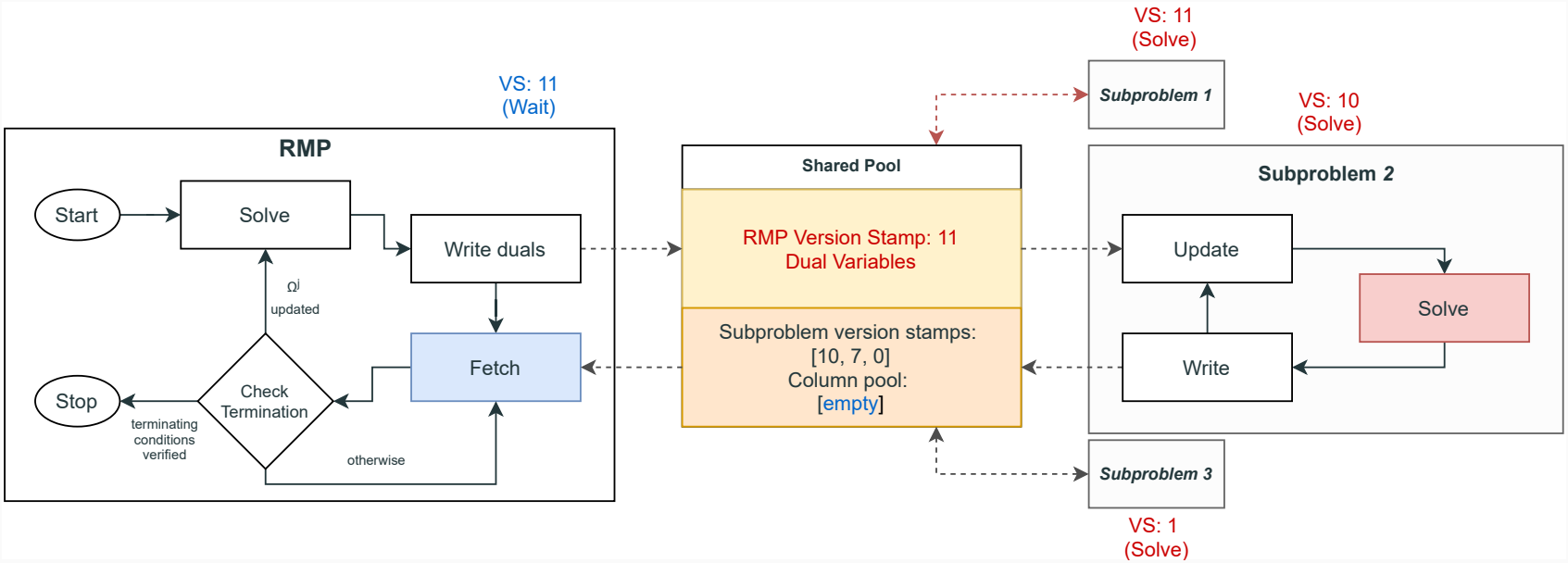
ACG example and terminating conditions



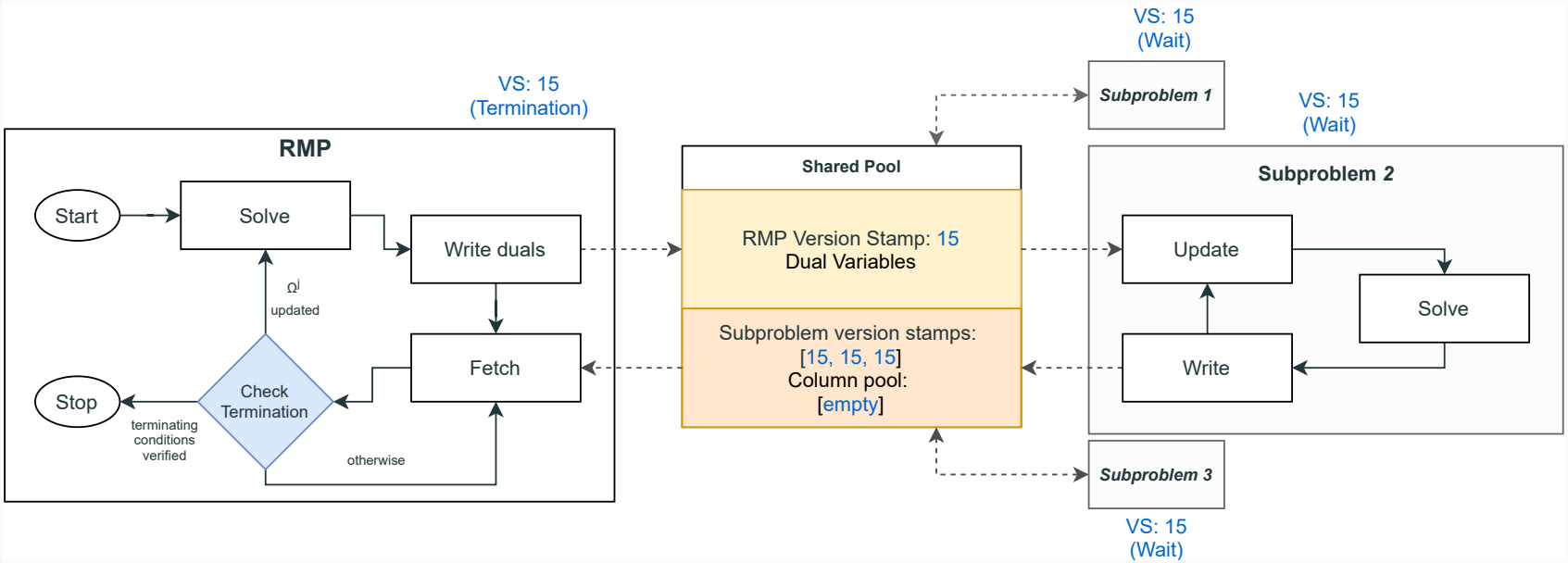
ACG example and terminating conditions



ACG example and terminating conditions



ACG example and terminating conditions

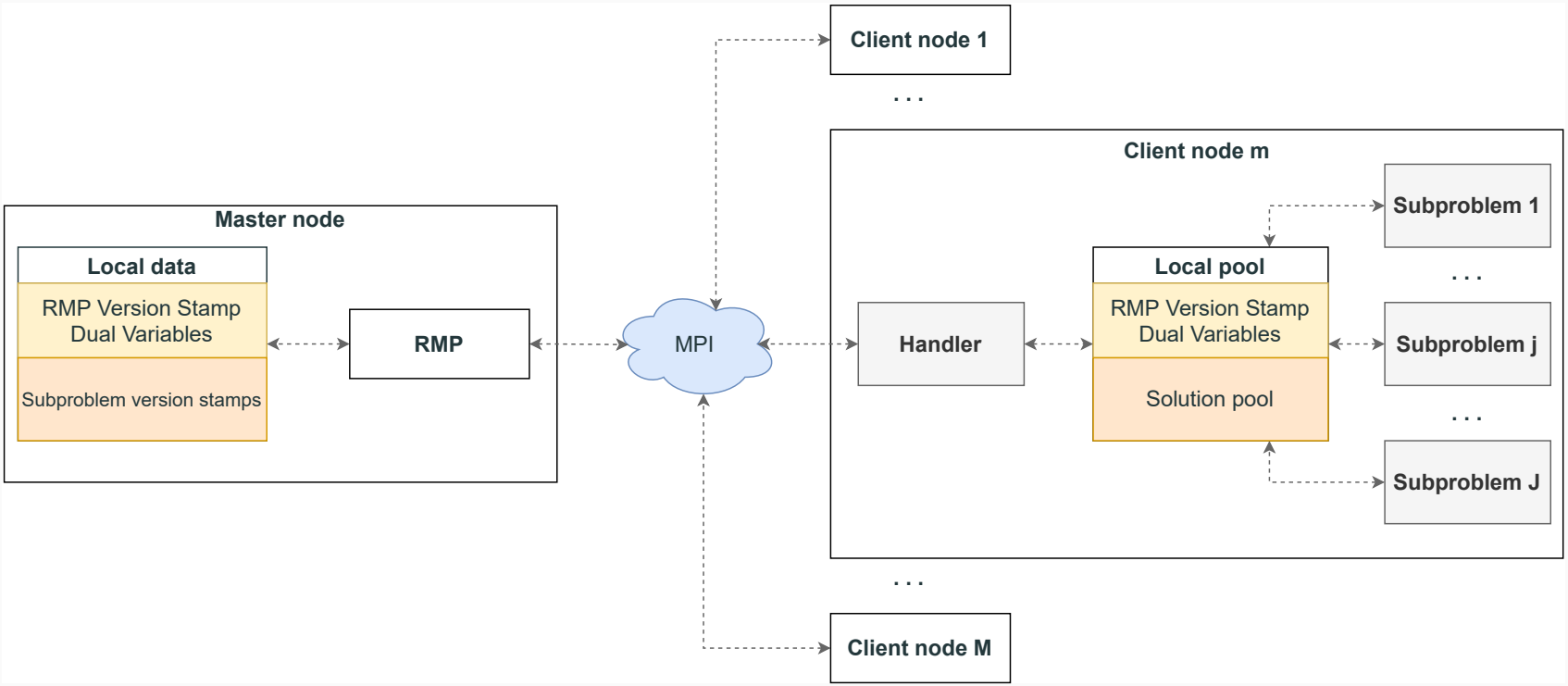


Proposition The following are sufficient and necessary conditions to guarantee convergence to the optimal solution

- each subproblem finished optimization with the latest RMP version stamp
- there are no new columns in the pool

Proof [...] □

Distributed Asynchronous Column Generation (DCG)



