DM841

Discrete Optimization

Part I

Lecture 3

Introduction to Gecode

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

[*Based on slides by Christian Schulte, KTH Royal Institute of Technology*]

Outline

1. Constraint Languages

2. Gecode

Resume

- Modelling in CP
  - Examples: graph labelling with consecutive numbers, cryptoarithmetic
- Overview on Constraint Programming
  - modelling
  - search = backtracking + branching
  - propagate (inference) + filtering

Constraint Programming:
representation (modeling language) + reasoning (search + propagation)

Outline

1. Constraint Languages

2. Gecode

Constraint Programming Systems

(modeling)
Expressive language stream

$+$

(efficient solvers)
Algorithm stream

Constraint Programming Systems

(modeling)
Expressive language stream
+
(efficient solvers)
Algorithm stream

CP systems typically include

- general purpose algorithms for constraint propagation (arc consistency on finite domains)
- built-in constraint propagation for various constraints (eg, linear, Boolean, global constraints)
- built-in for constructing various forms of search

Logic Programming

Logic programming is the use of mathematical logic for computer programming.

First-order logic is used as a purely declarative representation language, and a theorem-prover or model-generator is used as the problem-solver.

Logic programming supports the notion of logical variables

- Syntax – Language
  - Alphabet
  - Well-formed Expressions
    E.g., $4X + 3Y = 10$; $2X - Y = 0$
- Semantics – Meaning
  - Interpretation
  - Logical Consequence
- Calculi – Derivation
  - Inference Rule
  - Transition System

Logic Programming

Example: Prolog

> *A logic program is a set of axioms, or rules, defining relationships between objects.*
>
> *A computation of a logic program is a deduction of consequences of the program.*
>
> *A program defines a set of consequences, which is its meaning.*

Sterling and Shapiro: The Art of Prolog, Page 1.

Logic Programming

Example: Prolog

> *A logic program is a set of axioms, or rules, defining relationships between objects.*
>
> *A computation of a logic program is a deduction of consequences of the program.*
>
> *A program defines a set of consequences, which is its meaning.*

Sterling and Shapiro: The Art of Prolog, Page 1.

To deal with the other constraints one has to add other constraint solvers to the language. This led to Constraint Logic Programming

Prolog Approach

- Prolog II till Prolog IV [Colmerauer, 1990]
- CHIP V5 [Dincbas, 1988] http://www.cosytec.com (commercial)
- CLP [Van Hentenryck, 1989]
- Ciao Prolog (Free, GPL)
- GNU Prolog (Free, GPL)
- SICStus Prolog
- ECLiPSe [Wallace, Novello, Schimpf, 1997] http://eclipse-clp.org/ (Open Source)
- Mozart programming system based on Oz language (incorporates concurrent constraint programming) http://www.mozart-oz.org/ [Smolka, 1995]

Other Approaches

Libraries:
Constraints are modeled as objects and are manipulated by means of special methods provided by the given class.

- CHOCO (free) http://choco.sourceforge.net/
- Kaolog (commercial) http://www.koalog.com/php/index.php
- ILOG CP Optimizer `www.cpoptimizer.ilog.com` (ILOG, commercial)
- Gecode (free) `www.gecode.org`
  C++, Programming interfaces Java and MiniZinc
- G12 Project
  `http://www.nicta.com.au/research/projects/constraint_programming_platform`

Other Approaches

Modelling languages:

- OPL [Van Hentenryck, 1999] ILOG CP Optimizer
  www.cpoptimizer.ilog.com (ILOG, commercial)
- MiniZinc [] (open source, works for various systems, ECLiPSe, Geocode)
- Comet
- AMPL

- Catalogue of Constraint Programming Tools:
  `http://openjvm.jvmhost.net/CPSolvers/`
- Workshop "CPSOLVERS-2013"
  `http://cp2013.a4cp.org/node/99`

CP Languages

Greater expressive power than mathematical programming

- constraints involving disjunction can be represented directly
- constraints can be encapsulated (as predicates) and used in the definition of further constrains

CP Languages

Greater expressive power than mathematical programming

- constraints involving disjunction can be represented directly
- constraints can be encapsulated (as predicates) and used in the definition of further constrains

However, CP models can often be translated into MIP model by

- eliminating disjunctions in favor of auxiliary Boolean variables
- unfolding predicates into their definitions

CP Languages

- Fundamental difference to LP
  - language has structure (global constraints)
  - different solvers support different constraints
- In its infancy
- Key questions:
  - what level of abstraction?
    - solving approach independent: LP, CP, ...?
    - how to map to different systems?
  - Modeling is very difficult for CP
    - requires lots of knowledge and tinkering

Summary

- Model your problem via Constraint Satisfaction Problem
- Declare Constraints + Program Search
- Constraint Propagation
- Languages

