Minion:
Fast, Scalable Constraint Solving

Ian Gent, Chris Jefferson, Ian Miguel
60 Second Introduction to CSPs

Standard Definition

- A CSP is a tuple \(<V, D, C>\)
  - \(V\): list of variables
  - \(D\): a domain for each member of \(V\)
  - \(C\): set of constraints

- Solution: An assignment to each variable which satisfies all the constraints.
60 Second Introduction to CSPs

Simple Example

\[ X \in \{1, 2, 3\}, \ Y \in \{1, 2, 3\}, \ Z \in \{1, 2, 3\} \]

\[ X < Y \quad X + Y + Z > 6 \]

A solution: \( X = 2, \ Y = 3, \ Z = 2 \)
Solving CSPs

- **Store** list of values allowed for each variable.
- **Branch** on domains of variables.
- **Propagation** removes value from which are not part of a solution.
Example

\[ X \leq Y \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X\in)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Y\in)</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Example

\[
\begin{array}{ccc}
X \in & 1 & 2 & 3 \\
Y \in & 1 & 2 & 3 \\
\end{array}
\]

\[X < Y\]
Constraint Toolkits

- Most CP solvers are a toolkit:
  - Designed to be linked to own program.
  - Library of modelling/solving functions.
- Examples:
  - ILOG Solver, Choco, Eclipse, Gecode.
Constraint Toolkits

- Toolkits provide a high level of customisation
  - Write own search strategy, propagation, ...
  - Dynamically expand / rewrite problem during search to improve performance.
- Constraint toolkits very powerful in the hands of an expert.
- Complex flexible architecture gives rise to overheads.
Model and Run?

To achieve widespread acceptance, Puget (designer of original ILOG Solver) advocates a model and run paradigm.

- Avoid need for customisation.
- Avoid reliance on fine-tuning.
- Just feed the solver a model and press go.
- Want a **black-box** solver.
Motivation: SAT Solvers

- SAT is a subset of CSP.
  - Only Boolean variables.
  - Each constraint a disjunction of literals
    - e.g. \( x_1 \lor \neg x_2 \lor x_3 \).
- Most SAT solvers are highly optimised black boxes.
Black Box Solving

- Minion, an open source Constraint Solver.
  - [http://minion.sourceforge.net](http://minion.sourceforge.net)
- Optimised for solving large, hard problems.
- Number of design decisions inspired by SAT solver, such as zChaff.
  - e.g. careful design of data structures.
Minion Limitations

- No dynamic variable ordering.
- No dynamic symmetry breaking.
- No adding / removing variables during search.
- No adding / removing constraints during search.
- No hybrids with other solvers.
- Quite a few bugs. 😊
  - But fixing them when they are found!
Benchmarks

- Minion vs. ILOG Solver 5.3 (6.3)
- Hardest instance tried.
  - BIBD: 100 (2) times faster.
  - Peg Solitaire: 10 (15) times faster.
  - Steel Mill: 50 (30) times faster.
  - Golomb: 1.8 (2.5) times faster.
  - Quasigroup: 100 (200) times faster.
- At Quasigroup, MiniSAT 15 times faster than Minion
Minion Memory

- We wish to exploit cache-based architectures.
  - Has been shown to pay dividends in SAT solvers.
- During initialisation all required backtrackable memory is re-arranged into one contiguous block: locality of reference
  - Non-backtrackable data also stored together.
Minion Memory

Information which must be backtracked

Information not backtracked
Minion Memory

Copy

Stack of states

On Branch
Minion Memory

On Branch

Copy
Minion Memory

On Backtrack

Copy
Minion Memory

- Leaving variables in a fixed location speeds up access
  - Variables never move.
- At the moment, just copy all memory.
  - Stupid, but copy is VERY quick.
- (New) using OS lazy copy-on-write.
Requirements of CSP Variables

- Quickly check & remove domain values
- Quickly check & change bounds
- Boolean domains
- Small domains
- Large domains
Storing Variables

- Storing any subset of \([1..n]\) requires \(n\) bits.
- There are a number of ways of getting a better average case.
  - Example: Series of bounds
    - \([1..5] [8..20]\)
Adjustable sized variables
Adjustable sized variables

Constraints
Adjustable sized variables

Constraints
Minion Memory

Each variable takes a fixed sized block of memory.

- Constraints can point directly at the variables.
- No need for indirection.
Variables

- Rather than one “best” variable representation, Minion has multiple representations.
- User must choose between these.
  - In the future, heuristics will choose.
- Special tricks for Booleans, bound variables and general domains.
Implementing multiple variable types for one constraint:

- Abstract interface with variables chosen at run-time.

- Slow.

- No inlining.
Different Variable Types

- Implementing multiple variable types for one constraint:
  - Implement each constraint for each type of variable.

- Fast.

- Have to write too much.
Compile-time Interfaces

- Define a minimal interface and compile each constraint with each variable type.
- Compiler optimisation removes the interface.
- Allows most constraints to have a single implementation.
- Looking at assembler, often identical to specialised implementations.
Results and Discussion

- Minion performs well on the problems we have tried.
- Where is the gain?
  - The search trees are the same.
  - Purely from architectural advances.
- Could we tune the others to narrow the gap?
  - Yes... but hardly “model and run”!
Present (and future)

- Minion now uses more from SAT solvers.
  - Watched Literals in Minion (CP06)
- Two major areas:
  - Learning (needs to be in Minion)
  - Modelling (Solver independent?)
- Still a magnitude slower than SAT solvers in nodes/second.

http://minion.sourceforge.net
Any Questions?

http://minion.sourceforge.net
Adding New Features

- Do you really need it?
  - Need to gain back the speed lost.
- Can the solver choose when it should be used?
  - Avoid giving users choices they don’t want.
- Can it be implemented without overhead when it is not used?
- Solver synthesis - exactly the Solver you want.
Boolean Variables

- Simplest kind of variable.
- Many problems have huge number of Booleans.

