Introduction to the SCIP Optimization Suite

Matthias Miltenberger

Zuse Institute Berlin

28th September 2015

Research Center MATHEON
Mathematics for Key Technologies

Berlin
Mathematical
School

MODAL
Mathematical Optimization and Data Analysis Laboratories

Introduction to the SCIP Optimization Suite

SCIP Optimization Suite

- toolbox for generating and solving constraint integer programs
- free for academic use, available in source code

ZIMPL

- model and generate LPs, MIPs, and MINLPs

SCIP

- MIP, MINLP and CIP solver, branch-cut-and-price framework

SoPlex

- revised primal and dual simplex algorithm

GCG

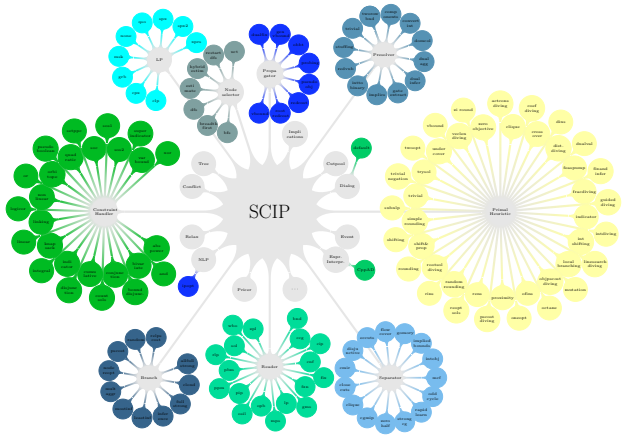
- generic branch-cut-and-price solver

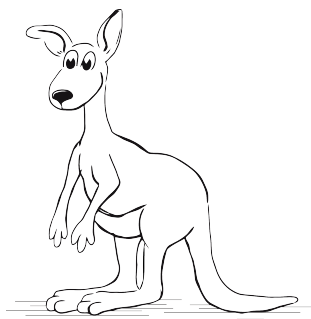
UG

- framework for parallelization of MIP and MINLP solvers

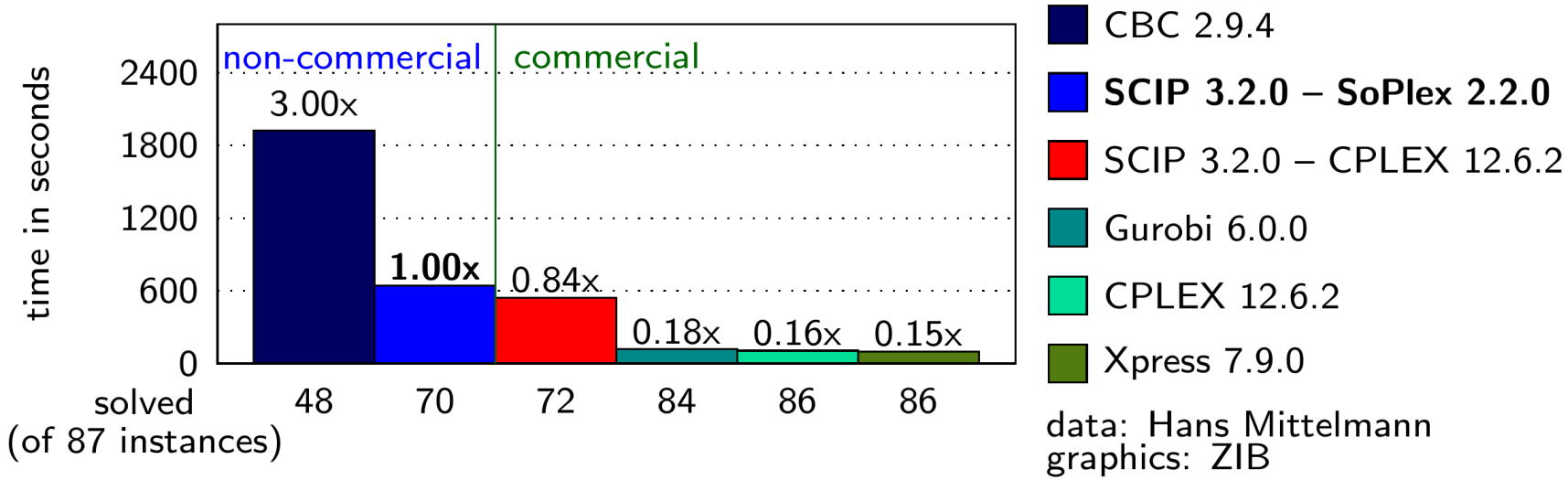
SCIP (Solving Constraint Integer Programs)

