Benchmarking and Performance Analysis of Optimization Software

TED RALPHS
ISE Department
COR@L Lab
Lehigh University
ted@lehigh.edu

CPAIOR, Bologna, 15 June 2010

Outline

1. Introduction

2. Benchmarking
   - Purpose
   - Sequential Codes
   - Parallel Codes

3. Performance Analysis

4. Conclusions

My Hats

- Research Scientist
- Lab Director (COR@L)
- Software Developer (SYMPHONY, CHiPPS, DIP, CBC, MiBS, ...)
- Open Source Project Leader (COIN-OR)
- Educator
- Thesis Advisor
- Industry Consultant

Caveats

- This talk is heavily biased towards LP-based branch-and-bound algorithms for solving *mathematical programming problems*.
- In such a setting, results can be "messy."
- Important aspects of this setting are that we have to account for
  - numerical error
  - failure of the algorithm to converge
- This talk contains a lot more questions than answers!

Background: COIN-OR

- The Common Optimization Interface for Operations Research Initiative was an initiative launched by IBM at ISMP in 2000.
- IBM seeded an open source repository with four initial projects and created a Web site.
- The goal was to develop the project and then hand it over to the community.
- The project has now grown to be self-sustaining and was spun off as a nonprofit educational foundation in the U.S. several years ago.
- The name was also changed to the Computational Infrastructure for Operations Research to reflect a broader mission.

What is COIN-OR Today?

The COIN-OR Foundation

- A non-profit foundation promoting the development and use of interoperable, open-source software for operations research.
- A consortium of researchers in both industry and academia dedicated to improving the state of computational research in OR.
- A venue for developing and maintaining standards.
- A forum for interaction and discussion of OR software.

The COIN-OR Repository

- A collection of interoperable software tools for building optimization codes, as well as a few stand-alone packages.
- A venue for peer review of OR software tools.
- A development platform for open source projects, including an SVN repository,

The COIN-OR Foundation

- The foundation has been up and running for more than five years.
- We have two boards.
  - A strategic board to set overall direction
  - A technical board to advise on technical issues
- The boards are composed of members from both industry and academia, as well as balanced across disciplines.
- Membership in the foundation is available to both individuals and institutions.
- The foundation Web site and repository are hosted by INFORMS.

My Hats: COIN-OR

- Member of Strategic Leadership Board
- Chair of Technical Leadership Council
- Project Manager
  - CoinBinary/CoinAll
  - SYMPHONY
  - CHiPPS
    - ALPS
    - BiCePS
    - BLIS
  - DIP
  - CBC
  - MiBS

What You Can Do With COIN

- We currently have 50+ projects and more are being added all the time.
- Most projects are now licensed under the EPL (very permissive).
- COIN has solvers for most common optimization problem classes.
  - Linear programming
  - Nonlinear programming
  - Mixed integer linear programming
  - Mixed integer nonlinear programming (convex and nonconvex)
  - Stochastic linear programming
  - Semidefinite programming
  - Graph problems
  - Combinatorial problems (VRP, TSP, SPP, etc.)
- COIN has various utilities for reading, building, and manipulating optimization models and feeding them to solvers.
- COIN has overarching frameworks that support implementation of broad algorithm classes.
  - Parallel search
  - Branch and cut (and price)
  - Decomposition-based algorithms

COIN-OR Projects Overview: Linear Optimization

- Clp: COIN LP Solver

  Project Manager: Julian Hall

- Cbc: COIN Branch and Cut

  Project Manager: T.R.

- SYMPHONY: a flexible integer programming package that supports shared and distributed memory parallel processing, biobjective optimization, warm starting, sensitivity analysis, application development, etc.

  Project Manager: T.R.

- BLIS: Parallel IP solver built to test the scalability of the CHiPPS framework.

  Project Manager: T.R.

COIN-OR Projects Overview: Nonlinear Optimization

- Ipopt: Interior Point OPTimizer implements interior point methods for solving nonlinear optimization problems.

  Project Manager: Andreas Wächter

- Bonmin: Basic Open-source Nonlinear Mixed INteger programming is for (convex) nonlinear integer programming.

  Project Manager: Pierre Bonami

- Couenne: Solver for nonconvex nonlinear integer programming problems.

  Project Manager: Pietro Belloti

COIN-OR Projects Overview: Modeling

- FLOPC++: An open-source modeling system.

  Project Manager: Tim Hultberg

- PuLP: Python-based modeling language for linear mathematical programs.

  Project Manager: Stu Mitchell

- Pyomo: Python-based modeling language for linear mathematical programs.

  Project Manager: Bill Hart

COIN-OR Projects Overview: Interfaces

- Osi: Open solver interface is a generic API for linear and mixed integer linear programs.

  Project Manager: Matthew Saltzman

- GAMSlinks: Allows you to use the GAMS algebraic modeling language and call COIN-OR solvers.

  Project Manager: Stefan Vigerske

- CoinMP: A callable library that wraps around CLP and CBC, providing an API similar to CPLEX, XPRESS, Gurobi, etc.

  Project Manager: Bjarni Kristjansson

- Optimization Services: A framework defining data interchange formats and providing tools for calling solvers locally and remotely through Web services.

  Project Managers: Jun Ma, Gus Gassmann, and Kipp Martin

COIN-OR Projects Overview: Frameworks

- Bcp: A generic framework for implementing branch, cut, and price algorithms.

  Project Manager: Laci Ladanyi

- CHiPPS: A framework for developing parallel tree search algorithms.

  Project Manager: T.R./Yan Xu

- DIP: A framework for implementing decomposition-based algorithms for integer programming, including Dantzig-Wolfe, Lagrangian relaxation, cutting plane, and combinations.

  Project Manager: T.R./Matthew Galati

COIN-OR Projects Overview: Miscellaneous

- CppAD: a package for doing algorithmic differentiation, a key ingredient in modern nonlinear optimization codes.

  Project Manager: Brad Bell

- CSDP: A solver for semi-definite programs

  Project Manager: Brian Borchers

- DFO: An algorithm for derivative free optimization.

  Project Manager: Katya Scheinburg

CoinAll, CoinBinary, BuildTools, and TestTools

- Many of the tools mentioned interoperate by using the configuration and build utilities provided by the `BuildTools` project.
- The `BuildTools` includes autoconf macros and scripts that allow PMs to smoothly integrate code from other projects into their own.
- The `CoinAll` project is an über-project that includes a set of mutually interoperable projects and specifies specific sets of versions that are compatible.
- The `TestTools` project is the focal point for testing of COIN code.
- The `CoinBinary` project is a long-term effort to provide pre-built binaries for popular platforms.
  - Installers for Windows
  - RPMs for Linux
  - .debs for Linux
- You can download `CoinAll` (source and/or binaries) here:

  http://www.coin-or.org/download/source/CoinAll
  http://www.coin-or.org/download/binary/CoinAll

Outline

1 Introduction

2 Benchmarking
- Purpose
- Sequential Codes
- Parallel Codes

3 Performance Analysis

4 Conclusions

The Different Roles of Benchmarking

- Comparing performance of different codes
- Comparing performance of different versions of the same code
- Debugging sofware
- Setting a direction/goal for future research
- Tuning software

Academy versus Industry

- The role of benchmarking in academia is different than in the commercial sector.
  - Commercial codes: Primary goal is to satisfy users.
  - Academic codes: Primary goal is to test ideas and generate papers.
- The importance of software to the progress of academic research is evident.
- However, academic research is (unfortunately) still driven primarily by publication in archival journals.
- Software is difficult to evaluate as an intellectual product on its own merits.
- Developers are forced to publish papers in archival journals about software instead of publishing the software itself.
- Publications about software necessitate the use of benchmarks.

Developing/Maintaining Benchmarks

- Many academic test sets are developed in an ad hoc fashion specifically to support findings reported in a paper.
- Hence, they are essentially only vetted by the referees of the paper who may not even examine the test set closely.
- Once cited in a paper, the test set is established and may drive the research agenda.
- Many codes become tuned to the benchmark.
- This introduces undesirable biases into the literature.
- Fortunately, there are some exceptions.

The Role of Open Source

- Open source projects can play an important role in benchmarking.
- Reference implementations released in open source provide a well-understood baseline for comparison.
- Without such implementations, it is virtually impossible to do a properly designed and controlled experiment.
- Comparisons against black-box software are often not very meaningful.
- This was one of the central motivation for the founding of COIN-OR.

Benchmarking within Open Source

- Within open source projects, benchmarking plays a role somewhere between academia and industry.
- Since development is decentralized, benchmarking can provide an “early warning system” for problems.
- As in industry, they can also make it easier to track progress.
- There may still be a tendency to “develop to the benchmark” that has to be guarded against.
- COIN-OR uses nightly builds and standard benchmarks to track development.

Issue 1: What is Really Being Tested?

- In general, the challenge is to test only a particular aspect of a given algorithm.
- To do so, we want to hold all other aspects of the algorithm constant.
- This is most easily accomplished when all experiments are done within a common software framework on a common experimental platform.
- Even in the most ideal circumstances, it can be difficult to draw conclusions.
  - Should the values of parameters be re-tuned?
  - Should "unrelated" parameter settings be held constant?
- How do you show that a new technique will be effective within a state-of-the-art implementation without access to the implementation?

Issue 2: How To Measure Performance?

Most papers in mathematical programming use measures such as

Without time limit

- Running time (wallclock or CPU?)
- Tree size (which nodes to count?)

With time limit

- Fraction solved (tolerance?)
- Final gap (how measured?)
- Quality of solution (what is optimum?)
  - Cost
  - Feasibility
- Time to first solution (quality?)

Are these good choices? Probably not.

Issue 3: What Is a Fair Comparison?

- How do we really compare two different codes “fairly”?
- Codes may have inconsistent default parameters
  - Error tolerances
  - Gap tolerances
- Two codes claiming to have found an optimal solution may nevertheless produce a different optimal value.
- In the case of nonlinear optimization, we may also have to deal with the fact that codes can produce local optima.
- Details of implementation
  - Who implemented the code and how well is it optimized?
  - Are there differences in the implementation of common elements that are tangential to what is being tested?

Benchmarking Parallel Codes

- For the foreseeable future, increases in computing power will come in the form of additional cores rather than improvements in clock speeds.
- For this reason, most codes will need to be parallelized in some way to remain competitive.
- All of the previously mentioned issues are brought into even greater contrast when benchmarking such codes.
- In addition to traditional performance measures, we must also consider *scalability*.
  - What is it?
  - What are the tradeoffs?

Parallel Scalability

- *Parallel scalability* measures how well an algorithm is able to take advantage of increased resources (primarily cores/processors).
- Generally, this is measured by executing the algorithm with different levels of available resources and observing the change in performance.
- The most clear-cut and often-cited measure is *speedup*, which measures time to optimality for different numbers of processors.
- This is not necessarily a relevant measure for real-world performance.

Traditional Measures of Performance

- *Parallel System*: Parallel algorithm + parallel architecture.
- *Scalability*: How well a parallel system takes advantage of increased computing resources.

Terms

- Sequential runtime: T_s
- Parallel runtime: T_p
- Parallel overhead: $T_o = NT_p - T_s$
- Speedup: $S = T_s/T_p$
- Efficiency: $E = S/N$

- Standard analysis considers change in efficiency on a fixed test set as number of processors is increased.
- This analysis is purely "compute-centric," and does not take into account the effects of limitations on memory and storage.

Amdahl's Law

- Amdahl's Law postulates a theoretical limit on speed-up based on the amount of *inherently sequential* work to be done.
- If s is the fraction of work to be done that is sequential, then efficiency on p processors is limited to $s + (1 - s)/p$.
- In other words, efficiency is bounded by the sequential fraction s.
- In reality, there is no well-defined "sequential fraction."
- The analysis also assumes a single, fixed test set.
- *Isoefficiency analysis* considers the increase in problem size to maintain a fixed efficiency as number of processors is increased.
- This is perhaps a more reasonable measure.

Parallel Overhead

- In practice, the amount of *parallel overhead* essentially determines the scalability.

Major Components of Parallel Overhead in Tree Search

- Communication Overhead (cost of sharing information)
- Idle Time
    - Handshaking/Synchronization (cost of sharing information)
    - Task Starvation (cost of *not* sharing information)
    - Ramp Up Time
    - Ramp Down Time
- Performance of Redundant Work (cost of *not* sharing information)

- Information sharing is the main driver of efficiency.
- There is a fundamental tradeoff between centralized and decentralized information storage and decision-making.

Effect of Architecture

- Architectures are getting more complex and each has its own bottlenecks.
  - “Traditional” architectures are fast becoming extinct.
  - Multi-core desktops are now common.
  - Clusters of multi-core machines are becoming a standard.
  - GPUs are still a bit unknown.
- Performance is affected by
  - Memory
  - Bandwidth
  - Latency
- Ultimately, one can think of the architecture primarily in terms of an extended memory hierarchy.
- Performance measures are only really valid for practically identical architectures.
- It’s extremely difficult to extrapolate.

Challenges in Measuring Performance

- Traditional measures may not be appropriate.
  - The interesting problems are the ones that take too long to solve sequentially.
  - Need to account for the possibility of failure.
- It's exceedingly difficult to construct a test set
  - Scalability varies substantially by instance.
  - Hard to know what test problems are appropriate.
  - A fixed test set will probably fail to measure what you want.
- Results are highly dependent on architecture
  - Difficult to make comparisons
  - Difficult to tune parameters
- Hard to get enough time on large-scale platforms for tuning and testing.
- Results are non-deterministic!
  - Determinism can be a false sense of security.
  - Lack of determinism requires more extensive testing.

Sample Scalability Analysis

Solved difficult knapsack instances by branch and bound on SDSC Blue Gene,

SDSC Blue Gene System

Machine:	IBM Blue Gene with 3,072 compute nodes
Node:	dual processor, speed 700 MHz
Memory:	512 MB RAM each node
Operating System:	Linux
Message Passing:	MPICH

P	Node	Ramp-up	Idle	Ramp-down	Wallclock	Eff
64	14733745123	0.69%	4.78%	2.65%	6296.49	1.00
128	14776745744	1.37%	6.57%	5.26%	3290.56	0.95
256	14039728320	2.50%	7.14%	9.97%	1672.85	0.94
512	13533948496	7.38%	4.30%	14.83%	877.54	0.90
1024	13596979694	8.33%	3.41%	16.14%	469.78	0.84
2048	14045428590	9.59%	3.54%	22.00%	256.22	0.77

Note the increase in ramp-up and ramp-down.

Scalability for Generic MILPs

- Selected 18 MILP instances from Lehigh/CORAL, MIPLIB 3.0, MIPLIB 2003, BCOL, and markshare.
- Tested on the Clemson cluster with BLIS.

Instance	Nodes	Ramp -up	Idle	Ramp -down	Comm Overhead	Wallclock	Eff
1 P	11809956	–	–	–	–	33820.53	1.00
Per Node		–	–	–	–	0.00286	
4P	11069710	0.03%	4.62%	0.02%	16.33%	10698.69	0.79
Per Node		0.03%	4.66%	0.00%	16.34%	0.00386	
8P	11547210	0.11%	4.53%	0.41%	16.95%	5428.47	0.78
Per Node		0.10%	4.52%	0.53%	16.95%	0.00376	
16P	12082266	0.33%	5.61%	1.60%	17.46%	2803.84	0.75
Per Node		0.27%	5.66%	1.62%	17.45%	0.00371	
32P	12411902	1.15%	8.69%	2.95%	21.21%	1591.22	0.66
Per Node		1.22%	8.78%	2.93%	21.07%	0.00410	
64P	14616292	1.33%	11.40%	6.70%	34.57%	1155.31	0.46
Per Node		1.38%	11.46%	6.72%	34.44%	0.00506	

Impact of Instance Properties

- Instance `input150_1` is a knapsack instance. When using 128 processors, BLIS achieved super-linear speedup mainly to the decrease of the tree size
- Instance `fc_30_50_2` is a fixed-charge network flow instance. It exhibits very significant increases in the size of its search tree.
- Instance `pk1` is a small integer program with 86 variables and 45 constraints. It is relatively easy to solve.

Instance	P	Node	Ramp-up	Idle	Ramp-down	Wallclock	Eff
input150_1	64	75723835	0.44%	3.38%	1.45%	1257.82	1.00
	128	64257131	1.18%	6.90%	2.88%	559.80	1.12
	256	84342537	1.62%	5.53%	7.02%	380.95	0.83
	512	71779511	3.81%	10.26%	10.57%	179.48	0.88
fc_30_50_2	64	3494056	0.15%	31.46%	9.18%	564.20	1.00
	128	3733703	0.22%	33.25%	21.71%	399.60	0.71
	256	6523893	0.23%	29.99%	28.99%	390.12	0.36
	512	13358819	0.27%	23.54%	29.00%	337.85	0.21
pk1	64	2329865	3.97%	12.00%	5.86%	103.55	1.00
	128	2336213	11.66%	12.38%	10.47%	61.31	0.84
	256	2605461	11.55%	13.93%	20.19%	41.04	0.63
	512	3805593	19.14%	9.07%	26.71%	36.43	0.36

Properties Affecting Scalability

- Shape of search tree (balanced or not)
- Time to process a node
- Number/distribution of feasible solutions
- Relative strength of upper/lower bound (proving optimality)
- Sizes of node descriptions

Benchmarking Tests

Scalability can be tested separately from sequential performance.

Scalability Tests

- Test set with known optima (prove optimality)
- Instances known to have balanced trees
- Instances with small node processing times and large trees
- Instances with large node processing times and small trees
- Instances with large node descriptions

Alternative Measures of Parallel Performance

- Time to optimality may not be the most appropriate measure.
- Most interesting problems cannot be solved easily with small numbers of processors.