Outline

1. Constraint Languages

2. Gecode

Gecode

an open constraint solving library

Christian Schulte
KTH Royal Institute of Technology, Sweden

Gecode People

- Core team
  - Christian Schulte, Guido Tack, Mikael Z. Lagerkvist.
- Code
  - contributions: Christopher Mears, David Rijsman, Denys Duchier, Filip Konvicka, Gabor Szokoli, Gabriel Hjort Blindell, Gregory Crosswhite, Håkan Kjellerstrand, Joseph Scott, Lubomir Moric, Patrick Pekczynski, Raphael Reischuk, Stefano Gualandi, Tias Guns, Vincent Barichard.
  - fixes: Alexander Samoilov, David Rijsman, Geoffrey Chu, Grégoire Dooms, Gustavo Gutierrez, Olof Sivertsson, Zandra Norman.
- Documentation
  - contributions: Christopher Mears.
  - fixes: Seyed Hosein Attarzadeh Niaki, Vincent Barichard, Pavel Bochman, Felix Brandt, Markus Böhm, Roberto Castañeda Lozano, Gregory Crosswhite, Pierre Flener, Gustavo Gutierrez, Gabriel Hjort Blindell, Sverker Janson, Andreas Karlsson, Håkan Kjellerstrand, Chris Mears, Benjamin Negrevergne, Flutra Osmani, Max Ostrowski, David Rijsman, Dan Scott, Kish Shen.

Gecode

Generic Constraint Development Environment

- **open**
  - easy interfacing to other systems
  - supports programming of: constraints, branching strategies, search engines, variable domains
- **comprehensive**
  - constraints over integers, Booleans, sets, and floats
    - different propagation strength, half and full reification, …
  - advanced branching heuristics (accumulated failure count, activity)
  - many search engines (parallel, interactive graphical, restarts)
  - automatic symmetry breaking (LDSB)
  - no-goods from restarts
  - MiniZinc support

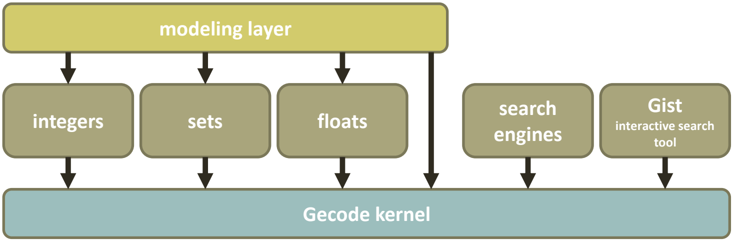
Gecode

Generic Constraint Development Environment

- **efficient**
  - *all* gold medals in *all* categories at *all* MiniZinc Challenges
- **documented**
  - tutorial (> 500 pages) and reference documentation
- **free**
  - MIT license, listed as free software by FSF
- **portable**
  - implemented in C++ that carefully follows the C++ standard
- **parallel**
  - exploits multiple cores of today's hardware for search
- **tested**
  - some 50000 test cases, coverage close to 100%

SOME BASIC FACTS

Architecture



- Small domain-independent kernel
- Modules
  - per variable type: variables, constraint, branchings, …
  - search, FlatZinc support, …
- Modeling layer
  - arithmetic, set, Boolean operators; regular expressions; matrices, …
- All APIs are user-level and documented (tutorial + reference)

Openness

- MIT license permits commercial, closed-source use
  - motivation: public funding, focus on research
  - not a reason: attitude, politics, dogmatism
- More than a license

license	restricts what users	**may do**
code and documentation	restrict what users	**can do**

- Modular, structured, documented, readable
  - complete tutorial and reference documentation
  - new ideas from Gecode available as scientific publications
- Equal rights: Gecode users are first-class citizens
  - you can do what we can do: APIs
  - you can know what we know: documentation
  - on every level of abstraction

Constraints in Gecode

- Constraint families
  - arithmetics, Boolean, ordering, ….
  - alldifferent, count (global cardinality, …), element, scheduling, table and regular, sorted, sequence, circuit, channel, bin-packing, lex, geometrical packing, nvalue, lex, value precedence, …
- Families
  - many different variants and different propagation strength
- All global constraints from MiniZinc have a native implementation
- Gecode ⇆ Global Constraint Catalogue: > 70 constraints

  abs_value, all_equal, alldifferent, alldifferent_cst, among, among_seq, among_var, and, arith, atleast, atmost, bin_packing, bin_packing_capa, circuit, clause_and, clause_or, count, counts, cumulative, cumulatives, decreasing, diffn, disjunctive, domain, domain_constraint, elem, element, element_matrix, eq, eq_set, equivalent, exactly, geq, global_cardinality, gt, imply, in, in_interval, in_intervals, in_relation, in_set, increasing, int_value_precede, int_value_precede_chain, inverse, inverse_offset, leq, lex, lex_greater, lex_greatereq, lex_less, lex_lesseq, link_set_to_booleans, lt, maximum, minimum, nand, neq, nor, not_all_equal, not_in, nvalue, nvalues, or, roots, scalar_product, set_value_precede, sort, sort_permutation, strictly_decreasing, strictly_increasing, sum_ctr, sum_set, xor