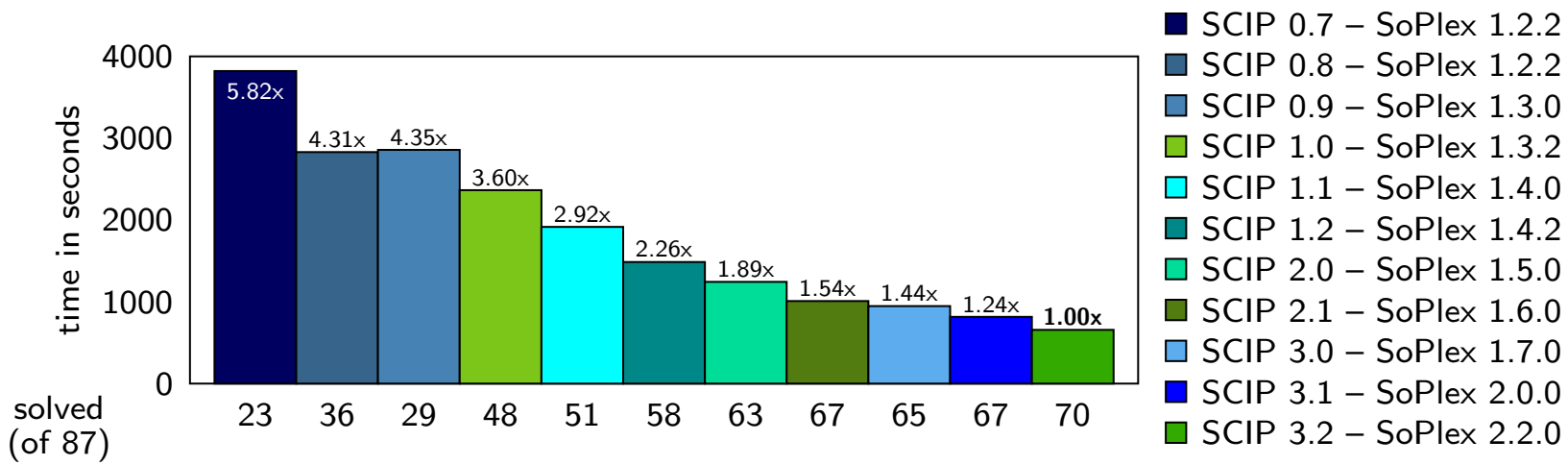
- is a branch-cut-and-price framework,
- is constraint based,
- incorporates
  - CP features (domain propagation),
  - MIP features (cutting planes, LP relaxation),
  - SAT-solving features (conflict analysis, restarts), and
  - MINLP features (expression trees, convex underestimators)
- has a modular structure via plugins,
- provides a full-scale MIP and MINLP solver,
- is free for academic purposes
- has just been released in its new version 3.2





Fastest MIP solver available in Source Code





Benchmark results from Hans Mittelmann, `http://plato.asu.edu/ftp/milpc`

Some facts about SCIP

- documentation and guidelines
  - more than 450 000 lines of C code, 20% documentation with 30 000 assertions and 4 000 debug messages
  - HowTos: plugin types, debugging, automatic testing
  - 9 examples and 4 applications illustrating the use of SCIP
  - active mailing list `scip@zib.de` (300 members)

