ID2204: Constraint Programming

Getting Practical… Modeling Introduction

Lecture 02, 2018-03-23

Christian Schulte
cschulte@kth.se

Software and Computer Systems
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden

Constraint Programming with Gecode

Constraint Programming in Practice

- There are two options, use
  - modeling language, for example: MiniZinc
  - constraint system, for example: Gecode
- Modeling language advantages
  - high-level, easy to learn, easy to model
  - models can be tried with different systems
- Key disadvantage for us: you can only model!
  - you will also learn how to implement constraints, …

Modeling in Gecode

- Gecode offers two interface layers
  - primitive layer for interfacing

    Chapter 2 in MPG
  - modeling support for making modeling easier

    Chapter 3 in MPG
- Plan for today
  - look at the ugly side first to see the primitives
  - look at the easy side then
- When you model, go with the easy side!
  - always! really, really!

In MPG: Getting Started, Chapter 2

PRIMITIVE MODELING

Overview

- Program problem as ***script***
  - declare variables
  - post constraints (creates propagators)
  - define branching
- Solve script
  - basic search strategy: first, all, best solution(s)
  - Gist: interactive visual search

PROGRAM PROBLEM 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(SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(void) { … }
  // Print solution
  void print(void) { … }
};

…
```

Script for SMM: Structure

```
#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(SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(void) { … }
  // Print solution
  void print(void) { … }
};

…
```

array of integer variables stores solution

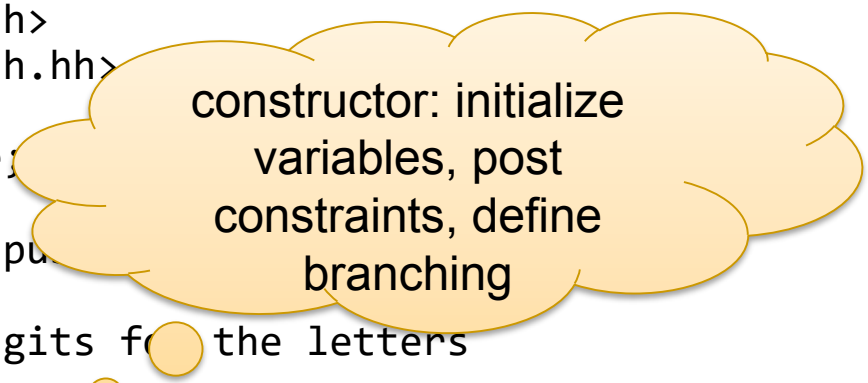
Script for SMM: Structure

```
#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(SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(void) { … }
  // Print solution
  void print(void) { … }
};

…
```



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(SendMoreMoney& s) … { … }
  // Perform copying during cloning
  virtual Space* copy(void) { … }
  // Print solution
  void print(void) { … }
};

…
```

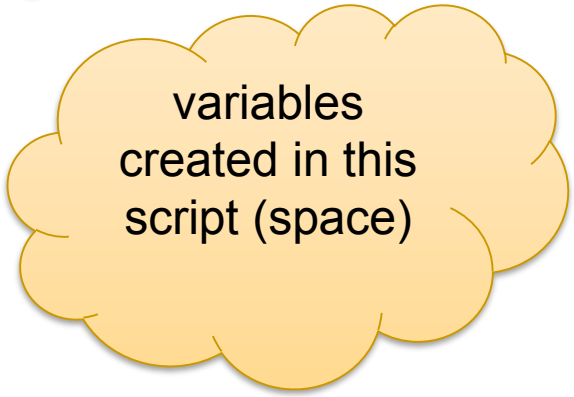
copy constructor and copy function

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
  …
}
```



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
  …
}
```

