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 Mistral IS a Terrific Recursive Acronym for a Library.
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 Mistral IS a Terrific Recursive Acronym for a Library.
Model: Golomb ruler

```cpp
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

```c
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

```c
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

```cpp
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

```c
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

```c
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 rewriting?
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 rewriting?

• 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

- **Search**
  - Search algorithms
  - Heuristics

- **Data Structures**
  - Variables
  - Backtrackable types

- **Propagation**
  - Library of constraints
  - Generic algorithms

- **Decision**

- **Backtrack**

- **Domain events**

- **Filtering**

- **Learning**
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
Domain as a Bitset

- One 32 bits word for every value in \([\min(D) .. \max(D)]\)
Domain as a Bitset

- One 32 bits word for every value in \([\text{min}(D)\ldots\text{max}(D)]\)
- For every word, we allocate as many word as values in that word:
  - \(O((\text{max}-\text{min}+1) + 32|D|)\) bits
Domain as a Bitset

- One 32 bits word for every value in $[\text{min}(D) \ldots \text{max}(D)]$
- For every word, we allocate as many word as values in that word:
  - $0((\text{max}-\text{min}+1) + 32 \times |D|)$ bits

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

Allocated statically
Domain as a Bitset

\[ X \in \{0,1,2,5,7,18,19,21\} \]
Domain as a Bitset

X in \{0, 1, 2, 5, 7, 18, 19, 21\}
Domain as a Bitset

decision 1 \rightarrow \{0,5,7,18,19,21\}

X in \{0,1,2,5,7,18,19,21\}
Domain as a Bitset

\[ \text{decision 1} \rightarrow \{0, 5, 7, 18, 19, 21\} \]

\[ X \text{ in } \{0, 1, 2, 5, 7, 18, 19, 21\} \]
Domain as a Bitset

**Decision 1** → \{0,5,7,18,19,21\}

**Decision 2** → \{0,5,7,18,19\}

\[ w_1 \]

\[ X \text{ in } \{0,1,2,5,7,18,19,21\} \]
Domain as a Bitset

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

decision 1 → \{0, 5, 7, 18, 19, 21\}

decision 2 → \{0, 5, 7, 18, 19\}

w1

w2

00110100
00110000
00000000
00000000
11100101
10000101
10000101
Domain as a Bitset

- Decision 1: \{0, 5, 7, 18, 19, 21\}
- Decision 2: \{0, 5, 7, 18, 19\}
- Decision 3: \{0, 5, 18\}

\[ X \in \{0, 1, 2, 5, 7, 18, 19, 21\} \]
Domain as a Bitset

- Decision 1: \{0,5,7,18,19,21\}
- Decision 2: \{0,5,7,18,19\}
- Decision 3: \{0,5,18\}

\( X \) in \{0,1,2,5,7,18,19,21\}
Domain as a Bitset

- **decision 1**: \{0,5,7,18,19,21\}
- **decision 2**: \{0,5,7,18,19\}
- **decision 3**: \{0,5,18\}
- **decision 4**: \{0,5\}

X in \{0,1,2,5,7,18,19,21\}
Domain as a Bitset

- Decision 1: \{0,5,7,18,19,21\}
- Decision 2: \{0,5,7,18,19\}
- Decision 3: \{0,5,18\}
- Decision 4: \{0,5\}

X in \{0,1,2,5,7,18,19,21\}
Domain as a Bitset

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

\begin{align*}
\text{decision 1} & \rightarrow \{0,5,7,18,19,21\} \\
\text{decision 2} & \rightarrow \{0,5,7,18,19\} \\
\text{decision 3} & \rightarrow \{0,5,18\} \\
\text{decision 4} & \rightarrow \{0,5\}
\end{align*}
Domain as a Bitset

decision 1 \rightarrow \{0,5,7,18,19,21\}
decision 2 \rightarrow \{0,5,7,18,19\}
decision 3 \rightarrow \{0,5,18\}

