Operations Research and Algorithms at Google

Laurent Perron

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

Operations Research @ Google

- Operations Research team based in Paris
- Started ~7 years ago
- Currently, ~12 people
- Mission:
  - Internal consulting: build and help build optimization applications
  - Tools: develop core optimization algorithms
- A few other software engineers with OR background distributed in the company

OR-Tools Overview

- https://code.google.com/p/or-tools/
- Open sourced under the Apache License 2.0
- C++, java, Python, and .NET interface
- Known to compile on Linux, Windows, Mac OS X
- Constraint programming + Local Search
- Wrappers around GLPK, CLP, CBC, SCIP, Sulum, Gurobi, CPLEX
- OR algorithms
- ~200 examples in Python and C++, 120 in C#, 40 in Java
- Interface to Minizinc/Flatzinc

OR-Tools: Constraint Programming

- Google Constraint programming:
  - Integer variables and constraints
  - Basic Scheduling support
  - Strong Routing Support.
  - No floats, no sets
- Design choices
  - Geared towards Local Search
  - No strong propagations (JC's AllDifferent)
  - Very powerful callback mechanism on search.
  - Custom propagation queue (AC5 like)

OR-Tools: Local Search

- Local search: iterative improvement method
  - Implemented on top of the CP engine
  - Easy modeling
  - Easy feasibility checking for each move
- Large neighborhoods can be explored with constraint programming
- Local search
- Large neighborhood search
- Default randomized neighborhood
- Metaheuristics: simulated annealing, tabu search, guided local search

OR-Tools: Algorithms

- Min Cost Flow
- Max Flow
- Linear Sum Assignment
- Graph Symmetries
- Exact Hamiltonian Path
- And more to be implemented as needed

OR-Tools: Linear Solver Wrappers

- Unified API on top of CLP, CBC, GLPK, SCIP, Sulum, Gurobi, and CPLEX, GLOP.
- On top of our solvers: GLOP (LP), and BOP (Boolean MIPs)
- Implemented in C++ with Python, java, and C# wrapping.
- Expose basic functionalities:
    - variables, constraints, reduced costs, dual values, activities...
    - Few parameters: tolerances, choice of algorithms, gaps

OR-Tools: Simplex (GLOP)

- Simplex implementation in C++ (25k lines)
  - Coin LP is at least 300k lines of code
- Better than lpsolve, glpk, soplex
- Usually better than Coin LP, except on wide problems (misses sifting)
- Focus on numerical stability

OR-Tools: (Max)SAT Solver

- Competitive SAT/MaxSAT Solver
- In 2014, should have won industrial, and half of crafted SAT competition.
- MaxSAT based on core algorithm

OR-Tools: Binary Optimizer (BOP)

- Based on SAT
- + Simplex (Glop)
- + Local Search (inspired from LocalSolver)
- + Large Neighborhood Search

Competitive with CPLEX/Gurobi on binary models from the MIPLIB (actually better as it find solutions to more problems)

More on this later

My Job at Google

- Tech Lead of the OR team:
  - Find project, establish collaboration
  - Help setup plan, milestones, deliverables
  - Decide on the technology, implement.
- Implement applications
- Implement technology

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

Consulting is hard

Really hard!

- Getting the right problem with the right people is hard.
- Getting clean data is hard.
- Solving the problem is easy.
- Reporting the result/explaining the implications is hard

Time spent is 50 / 25 / 5 / 20 %

Convincing the User

You need to prove your anticipated gains to sign the contract

You need the trust of the client

You need to polish your results (the easy swap syndrome)

Stability is an issue

Running time/precise modeling are also an issue

The objective function is never straightforward

A network routing problem

Here is the customer description:

I have a network, each arc has a maximum capacity.

I have a set of demand, each demand has a source, a destination, a monetary value, and a traffic usage.

I want to select demands and how to route them in order to maximize the total value of routed demands.

On a given arc, the sum of traffic is <= capacity.