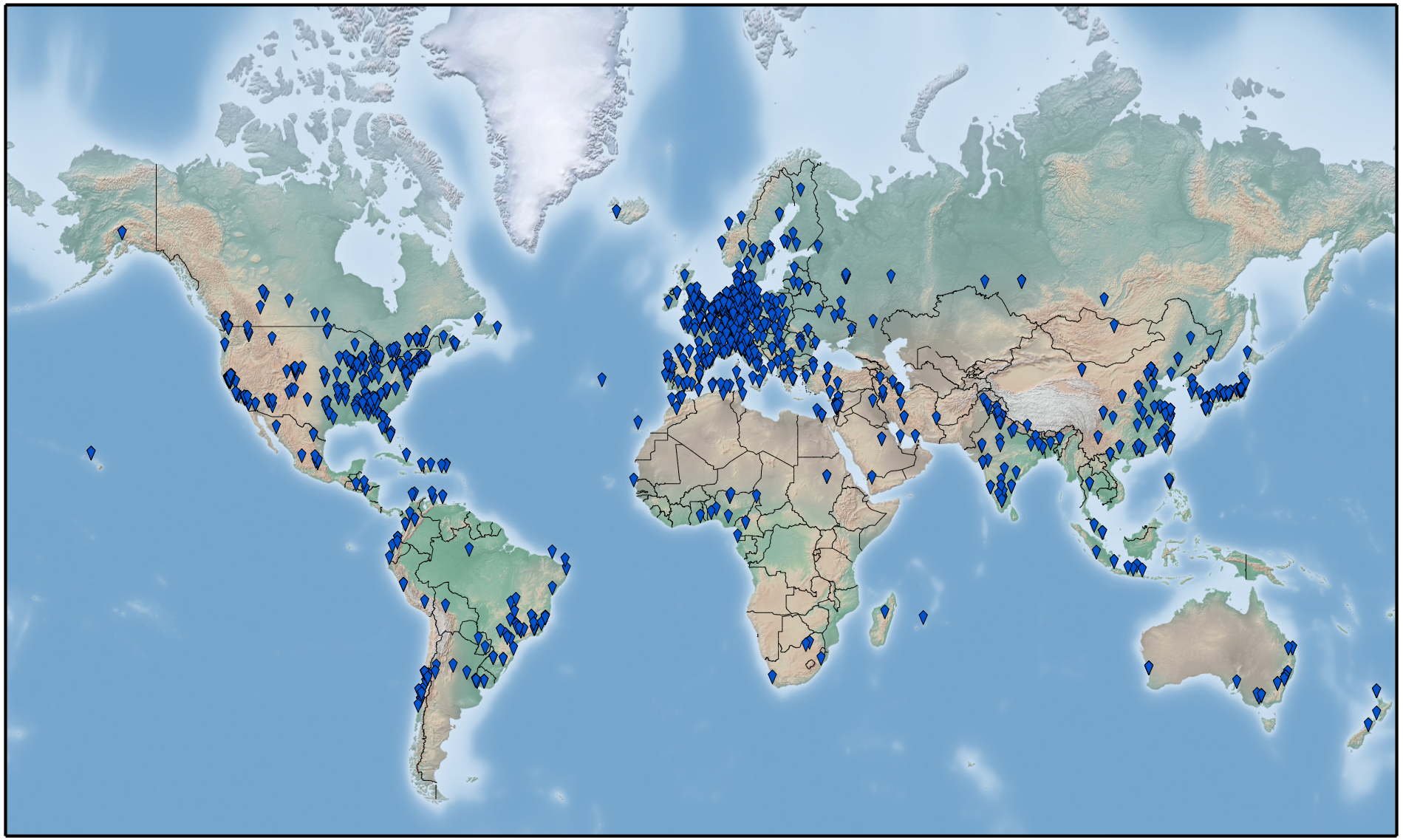
- interface and usability
  - user-friendly interactive shell
  - interfaces to AMPL, GAMS, ZIMPL, MATLAB, Python and Java
  - C++ wrapper classes
  - LP solvers: CLP, CPLEX, Gurobi, MOSEK, QSopt, SoPlex, Xpress
  - over 1 600 parameters and 15 emphasis settings