- RMP and Subproblem tasks run on different machines
- Communication through MPI protocol (Master-Client architecture)
- Local pool (exclusive access) on every node
- Additional handler task on every Client node

S. Basso, A. Ceselli "Distributed Asynchronous Column Generation", Computers & OR (2022)

Maximizing asynchronous computation

Experimentally, short optimization times maximize asynchronous computation:

- Dual variables are updated earlier
- RMP fetches new columns faster

Minimizing RMP optimization time:

- Rebalancing (from 10k to 20k solutions)
- RMP update policy (store only promising solutions for the current set of dual variables)

Minimizing Subproblems optimization time:

- Short timeouts during the `solve` step (from 1 to 60 seconds)

Test-bed Instances and Benchmark Algorithms

Configurations:

- CPLEX (version 12.6.3)
- Synchronous Column Generation (SCG)
- Asynchronous Column Generation (ACG)
- Distributed Asynchronous Column Generation (DCG)

Implementation: C++11 with Boost, OpenMP and OpenMPI libraries

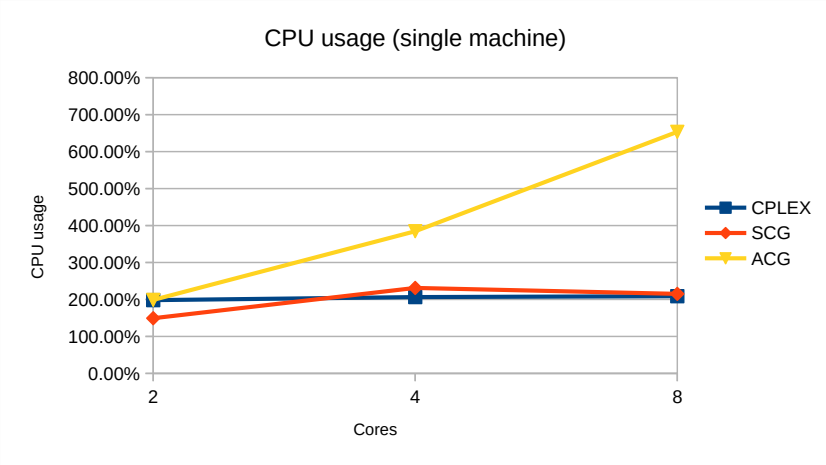
Experimental Setup: up to 4 machines with Intel i7-6700K CPU and 32GB RAM (Ubuntu 16.04)