Analysis

This looks like a multi flow problem

or is it?

value is disconnected from traffic for a given demand, so we add a knapsack component to the problem.

Question we should ask the client

- Are partially fulfilled demands accepted?
  - if yes, is the gain linear w.r.t. the fulfilled traffic?
- Demands can be split? Capacity is for all traffic, on per direction?
  - Are the constraints soft or hard?
- Are there side constraints:
  - Max number of demands per arc, per node
  - Symmetric routing
  - Comfort zone on an arc, penalty on congestion
  - Priorities in demands
  - Special cost function, grouping, exclusion...

Choosing a strategy

At this point, you have no idea what a good solution looks like.

You have no idea what the input format looks like.

There is no point in starting a complex optimization model.

Choosing a strategy - 2

As a rule of the thumb, on an optimization problem, after you are sure of the problem:

- 50% of the time is spent getting clean data
- 10% is done working on the optimization problem
- 40% of the time is spent in the output part, getting feedback, qualifying the result

Choosing a strategy - 3

The best strategy going forward is to:

- Create an end to end solution.
- Spent the minimum amount of time needed to find a solution to the optimization problem.
- Showing the result and learning implicit constraints.

The minimal optimization problem is often a greedy algorithm.

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

Why focus on 0-1 LPs?

- Many engineers familiar with MIP/LP
- Many applications can easily be modeled as a 0-1 LPs
- One-line switch between classic MIP solver and our specialized 0-1 one
- “Easier” for us to do what we had in mind...

Why focus on generic LS and LNS?

- Efficient approach on large problem
- Using Constraint Programming is “hard”
- New applications often require special local moves or neighborhood to be created

Our intuition: automatically generated moves and neighborhood from linear binary representation alone can be good enough

Binary Solver Details

Efficient “extended” SAT solver

- Start with efficient state of the art CDLC (Conflict Driven Clause Learning) solver
- Add support for pseudo-Boolean constraint propagation and explain them. Ex:
  - b1 + b2 - 3*b3 + 5*b4 <= 5
  - trail: (b1 true, b2 true, b3 false) => b4 false
  - 1 reason for b3 assignment is clause “~b1 v b3 v ~b4”

“Max-SAT” complete solver

2 main ways to use SAT solver for optimization:

- linear scan (better and better solutions)
  - find a better solution by adding a constraint:
    objective < current best objective value
- core-based (better and better lower bounds)
  - Start by constraining all objective variable to their lower cost value.
    ex: all objective variable are false.
  - If UNSAT, identify a small core (subset of clauses) to explain this, relax just enough, and repeat until SAT.

Good first solution strategies

- SAT with many “random” heuristics:
  - variable branching order (in order, reverse, random)
  - branch choice (always true, always false, best objective, random, …)
  - also try different solver parameters.

- SAT guided by LP: Solve the LP relaxation, use optimal value to drive branching choices.

Improving feasible solution with LS

One idea is simply to explore one-flip “repairs”

Over-constrain objective so that initially it is the only infeasible constraint and:

1. Pick infeasible constraint (set is incrementally maintained).
2. Explore all the possible way to repair it by flipping 1 variable.
3. Enqueue each repair and propagate using underlying SAT solver.
4. Abort if SAT, otherwise if depth is not too big continue at 1.

Usually we limit the depth to 1,2,3 or 4 one-flip repairs.

The SAT solver can detect conflicts and learn new clauses in the process (related to probing in SAT/MIP presolve).

Improving feasible solution with LNS

- Fix some variables using current solution
- Use SAT with low deterministic time limit to try to find a better solution

Notes:

- Various heuristics to choose what to fix (random variables, random constraints, local neighborhood in var-constraint graph, …).
- We exploit SAT propagation to construct the neighborhood.
- Dynamically adapt the neighborhood size according to the result.

Another “LNS” approach

Use SAT solver with 2 extra constraints:

- Objective < current feasible solution value
- Hamming distance (potentially restricted to a subset of variables) from current solution is lower than a constant parameter.

