CONNECTING CONSTRAINT SOLVERS TO AMPL

Victor Zverovich, Robert Fourer

AMPL Optimization

CPAIOR 2013, Yorktown Heights, NY, USA May 18-22, 2013

WHY AMPL?

- AMPL is a popular modeling system
  - used in businesses, government agencies, and academic institutions (over 100 courses in 2012)
  - large community
    (~ 1,400 members in AMPL Google Group alone)
  - the most popular input format on NEOS
    (> 200,000 or 57% submissions in 2012)
- AMPL is high-level, solver-independent and efficient.
- Supports a variety of solvers and problem types: linear, mixed integer, quadratic, second-order cone, nonlinear, complementarity problems and more.

HISTORY OF CP SUPPORT IN AMPL

- 1996: first experiments with adding logic programming features to AMPL.
- Fourer (1998). *Extending a General-Purpose Algebraic Modeling Language to Combinatorial Optimization: A Logic Programming Approach* [1]
- Fourer and Gay (2001). *Hooking a Constraint Programming Solver to an Algebraic Modeling Language*
- Fourer and Gay (2002). *Extending an Algebraic Modeling Language to Support Constraint Programming* [2]
- First **IBM/ILOG CP Optimizer** was connected.
- Gecode was connected in 2012, JaCoP - in early 2013.

SUPPORTED CP CONSTRUCTS

- Logical operators: `and`, `or`, `not`; iterated `exists`, `forall`
- Conditional operators: `if-then`, `if-then-else`, `==>`, `==> else`, `<==`, `<==>`
- Counting operators: iterated `count`, `atmost`, `atleast`, `exactly`, `numberof`
- Pairwise operators: `alldiff`

See **http://www.ampl.com/NEW/LOGIC/**

CONNECTED CP SOLVERS

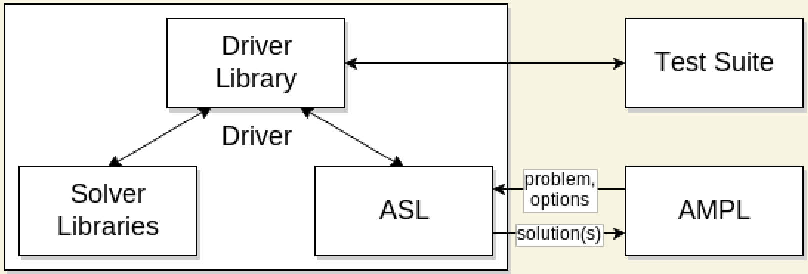
- Solvers:
  - ilogcp: **IBM/ILOG CP Optimizer**
  - gecode: **Generic constraint development environment**. **New:** Gecode 4.0, more solver options
  - **New:** jacop: **Java constraint solver**
- How to get:
  - Ilogcp is available to all CPLEX-for-AMPL users
  - AMPL Gecode and JaCoP (soon) downloads: **https://code.google.com/p/ampl/**
  - Source code: **https://github.com/vitaut/ampl** **solvers/gecode** | **solvers/ilogcp** | **solvers/jacop**

AMPL SOLVER LIBRARY

AMPL Solver Library (ASL) is an open-source library for connecting solvers to AMPL.

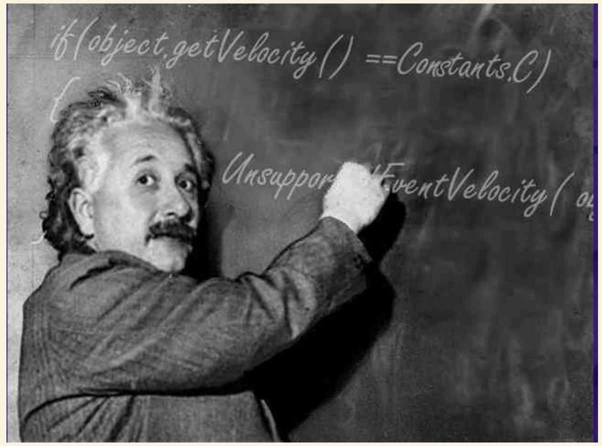
- C interface:
  - described in **Hooking Your Solver to AMPL**
  - used by most non-CP solvers
- **C++ interface**:
  - makes connecting new solvers super easy
  - type-safe: no casts needed when working with expression trees
  - efficient: no overhead compared to the C interface
  - used by the gecode, ilogcp and jacop (via JNI) drivers