Alternative Measures

- Final gap in fixed time
- Time to prove optimality (post facto)
- Time to target gap
- Time to target solution quality
- Time to target upper/lower bound

Tradeoffs

- How important is scalability versus sequential performance?
- The answer depends on the availability of computing resources.
- With large numbers of processors available, good scalability may overcome sub-standard performance.
- Keep in mind, however, that going on level deeper in a balanced tree doubles the size.
- Hence, parallelism is unlikely to be much of a silver bullet.

Outline

1. Introduction
2. Benchmarking
   - Purpose
   - Sequential Codes
   - Parallel Codes
3. Performance Analysis
4. Conclusions

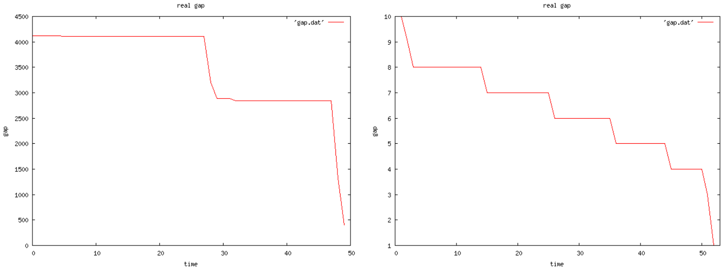
Performance Tuning

- One of the goals of benchmarks is performance tuning.
- Does the information used to benchmark help us to tune?
- Not really, we need more in-depth analysis.
- This section focuses on branch and bound algorithms generally.

Assessing the Performance of B&B

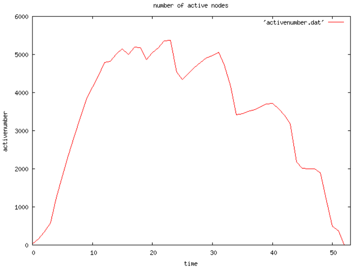
- Benchmarking focuses on aggregate measures, but these measures are not very useful for performance tuning.
- Most commercial and open-source solvers report:
  - optimality gap (global lower and upper bound)
  - number of candidate nodes
  - statistics to indicate use/effectiveness of various components of the algorithm
    - Preprocessing
    - Cutting plane generators
    - Primal heuristics
- These are ultimately not very useful in identifying strategies for performance improvement.

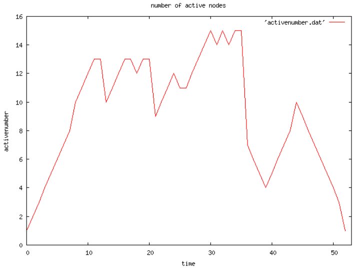
Optimality gap



- Strength: Gives indication of quality of solution
- Strength: Nonincreasing measure
- Weakness: may remain constant for long periods, then drop suddenly

Number of active nodes





- Strength: Indicates "work done" and "work remaining."
- Weakness: may go up and down
- Weakness: each active node counts equally

Deeper Analysis

In principle, there is a wealth of additional information available that can be used to visualize performance.

- Number of nodes of different statuses
  - Candidate
  - Infeasible
  - Branched
  - Fathomed
- For each “feasible” node:
  - LP relaxation value
  - integer infeasibility
  - history/position in tree (e.g., depth and parent)
  - statistics about methods applied

How can we use this information to better assess performance?

The Branch and Bound Analysis Kit (BAK)

- Works with any instrumented solver (currently open-source solvers GLPK, SYMPHONY, and CBC).
- Solver must be modified to provide output when nodes are added and processed.
- A processing script creates visual representations of the data by parsing the output file
  - Output file can be processed at any point during the solving process
  - Parsing is done in Python, images are created with Gnuplot
- Available for download at `http://www.rosemaryroad.org/brady/software/index.html`

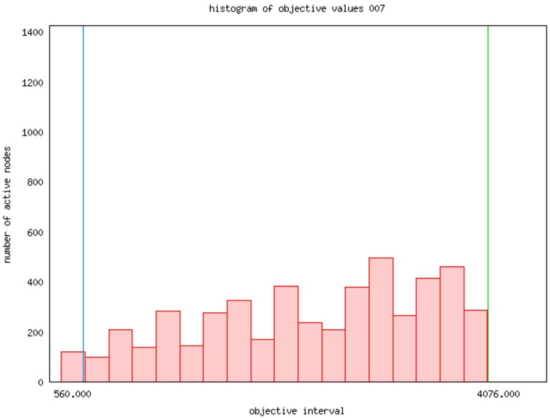
Example of output from solver

```
# CBC
0.040003 heuristic -28.000000
2.692169 branched 0 -1 N -39.248099 16 0.169729
2.692169 pregnant 2 0 R -39.248063 14 105.991922
2.708170 pregnant 3 0 L -38.939929 6 0.105246
2.764173 pregnant 5 2 R -39.244862 12 49.115388
2.764173 branched 2 0 R -39.248063 14 105.991922
```

Visual Representations

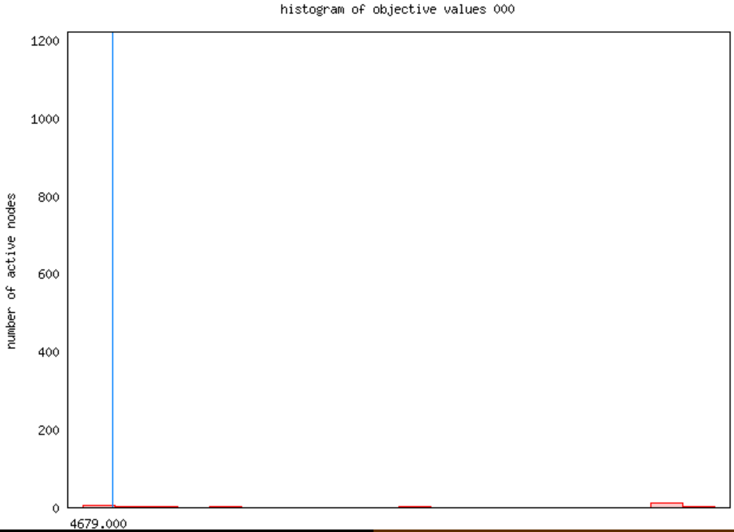
- Histogram of active node LP bounds
- Scatter plot of active node LP bounds & integer infeasibility
- Incumbent node history in scatter plot
- B&B trees showing the LP bound of each node

Visualization tools: Histogram of active node LP bounds

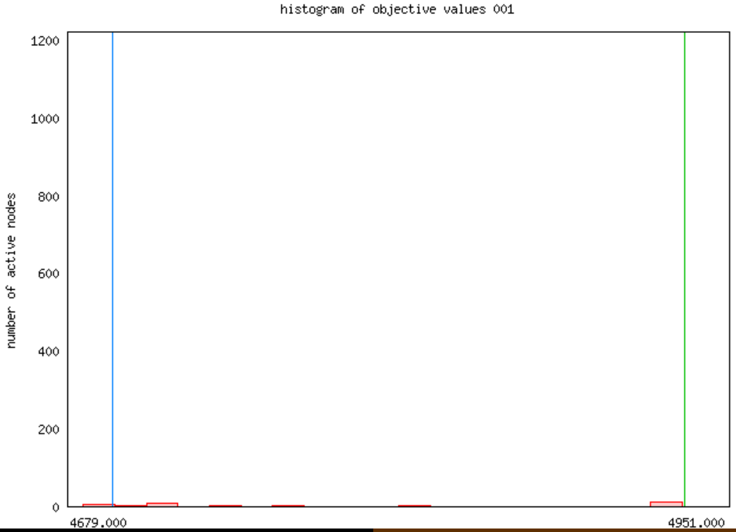


- Horizontal axis is the LP bound
- Vertical axis is number of active nodes
- Green vertical line shows the current incumbent value and the blue one