Putting it all together

- Each "Optimizer" can be run with a small deterministic time limit.
- Main loop picks a random Optimizer to run for a short time according to its "score".
- Scores are dynamically updated depending on the amount of "learned" information on the problem.

0-1 MIPLIB 2010 results

The “benchmark”

MIPLIB, 59 0-1 linear problems, available at:

http://miplib.zib.de/miplib2010-BP.php

Caveats:

- Only 5 minutes time-limit
- MIPLIB are “hard” problem for MIP

Results (short version)

If feasibility is hard, LP relaxation is bad

BOP wins

If LP relaxation is very good, and/or the problem is nearly unimodular

Gurobi/CPLEX wins

The MIPLIB is biased towards the first case

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

My CP Experience

- Built the OR team in Google:
  - Introduced CP at Google.
  - Google does not care about technology.
  - But they care about testing/quality/security.

- On demand implementation:
  - So much to implement.
  - Constraint Catalog has more than 400 entries.
  - You have to concentrate on what is useful.

Must have

- Very few constraints/expressions.
- Add optionality as a first class concept.
- Debugging/explanations.
- Strong consulting experience

- Performance

Nice to have

- Diagnostic on my code and my model
  - Look at the generated model
  - Compute statistics
  - Profile the model

- Automatic behaviors
  - Automatic Search
  - Automatic LNS
  - Presolve, decomposition

How to gain performance

- Standard techniques
  - Fast algorithms
  - Fast data structures
- Branch and Bound techniques
  - Fast repetitive algorithms
  - Incremental algorithms
  - The fast code is the one that do not run
- Constraint Programming techniques
  - Better propagation
  - Model reinforcement

The Quest for the Perfect Sum

The standard algorithm

Sum(xi) = z

- Sum(min(xi)) <= min(z)
- Sum(max(xi)) >= max(z)

What can we deduce from the bounds of Z?

[0..1] + [0..1] + [0..1] = [0..1] Nothing

[0..1] + [0..1] + [0..10] = [4..7] -> [2..9] for 3rd term

Back-Propagation of the Sum

- [0..1] + [0..1] + [0..10] = [4..7]
  - Sum of mins: 0 + 0 + 0 = 0
  - Sum of max: 1 + 1 + 10 = 12
- [0..10] = [4..7] - [0..1] - [0..1]
  - min(0..10) >= min(4..7) - max(0..1) - max(0..1)
  - min(0..10) >= min(4..7) - (sum of max) + max(0..10)
  - min(0..10) >= 4 - 12 + 10 = 2
  - thus [0..10] -> [2..10]
  - and [2..10] -> [2..9]
- By default, all 3 terms will be checked.

Complexity of the Sum

Linear between the xi and z, in both directions

How to improve it:

- propagate delta:
  - xi[a + da, b - db] -> sum [zmin+da, zmax-db]
- Divide and conquer on the array
  - tree based split, in nothing to deduce -> complexity based on the size of the block
  - but, propagation of delta in log(n) instead of constant time
  - Use diameter optimization, if xi greater than the slack between the sum of xi and the bound of the z, it can absorb any reduction

More work on the sum

If sum is a scalar product sum(ai.xi) = b.z

- We can add a gcd constraint
  - gcd(ai) divides b
- if z is constant,
  - gcd(ai|xi non bound) divides
    - b*z - ai.xi (xi bound)
- If z is not constant, move to left part and move constants to the right hand side.

Even More Work on the Sum

If sum(ai * bi) = z, ai > 0, bi boolean variables

Then sort ai increasingly,

Start from the end, if bi is unbound:

- if ai > zmax - sum min(xi), then bi = 0,
  - continue
- if ai > sum max(xi) - zmin, then bi = 1
- else stop

This is the perfect propagation

Complexity is linear down a branch

The Next Level

Can we achieve arc consistency in the sum?

i.e. :

{1, 5, 6} + [0..2] = {1, 2, 3, 5, 6, 7, 8}?

