Implementing a Constraint Solver: A Case Study

Emmanuel Hebrard
Cork Constraint Computation Centre & University College Cork

Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
- Search
- Code Optimization
- Competition

Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
- Search
- Code Optimization
- Competition

Goal

Goal

- "Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." [E. Freuder]

Goal

- "Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." [E. Freuder]
- ...if given enough time!

Does Mistral achieve this goal?

Does Mistral achieve this goal?

- Library in C++

Does Mistral achieve this goal?

- Library in C++
- Developed during my PhD
  - Ilog Solver is not open source
  - Good substitutes (Gecode, Choco and others) did not exist yet

Does Mistral achieve this goal?

- Library in C++
- Developed during my PhD
  - Ilog Solver is not open source
  - Good substitutes (Gecode, Choco and others) did not exist yet
- Under GNU General Public License

Does Mistral achieve this goal?

- Library in C++
- Developed during my PhD
  - Ilog Solver is not open source
  - Good substitutes (Gecode, Choco and others) did not exist yet
- Under GNU General Public License
- Why Mistral?
  - because **M**istral **IS** a **T**errific **R**ecursive **A**cronym for a **L**ibrary.

Does Mistral achieve this goal?

- Library in C++
- Developed during my PhD
  - Ilog Solver is not open source
  - Good substitutes (Gecode, Choco and others) did not exist yet
- Under GNU General Public License
- Why Mistral?
  - because **M**istral **IS** a **T**errific **R**ecursive **A**cronym for a **L**ibrary.

Model: Golomb ruler

```
int main(int argc, char *argv[])
{
  // input
  int nbMarks = (argc > 1 ? atoi(argv[1]) : 8);
  int rulerSize = ( 2 << (nbMarks-1) );

  // declare model and variables
  CSP model;
  VarArray mark(nbMarks, 0, rulerSize-1);
  VarArray distance(nbMarks*(nbMarks-1)/2, 1, rulerSize-1);

  // post constraints
  int i, j, k=0;
  for(i=1; i<nbMarks; ++i) {
    model.add( mark[i-1] < mark[i] );
    for(j=0; j<i; ++ j)
      model.add( mark[i] == (mark[j] + distance[k++]) );
  }
  model.add( BoundAllDifferent(distance) );

  // post objective function
  model.add( Minimise( Max(mark) ) );

  // solve
  Solver s( model, mark );
  s.setVerbosity(1);
  s.solve();

  // print search statistics
  s.printStatistics( cout, (Solver::NDS | Solver::RUNTIME) );
  cout << endl;
}
```

Model: Golomb ruler

```
int main(int argc, char *argv[])
// input
int nbMarks = (argc > 1 ? atoi(argv[1]) : 8);
int rulerSize = ( 2 << (nbMarks-1) );

CSP model;
VarArray mark(nbMarks, 0, rulerSize-1);
VarArray distance(nbMarks*(nbMarks-1)/2, 1, rulerSize-1);

// post constraints
int i, j, k=0;
for(i=1; i<nbMarks; ++i) {
  model.add( mark[i-1] < mark[i] );
  for(j=0; j<i; ++ j)
    model.add( mark[i] == (mark[j] + distance[k++]) );
}
model.add( BoundAllDifferent(distance) );

// post objective function
model.add( Minimise( Max(mark) ) );

// solve
Solver s( model, mark );
s.setVerbosity(1);
s.solve();

// print search statistics
s.printStatistics( cout, (Solver::NDS | Solver::RUNTIME) );
cout << endl;
}
```

Model: Golomb ruler

```
int main(int argc, char *argv[])
{
  // input
  int nbMarks = (argc > 1 ? atoi(argv[1]) : 8);
  int rulerSize = ( 2 << (nbMarks-1) );

  // declare model and variables
  CSP model;
  VarArray mark(nbMarks, 0, rulerSize-1);
  VarArray distance(nbMarks*(nbMarks-1)/2, 1, rulerSize-1);

  for(i=1; i<nbMarks; ++i) {
    model.add( mark[i-1] < mark[i] );
    for(j=0; j<i; ++ j)
      model.add( mark[i] == (mark[j] + distance[k++]) );
  }
  model.add( BoundAllDifferent(distance) );

  // post objective function
  model.add( Minimise( Max(mark) ) );

  // solve
  Solver s( model, mark );
  s.setVerbosity(1);
  s.solve();

  // print search statistics
  s.printStatistics( cout, (Solver::NDS | Solver::RUNTIME) );
  cout << endl;
}
```

Model: Golomb ruler

```
int main(int argc, char *argv[])
{
  // input
  int nbMarks = (argc > 1 ? atoi(argv[1]) : 8);
  int rulerSize = ( 2 << (nbMarks-1) );

  // declare model and variables
```

```
// post constraints
int i, j, k=0;
for(i=1; i<nbMarks; ++i) {
  model.add( mark[i-1] < mark[i] );
  for(j=0; j<i; ++ j)
    model.add( mark[i] == (mark[j] + distance[k++]) );
}
model.add( BoundAllDifferent(distance) );
```

```
  // solve
  Solver s( model, mark );
  s.setVerbosity(1);
  s.solve();

  // print search statistics
  s.printStatistics( cout, (Solver::NDS | Solver::RUNTIME) );
  cout << endl;
}
```