Represent with two bits:

- Is Assigned
- Value Assigned
Boolean Variables

Assigned True

Assigned False

Boolean Unassigned
Boolean Variables

- The “assignment” bit is not backtracked!
  - If variable still assigned, has same value.
  - If unassigned, value unused.
Booleans in Search

- Start of search.
- "Value" is set to a random value.
- "Assigned" = 0
Booleans in Search

- Search branches.
- The variable is assigned ‘1’.
- Value and Assigned bits both set.
Booleans in Search

- Search branches.
- The variable is assigned ‘1’.
- Value and Assigned bits both set.
- Under this node, search leaves variable the same.
Booleans in Search

- Search branches.
- The variable is assigned ‘1’.
- Value and Assigned bits both set.
- Under this node, search leaves variable the same.
Booleans in Search

- Search branches.
- The variable is assigned ‘1’.
- Value and Assigned bits both set.
- Under this node, search leaves variable the same.
Booleans in Search

- Search branches.
- The variable is assigned ‘1’.
- Value and Assigned bits both set.
- Under this node, search leaves variable the same.
Booleans in Search

- On backtrack here, “Assigned” set to 0.
- “Value” will be ignored until Boolean is next assigned.
Booleans in Search

Value

```
1
```

```
0
```

```
1
```

```
1
```

```
0
```

```
1
```

```
1
```
Booleans in Search

- Variable assigned 0.
- “Assigned” set to 1.
- “Value” set to 0.
Booleans in Search

- “Assigned” reset on backtrack
- “Value” left alone.
- Search continues...

Value

0

1

1

1

0
Results and Discussion

- Minion performs well on the problems we have tried.
- Where is the gain?
  - The search trees are the same.
  - Purely from architectural advances.
- Could we tune the others to narrow the gap?
  - Yes... but hardly “model and run”!
Present (and future)

- Minion now uses more from SAT solvers.
  - Watched Literals in Minion (CP06)
- Two major areas:
  - Learning (needs to be in Minion)
  - Modelling (Solver independent?)
- Still a magnitude slower than SAT solvers in nodes/second.

- [http://minion.sourceforge.net](http://minion.sourceforge.net)
Any Questions?

http://minion.sourceforge.net
Adding New Features

- Do you really need it?
  - Need to gain back the speed lost.
- Can the solver choose when it should be used?
  - Avoid giving users choices they don’t want.
- Can it be implemented without overhead when it is not used?
- Solver synthesis - exactly the Solver you want.
Conclusions

- Removing the ability to dynamically change search is beneficial, both in simplicity and speed.
- Paying close attention to data structures pays off.
- Often the simplest algorithm is the fastest!
  - Simple algorithms avoid the need for complex systems to speed them back up.
Booleans in Search

- Start of search.
- "Value" is set to a random value.
- "Assigned" = 0
Booleans in Search

- Search branches.
- The variable is assigned ‘1’.
- Value and Assigned bits both set.
On backtrack here, “Assigned” set to 0.

“Value” will be ignored until Boolean is next assigned.
Booleans in Search

Value

1  0

1  0
Booleans in Search

- Variable assigned 0.
- “Assigned” set to 1.
- “Value” set to 0.
Booleans in Search

- “Assigned” reset on backtrack
- “Value” left alone.
- Search continues...

Value

0

0

1

0

1

58
Minion Is Not...
Minion Is Not...

- Bug free.
Minion Is Not...

- Bug free.
- Able to hybrid with Linear Programming solvers.
Minion Is Not...

- Bug free.
- Able to hybrid with Linear Programming solvers.
- Able to add new variables or constraints during search.
Minion Is Not...

- Bug free.
- Able to hybrid with Linear Programming solvers.
- Able to add new variables or constraints during search.
- Able to change search strategy.
Minion Is Not...

- Bug free.
- Able to hybrid with Linear Programming solvers.
- Able to add new variables or constraints during search.
- Able to change search strategy.
- Designed to be easily extended and adjusted by users.
Bound Variables

- Store only the current upper and lower bounds.
- Very low memory requirements.
- Loss of information
  - \( \{1,3,5\} \rightarrow [1 .. 5] \rightarrow \{1,2,3,4,5\} \)
- Space / Time tradeoff.
Bound Variables

- Store only the current upper and lower bounds.
- Very low memory requirements.
- Loss of information
  - \( \{1,3,5\} \rightarrow [1..5] \rightarrow \{1,2,3,4,5\} \)
- Space / Time tradeoff.
- In some problems, can prove only bounds needed!
Discrete Variables

- Use a Boolean array for domain values.
- Store upper and lower bounds for optimisation reasons.
Discrete Variables

- Domain \{1,2,3,4,5,6\}
Discrete Variables

- Remove 3 from Domain.

```
 1 1 0 1 1 1 1
```

Lower  Upper

63
Discrete Variables

- Update **Lower** to 3

```
  1 1 0 1 1 1 1
```

Lower  Upper
Discrete Variables

- Update **Lower** to 3

- Optimisation:
  
  Boolean array only valid between **Lower** and **Upper**.

  - Gives fast bounds update.
Discrete Variables

- Have to move Lower until the first value in Domain is found.

1 1 0 1 1 1

Lower Upper
Minion

- Minion is a black-box constraint solver.
  - Not designed to be extended per problem.
  - New constraints and extensions have been added.
- Version for this paper about 3 man-months.
- Minion is up to 100x faster than ILOG Solver 5.3
  - Solver 6 faster on some problems, slower on others (don’t know why...)