The SCIP Community

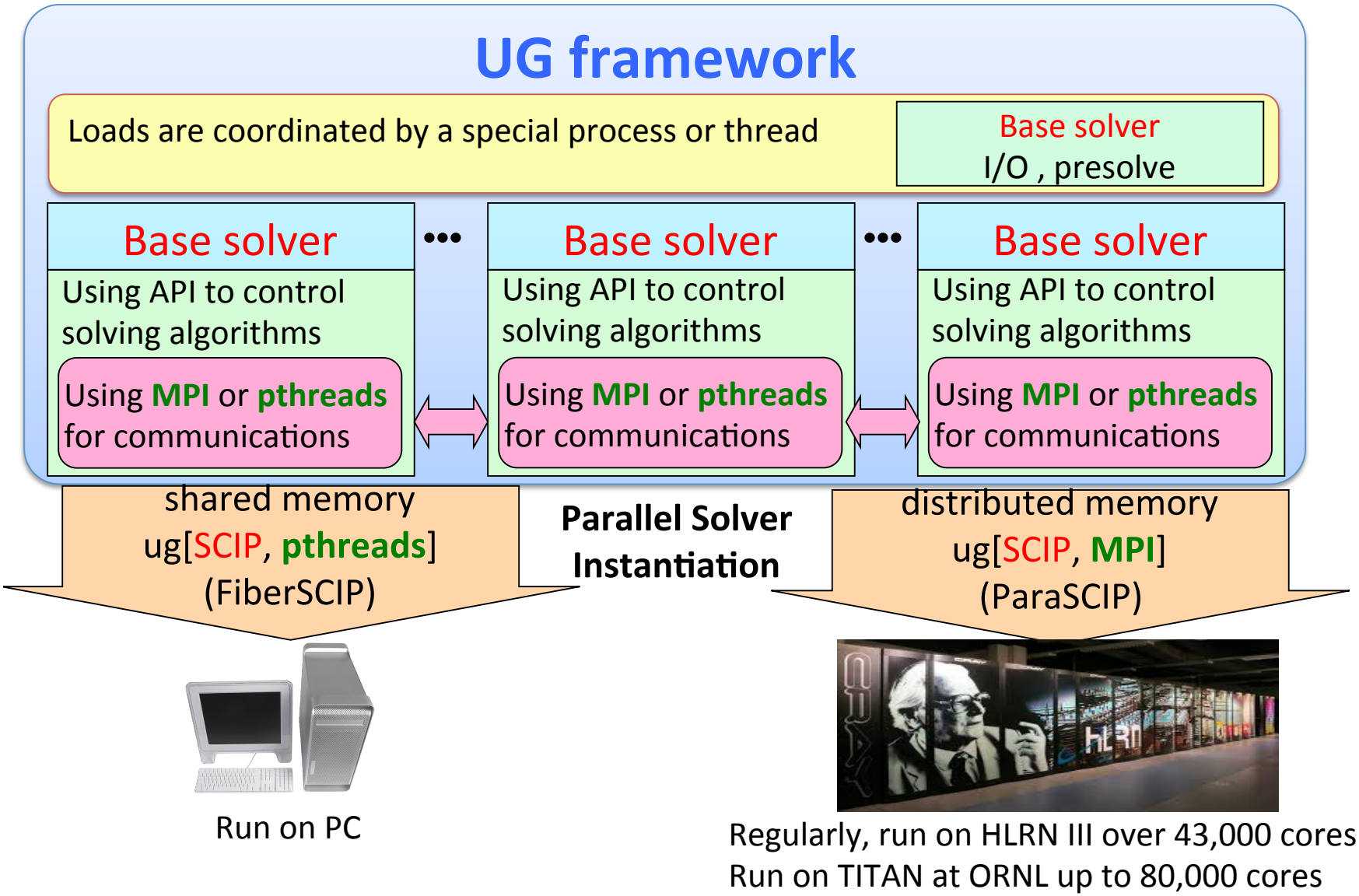
- 27 active developers
  - 4 running Bachelor and Master projects
  - 15 running PhD projects
  - 8 postdocs and professors
- 4 development centers in Germany
  - Aachen: GCG
  - Berlin: SCIP, SoPlex, UG, ZIMPL
  - Darmstadt: SCIP and SCIP-SDP
  - Erlangen-Nürnberg: SCIP

