# Gecode

**Christian Schulte**[1]
Guido Tack[2]
Mikael Z. Lagerkvist[1]

[1] KTH – Royal Institute of Technology, Sweden
[2] Saarland University, Germany

# Gecode

- Generic constraint development environment

  www.gecode.org

- Open source C++ library

  - open — programming interfaces
  - free — MIT license
  - portable — whatever hardware/software environment
  - accessible — extensively documented
  - efficient — competetive performance (space/time)

    see webpage for comparisons

- Gecode 2.0.0 to be released end of October

# Basic Facts

# Open Platform

- Research
  - extensibility, openness
- Education
  - modern free platform for teaching CP
- Deployment
  - do whatever you want
- Efficiency
  - be useful

# Gecode Architecture

- Generic kernel
  - kernel core
  - domain-independent abstractions (branching, propagators, …)
- Modules
  - typically, one per variable domain (as many as you want)
    - finite domain integers, finite sets, complete finite sets
  - search engines
  - modeling support
  - serialization
  - …

# Search

- Search based on recomputation
  - expressive for programming search [Schulte, 2002]
  - adaptive and batch recomputation for efficiency
- Standard engines
  - depth-first search
  - limited discrepancy search
  - branch-and-bound optimization
  - DFS restart optimization

# Finite Domain Integers

- Use generic kernel interfaces (no special pet)
- Standard constraints
  - arithmetic, Boolean, and linear constraints
  - reified versions of the above
- Global constraints
  - all-different, global cardinality, count, element, regular, lexicographic ordering, inverse, sortedness, cumulatives, circuit, channel, extensional (table)
  - typically supporting various consistency levels

# Finite Sets
# Complete Finite Sets

- Two variable kinds
  - bounds and cardinality approximation [Puget, SPICIS 1992] [Gervet, Constraints 1997]
  - complete domain representation [Hawkins et al., JAIR 2005]
- Standard constraints
  - set relations and operations
- Global constraints
  - convexity, distinctness, atmost, selection, channel
- Compiler for generating propagators from formulas [CP 2006]

# Modeling

- Orthogonal to rest of system
- Natural representation of expressions

```
post(this, x + 3*y >= z);
post(this, tt(x & y | z));
regular(this, x, *(REG(0)+REG(1)));
```

- Matrices of variables

```
for (int i = 0; i < 9; ++i)
  distinct(this, sudoku.row(i));
```

- Expressions, …

# Quality: Systematic Testing

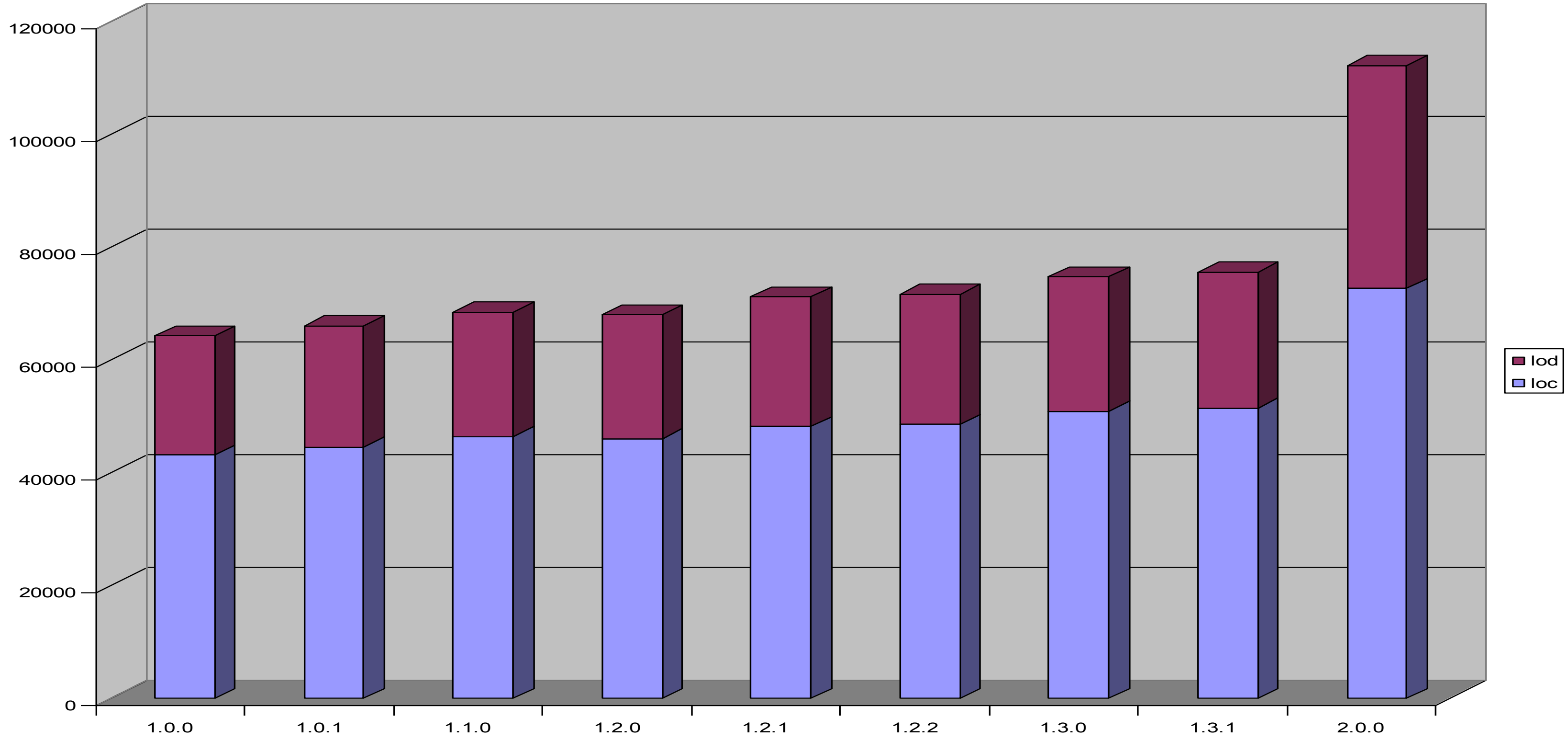
- Extensive test-suite for all constraints in the system
  - randomized tests
  - good coverage (> 97%)
- Indispensable: users, reproducible research