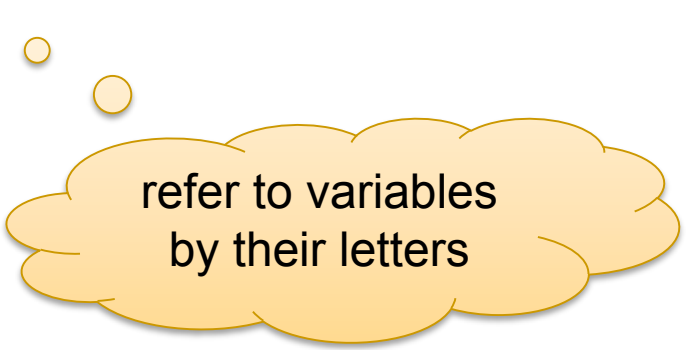
Script for SMM: Constructor

```
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
  …
}
```

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
  …
}
```



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
  …
}
```

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

Script for SMM: Copying

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

update all variables needed for solution

Script for SMM: Copying

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

Copying

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

In MPG: Getting Started

SOLVING SCRIPTS

Available Search Engines

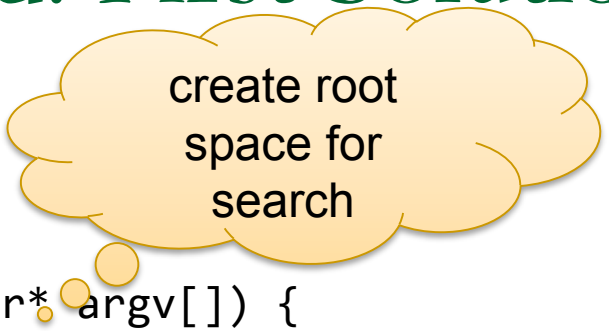
- Returning solutions one by one for script
  - `DFS` depth-first search
  - `BAB` branch-and-bound
- Interactive, visual search
  - `Gist`

Main Method: First Solution

…

```
int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  if (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

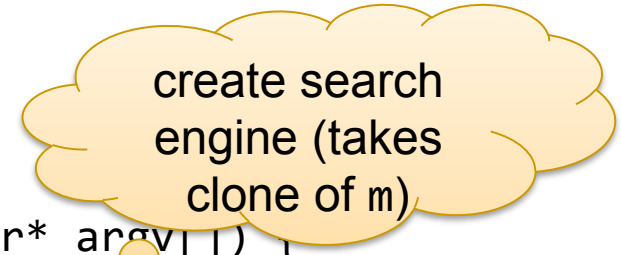
Main Method: First Solution



…

```
int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  if (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

Main Method: First Solution




```
…

int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  if (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

Main Method: First Solution

root space not any longer needed

…

```
int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  if (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

Main Method: First Solution

```
…

int main(int argc, char* argv[
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  if (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

Main Method: All Solutions

…

```
int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  DFS<SendMoreMoney> e(m);
  delete m;
  while (SendMoreMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