- many international contributors and users
  - more than 8 000 downloads per year from over 100 countries
- careers
  - 10 awards for Masters and PhD theses: MOS, EURO, GOR, DMV
  - 7 former developers are now building commercial optimization software at CPLEX, FICO Xpress, Gurobi, MOSEK, and GAMS

8 000 Downloads from more than 100 Countries



UG – Ubiquity Generator

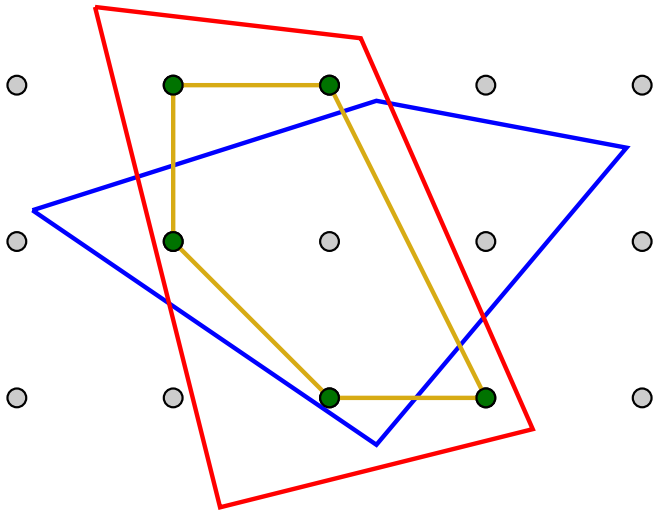


Run on PC

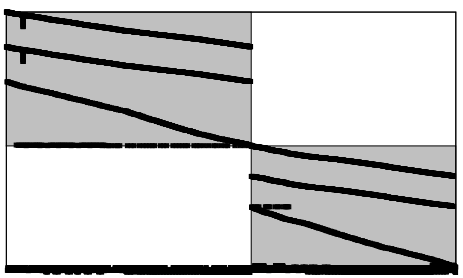
Regularly, run on HLRN III over 43,000 cores
Run on TITAN at ORNL up to 80,000 cores

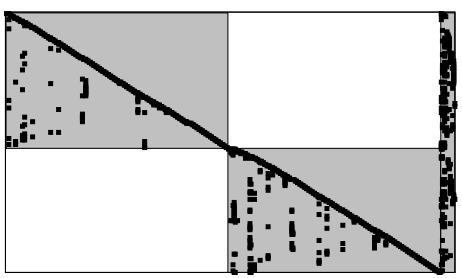
GCG – Generic Column Generation

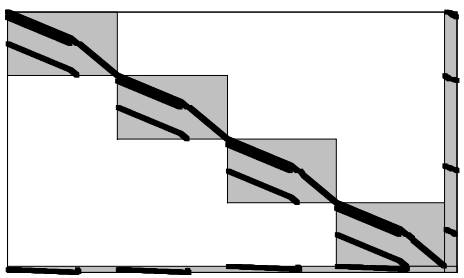
- Goal of GCG:
  - extend branch-cut-and-price framework SCIP to generic solver
  - based on Dantzig-Wolfe decomposition
  - easy use of branch-cut-and-price



- How does it work?
  - structure of problem provided or detected
  - pricing problems solved as general MIPs