Model: Golomb ruler

```
int main(int argc, char *argv[])
{
  // input
  int nbMarks = (argc > 1 ? atoi(argv[1]) : 8);
  int rulerSize = ( 2 << (nbMarks-1) );

  // declare model and variables
  CSP model;
  VarArray mark(nbMarks, 0, rulerSize-1);
  VarArray distance(nbMarks*(nbMarks-1)/2, 1, rulerSize-1);

  // post constraints
  int i, j, k=0;
  for(i=1; i<nbMarks; ++i) {
    model.add( mark[i-1] < mark[i] );
    for(j=0; j<i; ++ j)
      model.add( mark[i] == (mark[j] + distance[k++]) );
  }

// post objective function
model.add( Minimise( Max(mark) ) );

  s.setVerbosity(1);
  s.solve();

  // print search statistics
  s.printStatistics( cout, (Solver::NDS | Solver::RUNTIME) );
  cout << endl;
}
```

Model: Golomb ruler

```
int main(int argc, char *argv[])
{
  // input
  int nbMarks = (argc > 1 ? atoi(argv[1]) : 8);
  int rulerSize = ( 2 << (nbMarks-1) );

  // declare model and variables
  CSP model;
  VarArray mark(nbMarks, 0, rulerSize-1);
  VarArray distance(nbMarks*(nbMarks-1)/2, 1, rulerSize-1);

  // post constraints
  int i, j, k=0;
  for(i=1; i<nbMarks; ++i) {
    model.add( mark[i-1] < mark[i] );
    for(j=0; j<i; ++ j)
      model.add( mark[i] == (mark[j] + distance[k++]) );
  }
  model.add( BoundAllDifferent(distance) );

    // solve
    Solver s( model, mark );
    s.setVerbosity(1);
    s.solve();

  cout << endl;
}
```

Message:

Message:

- Efficient implementation
  - Details do matter

Message:

- Efficient implementation
  - Details do matter
- Modeling choices
  - Automatic choices of the best representation/algorithm
    - ✓ Variable (Constant, Boolean, Interval, Bitset, List, ...)
    - ✓ Constraints (Specific algorithm, Decomposition, Generic algorithms, ...)
    - ✓ Heuristics
  - Automatic rewritting?

Message:

- Efficient implementation
  - Details do matter
- Modeling choices
  - Automatic choices of the best representation/algorithm
    - ✓ Variable (Constant, Boolean, Interval, Bitset, List, ...)
    - ✓ Constraints (Specific algorithm, Decomposition, Generic algorithms, ...)
    - ✓ Heuristics
  - Automatic rewritting?
- Robustness
  - Worst case principle

Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
- Search
- Code Optimization
- Competition

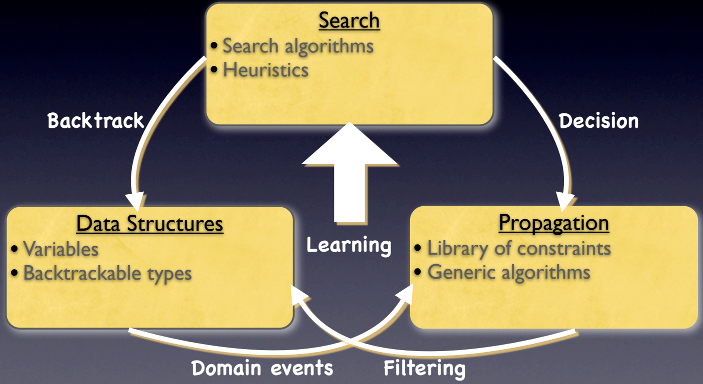
A little bit of structure

Search
- Search algorithms
- Heuristics

Data Structures
- Variables
- Backtrackable types

Propagation
- Library of constraints
- Generic algorithms

A little bit of structure



Road-map

- Goal
- Blueprint
- Data Structures
  - Variables (Baktracks)
- Propagation
- Search
- Code Optimization
- Competition

Backtrackable Data-Structures

- Copying/Trailing
  - See Shulte's papers and PhD Thesis
  - Copying
    - ✓ Easier to implement data structures
    - ✓ Easier to implement search strategies
    - ✓ Easier to parallelize
  - Trailing
    - ✓ Do only necessary work
    - ✓ Memory efficient

Copying

Trailing

Domain as a Bitset

- One 32 bits word for every value in [min(D)..max(D)]

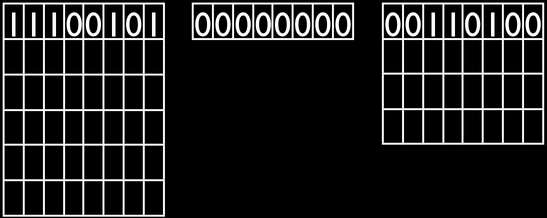
X in
{0,1,2,5,7,18,19,21}

11100101	00000000	00110100

Domain as a Bitset

- One 32 bits word for every value in [min(D)..max(D)]
- For every word, we allocate as many word as values in that word:
  - O((max-min+1) + 32*|D|) bits