\text{X in \{0,1,2,5,7,18,19,21\}}
Domain as a Bitset

Decision 1: \{0,5,7,18,19,21\}

Decision 2: \{0,5,7,18,19\}

\(X\) in \{0,1,2,5,7,18,19,21\}
Domain as a List
Domain as a List

9 1 5 12 2 14 6 4 list
Domain as a List

8 size

9 1 5 12 2 14 6 4 list
Domain as a List

8 size

9 1 5 12 2 14 6 4

∞ 1 ∞ 7 2 6 ∞ ∞ 0 ∞ ∞ 3 ∞ 5

list

index
Domain as a List

```
9 1 5 12 2 14 6 4
1 4 7 2 6 0 3 5
0 1 2 3 4 5 6 7 8 9 10 11 13 14
∞ ∞ ∞ ∞ ∞ ∞ ∞
8
```

membership: $index[v] < size$
Domain as a List

membership: index[v] < size
Domain as a List

membership: \textit{index}[v] < \textit{size}
Domain as a List

membership: \( \text{index}[v] < \text{size} \)
Domain as a List

```plaintext
6 12 5 1 4 14 9 2
3 7 4 2 0 6 1 5
0 1 2 3 4 5 6 7 8 9 10 11 13 14
∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

membership: \( \text{index}[v] < \text{size} \)
```
Domain as a List

membership: \( \text{index}[v] < \text{size} \)
Domain as a List

membership: \( \text{index}[v] < \text{size} \)
Pigeon holes
Pigeon holes

- Domain as a Bitset:
  - Space complexity in $O(\text{max-min})$
  - Restore up to 32 values at a time
  - 600,000 Bts/second
Pigeon holes

• Domain as a Bitset:
  ✓ Space complexity in $O(\text{max-min})$
  ✓ 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:

```xml
<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:

\[
\begin{align*}
X_1 &< X_2 \\
+ &85 \\
> &X_1 \\
+ &64 \\
&X_2
\end{align*}
\]
Nested Predicates: Decomposition

Example: open-shop scheduling:

\[ Y_1 = X_1 + 85 \]

\[
\begin{align*}
Y_1 &< X_2 \\
&> X_1 \\
&+ 64 \\
&= X_2
\end{align*}
\]
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:

\[
\begin{align*}
Y_1 &= X_1 + 85 \\
Y_2 &= (Y_1 < X_2) \\
Y_3 &= X_2 + 64
\end{align*}
\]
Nested Predicates: Decomposition

Example: open-shop scheduling:

\[
\begin{align*}
Y_1 &= X_1 + 85 \\
Y_2 &= (Y_1 < X_2) \\
Y_3 &= X_2 + 64 \\
Y_4 &= (Y_3 < X_1)
\end{align*}
\]
Nested Predicates: Decomposition

Example: open-shop scheduling:

\[
Y_1 = X_1 + 85 \\
Y_2 = (Y_1 < X_2) \\
Y_3 = X_2 + 64 \\
Y_4 = (Y_3 < X_1) \\
(Y_2 \lor Y_4)
\]
Nested Predicates: GAC-Checker

Example: open-shop scheduling:

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

X_1 + X_2 < X_1 + X_2 > 85 64

Or

X_1 X_2
Nested Predicates: GAC-Checker

Example: open-shop scheduling:

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