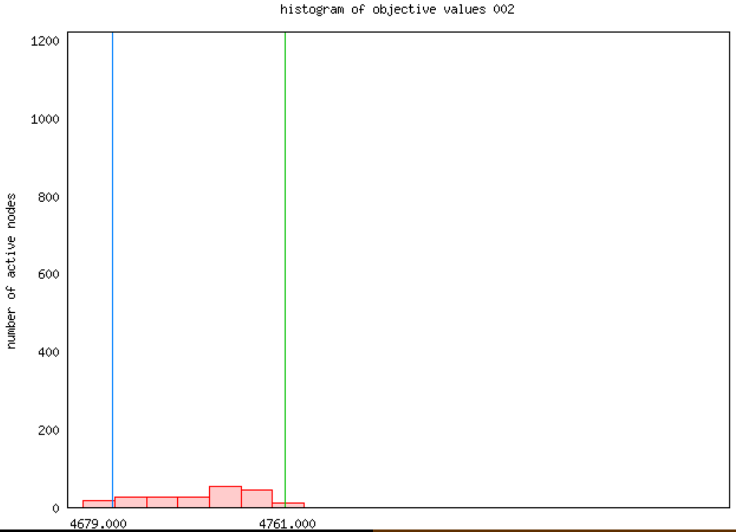
Example histogram series 1: 1152lav



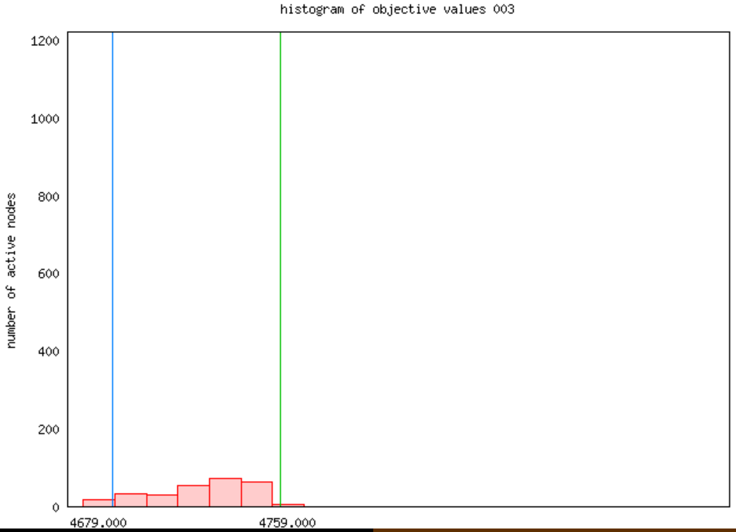
Example histogram series 1: 1152lav



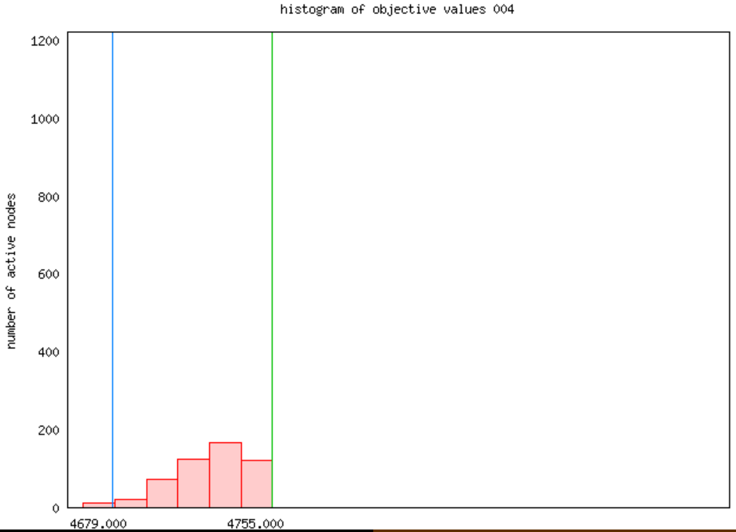
Example histogram series 1: 1152lav



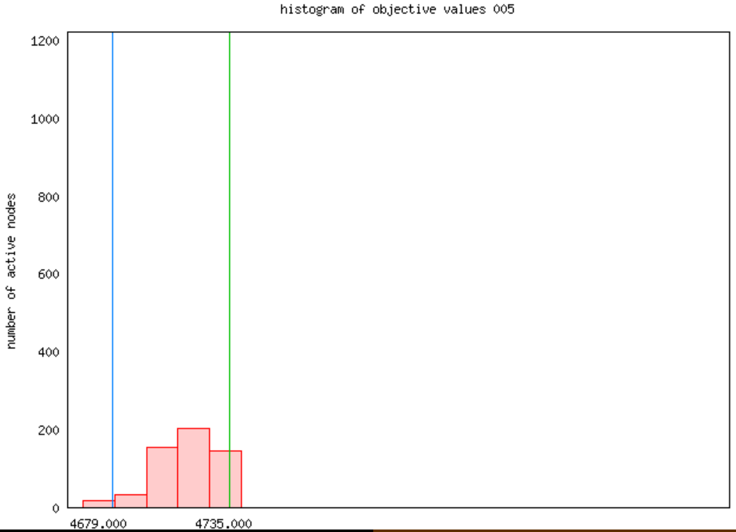
Example histogram series 1: 1152lav



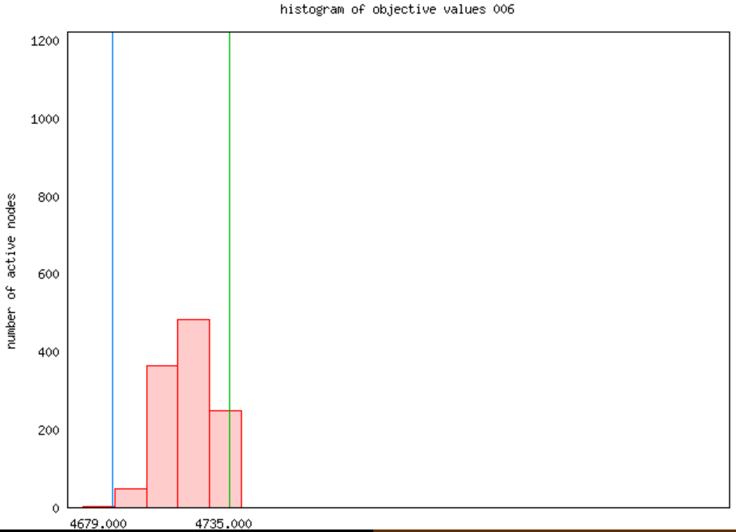
Example histogram series 1: 1152lav



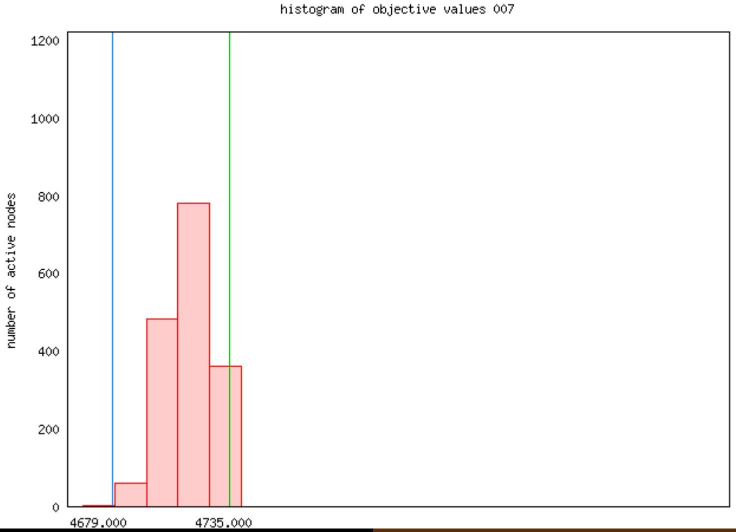
Example histogram series 1: 1152lav



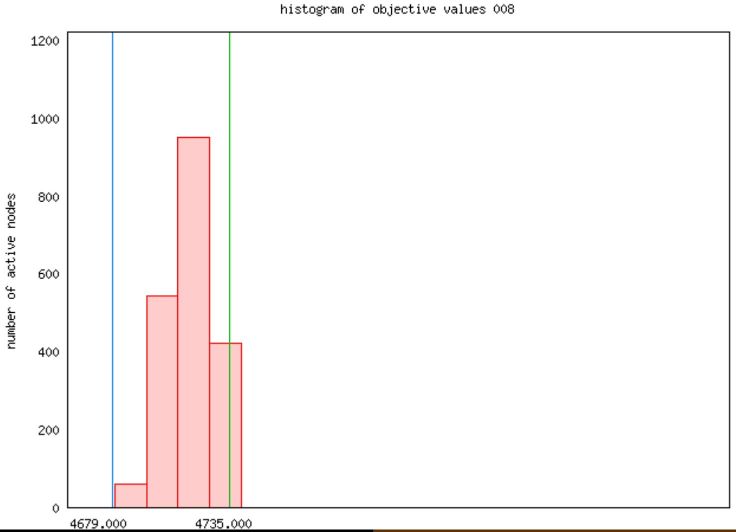
Example histogram series 1: 1152lav



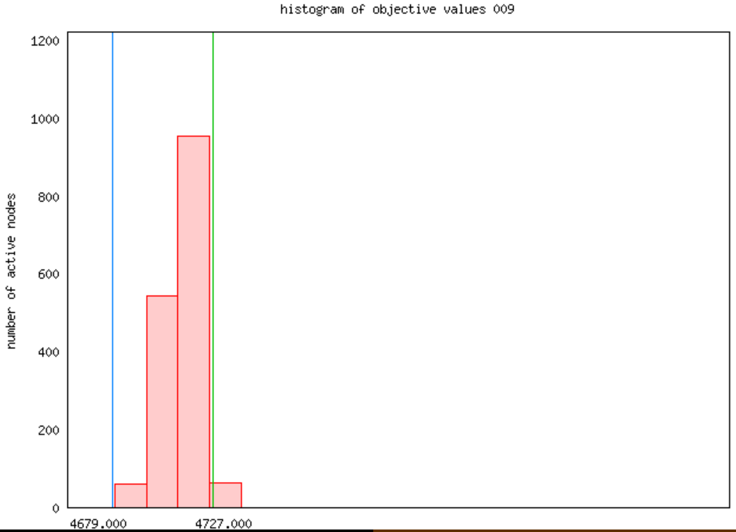
Example histogram series 1: 1152lav



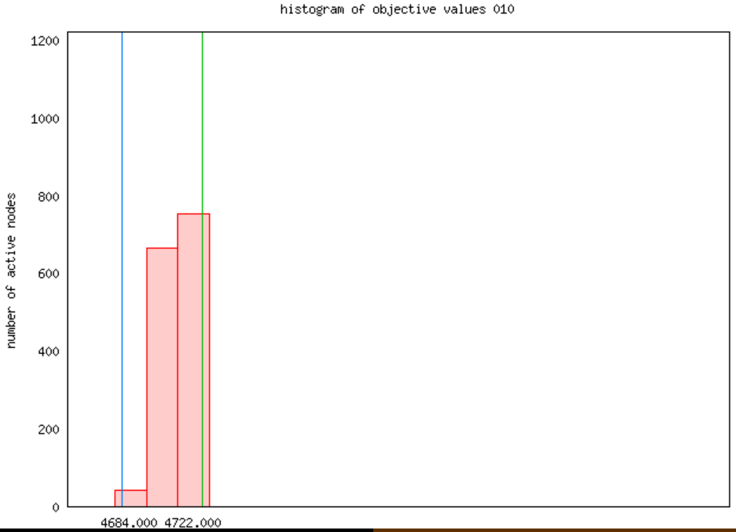
Example histogram series 1: 1152lav



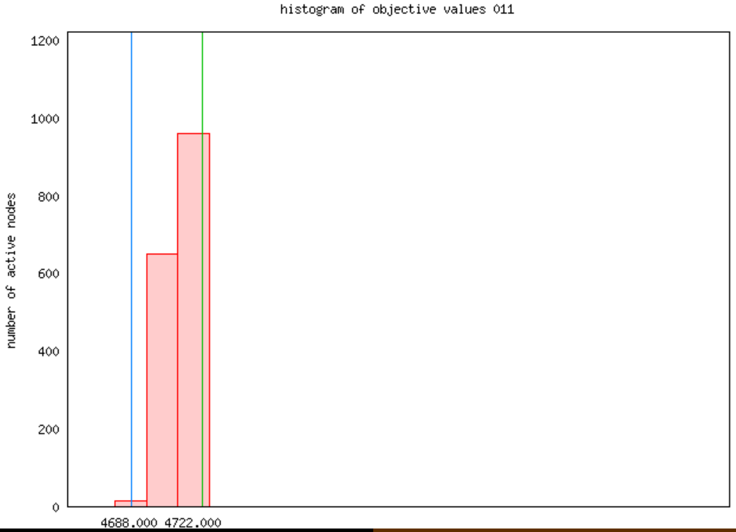
Example histogram series 1: 1152lav



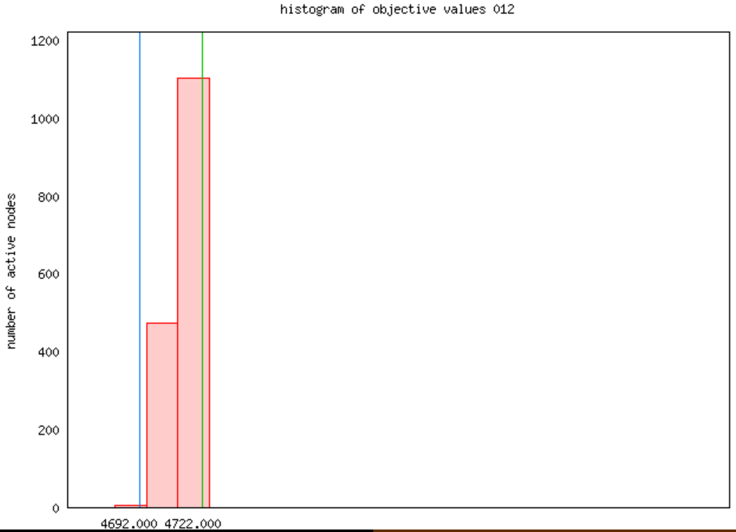
Example histogram series 1: 1152lav



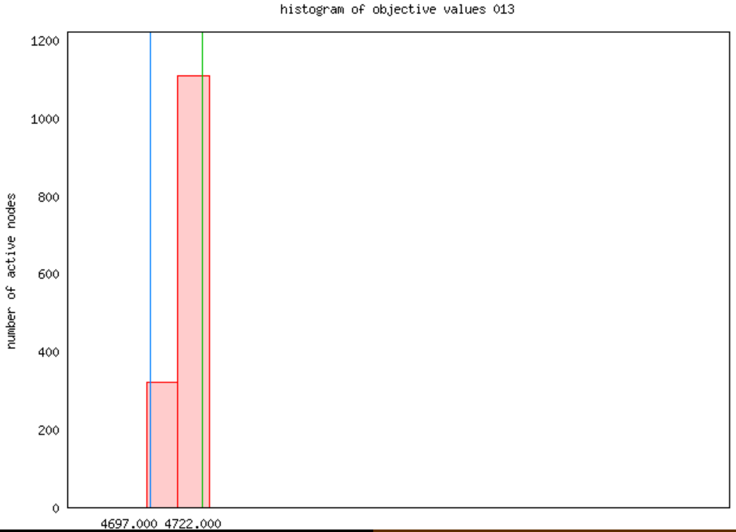
Example histogram series 1: 1152lav



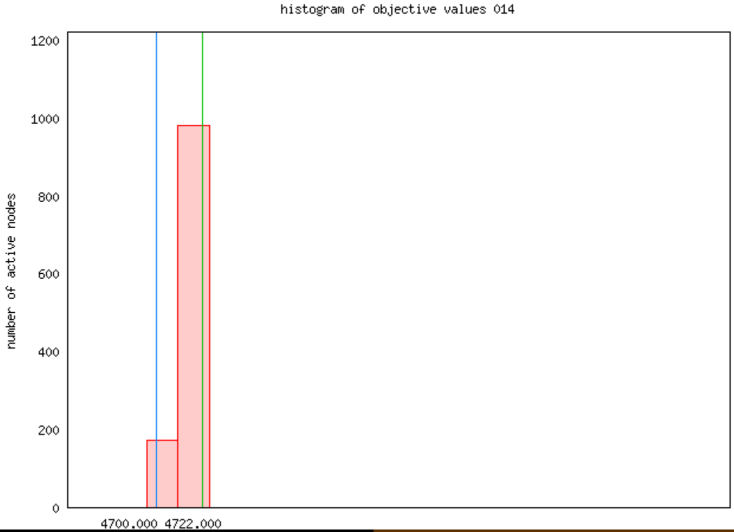
Example histogram series 1: 1152lav



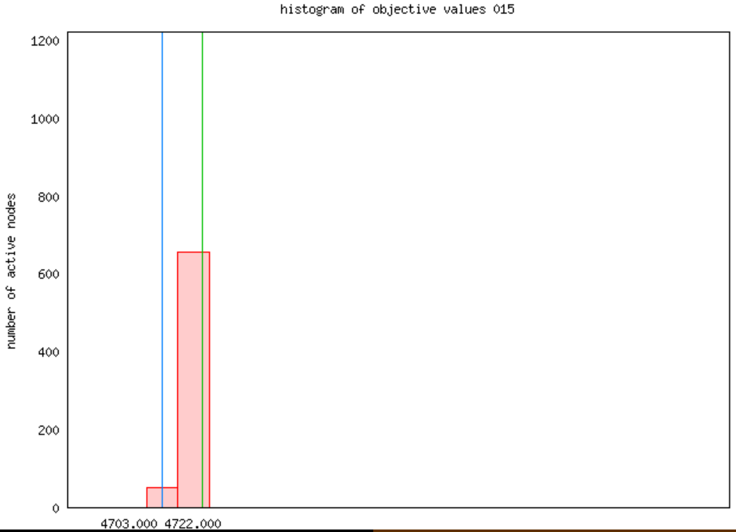
Example histogram series 1: 1152lav



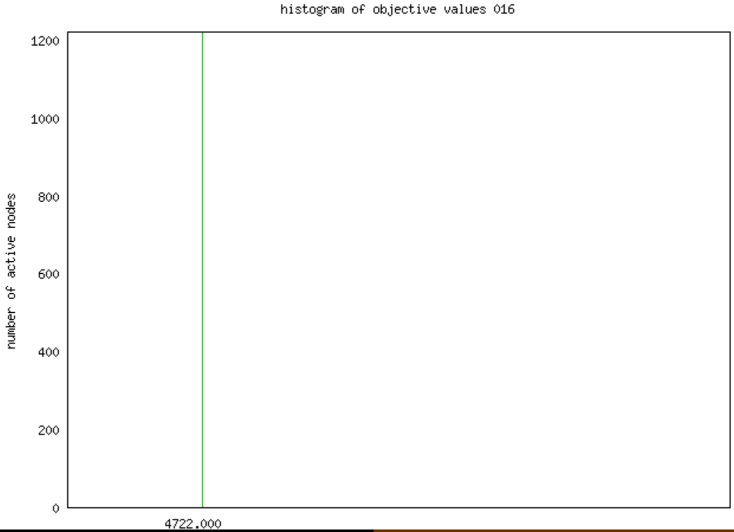
Example histogram series 1: 1152lav



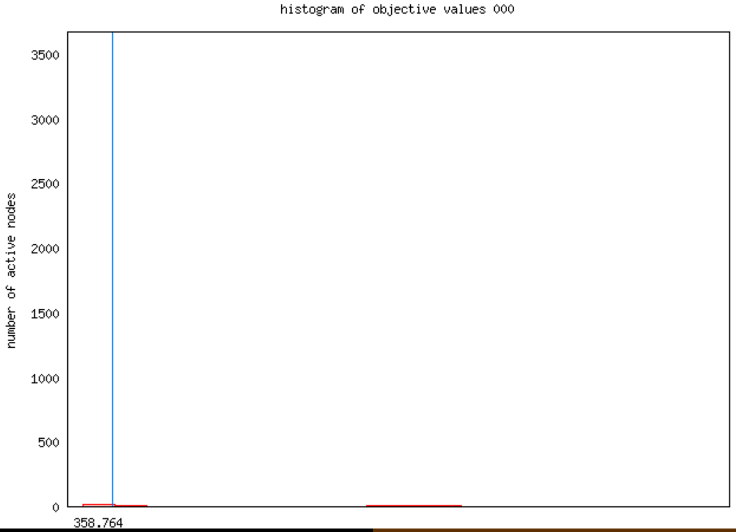
Example histogram series 1: 1152lav



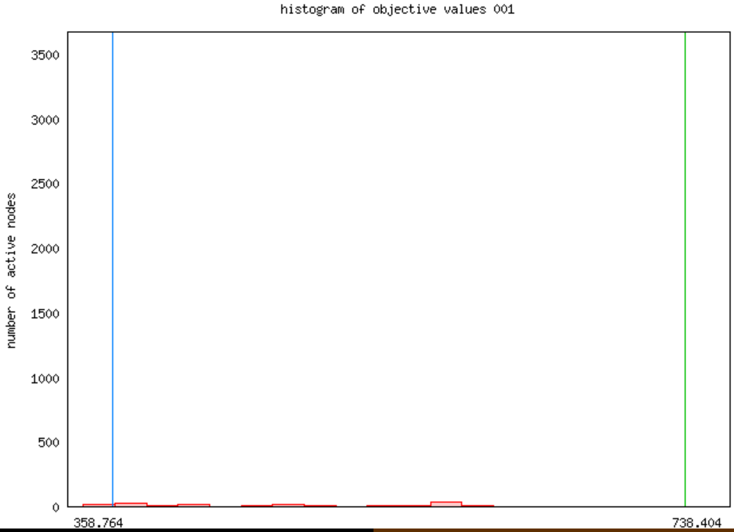
Example histogram series 1: 1152lav



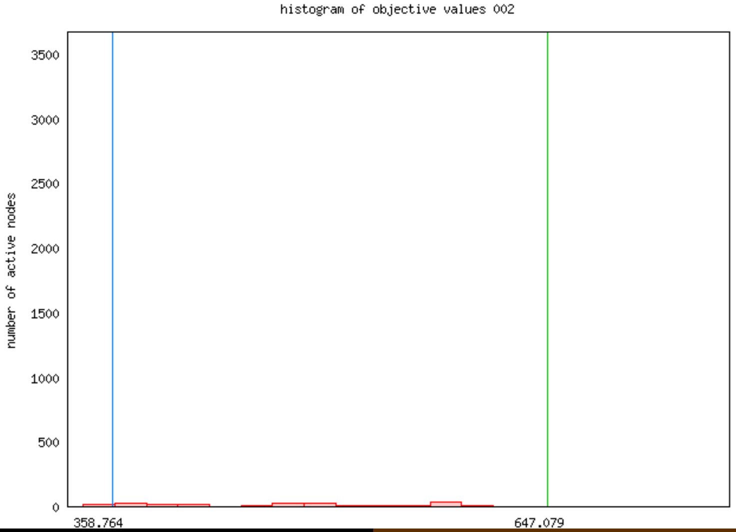