AMPL SOLVER ARCHITECTURE



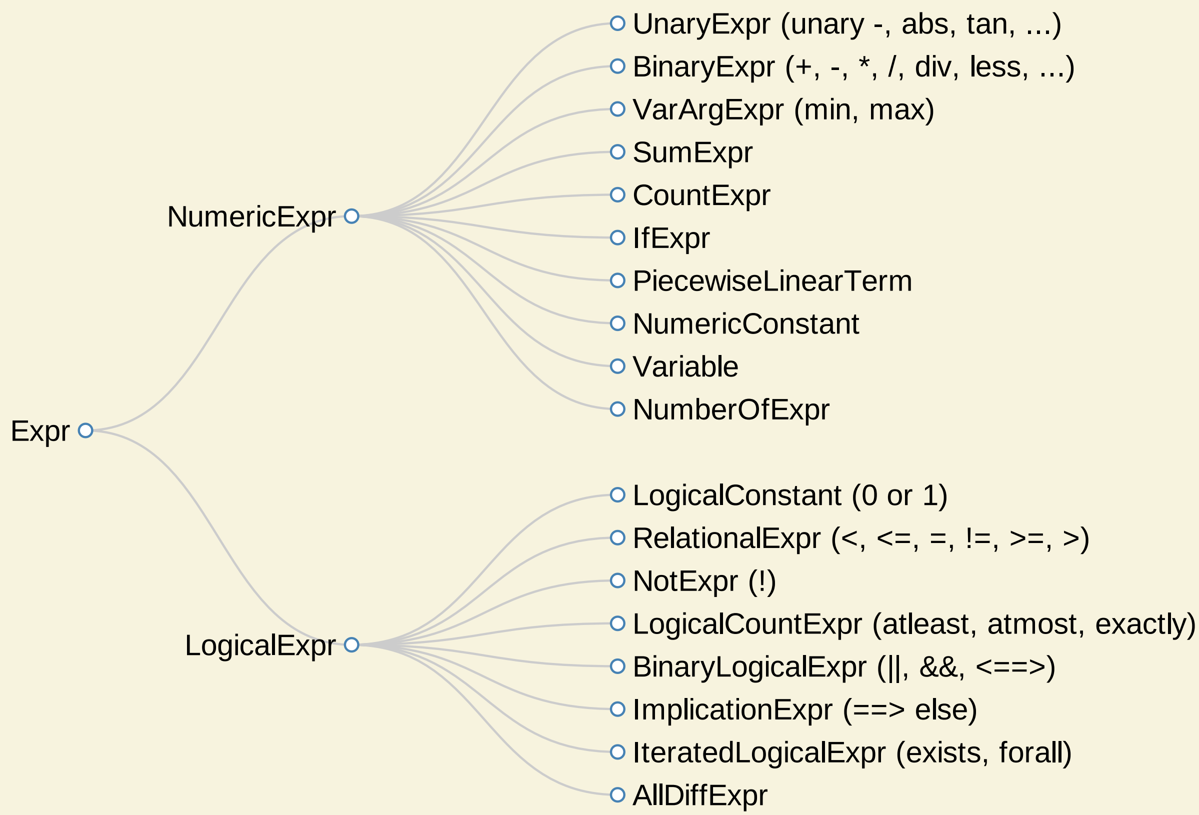
- Driver library is optional but facilitates testing.
- The test suite is used to verify correctness of driver implementation.
- The test suite is solver independent and can be reused when connecting a new solver.

CONNECTING JAVA SOLVERS



- JNI allows using ASL via the C++ API
- The JaCoP driver can be used as an example
- Initialization is tricky on Windows (delay loading of jvm.dll, check installation path in registry, ...)

EXPRESSION TREES



TREE TRAVERSAL WITH VISITORS

Expression tree traversal is implemented using a variant of the **visitor design pattern**.

```
class NLToConcertConverter :
  public ExprVisitor<NLToConcertConverter, IloExpr, IloConstraint>
{
  // Convert numeric expressions:
  IloExpr VisitNumericConstant(NumericConstant c);
  IloExpr VisitVariable(Variable v);
  IloExpr VisitPlus(BinaryExpr e);
  // ...

  // Convert logical expressions:
  IloConstraint VisitLogicalConstant(LogicalConstant c);
  IloConstraint VisitEqual(RelationalExpr e);
  IloConstraint VisitAnd(BinaryLogicalExpr e);
  // ...
};

NLToConcertConverter converter(...);
converter.Visit(expr); // Dispatches to specific Visit* method
                       // depending on dynamic type of expr
```

CONVERTING NUMERIC EXPRESSIONS

```
IloExpr VisitNumericConstant(NumericConstant n) {
  return IloExpr(env_, n.value());
}

IloExpr VisitVariable(Variable v) {
  return vars_[v.index()];
}

IloExpr VisitPlus(BinaryExpr e) {
  return Visit(e.lhs()) + Visit(e.rhs());
}

IloExpr VisitPow(BinaryExpr e) {
  return IloPower(Visit(e.lhs()), Visit(e.rhs()));
}

IloExpr VisitSum(SumExpr e) {
  IloExpr sum(env_);
  for (auto arg: e) // Iterating over arguments (C++11)
    sum += Visit(arg);
  return sum;
}
```