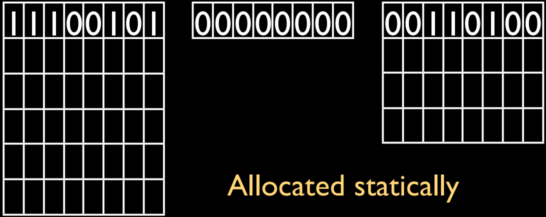
X in
{0,1,2,5,7,18,19,21}



Domain as a Bitset

- One 32 bits word for every value in [min(D)..max(D)]
- For every word, we allocate as many word as values in that word:
  - O((max-min+1) + 32*|D|) bits

X in
{0,1,2,5,7,18,19,21}



Allocated statically

Domain as a Bitset

X in {0,1,2,5,7,18,19,21}

Domain as a Bitset

1	1	1	0	0	1	0	1

0	0	0	0	0	0	0	0

0	0	1	1	0	1	0	0

X in {0,1,2,5,7,18,19,21}

Domain as a Bitset

decision 1 → {0,5,7,18,19,21}

1	1	1	0	0	1	0	1

0	0	0	0	0	0	0	0

0	0	1	1	0	1	0	0

X in {0,1,2,5,7,18,19,21}

Domain as a Bitset

decision 1 ⇒ {0,5,7,18,19,21} w1

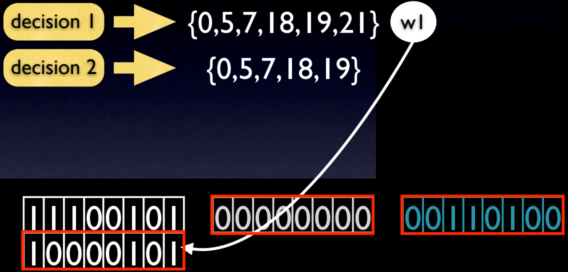
1	1	1	0	0	1	0	1
1	0	0	0	0	1	0	1

0	0	0	0	0	0	0	0

0	0	1	1	0	1	0	0

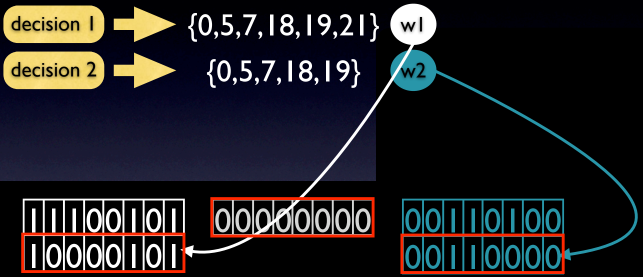
X in {0,1,2,5,7,18,19,21}

Domain as a Bitset



X in {0,1,2,5,7,18,19,21}

Domain as a Bitset



X in {0,1,2,5,7,18,19,21}

Domain as a Bitset

decision 1 → {0,5,7,18,19,21} w1

decision 2 → {0,5,7,18,19} w2

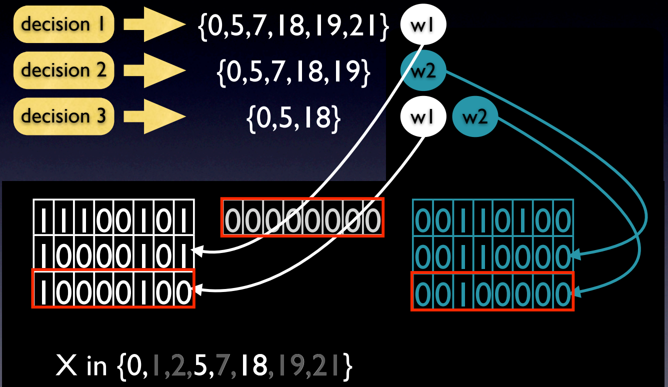
decision 3 → {0,5,18}