Example histogram series 2: swath



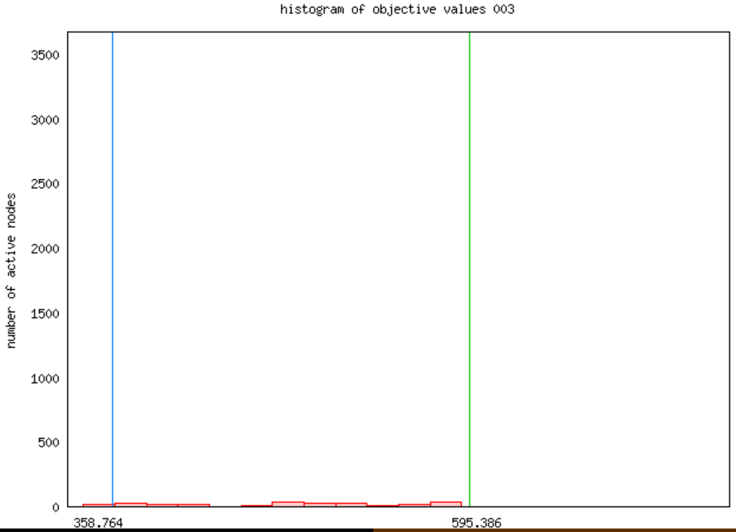
Example histogram series 2: swath



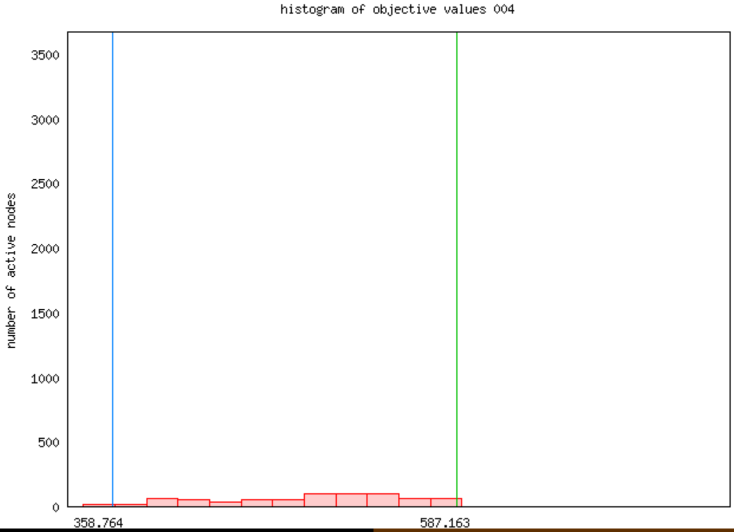
Example histogram series 2: swath



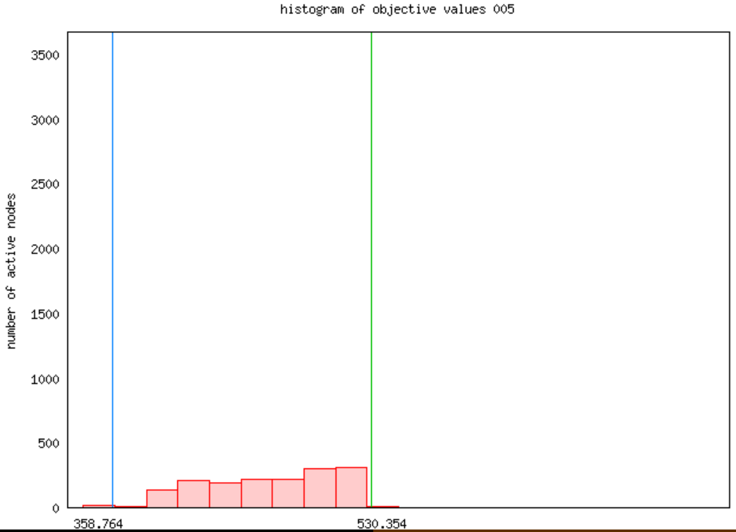
Example histogram series 2: swath



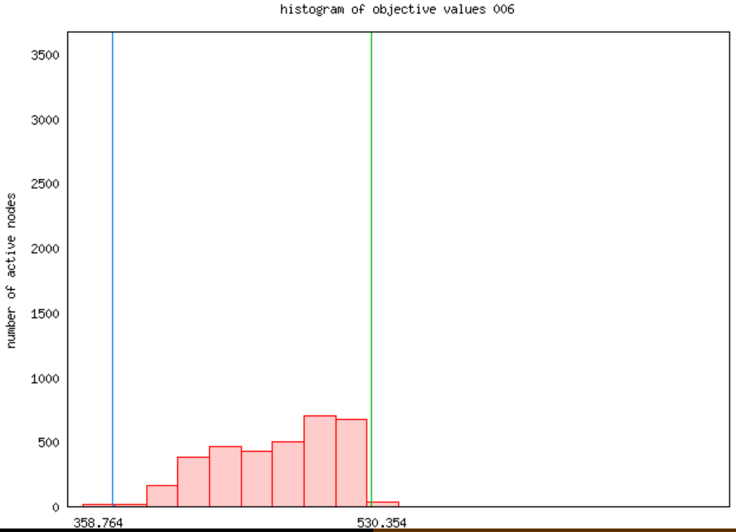
Example histogram series 2: swath



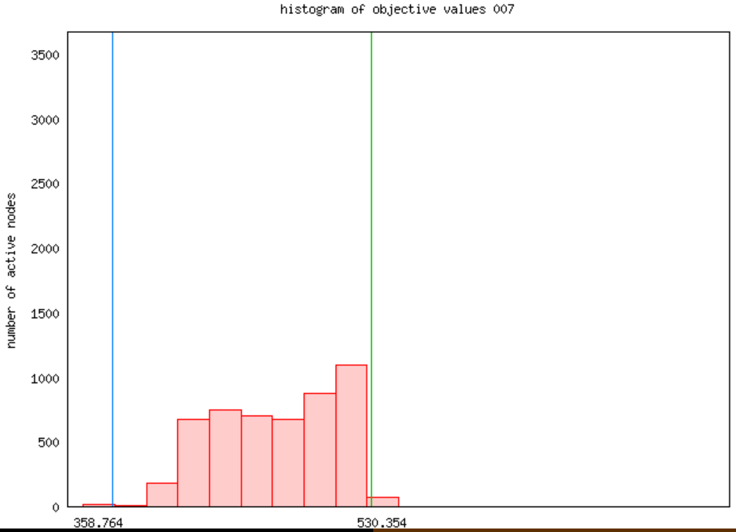
Example histogram series 2: swath



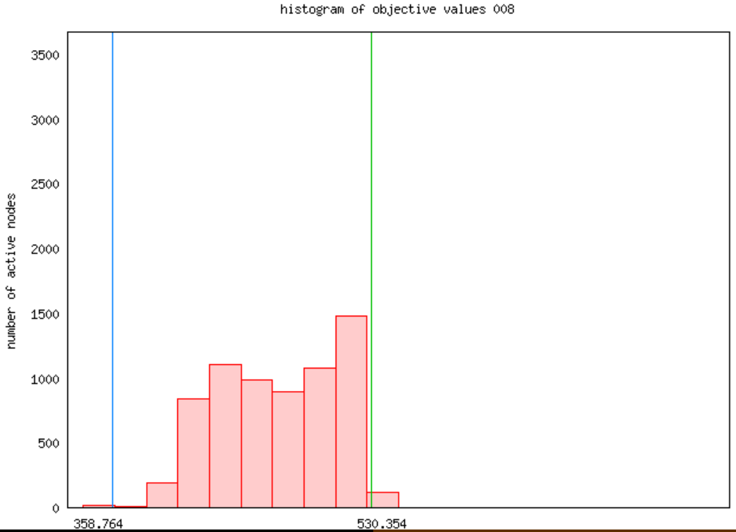
Example histogram series 2: swath



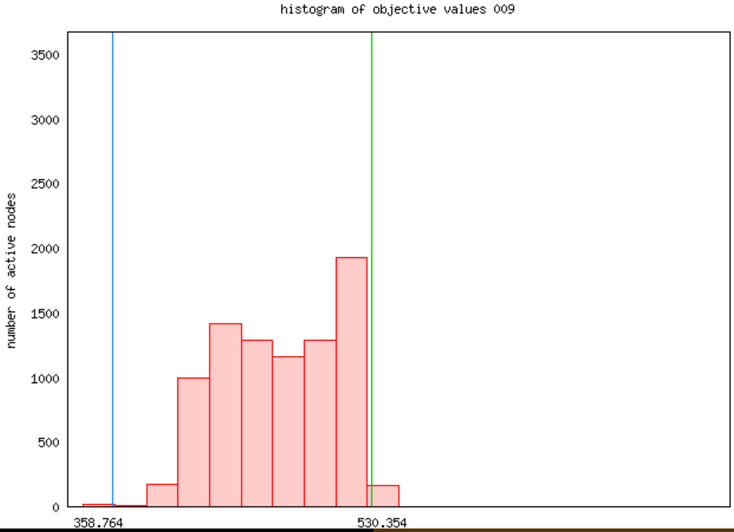
Example histogram series 2: swath



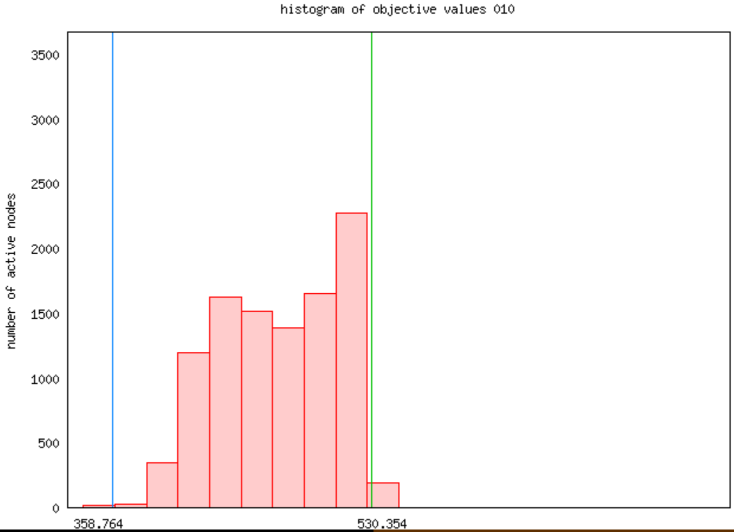
Example histogram series 2: swath



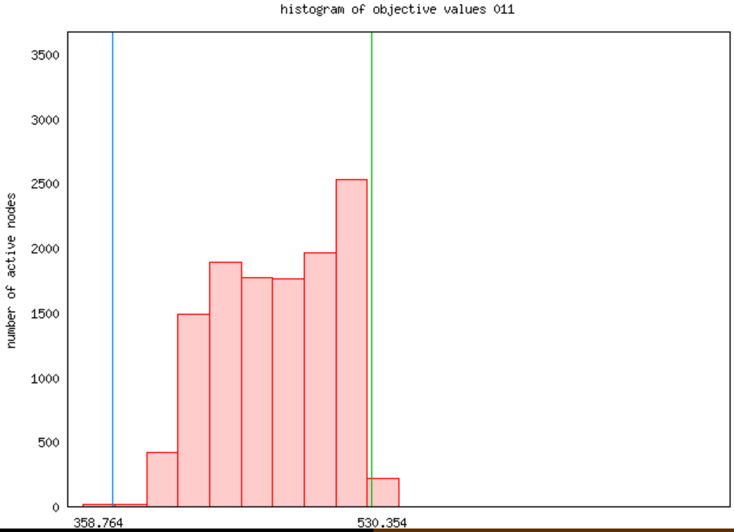
Example histogram series 2: swath



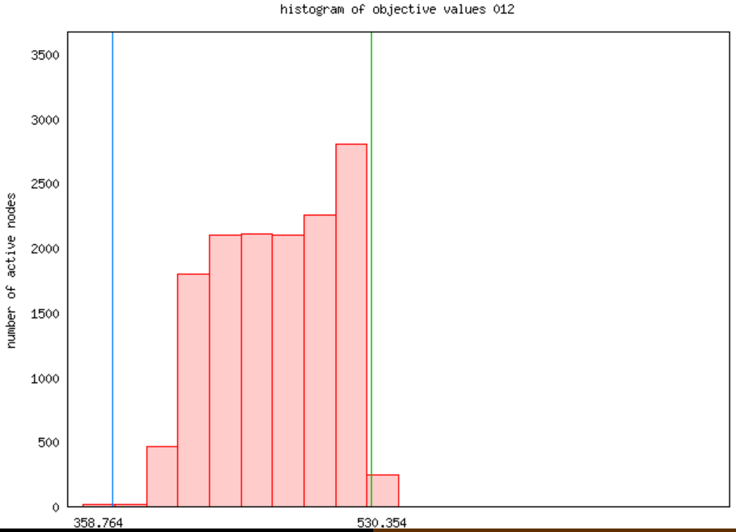
Example histogram series 2: swath



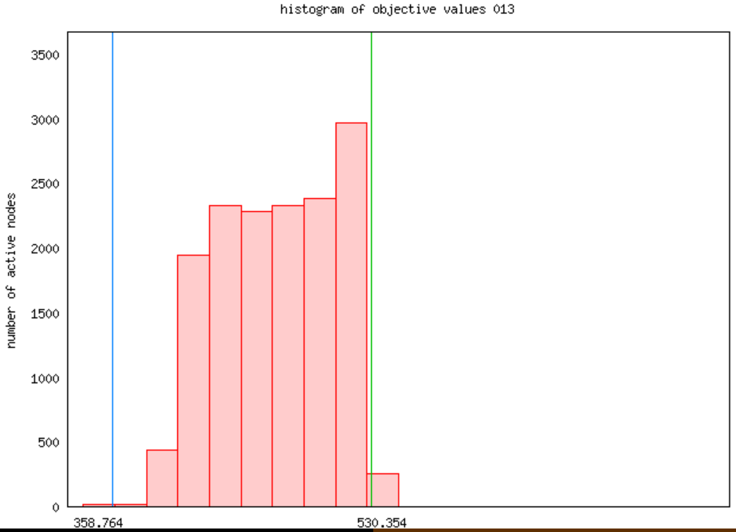
Example histogram series 2: swath



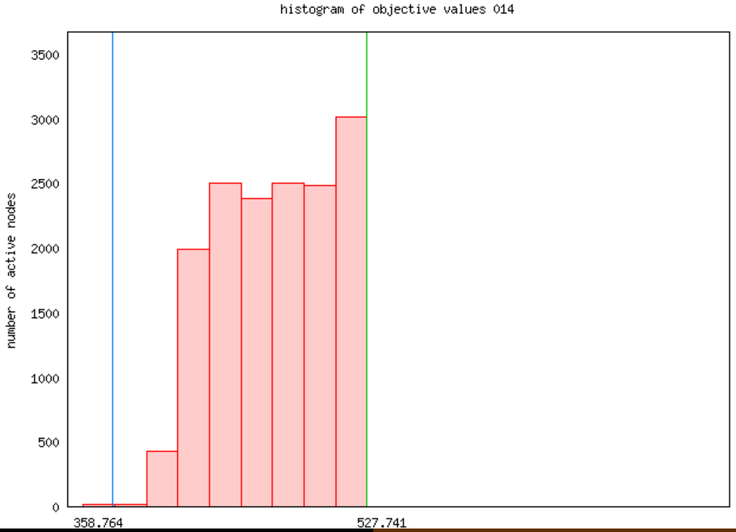
Example histogram series 2: swath



Example histogram series 2: swath

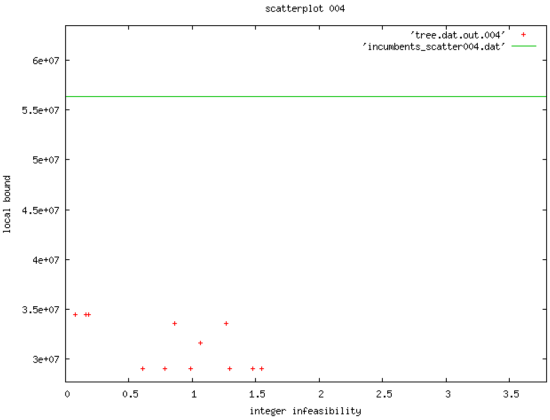


Example histogram series 2: swath



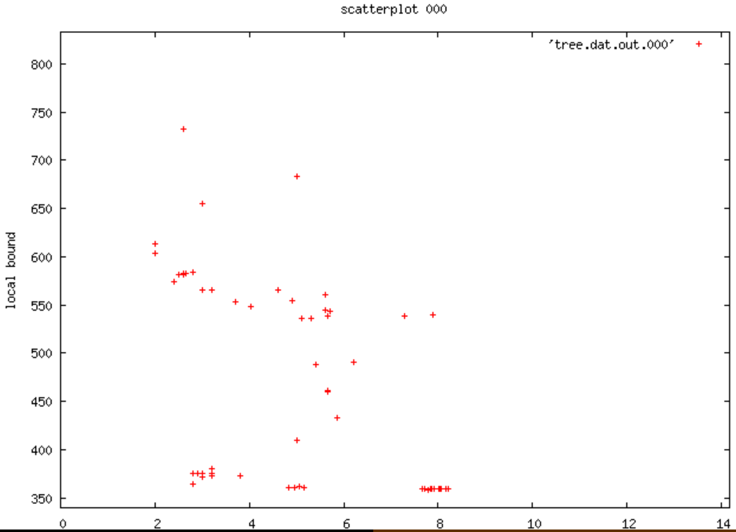
Example histogram series 2: swath

Visualization tools: Scatter plot

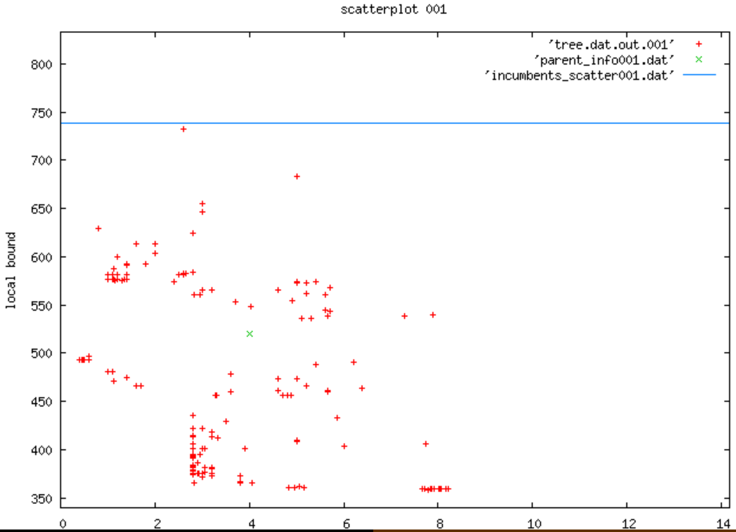


- Horizontal axis is the integer infeasibility
- Vertical axis is the LP bound
- Green horizontal line is the current incumbent value