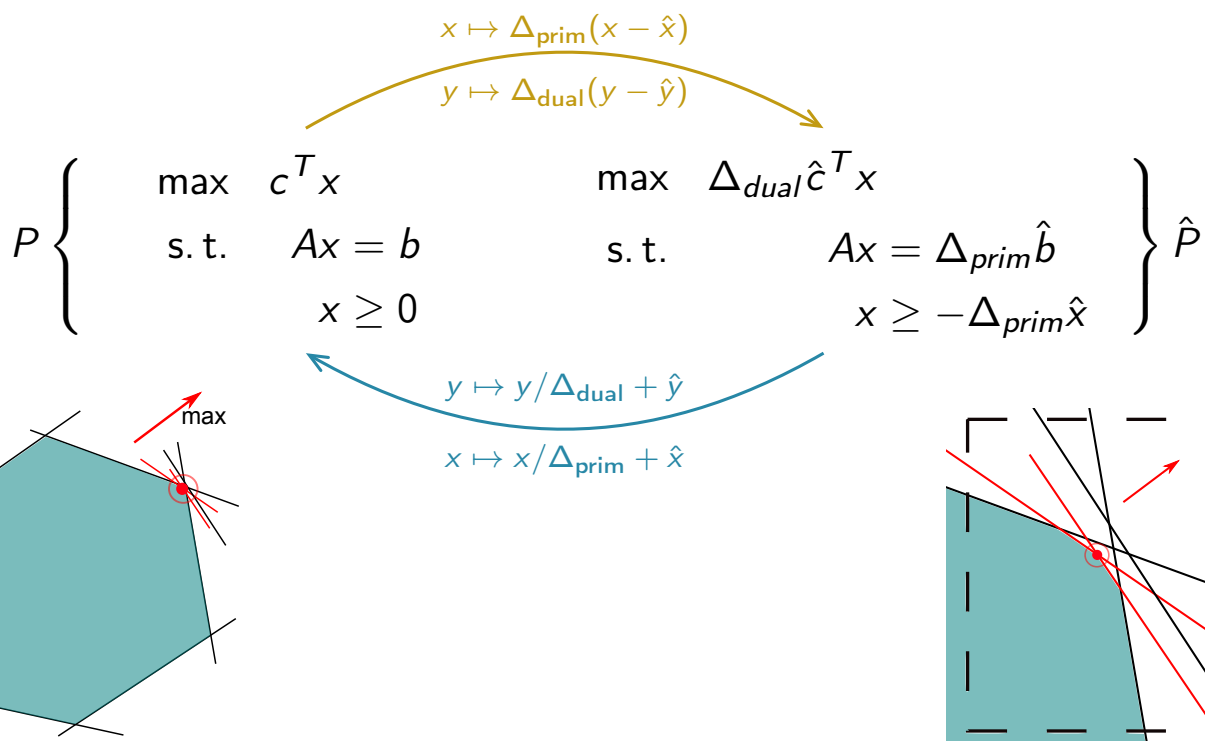






SoPlex – Sequential object-oriented simplex

- implementation of the revised simplex algorithm
- primal and dual solving routines for linear programs
- iterative refinement to overcome numerical problems
  - ▷ fast and accurate solutions by repeated floating-point solves



Most basic Linux commands

- SCIP is a command line tool:
  - no GUI (graphical user interface)
  - completely controlled from the console

- some Linux console commands:

`cd <dir>`	change current directory/navigate through directories
`cd ..`	move one directory up
`ls`	list contents of current directory
`rm <file>`	delete `<file>`
`cp <f1> <f2>`	copy `<f1>` to `<f2>`
`less <file>`	show contents of `<file>` (for text files, `q` to quit)
`./<binary>`	execute file `<binary>`

Installing SCIP

basic requirements:

- C/C++ compiler
- LP solver

- everything you need for a basic installation is contained in the SCIP Optimization Suite
- extract the package and type make:
  1. `tar -xvf scipoptsuite-3.2.0.tgz`
  2. `make`

Installing SCIP

basic requirements:

- C/C++ compiler
- LP solver

- everything you need for a basic installation is contained in the SCIP Optimization Suite
- extract the package and type make:
  1. `tar -xvf scipoptsuite-3.2.0.tgz`
  2. `make`

- will most likely not work on a standard Linux installation...
- additional packages/third-party software is required to fully enjoy SCIP
- carefully read the error messages to see what is missing

Third party software

- ZLIB
  - necessary to load compressed problem files
- Readline
  - comfortable interactive shell (tab-completion, command history, ...)
- GMP
  - Gnu Multiple Precision library
  - allows for higher precision arithmetic
- ncurses
  - necessary for output formatting
- ...

- You need the respective developer versions, indicated by `-devel` or `-dev` in your favorite package manager!

Alternative

What if you cannot install a package?