There are three options:

- Count the number of supports for each value of each variable.
- Use a table constraint (explicit representation of the graph of the constraint).
- Use bitset manipulation.

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

Failing a project

There are many ways to fail a project for non technical reasons

- Wrong problem
- No Pain
- Wrong person
- Bad timing
- Moving target
- Bad cost estimate

How to waste your time

- Complex Search Language
  - 2 months of work → ½ % gain
- Random LNS
  - 20 lines of code, 2 hours of work → 2% gain
  - Complex structure with portfolio, learning, deeper randomization → 6% gain
  - Smart fragment selection → 10% gain
  - Parallelism (4 core 1% gain, 8 core 1.5% gain)

Imagination is limited

- Sports Scheduling, team/opponent matrix model.
- What search strategy do you use:
  - Focus on constrained team
  - Focus on constrained period
  - Alternate
  - Randomize...
- This is limited, impact kills any of these

Imagination is limited - 2

Car Sequencing

- The problem is nearly killed by a good heuristics
- Let's try Large Neighborhood Search:
- Fragment is a sequence
- Fragment is a set of vehicles with given types
- Propagation Guided LNS kills it, in less effort.

The lure of propagation

- Let's look at Sum of All Different
  - Seems generic
  - Find a good BC propagation algorithm
  - Implement it

- And then you need to test it:
  - Magic Square
  - And that is all (sum → cost, all different → assignment). They do not mix.

Outline

- Operations Research at Google
- Consulting is Hard
- Binary Optimizer
- Implementing Constraint Programming
- Traps and Pitfalls
- Conclusion

Conclusion on consulting

Validate your model and your objective function

Demonstrate end to end first.

Always be smart when spending your development time.

Appendix

Problem "easy" for SAT

	BOP	CPLEX	GUROBI
acc-tight4	Optimal, 0.23s	Optimal, 27s	Optimal, 27s
acc-tight5	Optimal, 1.06s	Optimal, 206s	Optimal, 240s
acc-tight6	Optimal, 0.80s	Optimal, 110s	Optimal, 90s
ex10	Optimal, 14s (core-sat 2s)	Optimal, 33s	Optimal, 38s
ex9	Optimal, 0.8s	Optimal, 4s	Optimal, 8s
hanoi5	Optimal, 17s (sat 0.83s)	--, 300s	--, 300s
macrophage	Optimal, 15s (core-sat 0.15s)	Optimal, 191s	Optimal, 26s

Problem “easy” for SAT - continued

	BOP	CPLEX	GUROBI
neos18	Optimal, 0.27s	Optimal, 24s	Optimal, 77s
neos808444	Optimal, 0.47s	Optimal, 36s	Optimal, 3s
neos-849702	Optimal, 16s	--, 300s	--, 300s
pb-simp-nonunif	Optimal, (=42) 26s (core-sat 11s)	109, 300s	140, 300s
vpphard	Optimal, (=5) 138s (core-sat 6s)	15, 300s	6, 300s
ns1688347	Optimal, 7s	Optimal, 28s	Optimal, 18s

Problem with good SAT first solution

	BOP	CPLEX	GUROBI
circ10-3	320 (lb 0) (fs 354 in 60s)	--, 300s (lb 140)	--, 300s (lb 140)
ns1696083	47 (lb 34) (fs 55 in 50s)	--, 300s (lb 24)	--, 300s (lb 32)
ns894236	17 (lb 14) (fs 18 in 20s)	--, 300s (lb 14)	--, 300s (lb 14)
ns894244	16 (lb 15)	--, 300s (lb 15)	16 (lb 15)
ns894786	14 (lb 8)	--, 300s (lb 18)	--, 300s (lb 18)
ns903616	20 (lb 17)	--, 300s (lb 17)	22 (lb 16)