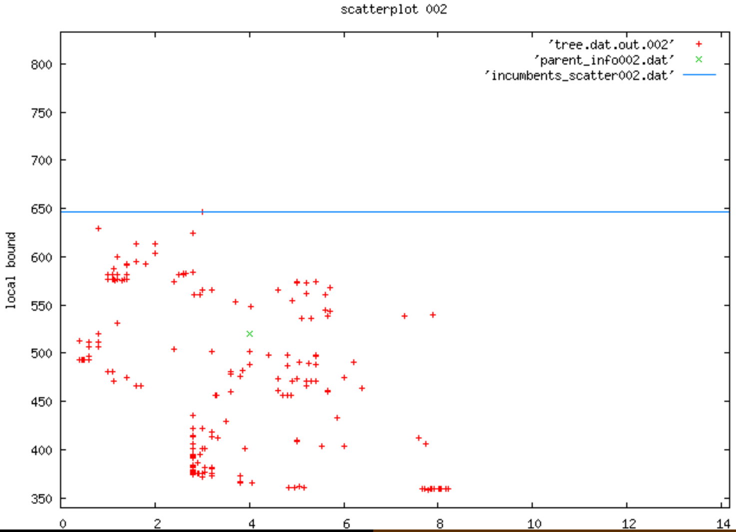
Example scatter plot series 1: swath



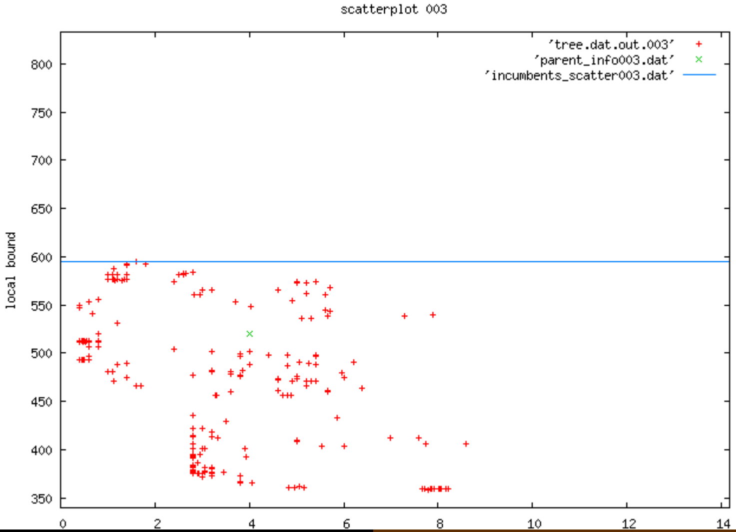
Example scatter plot series 1: swath



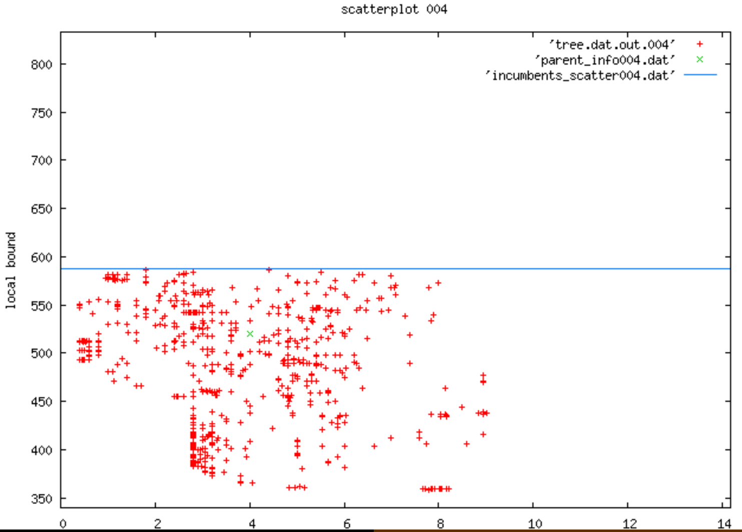
Example scatter plot series 1: swath



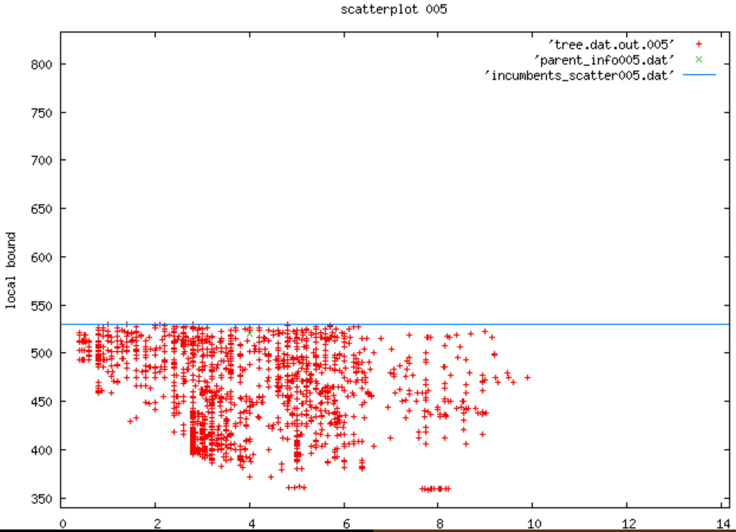
Example scatter plot series 1: swath



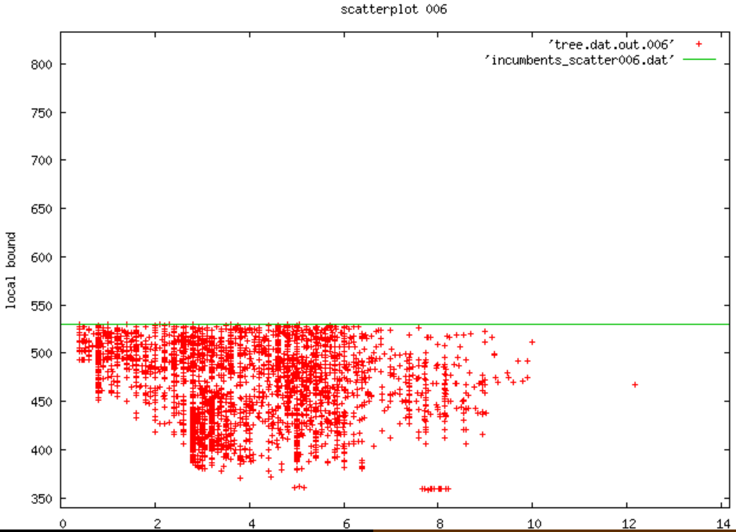
Example scatter plot series 1: swath



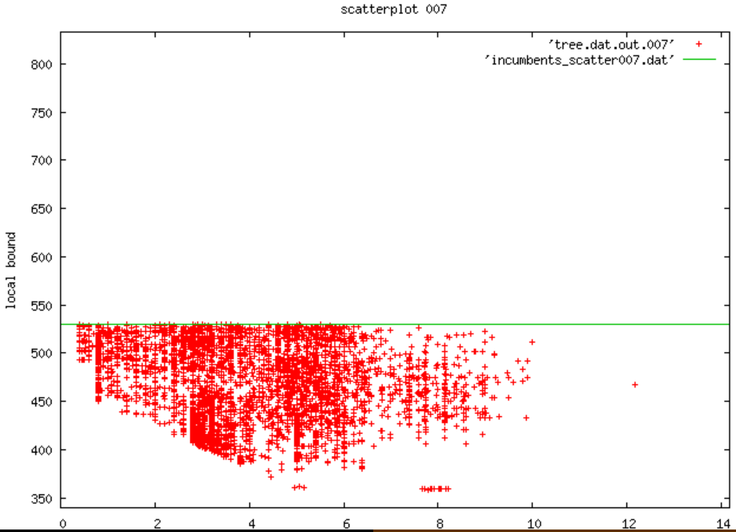
Example scatter plot series 1: swath



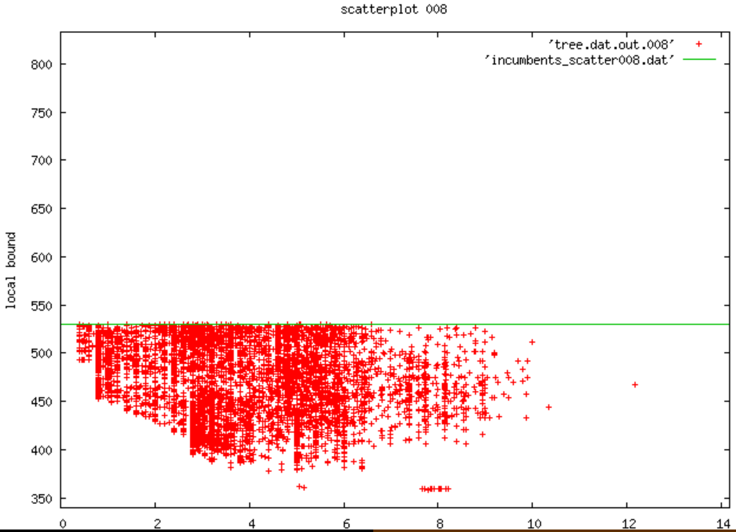
Example scatter plot series 1: swath



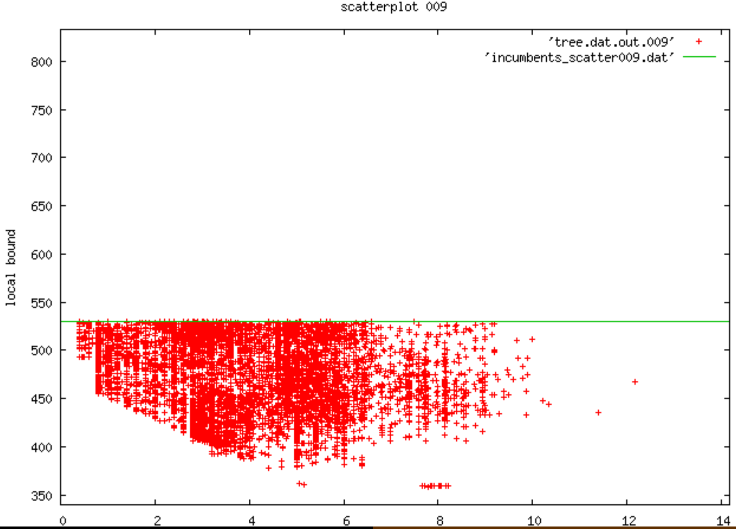
Example scatter plot series 1: swath



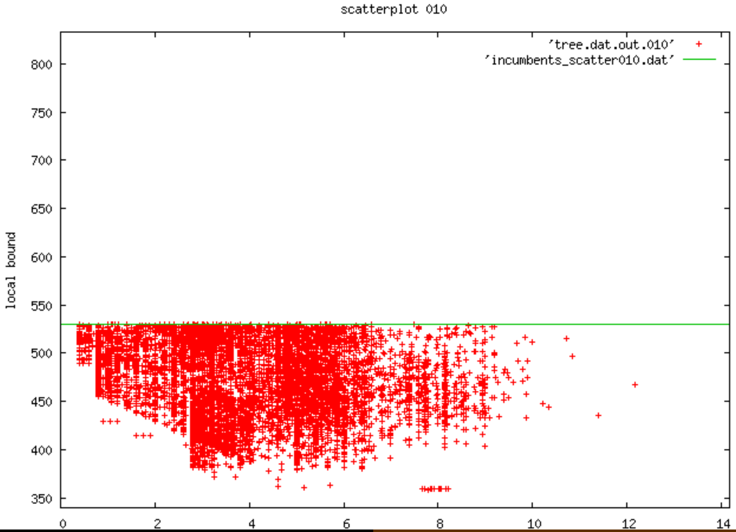
Example scatter plot series 1: swath



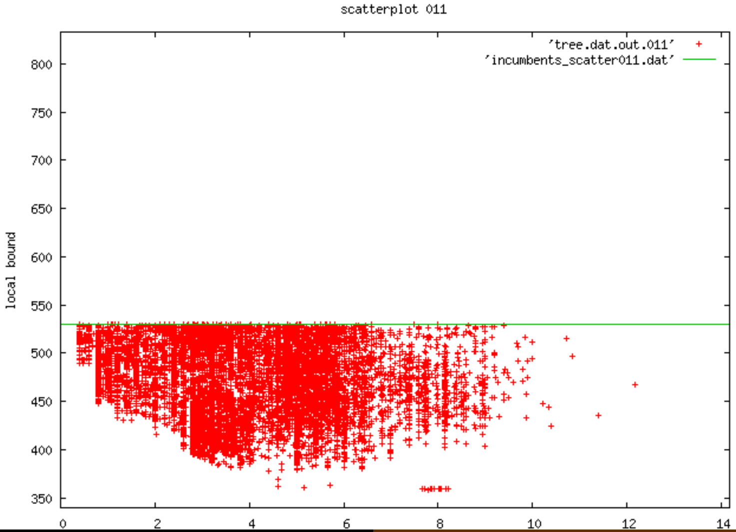
Example scatter plot series 1: swath



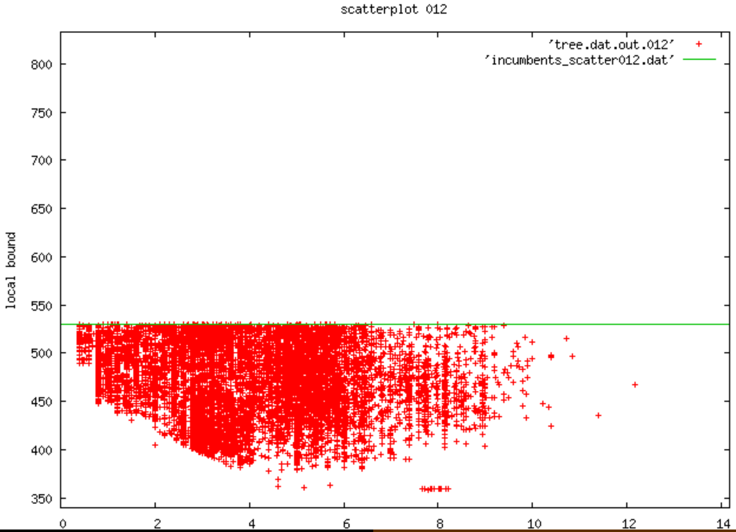
Example scatter plot series 1: swath



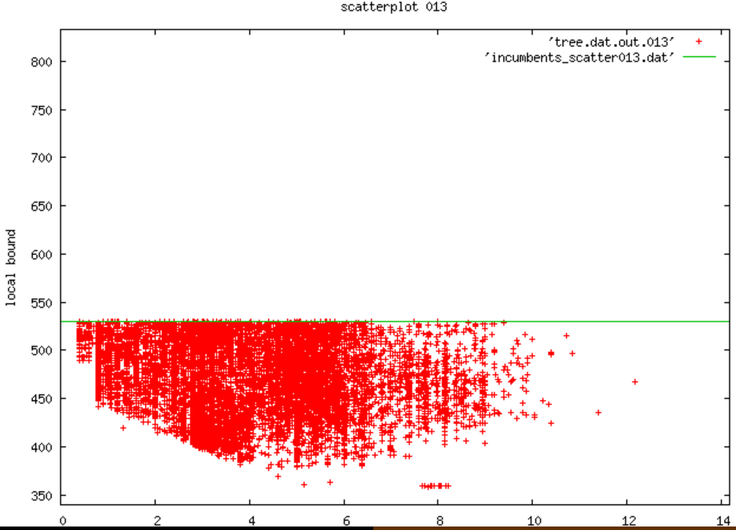
Example scatter plot series 1: swath



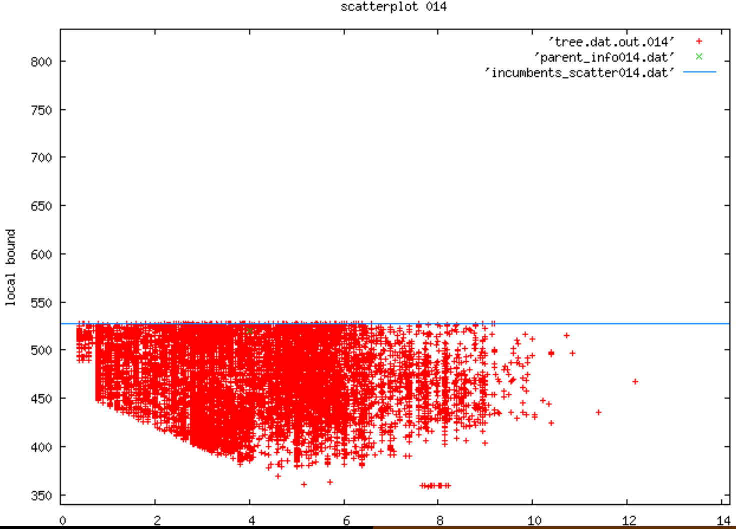
Example scatter plot series 1: swath



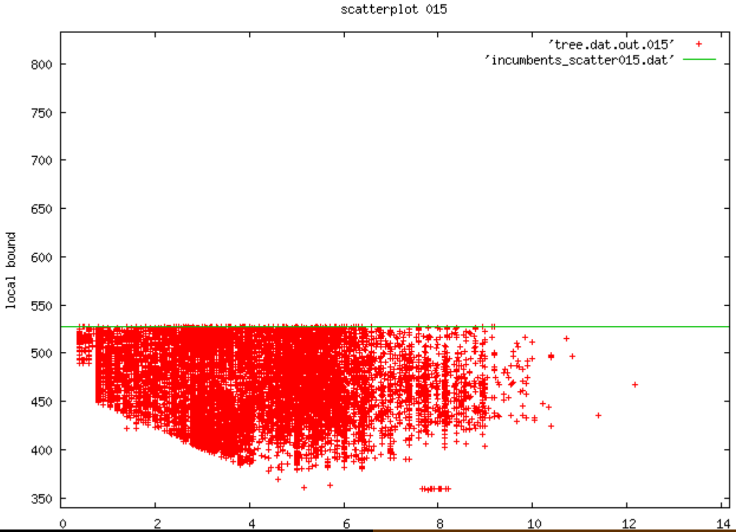
Example scatter plot series 1: swath



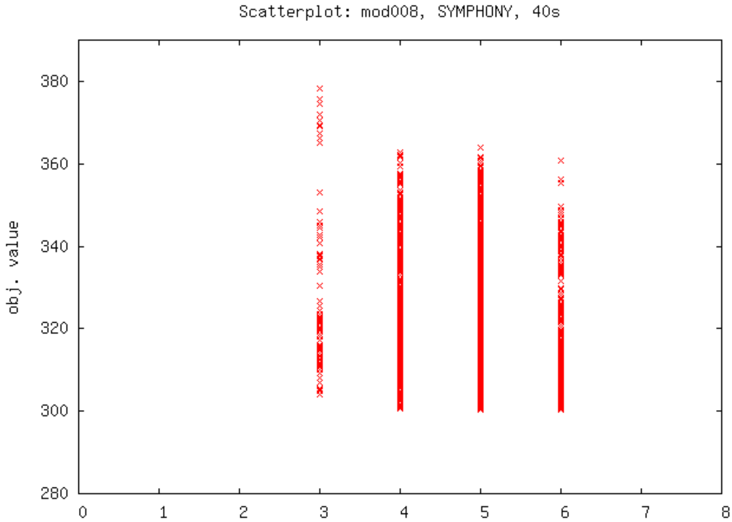
Example scatter plot series 1: swath



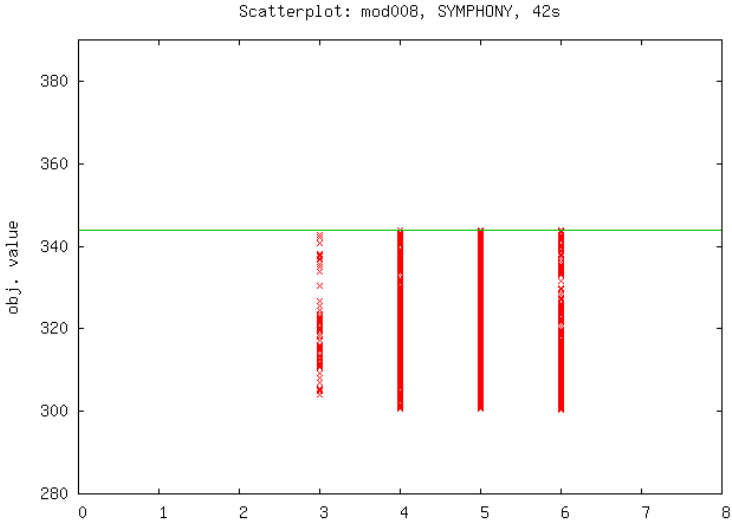
Example scatter plot series 1: swath



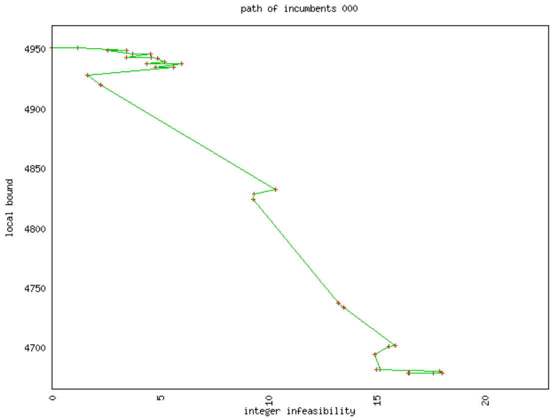
Patterns in integer infeasibility: SYMPHONY



