Handling first-order linear constraints with SCIP

James Cussens, University of York

KU Leuven, 2015-02-16

Mixed integer programs

We have

- **Variables** These can be binary, integer or real-valued
- **Constraints** These are linear inequalities (maybe equations)
- **Objective** $z = c_i x_i$ where each variable x_i has an objective coefficient c_i.

- Goal is to assign values to the variables which meet all constraints and maximise the objective.
- If all variables are real-valued we have a *linear program (LP)* which can be solved in polynomial time.
- Otherwise the *linear relaxation* provides a useful upper bound.

Mixed integer programs

We have

Variables These can be binary, integer or real-valued

Constraints These are linear inequalities (maybe equations)

Objective $z = c_i x_i$ where each variable x_i has an objective coefficient c_i.

- Goal is to assign values to the variables which meet all constraints and maximise the objective.
- If all variables are real-valued we have a *linear program (LP)* which can be solved in polynomial time.
- Otherwise the *linear relaxation* provides a useful upper bound.
- Can extend to *constraint integer program* (SCIP approach).

Facility location in ZIMPL

```
set PLANTS := { "A", "B", "C", "D" };
set STORES := { 1 .. 9 };
set PS     := PLANTS * STORES;
var x[PS]     binary;  # Is plant p suppling store s ?
var z[PLANTS] binary;  # Is plant p build ?
minimize cost: sum <p> in PLANTS : building[p] * z[p]
             + sum <p,s> in PS : transport[p,s] * x[p,s];
subto assign: forall <s> in STORES :
               sum <p> in PLANTS : x[p,s] == 1;
subto build: forall <p,s> in PS : x[p,s] <= z[p];
subto limit: forall <p> in PLANTS :
   sum <s> in STORES : demand[s] * x[p,s] <= capacity[p];
```

First-order representations for convenience

- A good modelling language makes it easy to define large mixed integer programs.
- Here's how to define a Markov logic network in the modelling language ZIMPL.
- Show `univ.mln` and `univ.zpl`

Grounding the first-order representation

- The ZIMPL description can be translated to a standard MIP format (e.g. .lp),
- or sent directly to SCIP.
- In either case the compact representation is grounded before solving begins
- Show `univ.lp`
- Demo solving of `univ.zpl`

What's wrong with grounding?

- If there are not 'too many' variables and constraints in the ground MIP then grounding is a good option.
- But what to do if there are too many?

The obvious solution

- Compact MIPs are often reformulated (typically by a human) to have very many linear constraints and/or very many variables.
- This is to end up with a MIP whose LP relaxation is 'tight' (possibly even integer).
- The (typically correct) assumption is that most constraints are not tight and most variables have value zero.

The obvious solution

- Compact MIPs are often reformulated (typically by a human) to have very many linear constraints and/or very many variables.
- This is to end up with a MIP whose LP relaxation is ‘tight’ (possibly even integer).
- The (typically correct) assumption is that most constraints are not tight and most variables have value zero.
- Most linear constraints are added only when/if useful during the course of solving (*cutting planes*).
- Most variables are created only when/if useful during the course of solving (*column generation*).
- Not-yet-existing variables implicitly have their value clamped to zero.

The obvious solution

- Instead of grounding a first-order linear constraint prior to solving, useful ground instances can be generated as cutting planes.
- Instead of creating all variables prior to solving, useful variables can be created using column generation.

First steps

- I have an initial implementation of this approach called `foilp`.
- It accepts input in a language ('FOZ') virtually identical to ZIMPL.
- Show `smoke.zpl` and `smoke.foz`
- A Python script converts the FOZ file to a ZIMPL file with special fake variables and constraints (Show `smoke_fo.zpl`).
- `foilp` uses that information to create first-order clausal constraints.
- Show solving with `smoke.zpl` and `smoke_fo.zpl`.

Using SCIP

- `foilp` is just SCIP extended with two new *constraint handlers*: `foclause` and `folinear`.
- A SCIP constraint handler must do (at least) the following:

  CONSCHECK Decide whether an arbitrary candidate solution satisfies all the constraints (of the constraint handler)

  CONSENFOLP Decide whether an LP solution satisfies all the constraints (of the constraint handler)

  CONSENFOPS Decide whether a ‘pseudo-solution’ satisfies all the constraints (of the constraint handler)

  CONSLOCK Specify whether the constraints lock variables up or down (or both).

When to add cutting planes

- `foilp` implements the `CONSSEPALP` method to generate cutting planes
- Recall that at the beginning no ground instances of a first-order constraint have been posted.
- So we have an initial under-constrained problem which may still be tough to solve if there are other constraints.
- We do **not** wait until this problem is solved before adding ground constraints—this could take a while.
- We add ground instances (cutting planes) as soon as the *LP relaxation* of the under-constrained problem is solved.
- The LP relaxation removes the constraint that variables have to take integer values, and can be solved quickly (polynomial time).
- Once the cutting planes are added we solve the new LP and so on . . .