- *We* found *many* bugs, users did not
  - one major bug since December 2005
  - new release with fix within two days

# Development

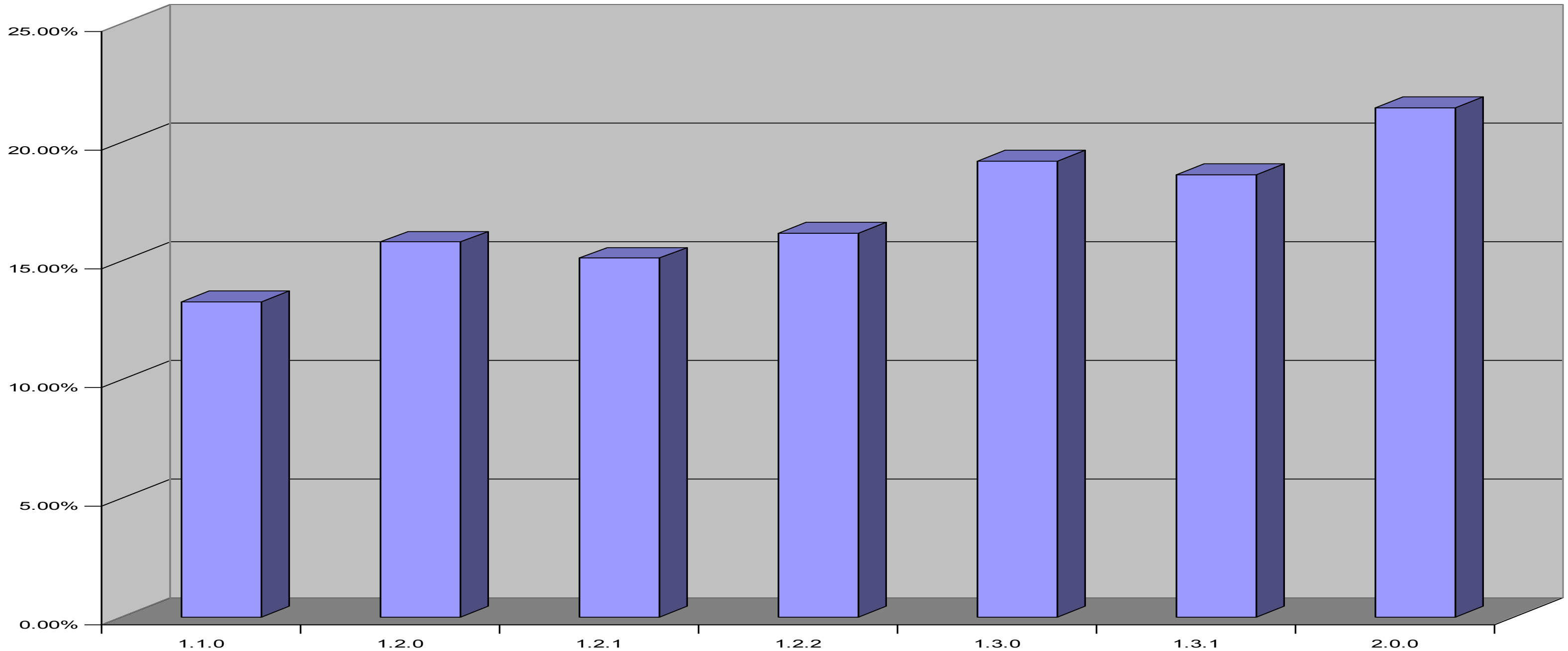
- First release 1.0.0, 12/6/2005
  - small improvements, user requests, fixes quickly
  - 1.0.1, 3/1/2006
  - 1.1.0, 4/10/2006
  - 1.2.0, 6/20/2006 1.2.1, 7/19/2006 1.2.2, 7/25/2006
  - 1.3.0, 9/19/2006 1.3.1, 10/25/2006
- Next major release: 2.0.0, 10/31/2007
  - support for incremental propagation: advisors [CP 2007]
  - interface to MiniZinc [CP 2007]
  - dramatically better 0/1 variables, many new constraints, complete set variables, reflection, much improved documentation, scalability, …
- Some 1000 downloads a month (mirrors excluded)