Gecode Gist

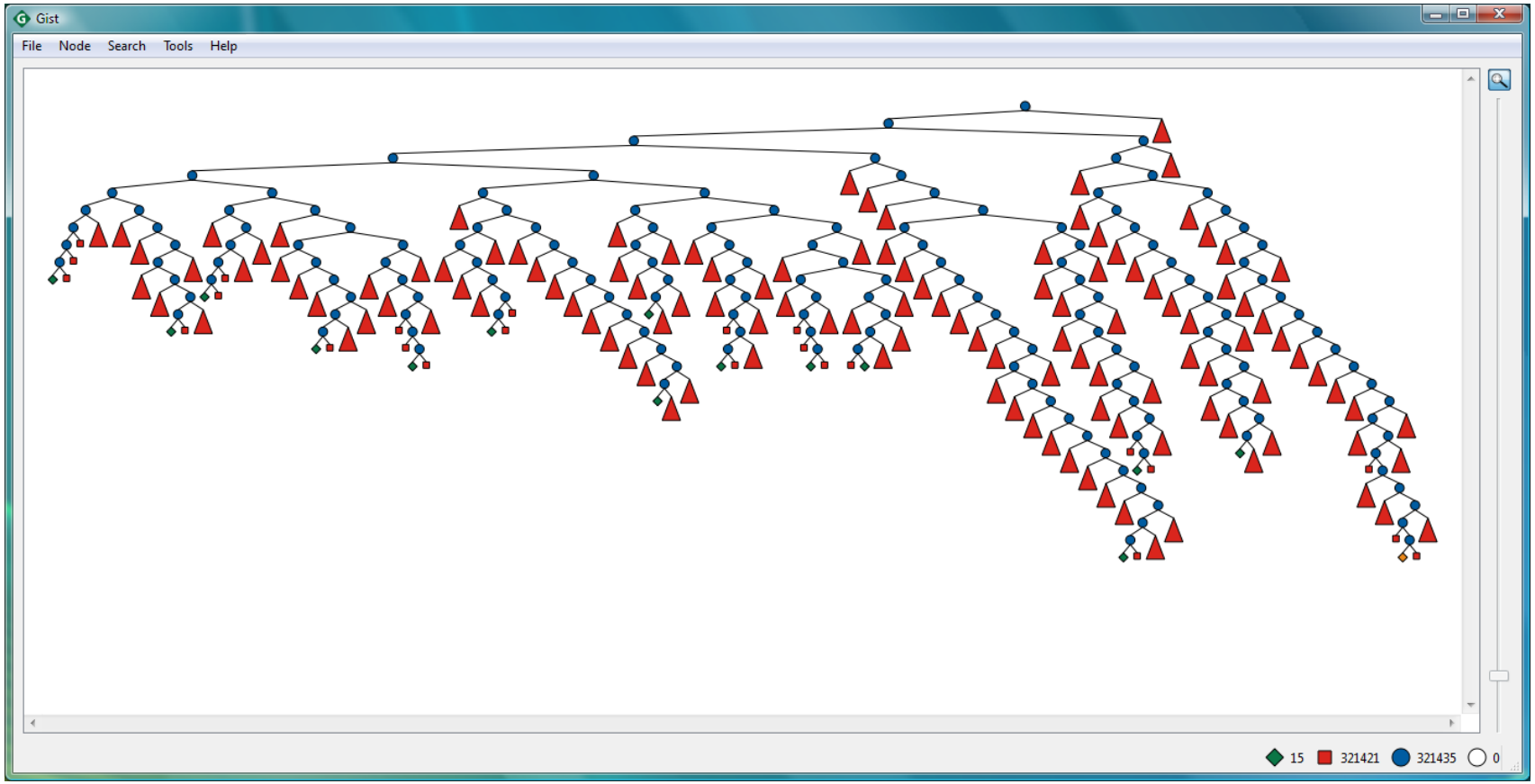
- A graphical tool for exploring the search tree
  - explore tree step by step
  - tree can be scaled
  - double-clicking node prints information: inspection
  - search for next solution, all solutions
  - …
- Best to play a little bit by yourself
  - hide and unhide failed subtrees
  - …

Main Function: Gist

```
#include <gecode/gist.hh>

int main(int argc, char* argv[]) {
  SendMoreMoney* m = new SendMoreMoney;
  Gist::dfs(m);
  delete m;
  return 0;
}
```

Gist Screenshot



Best Solution Search

Reminder: SMM++

- Find distinct digits for letters, such that

```
  SEND
+ MOST
-------
= MONEY
```

and **MONEY** maximal

Script for SMM++

- Similar, please try it yourself at home
- In the following, referred to by `SendMostMoney`

Solving SMM++: Order

- Principle
  - for each solution found, constrain remaining search for better solution
- Implemented as additional method

```
virtual void constrain(const Space& b) {
  …
}
```

- Argument b refers to so far best solution
  - only take values from b
  - never mix variables!
- Invoked on object to be constrained

Order for SMM++

```
…
#include <gecode/minimodel.hh>
…
  virtual void constrain(const Space& _b) {
    const SendMostMoney& b =
      static_cast<const SendMostMoney&>(_b);

    IntVar e(l[1]), n(l[2]), m(l[4]), o(l[5]), y(l[7]);

    IntVar b_e(b.l[1]), b_n(b.l[2]), b_m(b.l[4]),
           b_o(b.l[5]), b_y(b.l[7]);

    int money = (10000*b_m.val()+1000*b_o.val()+100*b_n.val()+
                 10*b_e.val()+b_y.val());

    rel(*this, 10000*m+1000*o+100*n+10*e+y > money);
  }
```