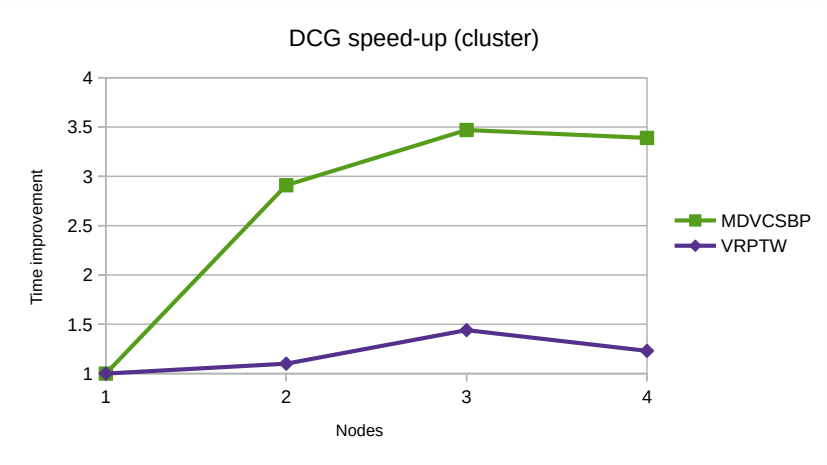
Test-bed instances (root node)

- 30 Large scale instances: Multi-dimensional Variable Size and Cost Bin-packing Problem (MDVCSBP) (250, 500, 750 subproblems)
- 21 "stress" test instances: Vehicle Routing Problem with Time Windows (VRPTW) (5 subproblems)
- 21 intermediate: Multi-Depot VRPTW (155 subproblems)

Scalability profiling

Single node (left) and multi node (right) scalability overview:

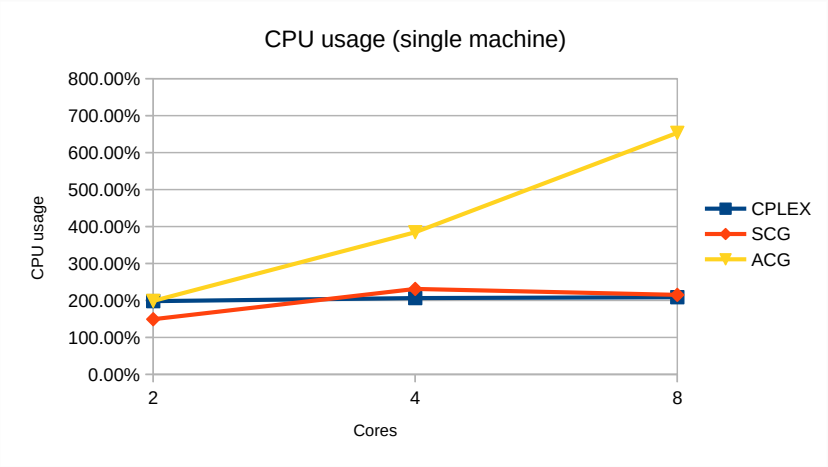


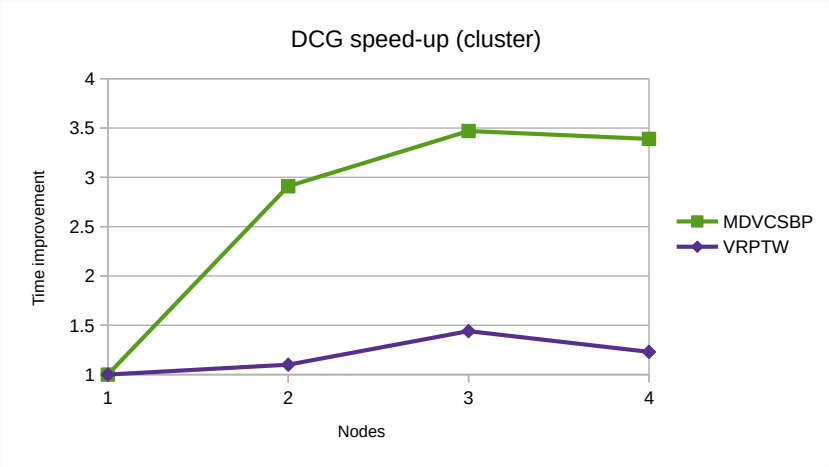


- Almost linear speed-ups for ACG on a single machine

Scalability profiling

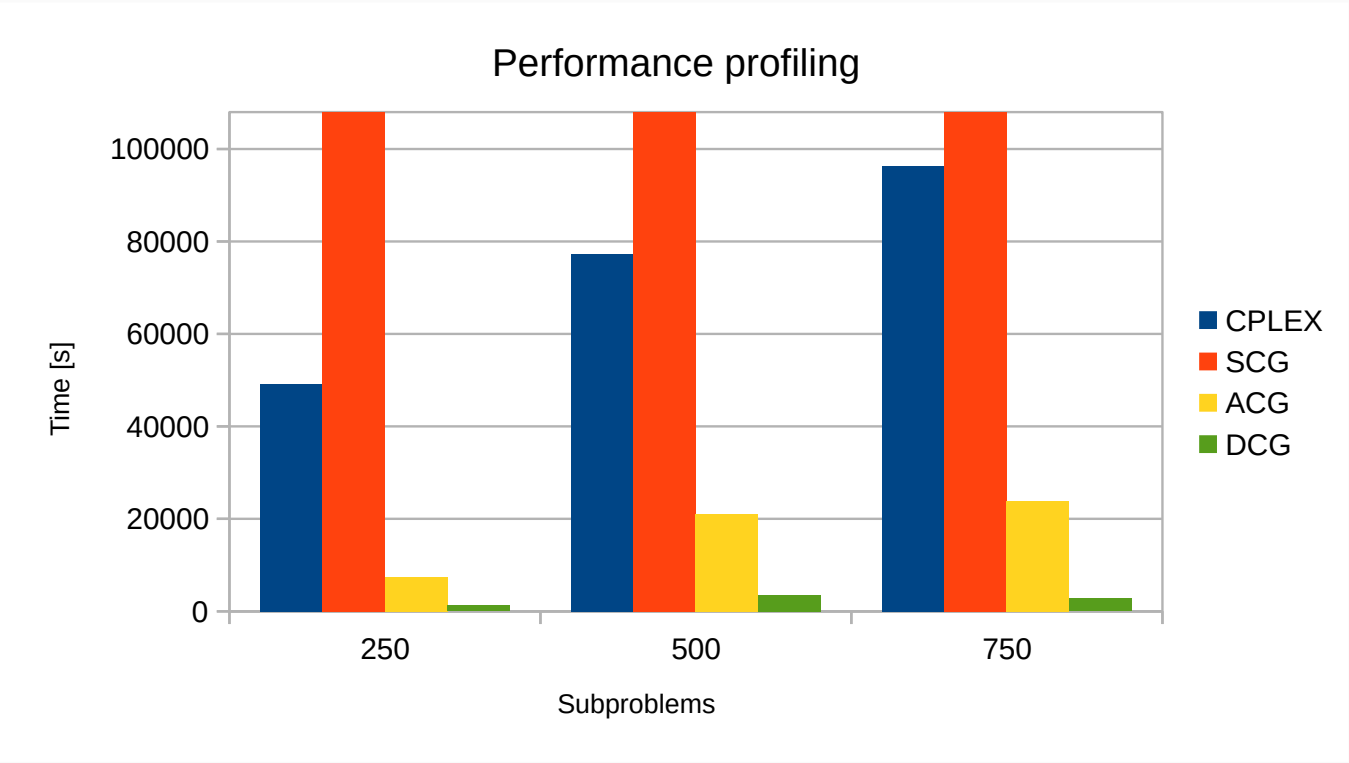
Single node (left) and multi node (right) scalability overview:





- Almost linear speed-ups for ACG on a single machine
- DCG scales well up to 3 machines on large scale instances
- Additional heuristic subproblems would improve performance