```
Or
  <
    +
      30
      85
  >
    +
      100
      30
        +
          64
          100
```
Nested Predicates: GAC-Checker

Example: open-shop scheduling:

check ([30, 100]) {
    assign leaves;
    query root;
}
Golomb ruler / FAPP

Instance:
- Decomposition
- GAC-Checker
<table>
<thead>
<tr>
<th>Instance:</th>
<th>Golomb Ruler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition</td>
<td>128 nodes</td>
</tr>
<tr>
<td></td>
<td>0.18 seconds</td>
</tr>
<tr>
<td>GAC-Checker</td>
<td>87 nodes</td>
</tr>
<tr>
<td></td>
<td>38.22 seconds</td>
</tr>
<tr>
<td>Instance</td>
<td>Golomb Ruler</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Decomposition</td>
<td>128 nodes 0.18 seconds</td>
</tr>
<tr>
<td>GAC-Checker</td>
<td>87 nodes 38.22 seconds</td>
</tr>
</tbody>
</table>
Making the Right Choice
<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decomp</th>
<th>OSP</th>
</tr>
</thead>
</table>

Making the Right Choice
## Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decomp</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>-</td>
<td>+</td>
<td>binary</td>
</tr>
</tbody>
</table>
# Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decomp</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>-</td>
<td>+</td>
<td>binary</td>
</tr>
<tr>
<td>Ratio node/leaves</td>
<td>+</td>
<td>-</td>
<td>5/2</td>
</tr>
</tbody>
</table>
## Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decompo</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>-</td>
<td>+</td>
<td>binary</td>
</tr>
<tr>
<td>Ratio node/leaves</td>
<td>+</td>
<td>-</td>
<td>5/2</td>
</tr>
<tr>
<td>Domain continuity</td>
<td>-</td>
<td>+</td>
<td>no holes</td>
</tr>
</tbody>
</table>
## Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decompo</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>–</td>
<td>+</td>
<td>binary</td>
</tr>
<tr>
<td>Ratio node/leaves</td>
<td>+</td>
<td>–</td>
<td>5/2</td>
</tr>
<tr>
<td>Domain continuity</td>
<td>–</td>
<td>+</td>
<td>no holes</td>
</tr>
<tr>
<td>Cartesian product cardinality</td>
<td>–</td>
<td>+</td>
<td>&gt;60,000</td>
</tr>
</tbody>
</table>
## Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decomp</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>-</td>
<td>+</td>
<td>binary</td>
</tr>
<tr>
<td>Ratio node/leaves</td>
<td>+</td>
<td>-</td>
<td>5/2</td>
</tr>
<tr>
<td>Domain continuity</td>
<td>-</td>
<td>+</td>
<td>no holes</td>
</tr>
<tr>
<td>Cartesian product cardinality</td>
<td>-</td>
<td>+</td>
<td>&gt;60,000</td>
</tr>
<tr>
<td>Boolean domains</td>
<td>-</td>
<td>+</td>
<td>no</td>
</tr>
</tbody>
</table>
## Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decomp</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>-</td>
<td>+</td>
<td>binary</td>
</tr>
<tr>
<td>Ratio node/leaves</td>
<td>+</td>
<td>-</td>
<td>5/2</td>
</tr>
<tr>
<td>Domain continuity</td>
<td>-</td>
<td>+</td>
<td>no holes</td>
</tr>
<tr>
<td>Cartesian product cardinality</td>
<td>-</td>
<td>+</td>
<td>&gt;60,000</td>
</tr>
<tr>
<td>Boolean domains</td>
<td>-</td>
<td>+</td>
<td>no</td>
</tr>
<tr>
<td>Total number of constraints</td>
<td>+</td>
<td>-</td>
<td>small</td>
</tr>
</tbody>
</table>
# Making the Right Choice

<table>
<thead>
<tr>
<th>Feature</th>
<th>GAC</th>
<th>Decomp</th>
<th>OSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Arity</td>
<td>-</td>
<td>+</td>
<td>binary</td>
</tr>
<tr>
<td>Ratio node/leaves</td>
<td>-</td>
<td>-</td>
<td>5/2</td>
</tr>
<tr>
<td>Domain continuity</td>
<td>+</td>
<td></td>
<td>no holes</td>
</tr>
<tr>
<td>Cartesian product cardinality</td>
<td>-</td>
<td>+</td>
<td>&gt;60,000</td>
</tr>
<tr>
<td>Boolean domains</td>
<td>-</td>
<td>+</td>
<td>no</td>
</tr>
<tr>
<td>Total number of constraints</td>
<td>+</td>
<td>-</td>
<td>small</td>
</tr>
</tbody>
</table>

Decomposition!
GAC Allowed v. GAC Valid

Backtracks/second

- Tighter
- Looser
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%
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)

Abscon  Choco  Mistral

BIN-EXT  BIN-INT  N-EXT  N-INT  GLOBAL  ALL
Quick Comparison
(#instances)

- Abscon
- Choco
- Mistral

Bar chart showing comparison of BIN-EXT, BIN-INT, N-EXT, N-INT, GLOBAL, and ALL categories.
Quick Comparison
(cpu time)

Abscon  Choco  Mistral
Quick Comparison (cpu time)
Quick Comparison (cpu time)
Backtracks v CPU-time

- **Mistral**
- **Abscon**
- **Choco**

**Bkts**

- 3,000,000
- 2,250,000
- 1,500,000
- 750,000
- 0

**CPU-time**

- rand-2-40-19-443-230-* and frb40-19
Backtracks v CPU-time

Bkts

0 500 1,000 1,500 2,000

Mistral Abscon Choco

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