Main Method: All Solutions

…

```
int main(int argc, char* argv[]) {
  SendMostMoney* m = new SendMostMoney;
  BAB<SendMostMoney> e(m);
  delete m;
  while (SendMostMoney* s = e.next()) {
    s->print(); delete s;
  }
  return 0;
}
```

Main Function: Gist

```
#include <gecode/gist.hh>

int main(int argc, char* argv[]) {
  SendMostMoney* m = new SendMostMoney;
  Gist::bab(m);
  delete m;
  return 0;
}
```

Summary: Solving

- Result-only search engines
  - `DFS, BAB`
- Interactive search engine
  - `Gist`

- Best solution search uses constrain-method for posting constraint
- Search engine independent of script and constrain-method

In MPG: Getting Comfortable, Chapter 3

USING THE GECODE MODELING LAYER

Modeling Layer and Driver

- Modeling layer
  - provides convenient base-classes for scripts
  - supports arithmetic expressions
  - supports cost functions
  - Chapter 7 in MPG (browse as needed)
- Driver
  - parses command line options used by scripts
  - most aspects for search can be controlled from commandline
  - Chapter 11 in MPG (browse as needed)

Predefined Scripts

```
…
#include <gecode/driver.hh>
#include <gecode/minimodel.hh>
…
class SendMoreMoney : public Script {
  …
public:
  SendMoreMoney(const Options& opt)
    : Script(opt), … { … }
  virtual void print(std::ostream& os) const { … }
};
```

- Instead of using `Space`, use `Script`
- The object `opt` captures command line options

Driver

```
…
int main(int argc, char* argv[]) {
  Options opt("SEND+MORE=MONEY");
  opt.parse(argc,argv);
  Script::run<SendMoreMoney,DFS,Options>(opt);
  return 0;
}
```

- Provides
  - commandline options
  - execution statistics (time, solutions, …)
  - support for different search engines

Using the Commandline

- Print first solution

  ```
  ./smm.exe
  ```

- Print all solutions

  ```
  ./smm.exe -solutions 0
  ```

- Use Gist instead

  ```
  ./smm.exe -mode gist
  ```

- What else can you do

  ```
  ./smm.exe -help
  ```

  - many pre-defined options that can come in handy

Arithmetic Expressions

```
…
SendMoreMoney(const Options& opt) … {
  …
  rel(*this,              1000*s+100*e+10*n+d
              +           1000*m+100*o+10*r+e
             == 10000*m+1000*o+100*n+10*e+y);
  …
}
```

- Function `rel` overloaded for arithmetic expressions
- Similar function `expr` returning a variable

Use Cost Functions!

```
…
class SendMostMoney : public IntMaximizeScript {
  …
  IntVar money;
public:
  SendMostMoney(const Options& opt) … {
    …
    rel(*this, money == 10000*m+1000*o+100*n+10*e+y);
    …
  }
  virtual IntVar cost(void) const {
    return money;
  }
}
```

- Class `IntMinimizeScript` similar

Driver

```
…
int main(int argc, char* argv[]) {
  Options opt("SEND+MOST=MONEY");
  opt.parse(argc,argv);
  Script::run<SendMostMoney,BAB,Options>(opt);
  return 0;
}
```

- Provides
  - commandline options
  - execution statistics (time, solutions, …)
  - support for different search engines

Getting Started with MPG

- Check the beginning of Part M for reading advice!
- Chapter 2 (Getting started)
  - **read all**
- Chapter 3 (Getting comfortable)
  - **read all**
- Chapter 4 (Integer and Boolean variables and constraints)
  - 4.1, 4.2, 4.3: read (maybe a little later)
  - 4.4: browse which constraints are available
- Chapter 7 (Modeling convenience: MiniModel)
  - browse when needed
- Chapter 8 (Branching)
  - 8.1, 8.2: read what you need
- Chapter 9 (Search)
  - 9.3: read what you need
- Chapter 11 (Script Commandline Driver)
  - browse when needed

Grocery

Grocery

- Kid goes to store and buys four items
- Cashier: that makes $7.11
- Kid: pays, about to leave store
- Cashier: hold on, I multiplied!
  let me add!
  wow, sum is also $7.11
- You: prices of the four items?