11100101 00000000 00110100

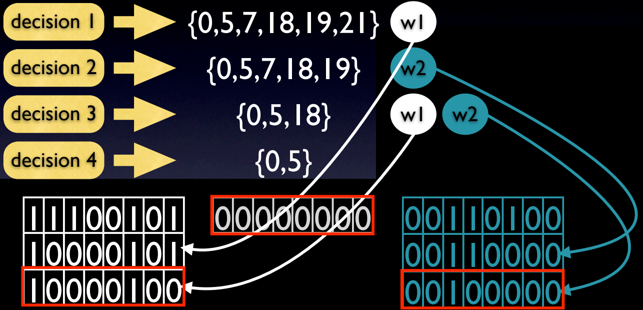
10000101 00110000

X in {0,1,2,5,7,18,19,21}

Domain as a Bitset

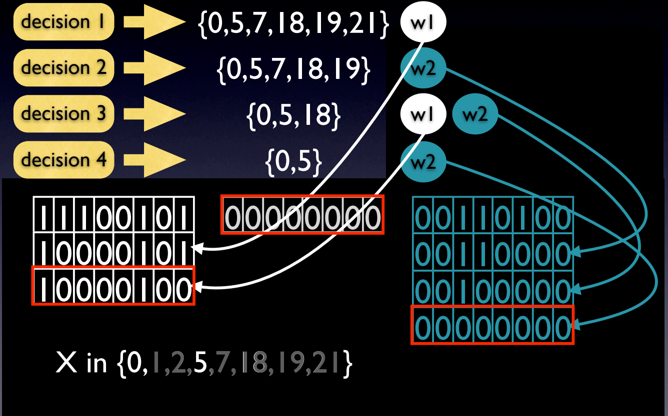


Domain as a Bitset

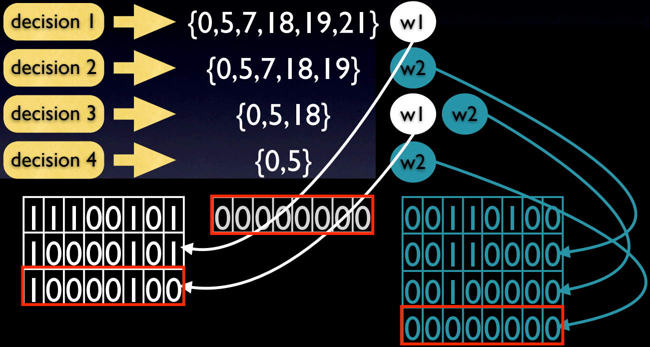


X in {0,1,2,5,7,18,19,21}

Domain as a Bitset

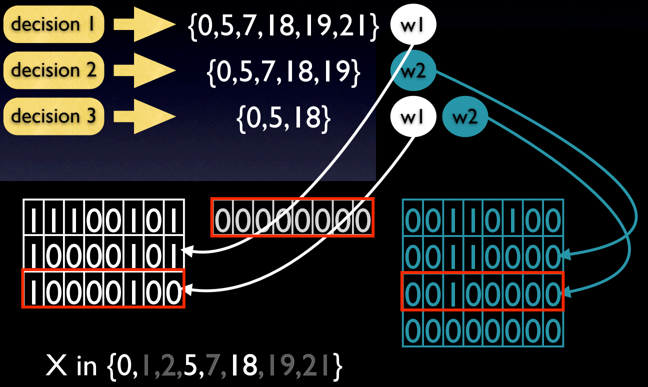


Domain as a Bitset

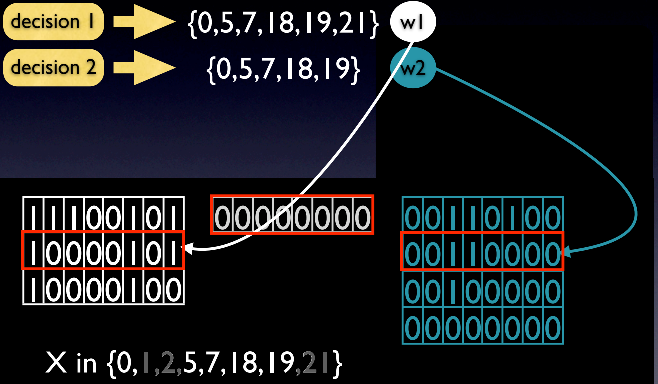


X in {0,1,2,5,7,18,19,21}

Domain as a Bitset



Domain as a Bitset



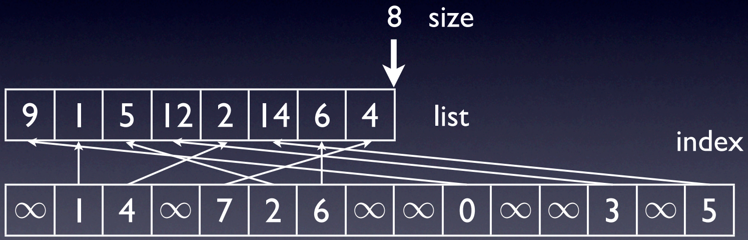
Domain as a List

Domain as a List

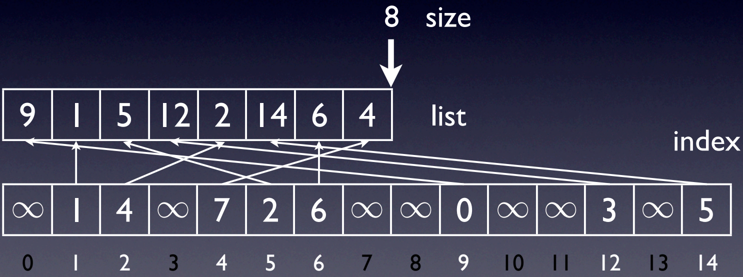
9	1	5	12	2	14	6	4	list

Domain as a List

Domain as a List

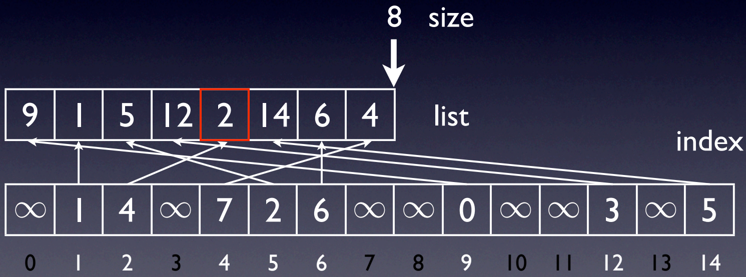


Domain as a List



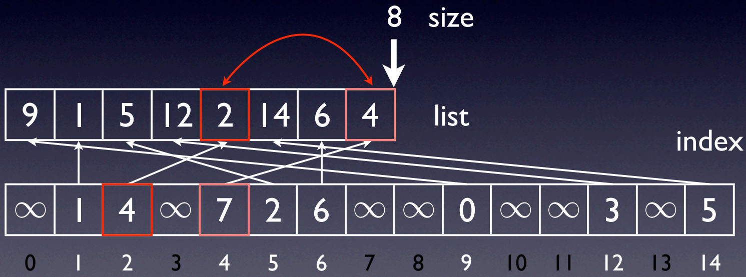
membership: *index[v] < size*

Domain as a List



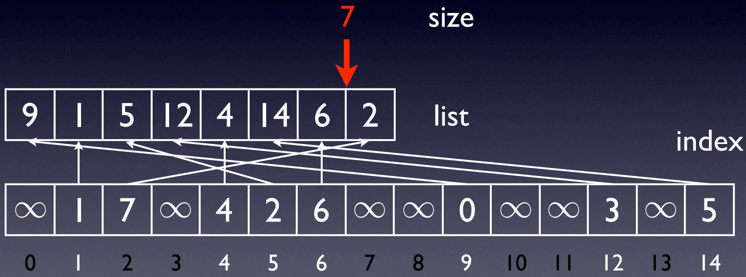
membership: *index[v] < size*

Domain as a List



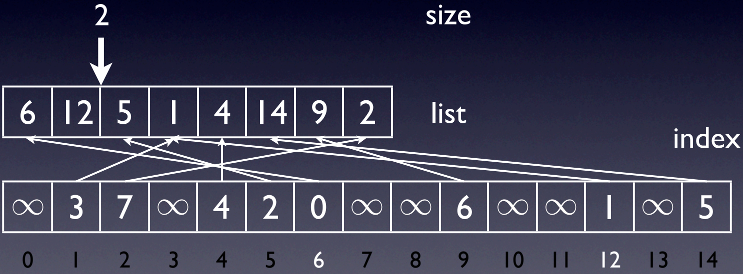
membership: *index[v] < size*

Domain as a List



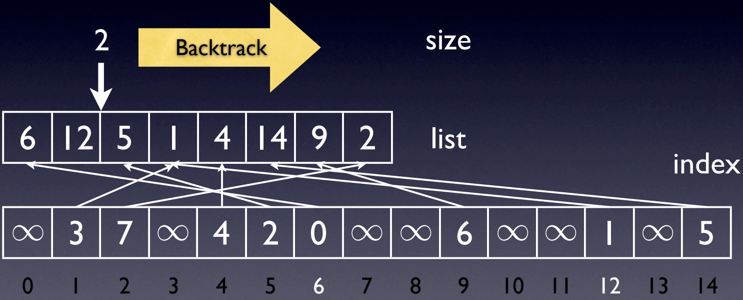
membership: *index[v] < size*