- most features can be disabled to solve installation problems:
  - `make ZLIB=false`
  - `make READLINE=false`
  - `make GMP=false`

- ZIMPL has additional requirements: `bison`, `flex`
- in case you cannot install ZIMPL:
  `make ZIMPL=false`

... even more options

- SCIP supports several different LP solvers:

SoPlex (default):	`make LPS=spx`
IBM CPLEX:	`make LPS=cpx`
FICO Xpress:	`make LPS=xprs`
Gurobi:	`make LPS=grb`
CoinOR CLP:	`make LPS=clp`
Mosek:	`make LPS=msk`
QSopt:	`make LPS=qso`
improved SoPlex interface:	`make LPS=spx2`

- for MINLP we recommend you install and use `Ipopt`
  - `make IPOPT=true`
  - a working installation of `Ipopt` is required (not covered here)
- give your binary a descriptive name:
  - `make VERSION=awesome-new-feature`
  - defaults to the current version number

Debugging with SCIP

SCIP can be compiled and run in debug mode

- 30 000 asserts will be checked
- additional checks will be performed
- more warnings may be printed
- debug information is available for `gdb`
- slower than the optimized mode (`OPT=opt`)
- useful when writing new code and when hunting for bugs

- `make OPT=dbg`

Examples/Applications contained in SCIP

- show how SCIP can be used and
  provide starting points for future developments
- examples:
  - Branch-and-Price
  - Branch-and-Cut
  - Callable Library
  - ...
- applications:
  - Coloring
  - Scheduler
  - Multi-objective Optimization
  - Steiner Tree Problem
- every example/application has its own Makefile
- go to `examples/` or `applications/` and type `make`
- code usually relies on a compiled main SCIP

Beyond SCIP...

- GCG, a generic column generation solver, can be compiled with `make gcg`
  - relies on bliss – for graph computations
- UG, the parallelization framework, can be compiled with `make ug`
  - default is to build FiberSCIP, a shared memory parallel SCIP extension

How to use the shell

- navigate to your SCIP installation: `cd projects/scip`
- start SCIP: `./bin/scip`
  (when installed globally just type: `scip`)
- display available commands: `help`
- commands in `<...>` open a new set of commands
  - press ⏎ to return to home menu
- commands can be shortened, but need to be unique:
  e.g. `o[ptimize]` or `q[uit]`
- cycle through previous commands with ↑ and ↓
- `Tab` completion (⇥), e.g. for finding files quickly, is also available
  (requires the `readline` package)

SCIP as a shell program

- run `./bin/scip --help` to see available options
- example:
  `./bin/scip -s fast.set -f mip.lp -l out.log`
- all commands from the interactive shell can be chained together in one call with `-c`:
  `./bin/scip -c "read mip.lp opt disp stat q"`

Modifying the Output

- customize (almost) everything with `SCIP> set display`

`freq`	in-/decrease frequency of node information line
`lpinfo`	enable output of LP solver
`verblevel`	change overall verbosity
	adjust which columns to display/hide
	...

Parameters

- parameter tuning is very important to achieve maximal performance
- SCIP has more than 1 600 parameters

Parameters

- parameter tuning is very important to achieve maximal performance
- SCIP has more than 1 600 parameters
- changing one setting may have unforeseen side effects
- trial and error is often unavoidable

Parameters

- parameter tuning is very important to achieve maximal performance
- SCIP has more than 1 600 parameters
- changing one setting may have unforeseen side effects
- trial and error is often unavoidable

Good news:

- SCIP's default settings are already pretty good!
- SCIP has several emphasis/meta settings to simplify tuning (these will modify multiple parameters at once)
  - `SCIP> set presol emph {off|fast|aggressive}`
  - `SCIP> set heur emph {off|fast|aggressive}`
  - `SCIP> set sepa emph {off|fast|aggressive}`
  - `SCIP> set emphasis ...` → *next slide*

Emphasis Settings

- `SCIP> set emphasis ...`

`feasibility`	for finding feasible solutions quickly
`optimality`	focus on proving optimality
`easycip`	for small and easy problems
`hardlp`	avoid solving many LP relaxations
`cpsolver`	rely on CP techniques instead of MIP
`counter`	for counting all feasible solutions

Analyzing a Run

- most important tool is the statistics output:
  `SCIP> display statistics`
- running times of (almost) all components can be compared

How to use the information?