# Development: Code Size




- Gecode 1.0.0: 43 kloc, 21 klod
- Gecode 2.0.0: 73 kloc, 40 klod

# Development: Speedup




- Gecode 2.0.0 > 20% speedup compared to Gecode 1.0.1
  - based on some 20 benchmarks
  - many: remain the same; some: twice as fast
  - new features without any slowdown

# Using Gecode

# How to Use Gecode…

- Interfacing
  - Java, MiniZinc, Ruby, Alice (SML), Python, …
- Direct modeling and solving
  - C++
- Adding
  - propagators, branchings, variables, search engines
  - not extending: based on well-documented programming interfaces
- Research vehicle
  - new variable domains and propagators, benchmarking, …

# Interfaces

- Gecode/J
  - comprehensive Java interface used for education (we)
- MiniZinc (through FlatZinc) [CP 2007]
- Gecode/R
  - modeling Ruby interface (Andreas Launilla, supported by Google summer of code grant)
- AliceML (dialect of Standard ML)
  - Gecode-bindings as a standard library (Smolka et al)
- GeOz
  - project to integrate Gecode into Mozart/Oz environment (AVISPA Group)
- Python, ...

# Gecode/J

- Complete interface in Java
  - modeling, propagators, branchings, etc
  - provides barrier-free and complete approach
  - released in lock step with Gecode

- Used in education
  - KTH, Sweden
  - Uppsala U, Sweden
  - UCL, Belgium
  - American U, Egypt
  - Saarland U, Germany
  - U Freiburg, Germany

# Some Use Cases

- For users with background in CP
- Integrate CP technology
    - companies (small): cheap access
- Extend CP
    - QeCode: quantified constraints [Benedetti ea, IJCAI 2007, CSCP 2006]
    - new variable domains CP(Graph), CP(Map): [Dooms ea, CP 2005] [Zampelli ea, CP 2005]
- Realistic experimentation platform
    - randomization in tail assignment [Otten ea, CP 2006]
    - abstractions for non-deterministic search [Michel ea, CP 2006]
    - ....