History

- 2002
    - development started
- 1.0.0
    - December 2005
- 2.0.0
    - November 2007
- 3.0.0
    - March 2009
- 4.0.0
    - March 2013
- 4.2.0 (current)
    - July 2013

Release	Size
1.0.0	43 kloc, 21 klod
2.0.0	77 kloc, 41 klod
3.0.0	81 kloc, 41 klod
4.0.0	164 kloc, 69 klod
4.2.0	168 kloc, 71 klod

34 releases

Tutorial Documentation

- 2002
  - development started
- 1.0.0 — 43 kloc, 21 klod
  - December 2005
- 2.0.0 — 77 kloc, 41 klod
  - November 2007
- 3.0.0 — 1 klod
  - March 2009
- 4.0.0 — 164 kloc, 69 klod
  - March 2013
- 4.2.0 (current)
  - July 2013

Modeling with Gecode (98 pages)

Modeling & Programming with Gecode (522 pages)

Future

- Large neighborhood search and other meta-heuristics
    - contribution expected
- Simple temporal networks for scheduling
    - contribution expected
- More expressive modeling layer on top of libmzn
- Grammar constraints
    - contribution expected
- Propagator groups
- …

- Contributions anyone?

Deployment & Distribution

- Open source ≠ Linux only
  - Gecode is native citizen of: Linux, Mac, Windows
- High-quality
  - extensive test infrastructure (around 16% of code base)
- Downloads from Gecode webpage
  - software: between 25 to 125 per day (total > 40000)
  - documentation: between 50 to 300 per day
- Included in
  - Debian, Ubuntu, Fedora, OpenSUSE, Gentoo, FreeBSD, …

Integration & Standardization

- Why C++ as implementation language?
  - good compromise between portability and efficiency
  - good for interfacing

  well demonstrated

- Integration with XYZ…
  - Gecode empowers users to do it
  - no “Jack of all trades, master of none”

  well demonstrated

- Standardization
  - any user can build an interface to whatever standard…
  - systems are the wrong level of abstraction for standardization
  - MiniZinc and AMPL are de-facto standards

Modeling & Programming

Constraint Programming with Gecode

Overview

- Program model as ***script***
  - declare variables
  - post constraints (creates propagators)
  - define branching

- Solve script
  - basic search strategy
  - Gist: interactive visual search

Program Model as Script

Script: Overview

- Script is class inheriting from class Space
  - members store variables regarded as solution
- Script constructor
  - initialize variables
  - post propagators for constraints
  - define branching
- Copy constructor and copy function
  - copy a Script object during search
- Exploration takes Script object as input
  - returns object representing solution
- Main function
  - invokes search engine

Script for SMM: Structure

```
#include <gecode/int.hh>
#include <gecode/search.hh>

using namespace Gecode;

class SendMoreMoney : public Space {
protected:
  IntVarArray l; // Digits for the letters
public:
  // Constructor for script
  SendMoreMoney(void) … { … }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(bool share) { … }
  // Print solution
  void print(void) { … }
};

…
```

Script for SMM: Structure

array of integer variables
stores solution

```
#include <gecode/int.hh>
#include <gecode/sear

using namespace Gec

class SendMoreMoney : public Space {
protected:
  IntVarArray l; // Digits for the letters
public:
  // Constructor for script
  SendMoreMoney(void) … { … }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(bool share) { … }
  // Print solution
  void print(void) { … }
};

…
```

Script for SMM: Structure

```
#include <gecode/int.hh>
#include <gecode/search.hh>

using namespace Gecode;

class SendMoreMoney : pu
protected:
  IntVarArray l; // Digits fo the letters
public:
  // Constructor for script
  SendMoreMoney(void) … { … }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(bool share) { … }
  // Print solution
  void print(void) { … }
};

…
```

constructor: initialize variables, post constraints, define branching

Script for SMM: Structure

```
#include <gecode/int.hh>
#include <gecode/search.hh>

using namespace Gecode;

class SendMoreMoney : public Space {
protected:
  IntVarArray l; // Digits for the letters
public:
  // Constructor for script
  SendMoreMoney(void) … { … }
  // Constructor for cloning
  SendMoreMoney(bool share, SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(bool share) { … }
  // Print solution
  void print(void) { … }
};

…
```

copy constructor and copy function

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  // Post constraints
  …
  // Post branchings
  …
}
```

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  // Post constraints
  …
  // Post branchings
  …
}
```

variables created in this script (space)