- Check for long running heuristics that don't find a solution!
- Are cut generators too costly or are more cuts necessary to increase the dual bound?
- ...

Working with Settings Files

- SCIP can save the current settings to a file:
  - either all of them: `SCIP> set save <file.set>`
  - or only the modified ones: `SCIP> set diffsave <file.set>`
- load a settings file:
  - `SCIP> set load <file.set>`
- settings files can also be modified by hand
- if a file called `scip.set` is present in SCIP's home directory, it is automatically loaded

Customizing Parameters

Example: Presolving

- `SCIP> read check/instances/MIP/bell5.mps`
  (models a fiber optic design network)
- disable presolving and compare with default
- `SCIP> set presol emph off`

Example: Presolving

- `SCIP> read check/instances/MIP/bell5.mps`
  (models a fiber optic design network)
- disable presolving and compare with default
- `SCIP> set presol emph off`

- presolving reduces the problem size
- smaller problems are usually easier to solve

Example: Node Selection

- `SCIP> read check/instances/MIP/dcmulti.mps`
  (models multi-period facility location problem)
- try different rules, e.g. depth-first search and breadth-first-search
- `SCIP> set nodesel dfs stdprio 1000000`
- `SCIP> set nodesel breadthfirst stdprio 1000000`

Example: Node Selection

- `SCIP> read check/instances/MIP/dcmulti.mps`
  (models multi-period facility location problem)
- try different rules, e.g. depth-first search and breadth-first-search
- `SCIP> set nodesel dfs stdprio 1000000`
- `SCIP> set nodesel breadthfirst stdprio 1000000`

- node selection rules
  - determine the order of open node processing
  - influence when solutions are found and how many nodes are needed

Example: Separating

- `SCIP> read check/instances/MIP/gt2.mps`
  (models truck routing problem)
- disable separation and compare with default

Example: Separating

- `SCIP> read check/instances/MIP/gt2.mps`
  (models truck routing problem)
- disable separation and compare with default

- cutting planes help to improve the dual bound
- Generating too many cuts can be costly!

Example: Branching

- `SCIP> read check/instances/MIP/enigma.mps`
- try different rules and find one that needs 50% more/less nodes than default
- `SCIP> set branching leastinf prio 1000000`
- branching rules determine on which fractional variable to branch

Example: Branching

- `SCIP> read check/instances/MIP/enigma.mps`
- try different rules and find one that needs 50% more/less nodes than default
- `SCIP> set branching leastinf prio 1000000`
- branching rules determine on which fractional variable to branch

- `leastinf` branching is very bad
- `fullstrong` branching is very good

Testing Environment

- SCIP package provides many scripts to make testing easy
- general setup:
  - set of instances put together in a test set:
    `check/testset/short.test`
  - verify solution values:
    `check/testset/short.solu` (optional)
  - run SCIP on the entire test set:
    (use the same options as for compiling)
    `make test TEST=short`
  - test non-default parameters:
    `make test TEST=short SETTINGS=heuristics-off`
  - use the interactive shell to create a new settings file
  - settings file needs to be placed in `settings/`

Default Values

```
      TEST = short
       OPT = opt
  SETTINGS = default
   VERSION = 3.2.0
       LPS = spx
       GMP = true
     ZIMPL = true
  READLINE = true
      ZLIB = true
     IPOPT = false
      TIME = 3600 (sec)
       MEM = 6144 (MB)
```

Output of a single test run

- `.out` file:
  concatenation of all logfiles
- `.err` file:
  concatenation of all occured errors/warnings
- `.set` file:
  copy of the used settings
- `.res` file:
  concise summary of test run
- `.tex` file:
  LaTeX table of results
- `.pav` file:
  machine readable table of results

- all files are stored in `check/results/`:

```
check.short.scip-3.2.0.linux.x86_64.gnu.opt.spx.opt49.default.out
```

Compare multiple runs

- multiple runs over the same test set can be compared
- use `check/allcmpres.sh`:
  1. `cd check`
  2. `./allcmpres.sh results/check.1.res results/check.2.res`

- more than two result files can be compared
- typical use cases:
  - compare different settings
  - compare different versions
  - ...