The separation problem for first-order clauses

Recall that literals are binary variables in the MIP which may have fractional values in the LP solution.
Suppose the LP solution includes the following values:
$p(a) = 0.2, q(a, b) = 0.7, r(b) = 0.$

And our problem has the following first-order clause:
$\forall x, y : p(x) \vee \neg q(x, y) \vee r(y)$
$\Leftrightarrow \forall x, y : p(x) + 1 - q(x, y) + r(y) \geq 1$

Then $p(a) + 1 - q(a, b) + r(b) \geq 1$
is a cutting plane worth adding—it *separates* the current LP solution, since
$0.2 + 1 - 0.7 + 0 < 1$

Efficient (?) separating

- In general deciding whether a first-order clause is satisfied by an interpretation is NP-complete (since we can encode any graph colouring problem like this).
- If the interpretation may be fractional then it will not get any easier!
- Current implementation of `foilp` does the sensible thing on clauses where the literals do not share variables.
- Otherwise just enumerates ground instances —it would not be hard to do better!

Proposal: Use a sub-IP to find a cutting plane

```
# Each first-order variable has exactly one grounding
I(x=a) + I(x=b) + ...+ I(x=c) == 1;
#Groundings of literals must be
# consistent with that of variables
I(x=a) = I(p(a,a)) + I(p(a,b)) + ... + I(p(a,c));
# Want grounding to violate first-order clause
minimize sum <x,y>: LPval(p[x])*I(p[x])
                    + 1-LPval(q[x,y])*I(q[x,y])
                    + LPval(r[y])*I(r[y])
```

Solution must be below 1 for there to be an associated cutting plane

What about variable creation?

There are two reasons to create a new variable:

1. If there is no feasible solution to the LP relaxation.
2. If there is a better solution where the new variable has a positive value.

Creating variables (column generation)

- Here we follow the presentation given be Gamrath [Gam10].
- Let the current LP be "min cx subject to $Ax \geq a$", where x is the vector of currently-existing variables. Suppose there is a feasible solution to this LP.
- The LP solver will provide us with π^*, the optimal solution to the *dual* of this LP. Each component of π^* corresponds to a row of A (a linear constraint)

Creating variables (column generation)

- Here we follow the presentation given be Gamrath [Gam10].
- Let the current LP be "min cx subject to $Ax \geq a$", where x is the vector of currently-existing variables. Suppose there is a feasible solution to this LP.
- The LP solver will provide us with π^*, the optimal solution to the *dual* of this LP. Each component of π^* corresponds to a row of A (a linear constraint)
- Let c_i be the objective coefficient for a potential new variable x_i.
- Let $A_{\cdot,i}$ be the vector of its coefficients in the current set of linear constraints.

The *pricing problem* is to find a new variable x_i with minimal *reduced cost*:

$$c^* = \min\{c_i - A_{\cdot,i}\pi^*\}$$

Recovering from infeasibility

If we have the clausal constraint:

$$p(x) + q(x, y) \geq 1$$

and, say, the MIP binary variables $p(a)$ and $p(a, b)$ do not exist yet then the constraint handler may generate the following cutting plane:

$$0 \geq 1$$

We need to create $p(a)$ or $p(a, b)$.

- If the LP is infeasible, the LP solver will provide Farkas multipliers u^T.
- Pricing problem is then $c^* = \min\{-A_{\cdot,i} u\}$.
- In practice typically better to create variables to ensure feasibility at the beginning.

Future work, etc

Future work, etc

- Implementation!

Future work, etc

- Implementation!
- Dealing with symmetries: "The most successful approach to date is to ensure that only one isomorphic copy of each node is kept in the enumeration tree" [CCZ14]

Future work, etc

- Implementation!
- Dealing with symmetries: "The most successful approach to date is to ensure that only one isomorphic copy of each node is kept in the enumeration tree" [CCZ14]
- Generate cuts using resolution as done by Chandru and Hooker [CH99].

Future work, etc

- Implementation!
- Dealing with symmetries: "The most successful approach to date is to ensure that only one isomorphic copy of each node is kept in the enumeration tree" [CCZ14]
- Generate cuts using resolution as done by Chandru and Hooker [CH99].
- Result: derived clauses can only be *facets* if they are prime implicants of the original set of clauses.

Future work, etc

- Implementation!
- Dealing with symmetries: "The most successful approach to date is to ensure that only one isomorphic copy of each node is kept in the enumeration tree" [CCZ14]
- Generate cuts using resolution as done by Chandru and Hooker [CH99].
- Result: derived clauses can only be *facets* if they are prime implicants of the original set of clauses.
- Generate new (useful!) first-order constraints from existing ones using first-order resolution.

Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli.
Integer Programming.
Springer, 2014.

Vijay Chandru and John Hooker.
Optimization methods for Logical Inference.
Wiley, 1999.

Gerald Gamrath.
Generic Branch-Cut-and-Price.
PhD thesis, TU Berlin, 2010.
Diploma Thesis.