Patterns in integer infeasibility: SYMPHONY

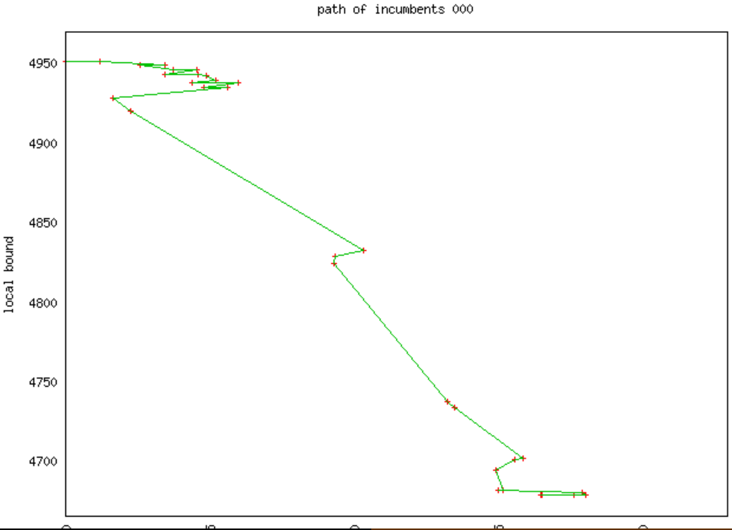


Visualization tools: Incumbent node history in scatter plot

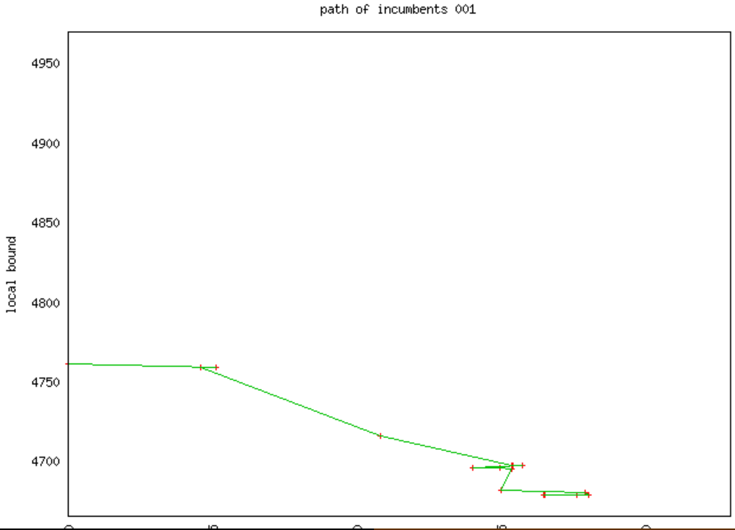


- Horizontal axis is the integer infeasibility
- Vertical axis is the LP bound
- Green line shows ancestors of the incumbent node

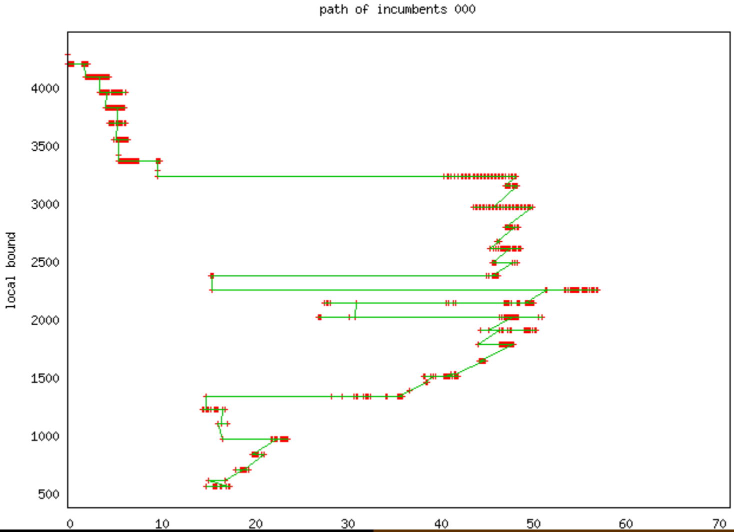
Example incumbent node history series 1: 1152lav



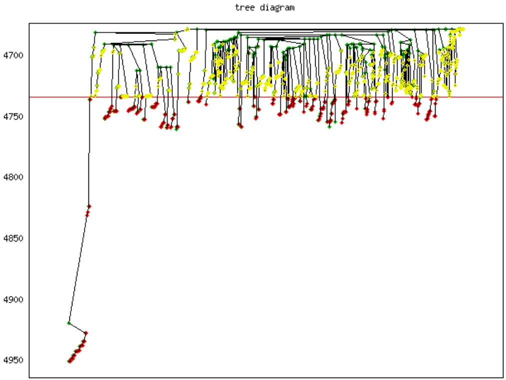
Example incumbent node history series 1: 1152lav



Example incumbent node history series 2: liu

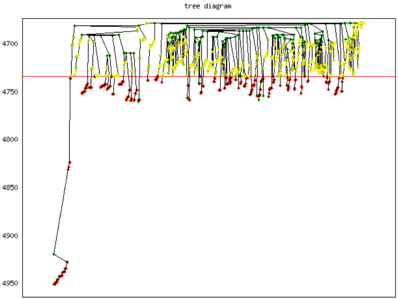


Visualization tools: B&B trees



- Vertical axis is the LP bound
- Nodes are horizontally positioned to make the pictures more readable
- Alternatively, horizontal positions may be fixed based on position in the tree

Visualization tools: B&B trees

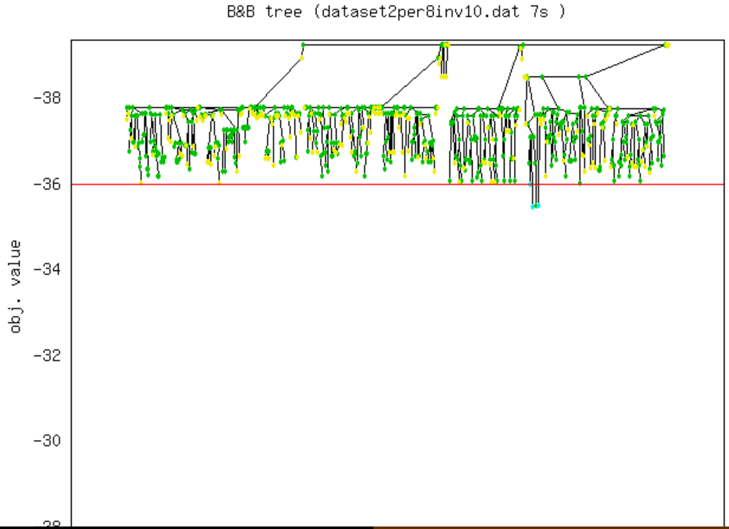


- Node color legend:
    - green: branched
    - yellow: candidate or pregnant
    - red: fathomed
    - blue: infeasible