Model

- Variables
  - for each item A, B, C, D
  - take values between {0, …, 711}
  - compute with cents: allows integers
- Constraints
  - A + B + C + D = 711
  - A * B * C * D = 711 * 100 * 100 * 100

Script

```
class Grocery : public Script {
protected:
    IntVarArray abcd;

    const int s = 711;
    const int p = s * 100 * 100 * 100;
public:
    Grocery(…) … { … }

    …
}
```

Script: Variables

```
Grocery(…) : …, abcd(*this,4,0,711) {
   …
}
```

Script: Sum

```
…
// Sum of all variables is s
linear(*this, abcd, IRT_EQ, s);

IntVar a(abcd[0]), b(abcd[1]),
       c(abcd[2]), d(abcd[3]);
```

Script: Product

```
IntVar t1(*this,1,p);
IntVar t2(*this,1,p);
IntVar t3(*this,p,p);

mult(*this, a, b, t1);
mult(*this, c, d, t2);
mult(*this, t1, t2, t3);
```

Branching

- Bad idea: try values one by one
- Good idea: split variables
  - for variable x
  - with $m = (\min(x) + \max(x)) / 2$
  - branch $x < m$ or $x \geq m$
- Typically good for problems involving arithmetic constraints
  - exact reason needs to be explained later

Script: Branching

```
branch(*this, abcd,
       INT_VAR_NONE(),
       INT_VAL_SPLIT_MIN());
```

Search Tree

- 2829 nodes for first solution
- Pretty bad…

Better Heuristic?

- Try branches in different order

  split with larger interval first
  - try: `INT_VAL_SPLIT_MAX()`
- Search tree: 2999 nodes
  - worse in this case

Symmetries

- Interested in values for A, B, C, D
- Model admits equivalent solutions
  - interchange values for A, B, C, D
- We can add order A, B, C, D:

  $A \leq B \leq C \leq D$
- Called “symmetry breaking constraint”

Script: Symmetry Breaking

```
…
rel(*this, a, IRT_LQ, b);
rel(*this, b, IRT_LQ, c);
rel(*this, c, IRT_LQ, d);
…
```

Effect of Symmetry Breaking

- Search tree size 308 nodes
- Let us try `INT_VAL_SPLIT_MAX()` again
  - tree size 79 nodes!
  - interaction between branching and symmetry breaking
  - other possibility: A ≥ B ≥ C ≥ D
  - we need to investigate more (later)!

Any More Symmetries?

- Observe: 711 has prime factor 79
  - that is: 711 = 79 × 9
- Assume: A can be divided by 79
  - add: $A = 79 \times X$ for some finite domain var X
  - remove $A \leq B$
  - the remaining B, C, D of course can still be ordered

Any More Symmetries?

- In Gecode

```
IntVar x(*this,1,p);
IntVar sn(*this,79,79);
mult(*this, x, sn, a);
```

- Search tree 44 nodes!
  - now we are talking!

Summary: Grocery

- Branching: consider also
  - how to partition domain
  - in which order to try alternatives
- Symmetry breaking
  - can reduce search space
  - might interact with branching
  - typical: order variables in solutions
- Try to really understand problem!

Another Observation

- Multiplication decomposed as

  $A \cdot B = T_1 \quad C \cdot D = T_2 \quad T_1 \cdot T_2 = P$

- What if

  $A \cdot B = T_1 \quad T_1 \cdot C = T_2 \quad T_2 \cdot D = P$

  - propagation changes: 355 nodes
  - propagation is not compositional!
  - another point to investigate