Operations Research @ Google

- Operations Research team based in Paris
- Started 5 years ago
- Currently, 10 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
- ~100k lines of C++ code (cp is 70k).
- 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
  - Scheduling support
  - 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 constraint programming 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
- Everything implemented on top of tree search and tree search callbacks (search monitors).
OR-Tools: Algorithms

- Bron-Kerbosch to find cliques in an undirected graph.
- Dijkstra algorithm to find shortest paths in a directed graph with nonnegative costs.
- Bellman-Ford algorithm to find shortest paths in a directed graph with arbitrary costs.
- Min Cost Flow
- Max Flow
- Linear Sum Assignment
- And more to be implemented as needed
OR-Tools: Linear Solver Wrappers

- Unified API on top of CLP, CBC, GLPK, SCIP, Sulum, Gurobi.
- 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
Distribution of optimization technology used for Google optimization applications developed by the operations research team.
Consulting is 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 %
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.
  - You have to concentrate on what is useful.
What is needed?

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

- Strong consulting experience
Nice to have functionalities

- Need diagnostic on my code and my model
  - Look at the generated model
  - Compute statistics
  - Profile the model
- Need to implement automatic behaviors
  - Automatic Search
  - Automatic LNS
  - Presolve, decomposition
Modelers, Languages

- Pure modelers are dead (AMPL, OPL...)
- It is too difficult to:
  - extend with new data source/visualization
  - integrate them in a middleware/software
- Python, Scala, and C# provide 80% of the gain at 5% of the cost
- Complete java/python/C# interface for 3600 lines of code.
```python
solver = pywrapcp.Solver('hidato-table')
positions = [solver.IntVar(0, r * c - 1, 'p of %i' % i)
             for i in range(r * c)]

for i in range(r):
    for j in range(c):
        if puzzle[i][j] > 0:
            solver.Add(positions[puzzle[i][j] - 1] == i * c + j)
```
IntVar[] a = solver.MakeIntVarArray(n+1-m, 0,Q+1, "a");
// Check that the final state is in F
solver.Add(a[a.Length-1].Member(F));
// First state is q0
solver.Add(a[m] == q0);

int i = 0; i < n-2; i++ ) {
    IntVar[] tmp = (
        from j in Enumerable.Range(0, n)
        where j > i
        select differences[i,j].ToArray();
    solver.Add(tmp.AllDifferent());
}
Future work

The flatzinc challenge has demonstrated the need for presolve and the need for an intermediate layer.

The next step is really the automatization of the work:

- automatic LNS and automatic local search
- automatic decomposition
- Transformation from a sat problem to an optimization problem