CONVERTING LOGICAL EXPRESSIONS

```
IloConstraint VisitLogicalConstant(LogicalConstant c) {
  return IloNumVar(env_, 1, 1) == c.value();
}

IloConstraint VisitEqual(RelationalExpr e) {
  return Visit(e.lhs()) == Visit(e.rhs());
}

IloConstraint VisitGreater(RelationalExpr e) {
  return Visit(e.lhs()) > Visit(e.rhs());
}

IloConstraint VisitAnd(BinaryLogicalExpr e) {
  return Visit(e.lhs()) && Visit(e.rhs());
}

IloConstraint VisitExists(IteratedLogicalExpr e) {
  IloOr disjunction(env_);
  for (auto arg: e) // Iterating over arguments (C++11)
    disjunction.add(Visit(arg));
  return disjunction;
}
```

PERFORMANCE CONSIDERATIONS

Map AMPL expressions into the most efficient solver equivalents.

Examples:

- `numberof` with constant values
  -> `IloDistribute` (ilogcp)
  controlled by the `usenumberof` option
- `if` *`logical-expr`* `then 1 (else 0)`
  -> channel constraint (gecode)

SUPPORTING MULTIPLE SOLVERS

- Separate hierarchies for logical and numeric expressions (ilogcp and gecode) are handled easily
- Possible to deal with more complex expression hierarchies, but with more efforts
- Not necessary to convert all expressions, solver will report an error when unhandled expression is encountered and exit gracefully.

EXAMPLES

Disjunctive constraint:

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
  sum {p in PROD} Trans[i,j,p] = 0 or
  minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

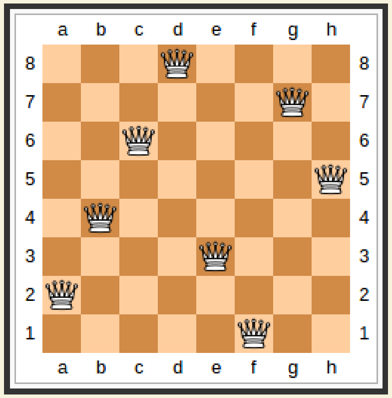
Implication:

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
  sum {p in PROD} Trans[i,j,p] > 0 ==>
    minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

SCHEDULING EXAMPLE WITH NUMBEROF

```
param n integer > 0;

set JOBS := 1..n;
set MACHINES := 1..n;

param cap {MACHINES} integer >= 0;
param cost {JOBS,MACHINES} > 0;

var MachineForJob {JOBS} integer >= 1, <= n;

minimize TotalCost:
   sum {j in JOBS, k in MACHINES}
      if MachineForJob[j] = k then cost[j,k];

subj to CapacityOfMachine {k in MACHINES}:
   numberof k in ({j in JOBS} MachineForJob[j]) <= cap[k];
```

N QUEENS EXAMPLE WITH ALLDIFF




```
# Place n queens on an n by n board
# so that no two queens can attack
# each other (nqueens.mod).

param n integer > 0;
var Row {1..n} integer >= 1 <= n;

subj to c1: alldiff ({j in 1..n} Row[j]);
subj to c2: alldiff ({j in 1..n} Row[j]+j);
subj to c3: alldiff ({j in 1..n} Row[j]-j);
```

More examples available at
http://www.ampl.com/NEW/LOGIC/examples.html.

USAGE EXAMPLE

```
ampl: model nqueens-mip.mod
ampl: let n := 100;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.4.0.0: optimal integer solution; objective 0
1416 MIP simplex iterations
0 branch-and-bound nodes
Objective = find a feasible point.
ampl: print _solve_time;
2.7680000000000002
ampl: model nqueens-cp.mod
ampl: option solver ilogcp;
ampl: solve;
ilogcp 12.4.0: feasible solution
1704 choice points, 781 fails
ampl: print _solve_time;
0.11599999999999966
ampl: option gecode_options 'val_branching=med';
ampl: solve;
gecode 4.0.0: val_branching=med
gecode 4.0.0: feasible solution
18382 nodes, 9145 fails
ampl: print _solve_time;
0.07600000000000273
```

SUMMARY

- AMPL now supports multiple constraint programming solvers.
- Connecting new solvers is super easy especially with the new C++ API.
- Java solvers are supported as well.

LINKS

- AMPL Logic and Constraint Programming Extensions: **http://www.ampl.com/NEW/LOGIC/**
- Trial version of AMPL with IBM/ILOG CP: **http://www.ampl.com/trial.html**
- Downloads for open-source AMPL solvers and libraries including Gecode: **https://code.google.com/p/ampl/**
- AMPL models by Hakan Kjellerstrand including 100 new CP models: **http://www.hakank.org/ampl/**
- Source code for ilogcp, gecode and jacop interfaces on GitHub: **https://github.com/vitaut/ampl**