Domain as a List



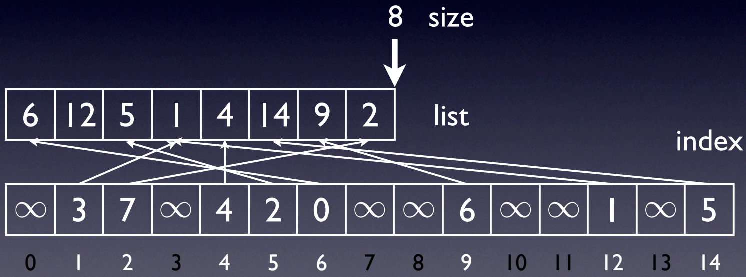
membership: *index[v] < size*

Domain as a List



membership: *index[v] < size*

Domain as a List



membership: *index[v] < size*

Pigeon holes

Pigeon holes

- Domain as a Bitset:
  - ✓ Space complexity in O(max-min)
  - ✓ Restore up to 32 values at a time
  - ✓ 600,000 Bts/second

Pigeon holes

- Domain as a Bitset:
  - ✓ Space complexity in O(max-min)
  - ✓ Restore up to 32 values at a time
  - ✓ 600,000 Bts/second
- Domain as a List:
  - ✓ Similar space complexity
  - ✓ Restore any number of values in one operation
  - ✓ 900,000 Bts/second
  - ✓ However, operations are much slower on the list (interval reasoning, set operations)

Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
  - Nested predicates
  - GAC valid v allowed
- Search
- Code Optimization
- Competition

Constraint Propagation

- Variable/Constraint Queue
- Specific Propagators
- Nested Predicates
- Generic AC algorithms
  - Binary: AC3Bitset
  - Tight: GAC2001Allowed
  - Loose: GAC3rValid

Pruning:

Nested Predicates

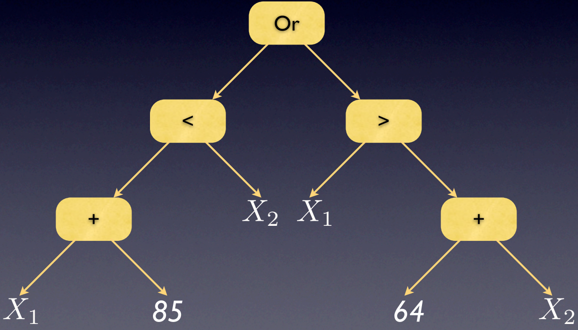
Example: open-shop scheduling:

```
<predicate name="P0">
  <parameters>int X0 int X1  int X2 int X3 int X4 int X5</parameters>
  <expression>
    <functional>or(le(add(X0,X1),X2),le(add(X3,X4),X5))<functional>
  </expression>
</predicate>

 <constraint name="C0" arity="2" scope="V0 V1" reference="P0">
   <parameters>V0 85 V1 V1  64 V0</parameters>
 </constraint>
```

Nested Predicates: Decomposition

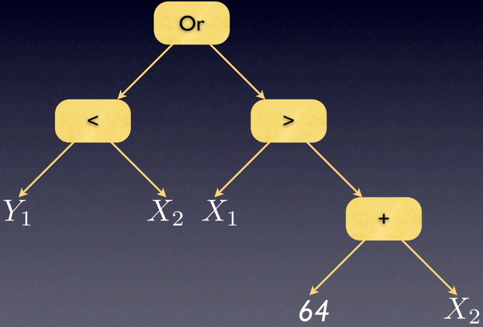
Example: open-shop scheduling:



Nested Predicates: Decomposition

Example: open-shop scheduling:

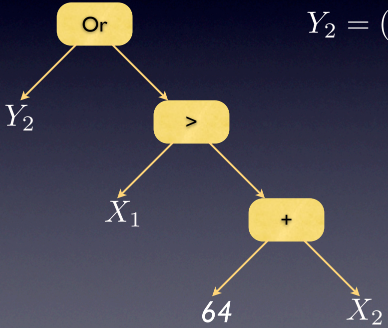
$$Y_1 = X_1 + 85$$



Nested Predicates: Decomposition

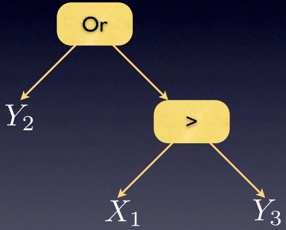
Example: open-shop scheduling:

$$Y_1 = X_1 + 85$$
$$Y_2 = (Y_1 < X_2)$$



Nested Predicates: Decomposition

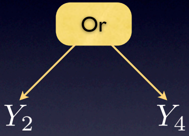
Example: open-shop scheduling:



$$Y_1 = X_1 + 85$$
$$Y_2 = (Y_1 < X_2)$$
$$Y_3 = X_2 + 64$$

Nested Predicates: Decomposition

Example: open-shop scheduling:



$$Y_1 = X_1 + 85$$
$$Y_2 = (Y_1 < X_2)$$
$$Y_3 = X_2 + 64$$
$$Y_4 = (Y_3 < X_1)$$

Nested Predicates: Decomposition

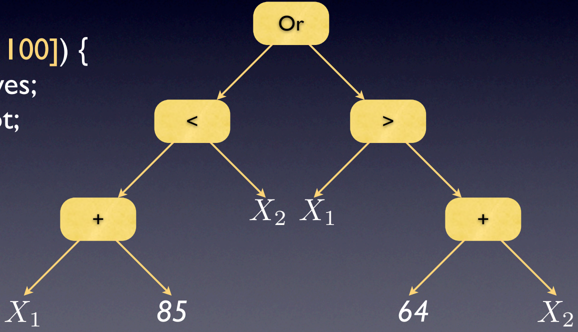
Example: open-shop scheduling:

$$Y_1 = X_1 + 85$$
$$Y_2 = (Y_1 < X_2)$$
$$Y_3 = X_2 + 64$$
$$Y_4 = (Y_3 < X_1)$$
$$(Y_2 \vee Y_4)$$

Nested Predicates: GAC-Checker

Example: open-shop scheduling:

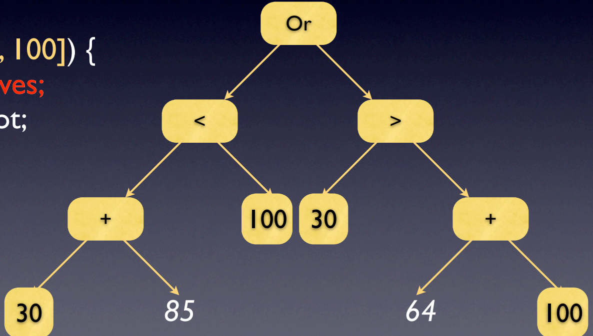
```
check ([30, 100]) {
  assign leaves;
  query root;
}
```



Nested Predicates: GAC-Checker

Example: open-shop scheduling:

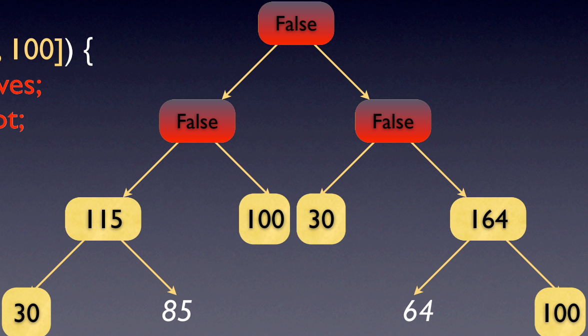
```
check ([30, 100]) {
  assign leaves;
  query root;
}
```



Nested Predicates: GAC-Checker

Example: open-shop scheduling:

```
check ([30, 100]) {
  assign leaves;
  query root;
}
```



Golomb ruler / FAPP

Instance:

Decomposition

GAC-Checker

Golomb ruler / FAPP

Instance:	Golomb Ruler
Decomposition	128 nodes 0.18 seconds
GAC-Checker	87 nodes 38.22 seconds

Golomb ruler / FAPP

Instance:	Golomb Ruler	FAPP
Decomposition	128 nodes 0.18 seconds	60181 nodes 55.18 seconds
GAC-Checker	87 nodes 38.22 seconds	374 nodes 1.18 seconds

Making the Right Choice

Making the Right Choice

Feature	GAC	Decomp	OSP

Making the Right Choice

Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary

Making the Right Choice

Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary
Ratio node/leaves	+	-	5/2

Making the Right Choice

Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary
Ratio node/leaves	+	-	5/2
Domain continuity	-	+	no holes

Making the Right Choice

Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary
Ratio node/leaves	+	-	5/2
Domain continuity	-	+	no holes
Cartesian product cardinality	-	+	>60,000

Making the Right Choice

Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary
Ratio node/leaves	+	-	5/2
Domain continuity	-	+	no holes
Cartesian product cardinality	-	+	>60,000
Boolean domains	-	+	no

Making the Right Choice

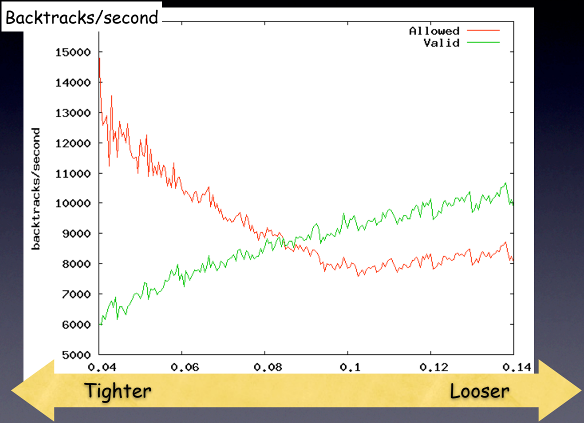
Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary
Ratio node/leaves	+	-	5/2
Domain continuity	-	+	no holes
Cartesian product cardinality	-	+	>60,000
Boolean domains	-	+	no
Total number of constraints	+	-	small

Making the Right Choice

Feature	GAC	Decomp	OSP
Constraint Arity	-	+	binary
Ratio node/leaves		-	5/2
Domain continuit		+	no holes
Cartesian product cardinality	-	+	>60,000
Boolean domains	-	+	no
Total number of constraints	+	-	small

Decomposition!

GAC Allowed v. GAC Valid



Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
- Search
  - Heuristics
- Code Optimization
- Competition

Search Strategies

- Depth-first, Breadth-first, LDS,...
- Branching Choices
  - Domain Splitting
  - Arbitrary Constraint
- Variable/Value Ordering
  - "Learning" heuristics
    - ✓ Weighted Degree, Impact

Weighted Heuristics

- The best general purpose orderings are based on some kind of learning (or weighting)
  - Weighted Degree [Boussemart, Hemery, Lecoutre, Sais 2004]
  - Impact [Refalo 2004]
- A "Weighter" can suscribe for different types of event
  - Weighted degree: failures
  - Impact: Decisions, success, failures
- This architecture allows easy development of variations around these models
  - Why isn't Impact/Weighted Degree any good?

Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
- Search
- Code Optimization
- Competition

Optimisation: Binary Extensional

- Standard algorithms:
  - AC3-bitset, Variable queue (fifo), revision condition
- Profiling, what does take time?
  - Propagation:.. 68%
    - “&” operation:.................................... 25.9%
    - Domain iteration:................................ 20.6%
    - AC3 (queuing/dequeuing):................ 11.4%
    - Revision condition + virtual call:.. 10.0%
  - Data structure modification:........... 19%
    - Domain modification:......................... 19.0%
  - Search:.. 12%
    - Trailing:.. 9.7%
    - Variable choice + Branching:........... 2.2%

Intersection on Bitsets (25%)

```
bool VariableList::wordIntersect(const MistralSet& s) const
{
  return values.wordIntersect(s);
}

inline bool MistralSet::wordIntersect(const MistralSet& s) const
{
  return ( table[neg_words] & s.table[neg_words] ) ;
}
```

Values Iteration (20%)