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  // Post constraints
  …
  // Post branchings
  …
}
```

8 variables

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  // Post constraints
  …
  // Post branchings
  …
}
```

take values
between 0 and 9
(digits)

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  // Post constraints
  …
  // Post branchings
  …
}
```

refer to variables by their letters

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  // No leading zeros (IRT: integer relation type)
  rel(*this, s, IRT_NQ, 0);
  rel(*this, m, IRT_NQ, 0);
  // All letters must take distinct digits
  distinct(*this, l);
  // The linear equation must hold
  …
  // Branch over the letters
  …
}
```

Posting Constraints

- Defined in namespace `Gecode`
- Check documentation for available constraints
- Take script reference as first argument
  - where is the propagator for the constraint to be posted!
  - script is a subclass of Space (computation space)

Linear Equations and Linear Constraints

- Equations of the form

  $$c_1 \cdot x_1 + \ldots + c_n \cdot x_n = d$$

  - integer constants: c_i and d
  - integer variables: x_i
- In Gecode specified by arrays
  - integers `(IntArgs)` c_i
  - variables `(IntVarArray, IntVarArgs)` x_i
- Not only equations
  - `IRT_EQ, IRT_NQ, IRT_LE, IRT_GR, IRT_LQ, IRT_GQ`
  - equality, disequality, inequality (less, greater, less or equal, greater or equal)

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  …
  // The linear equation must hold
  IntArgs c(4+4+5); IntVarArgs x(4+4+5);
  c[0]=1000; c[1]=100; c[2]=10; c[3]=1;
  x[0]=s;    x[1]=e;   x[2]=n;  x[3]=d;
  c[4]=1000; c[5]=100; c[6]=10; c[7]=1;
  x[4]=m;    x[5]=o;   x[6]=r;  x[7]=e;
  c[8]=-10000; c[9]=-1000; c[10]=-100; c[11]=-10; c[12]=-1;
  x[8]=m;      x[9]=o;     x[10]=n;    x[11]=e;   x[12]=y;
  linear(*this, c, x, IRT_EQ, 0);
  // Branch over the letters
  …
}
```

Linear Expressions

- Other options for posting linear constraints are available: minimodeling support
  - linear expressions
  - Boolean expressions
  - matrix classes
  - …
- See the examples that come with Gecode

Script for SMM: Constructor

```
…
#include <gecode/minimodel.hh>
…

  SendMoreMoney(void) : l(*this, 8, 0, 9) {
    …
    // The linear equation must hold
    post(*this,            1000*s + 100*e + 10*n + d
                         + 1000*m + 100*o + 10*r + e
                == 10000*m + 1000*o + 100*n + 10*e + y);
    // Branch over the letters
    …
  }
```

Script for SMM: Constructor

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  …
  // Branch over the letters
  branch(*this, l, INT_VAR_SIZE_MIN, INT_VAL_MIN);
}
```

Branching

- Which variable to choose
  - given order `INT_VAR_NONE`
  - smallest size `INT_VAR_SIZE_MIN`
  - smallest minimum `INT_VAR_MIN_MIN`
  - …
- How to branch: which value to choose
  - try smallest value `INT_VAL_MIN`
  - split (lower first) `INT_VAL_SPLIT_MIN`
  - …

Script for SMM: Copying

```
// Constructor for cloning
SendMoreMoney(bool share, SendMoreMoney& s) : Space(share, s) {
  l.update(*this, share, s.l);
}
// Perform copying during cloning
virtual Space* copy(bool share) {
  return new SendMoreMoney(share,*this);
}
```

Script for SMM: Copying

```
// Constructor for cloning
SendMoreMoney(bool share, SendMoreMoney& s) : Space(share, s) {
  l.update(*this, share, s.l);
}
// Perform copying during cloning
virtual Space* copy(bool share) {
  return new SendMoreMoney(share,*th
}
```

update all
variables needed
for solution

Script for SMM: Copying

```
// Constructor for cloning
SendMoreMoney(bool share, SendMoreMoney& s) : Space(share, s) {
  l.update(*this, share, s.l);
}
// Perform copying during cloning
virtual Space* copy(bool share) {
  return new SendMoreMoney(share,*this);
}
```

create a new copy of the space during cloning

Copying

- Required during exploration
  - before starting to guess: make copy
  - when guess is wrong: use copy
  - discussed later

- Copy constructor and copy function needed
  - copy constructor is specific to script
  - updates (copies) variables in particular

Copy Constructor And Copy Function

- Always same structure

- Important!
  - must update the variables of a script!
  - if you forget: crash, boom, bang, …

Script for SMM: Print Function

```
…
  // Print solution
  void print(void) {
    std::cout << l << std::endl;
  }
```

Summary: Script

- Variables
  - declare as members
  - initialize in constructor
  - update in copy constructor
- Posting constraints
- Create branching
- Provide copy constructor and copy function