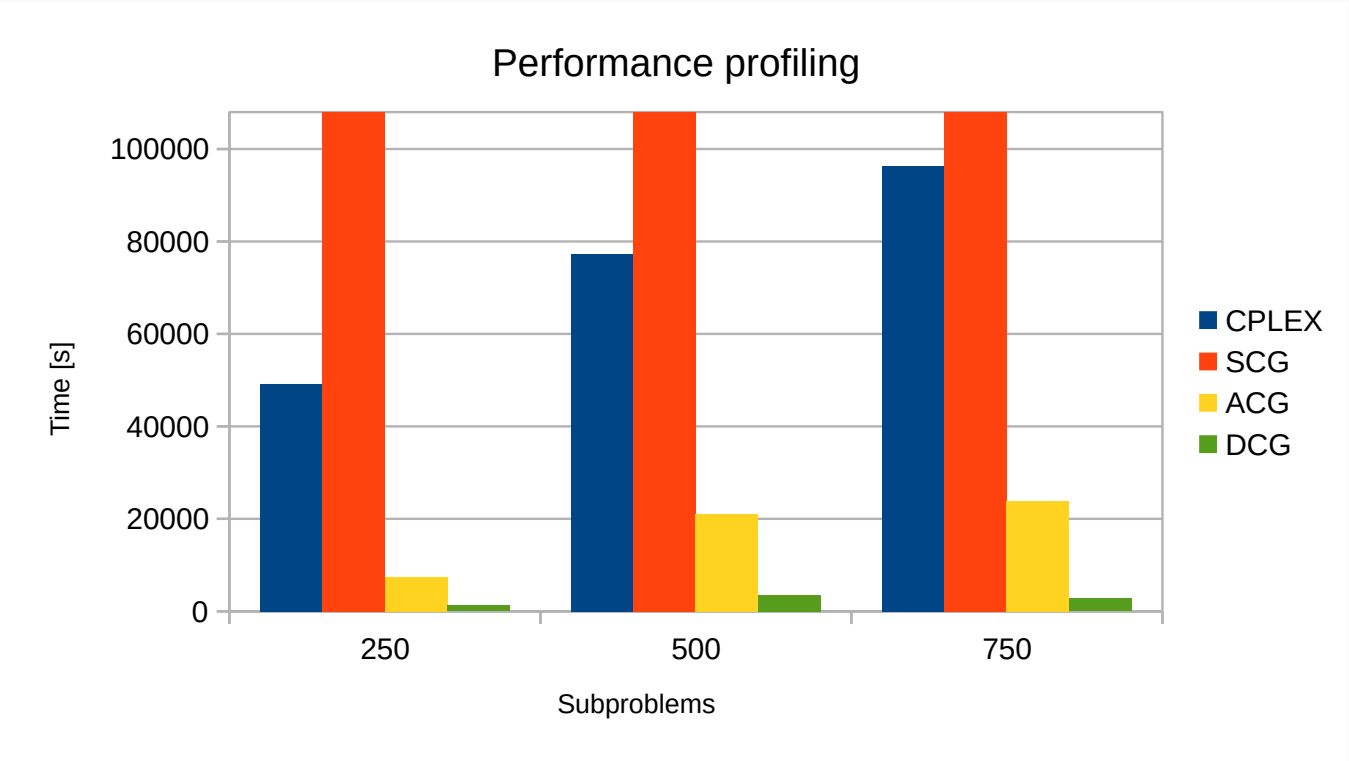
Overall results



30 Large scale instances (30h timeout):

- CPLEX hits 10 timeouts
- DCG is on average 87 times faster

Overall results



30 Large scale instances (30h timeout):

- CPLEX hits 10 timeouts
- DCG is on average 87 times faster

21 Stress test instances (10h timeout):

- CPLEX always out of memory (19 times) or timeout (2 times)
- DCG requires on average 14 minutes

Conclusions and Perspectives

Our data driven approach for the automatic decomposition of MIPs

- head-to-head with state-of-the-art detectors
- provides decompositions with a twisted flavor

Our parallel and distributed column generation approach

- performs one order of magnitude better than commercial solvers on large scale instances

There is still ground to cover for

- reaching commercial solvers on fully generic MIPs
- improving ML models and search algorithms
- using ML models as white boxes