Values Iteration (20%)

- Random binary CSP

Values Iteration (20%)

- Random binary CSP
- Domain as a Bitset:
  - 6,500 Bts/second

Values Iteration (20%)

- Random binary CSP
- Domain as a Bitset:
  - 6,500 Bts/second
- Domain as a List (hybrid bitset/list):
  - 10,000 Bts/second
  - Values are stored contiguously in an array
  - The order does not matter

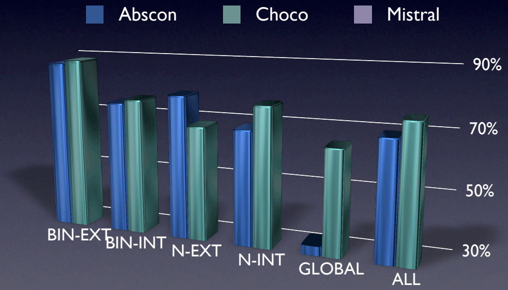
Road-map

- Goal
- Blueprint
- Data Structures
- Propagation
- Search
- Code Optimization
- Competition

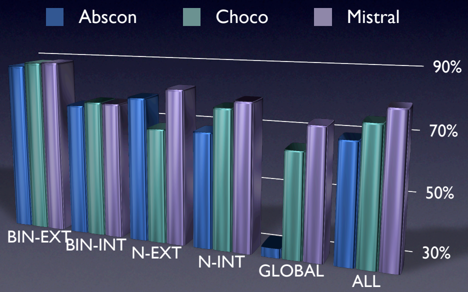
Quick Comparison
(#instances)

Abscon Choco Mistral

Quick Comparison
(#instances)



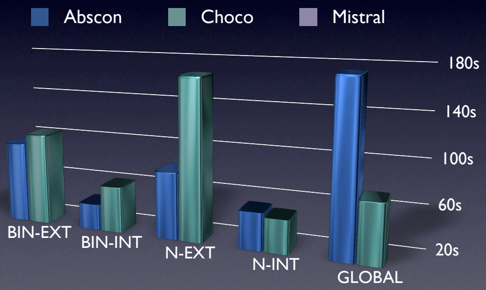
Quick Comparison
(#instances)



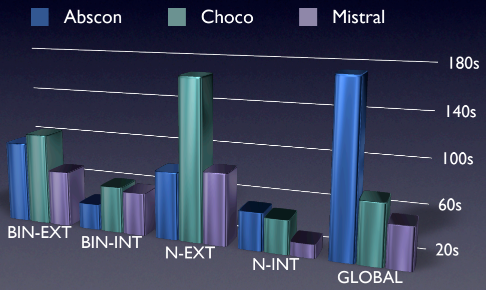
Quick Comparison
(cpu time)

Abscon Choco Mistral

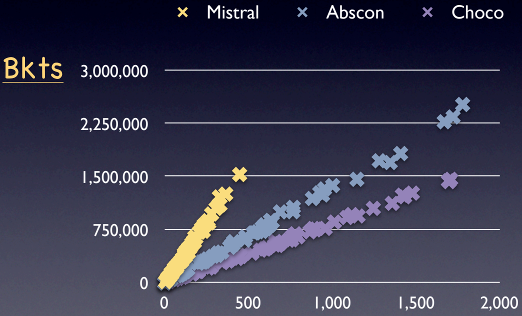
Quick Comparison
(cpu time)



Quick Comparison
(cpu time)

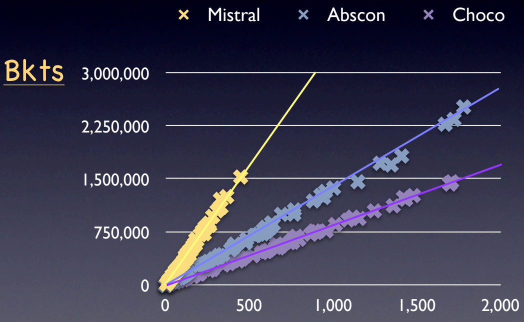


Backtracks v CPU-time



rand-2-40-19-443-230-* and frb40-19

Backtracks v CPU-time



rand-2-40-19-443-230-* and frb40-19

Conclusion

Conclusion

- Implementing a constraint solver may appear like a daunting task

Conclusion

- Implementing a constraint solver may appear like a daunting task
  - It is so.

Conclusion

- Implementing a constraint solver may appear like a daunting task
  - It is so.
  - But you'll learn a lot!

Conclusion

- Implementing a constraint solver may appear like a daunting task
  - It is so.
  - But you'll learn a lot!
- Adaptability

Conclusion

- Implementing a constraint solver may appear like a daunting task
  - It is so.
  - But you'll learn a lot!
- Adaptability
- Attention to details

Conclusion

- Implementing a constraint solver may appear like a daunting task
  - It is so.
  - But you'll learn a lot!
- Adaptability
- Attention to details
- Robustness

Conclusion

- Implementing a constraint solver may appear like a daunting task
  - It is so.
  - But you'll learn a lot!
- Adaptability
- Attention to details
- Robustness
  - Weaknesses are always more obvious to a user than strengths