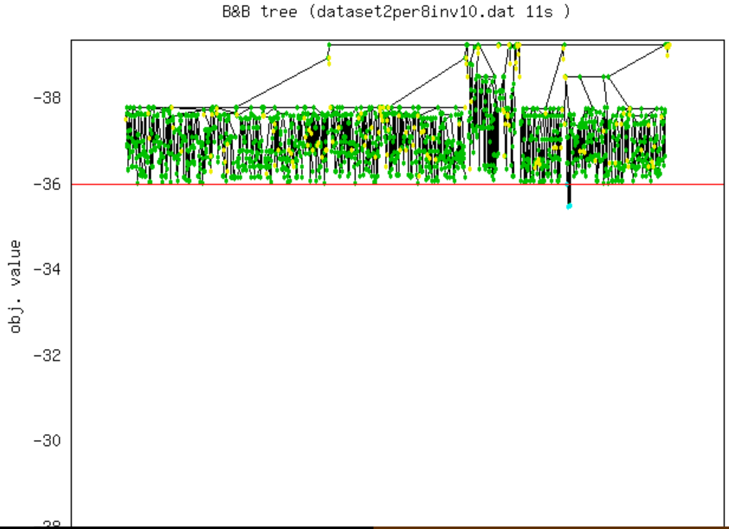
Example B&B trees

Example B&B trees

Example B&B trees

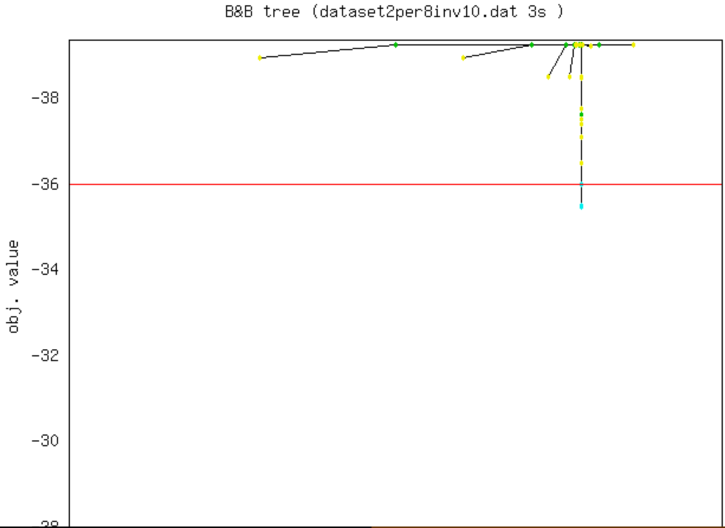


Example B&B trees



Example B&B trees

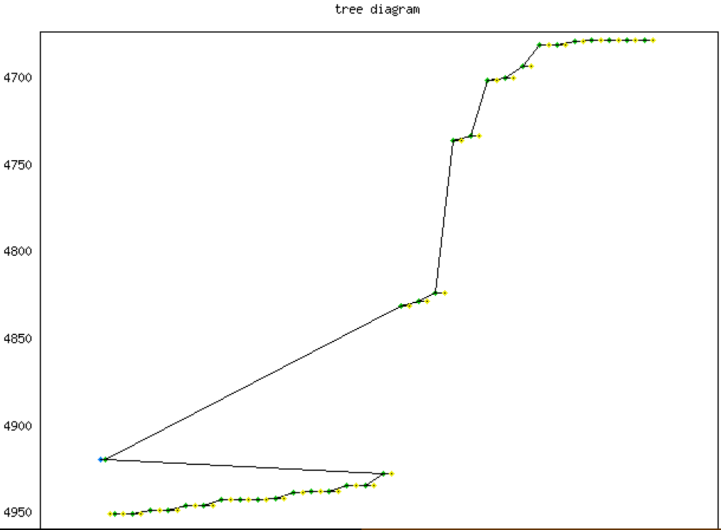
Example B&B trees



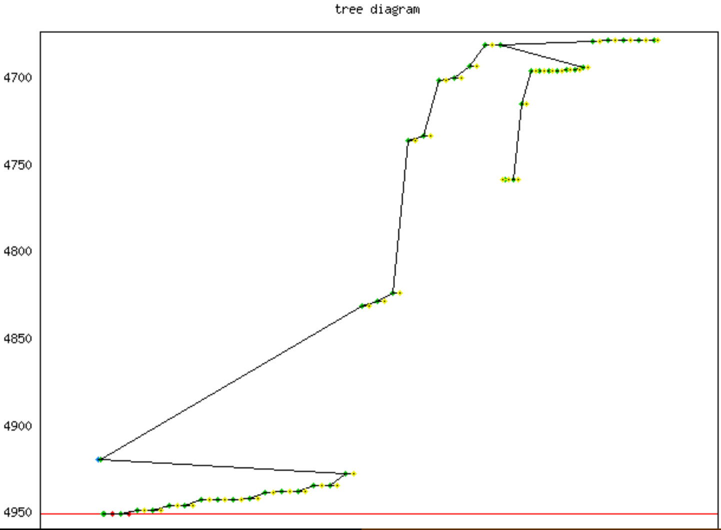
Example B&B trees

Example B&B trees

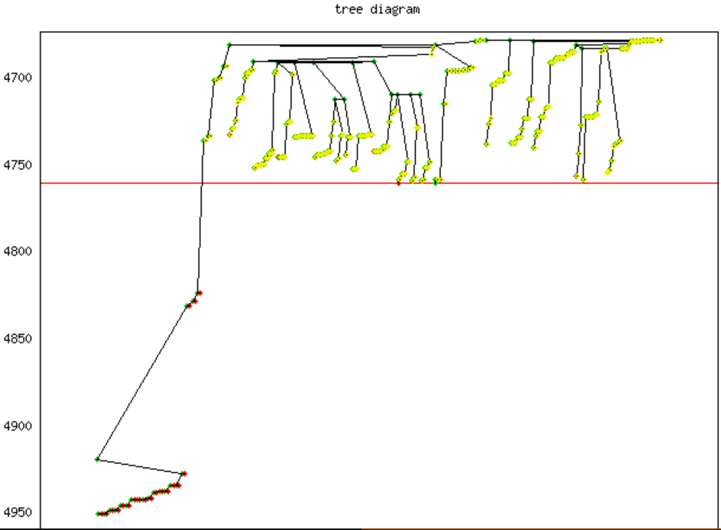
Example B&B tree series 1: l152lav



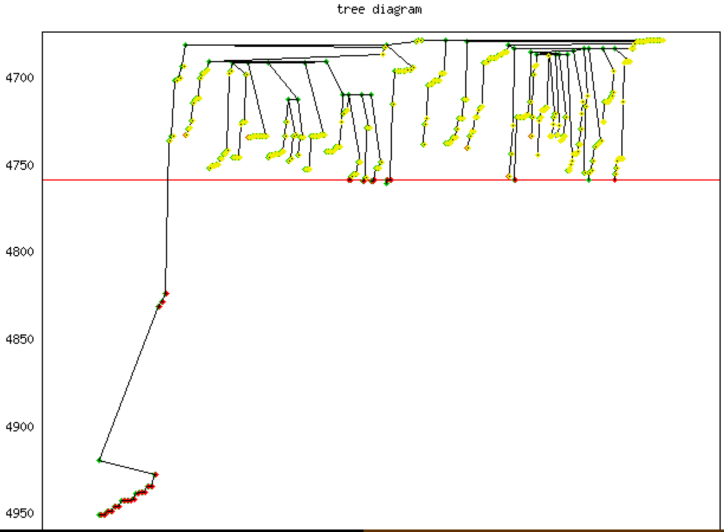
Example B&B tree series 1: l152lav



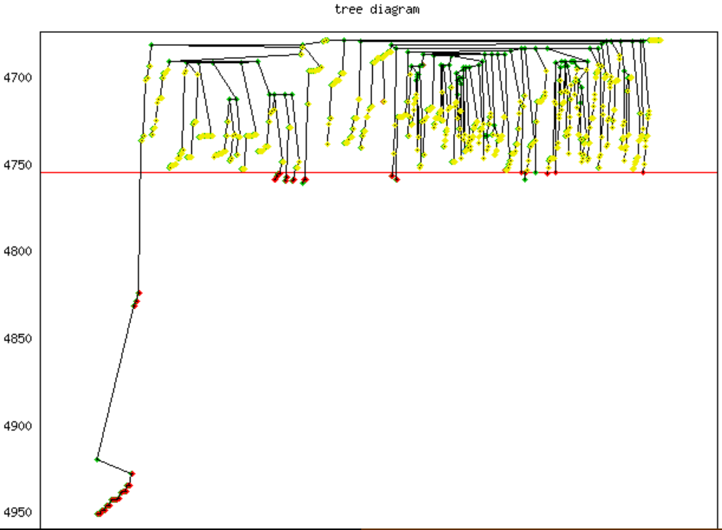
Example B&B tree series 1: l152lav



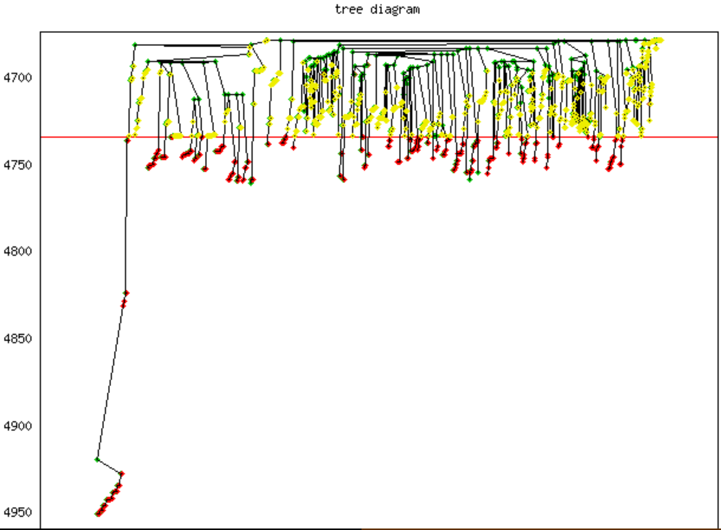
Example B&B tree series 1: l152lav



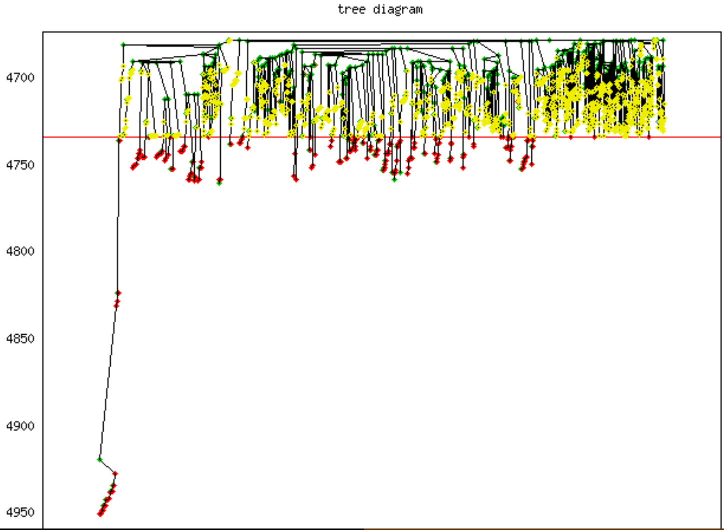
Example B&B tree series 1: l152lav



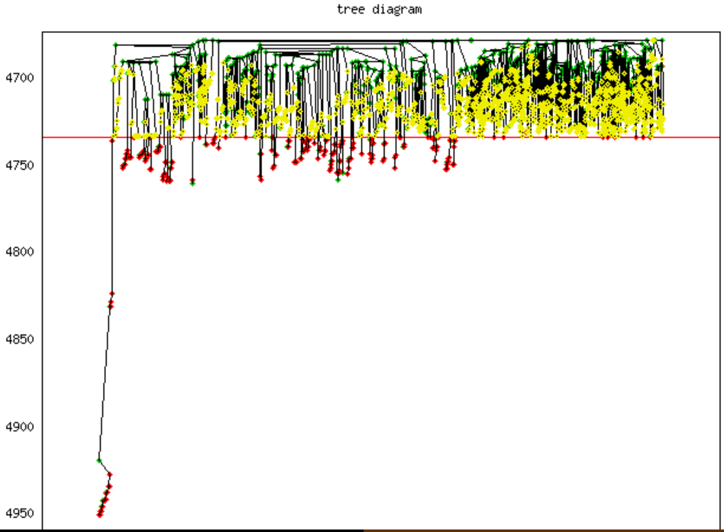
Example B&B tree series 1: l152lav



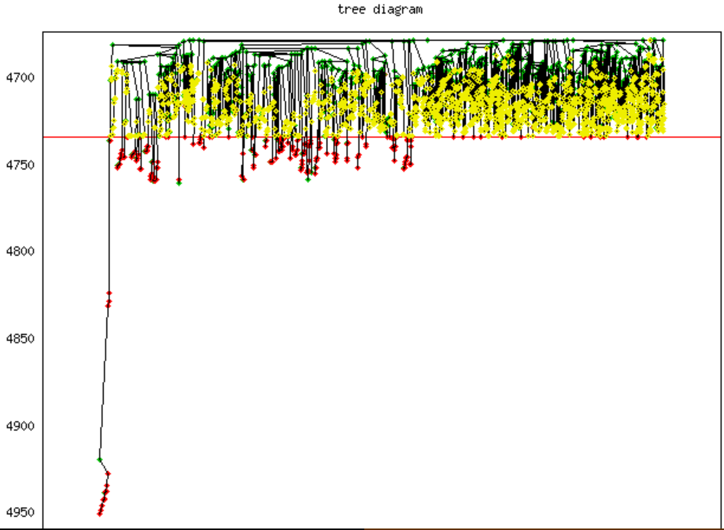
Example B&B tree series 1: l152lav



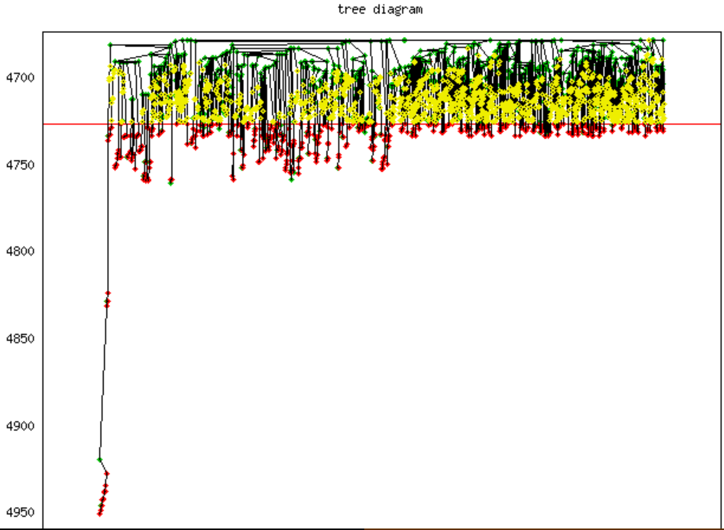
Example B&B tree series 1: l152lav



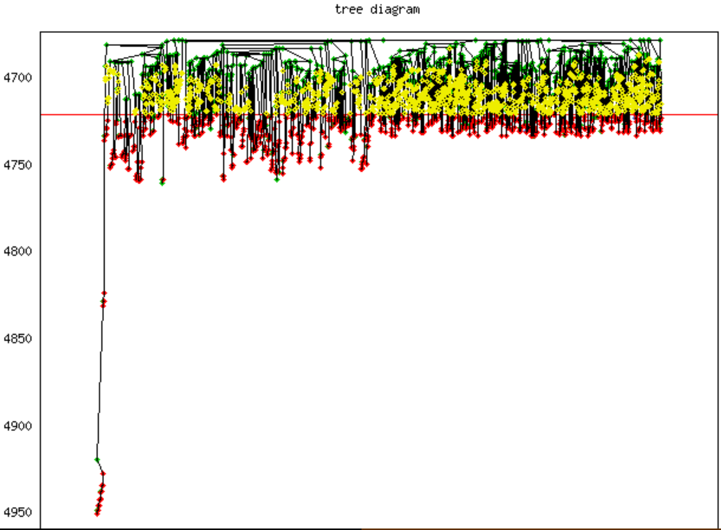
Example B&B tree series 1: l152lav



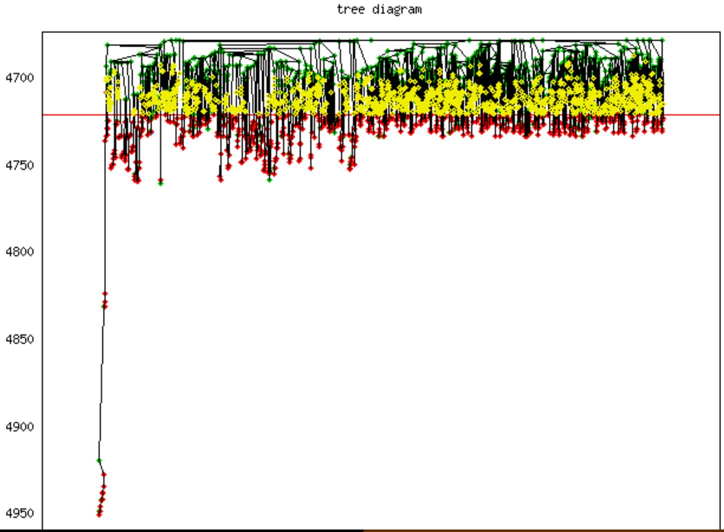
Example B&B tree series 1: l152lav



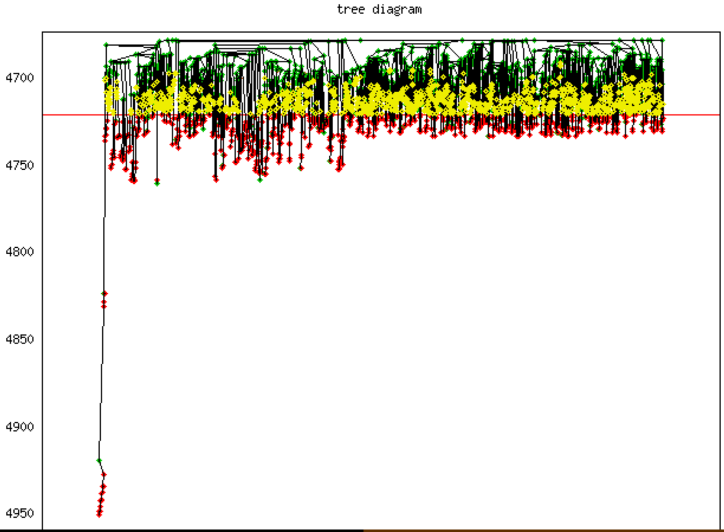
Example B&B tree series 1: l152lav



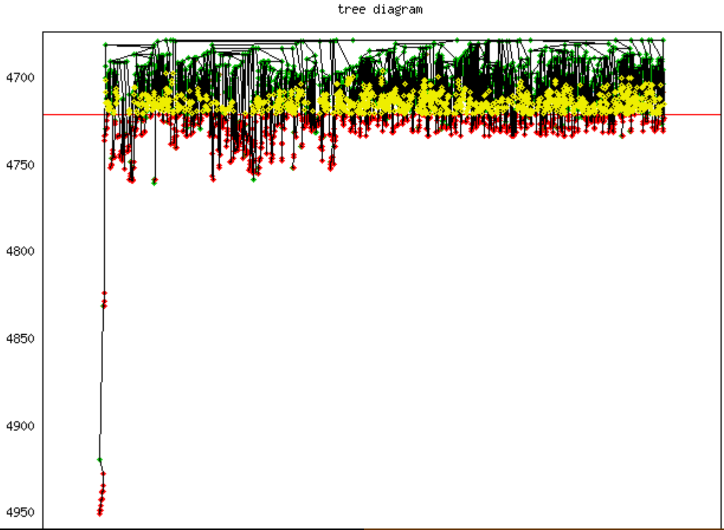
Example B&B tree series 1: l152lav



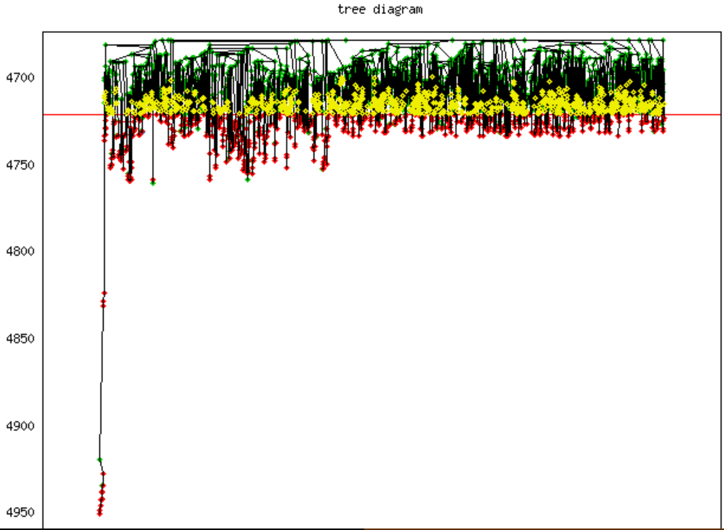
Example B&B tree series 1: l152lav



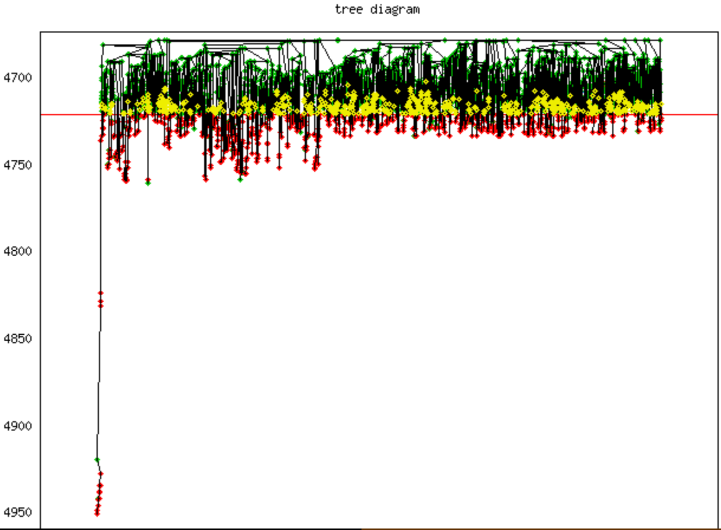
Example B&B tree series 1: l152lav



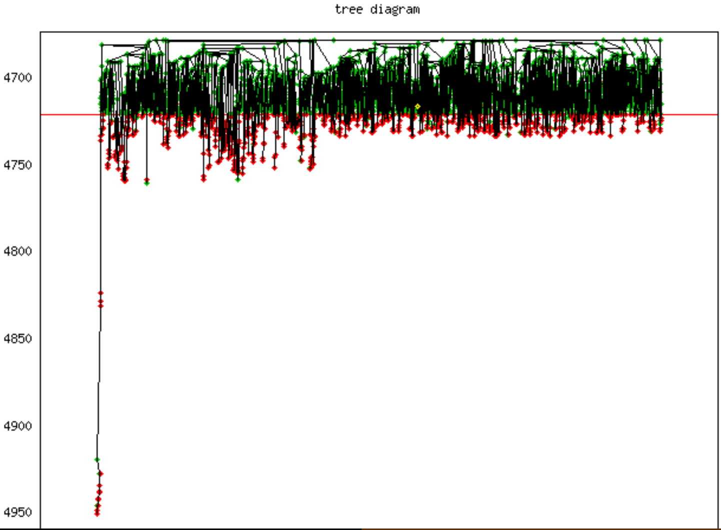
Example B&B tree series 1: l152lav



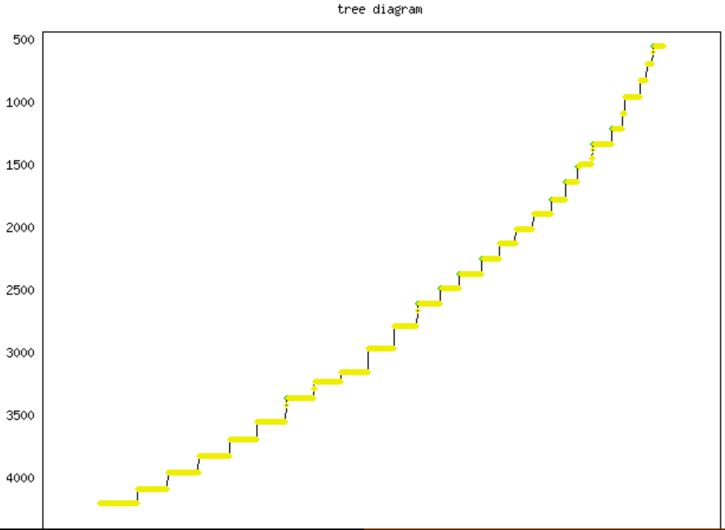
Example B&B tree series 1: l152lav



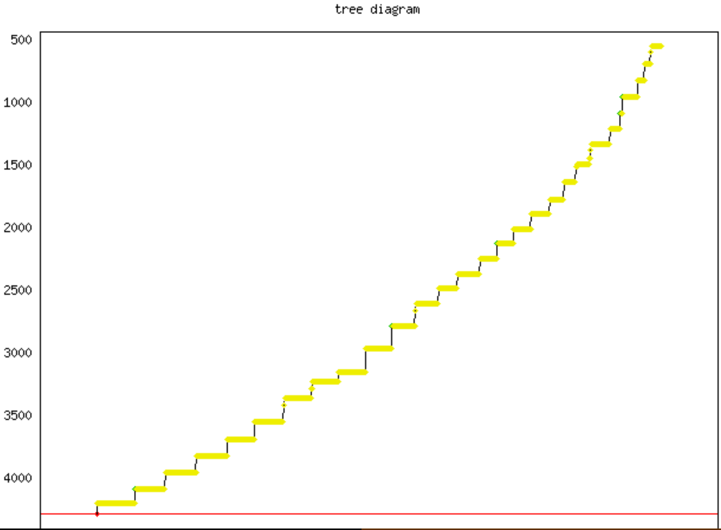
Example B&B tree series 1: l152lav



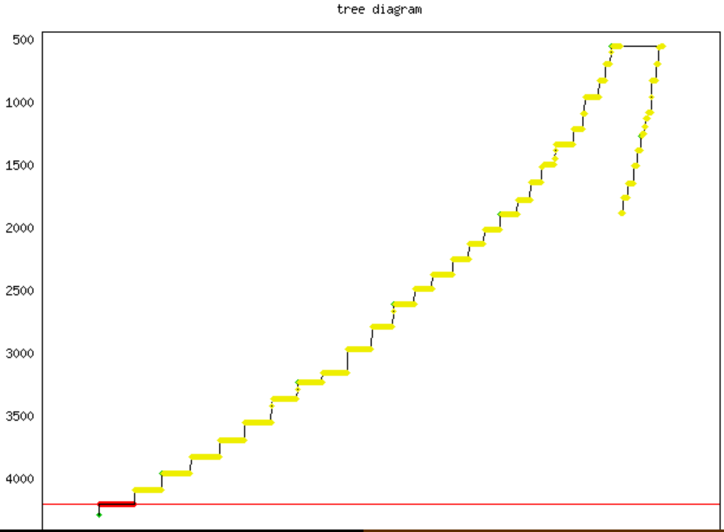
Example B&B tree series 2: liu



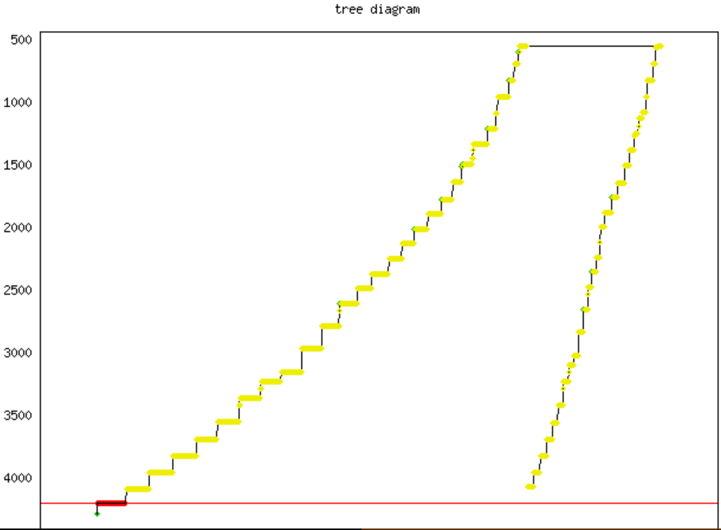