# Modeling in Gecode

- Model structure
  - subclass from class Space (node in search tree)
  - constructor: create variables, post constraints & branchings
  - two additional methods for copying (trivial)

- Solving model
  - create instance of model
  - pass to search engine, or apply search strategy

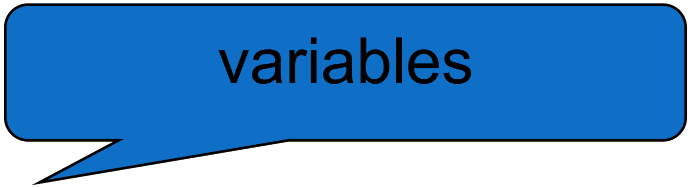
# Toplevel Struct

setting the stage

```
#include "gecode/int.hh"
#include "gecode/search.hh"
#include "gecode/minimodel.hh"
using namespace Gecode;

class Queens : public Space {
protected:
  IntVarArray q; // Position of queen
public:
  Queens(int n) : q(this,n,0,n-1) {
    ... post constraints & branchings
  }
  ... two additional methods for copying
  ... more methods (printing, etc)
};

int main(int argc, char* argv[]) {
  ... run model
}
```

# Toplevel Structure

```
#include "gecode/int.hh"
#include "gecode/search.hh"
#include "gecode/minimodel.hh"
using namespace Gecode;

class Queens : public Space {
protected:
  IntVarArray q; // Position of queen
public:
  Queens(int n) : q(this,n,0,n-1) {
    ... post constraints & branchings
  }
  ... two additional methods for copying
  ... more methods (printing, etc)
};

int main(int argc, char* argv[]) {
  ... run model
}
```



# Toplevel Structure

```
#include "gecode/int.hh"
#include "gecode/search.hh"
#include "gecode/minimodel.hh"
using namespace Gecode;

class Queens : public Space {
protected:
  IntVarArray q; // Position of
public:
  Queens(int n) : q(this,n,0,n-1) {
    ... post constraints & branchings
  }
  ... two additional methods for copying
  ... more methods (printing, etc)
};

int main(int argc, char* argv[]) {
  ... run model
}
```

initialize variables

# The Actual Model

```
class Queens : public Space {
protected:
  IntVarArray q; // Position of queen
public:
  Queens(int n) : q(this,n,0,n-1) {
    for (int i = 0; i<n; i++)
      for (int j = i+1; j<n; j++) {
        post(this, q[i] != q[j]);
        post(this, q[i]+i != q[j]+j);
        post(this, q[i]-i != q[j]-j);
      }
    branch(this, q, INT_VAR_SIZE_MIN, INT_VAL_MIN);
  }
  ...
};
```

# Remaining Methods

```
class Queens : public Space {
  ...
  // Constructor for cloning
  Queens(bool share, Queens& s) : Space(share,s) {
    q.update(this, share, s.q);
  }
  // Perform copying during cloning
  virtual Space* copy(bool share) {
    return new Queens(share,*this);
  }
  // Print solution
  void print(void) {
    std::cout << q << std::endl;
  }
};
```

# Solving

```
...
int main(int argc, char* argv[]) {
  int n = atoi(argv[1]);
  Queens* q = dfs(new Queens(n));
  if (q != NULL)
    q->print();
  delete q;
  return 0;
}
```

# Using distinct

```
Queens(int n) : q(this,n,0,n-1) {
  distinct(this, q);

  IntArgs c(n);
  for (int i=0; i<n; i++)
    c[i] = i;
  distinct(this, c, q);

  for (int i=0; i<n; i++)
    c[i] = -i;
  distinct(this, c, q);
  …
```

# Using domain-consistent distinct

```
Queens(int n) : q(this,n,0,n-1) {
  distinct(this, q, ICL_DOM);

  IntArgs c(n);
  for (int i=0; i<n; i++)
    c[i] = i;
  distinct(this, c, q, ICL_DOM);

  for (int i=0; i<n; i++)
    c[i] = -i;
  distinct(this, c, q, ICL_DOM);
  …
```

# More Programming

- Programming propagators and branchings
  - straightforward object-oriented interfaces

- Programming new variable types
  - system-specific aspects generated from simple specification
  - only domain implementation required

- Programming search engines (exploration)
  - based on spaces, similar to Oz [Schulte, CP 1997]

# Contributions

- System contributions
  - fully open design, particular: program variable domains
  - also: systematic tests
- Model contributions
  - organization of propagation
    [Schulte, Stuckey, CP 2004] [Schulte, Stuckey, CoRR, submitted, 2006]
  - views and iterators for generic propagators
    [Schulte, Tack, CP 2005]
  - automatic generation of set propagators
    [Tack, Schulte, Smolka, CP 2006]
  - advisors for incremental propagation
    [Lagerkvist, Schulte, CP 2007]
  - search based on recomputation

# Never Ask Me…

- I have introduced XYZ in my paper…

  …when will you implement it?

- You can do it yourself in Gecode
  - simple, efficient, cheap

- We might even ship it with Gecode
- The community will actually use it