Problem “easy” for MIP

	BOP	CPLEX	GUROBI
air04	57456 (lb 55536) Optimal is 56137	Optimal, 29s	Optimal, 12s
cov1075	20 (lb 18) Optimal is 20	Optimal, 11s	Optimal, 9s
harp2	-73467600 Optimal is -73899770	Optimal, 38s	Optimal, 49s
neos-1109824	383 (lb 278) Optimal is 378	Optimal, 27s	Optimal, 94s
neos-1337307	-- (lb -203123) Optimal is -202319	Optimal, 28s	Optimal, 55s
neos-777800	Optimal, 231s	Optimal, 1s	Optimal, 1s

Problem “easy” for MIP - continued

	BOP	CPLEX	GUROBI
tanglegram1	Optimal, 109s	Optimal, 34s	Optimal, 6s
tanglegram2	Optimal, 15s	Optimal, 0.8s	Optimal, 0.8 s

	BOP	CPLEX	GUROBI
mspp16	364 (lb 341)	407 (lb 341)	Optimal (=363) in 265s
n3seq24	52800 (lb 52000)	52200 (lb 52000)	Optimal (=52200) in 139s
vpphard2	311 (lb 15)	146 (lb 0)	Optimal (=81), 300s

Problem with good LS/LNS (300s)

	BOP	CPLEX	GUROBI
opm2-z10-s2	-33190 (lb -49308)	0 (lb -48954)	-19601 (lb -49308)
opm2-z11-s8	-42640 (lb -62971)	0 (lb -62971)	-21661 (lb -62953)
opm2-z12-s14	-63580 (lb -965157)	0 (lb -91524)	-28855 (lb -91524)
opm2-z12-s7	-65483 (lb -963536)	0 (lb -90514)	-28549 (lb -90514)
opm2-z7-s2	-10279 (lb -12879)	Optimal -10280, 174s	Optimal, 80s

Problem with good LS/LNS - cont

	BOP	CPLEX	GUROBI
queens-30	-39 (lb -70)	-37 (lb -69)	-36 (lb -70)
ramos3	254 (lb 164)	274 (lb 146)	277 (lb 146)
sts405	343 (lb 222)	405 (lb 137)	344 (lb 151)
sts729	642 (lb 325)	729 (lb 249)	647 (lb 259)
t1717	199029 (lb 134532)	208954 (lb 135183)	239381 (lb 135248)
t1722	126762 (lb 98816)	130615 (lb 99990)	151713 (lb 99863)

Problem hard for both (300s)

	BOP	CPLEX	GUROBI
ex10-10pi	250 (lb 221)	250 (lb 222)	245 (lb 222)
go19	88 (lb 77)	84 (lb 80)	84 (lb 81)
iis-100-0-cov	29 (lb 23)	29 (lb 24)	29 (lb 25)
iis-bupa-cov	36 (lb 27)	37 (lb 31)	36 (lb 31)
iis-pima-cov	34 (lb 27)	34 (lb 30)	34 (lb 31)
m100n500k4r1	-23 (lb -25)	-24 (lb -25)	-24 (lb -25)
methanosarcina	2897 (lb 2124)	2823 (lb 881)	2852 (lb 1019)
n3div36	139000 (lb 114400)	132800 (lb 121028)	130800 (lb 125000)

Problem hard for both (300s) - cont

	BOP	CPLEX	GUROBI
neos-1616732	161 (lb 146)	160 (lb 152)	159 (lb 151)
neos-1620770	9 (lb 8)	9 (lb 7)	9 (lb 8)
neos-631710	230 (lb 10)	490 (lb 188)	203 (lb189)
p6b	-60 (lb -90)	-62 (lb -68)	-62 (lb -68)
protfold	-29 (lb -41)	-22 (lb -37)	-21 (lb -38)
seymour	429 (lb 404)	426 (lb 415)	432 (lb 417)
toll-like	664 (lb 590)	613 (lb 500)	612 (lb 520)
wnq-n100-mw99-14	503 (lb 186)	268 (lb 231)	269 (lb 231)