Example B&B tree series 2: liu



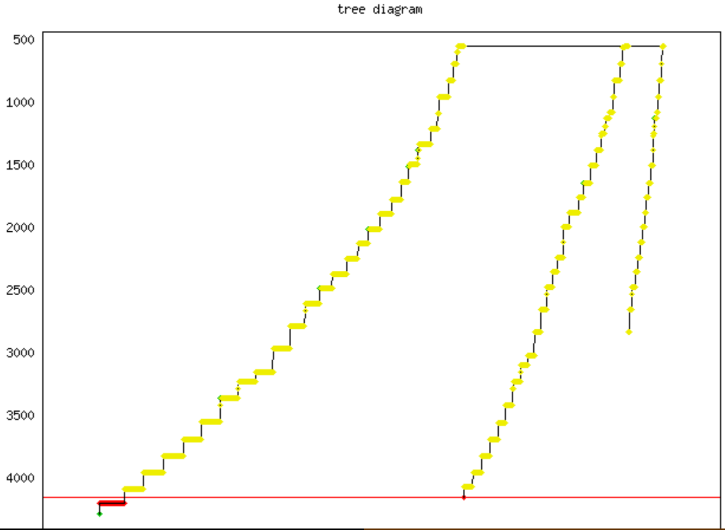
Example B&B tree series 2: liu



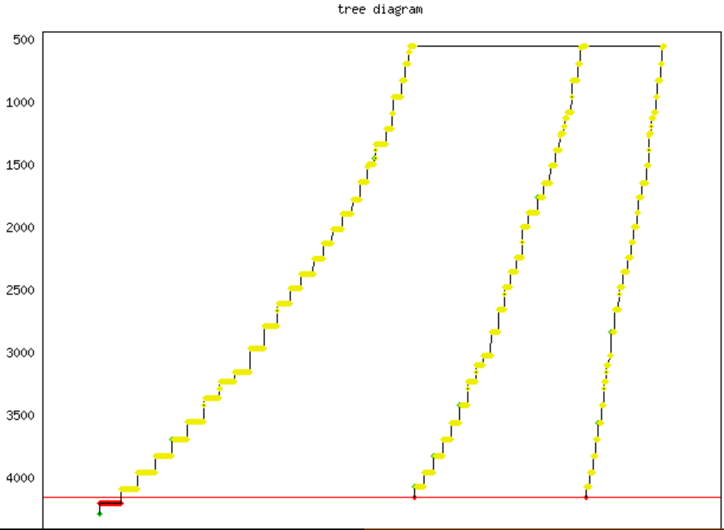
Example B&B tree series 2: liu



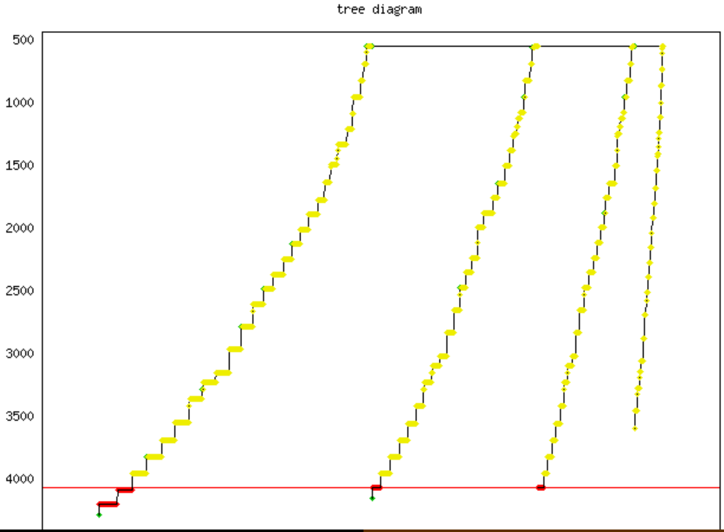
Example B&B tree series 2: liu



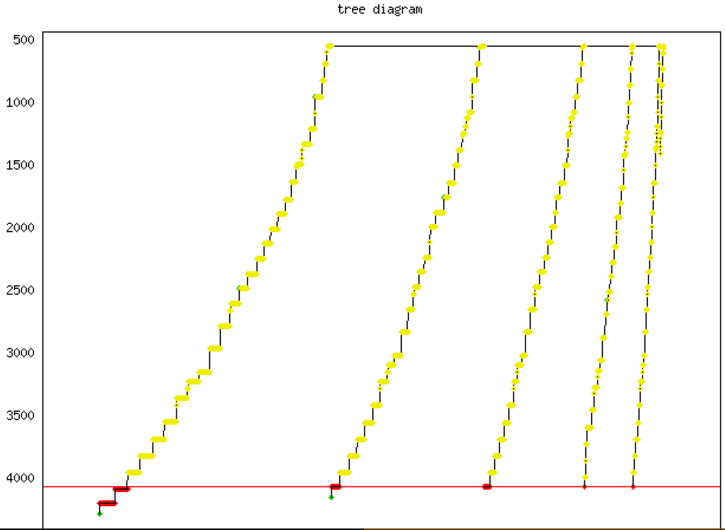
Example B&B tree series 2: liu



Example B&B tree series 2: liu



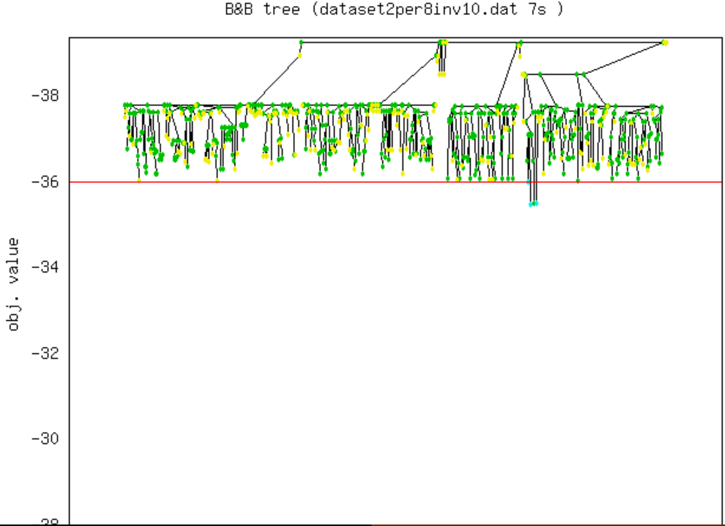
Example B&B tree series 2: liu



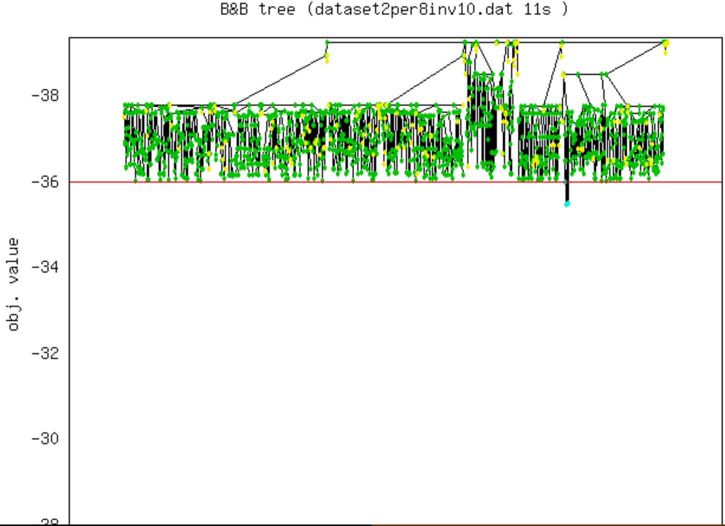
Example B&B tree series 3

Example B&B tree series 3

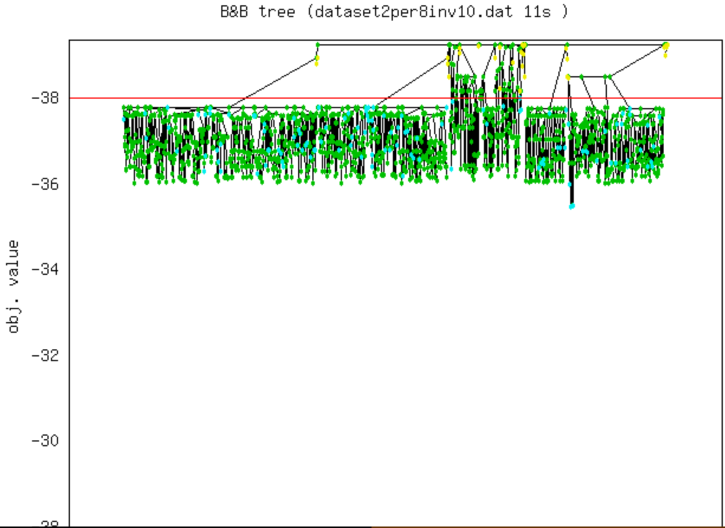
Example B&B tree series 3



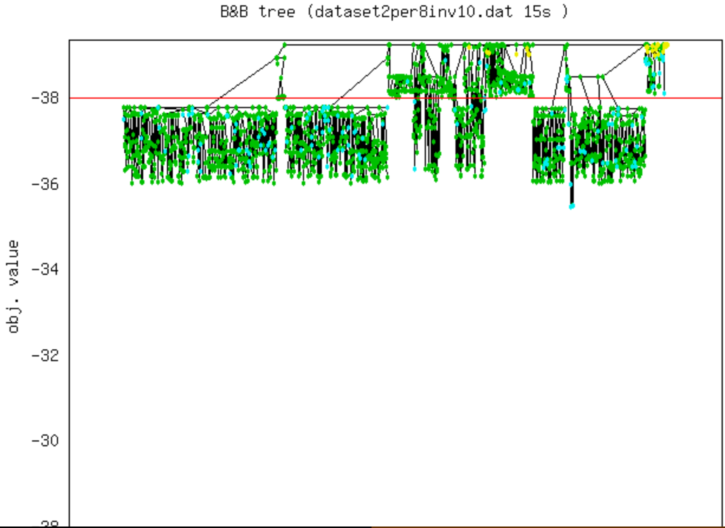
Example B&B tree series 3



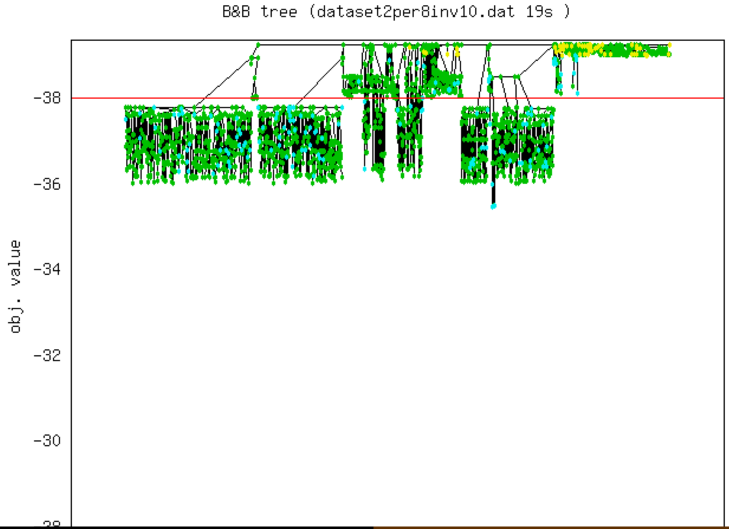
Example B&B tree series 3



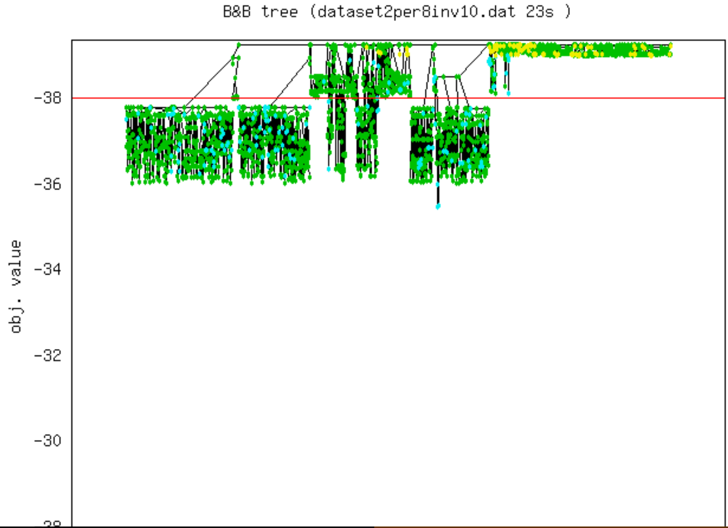
Example B&B tree series 3



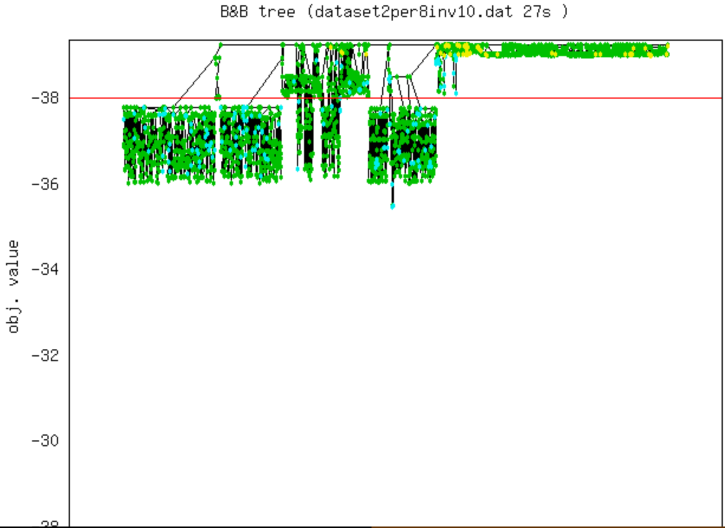
Example B&B tree series 3



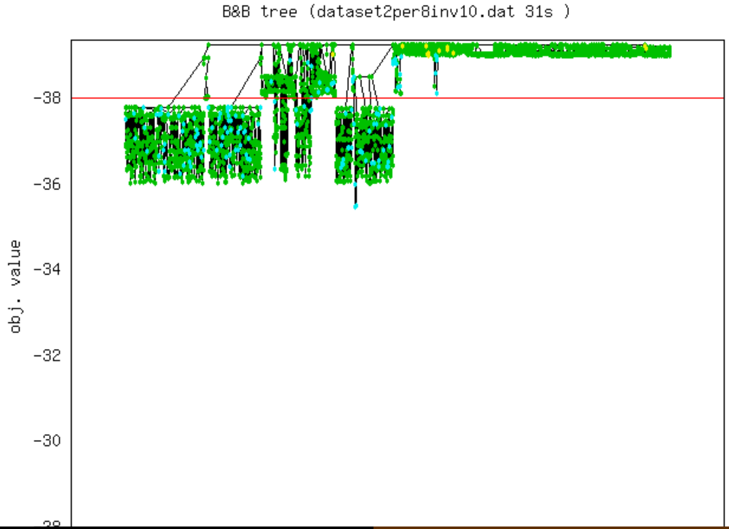
Example B&B tree series 3



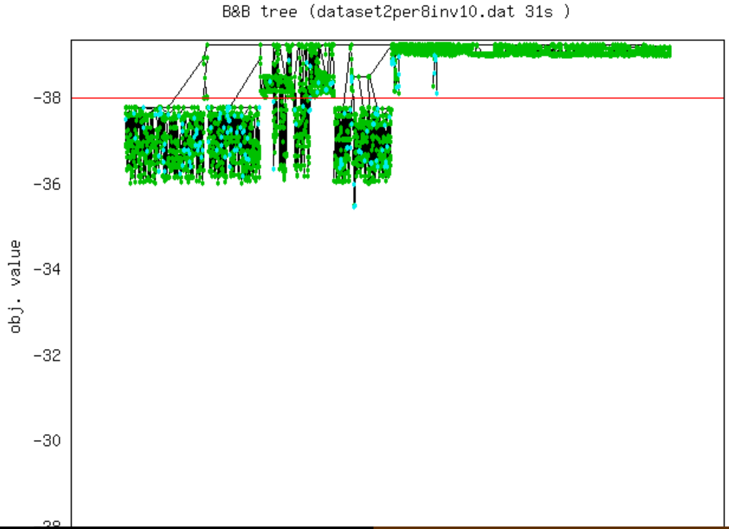
Example B&B tree series 3



Example B&B tree series 3



Example B&B tree series 3



Outline

1. Introduction
2. Benchmarking
   - Purpose
   - Sequential Codes
   - Parallel Codes
3. Performance Analysis
4. Conclusions

Other Tools

- Performance profiles
- Hudson (https://software.sandia.gov/hudson/)
- Hans Mittelman's Optimization Benchmarks (http://plato.asu.edu/bench.html)
- STOP (http://www.rosemaryroad.org/brady/software/index.html)

Final Remarks

- Benchmarking must be done with extreme care, especially with parallel codes.
- Open source can play a critical role in allowing researchers to carry out properly designed and controlled experiments.
- Please consider putting your codes into the COIN-OR repository or elsewhere for others to build on.