GHOST: A stealth solver

Florian Richoux - Jean-François Baffier - Alberto Uriarte

October 4th, 2014
1. A short introduction to CSP/ COP
2. GHOST
3. Our models and results
4. Conclusion and discussion
1. A short introduction to CSP/ COP

2. GHOST

3. Our models and results

4. Conclusion and discussion
Constraint Satisfaction Problems (CSP)

CSP is a homogeneous framework to model combinatorial problems.

Constraint Optimization Problems (COP)

Same for optimization problems.

Examples

- AI,
- Graph problems,
- Database problems,
- Bio-informatics,
- ...
Definition of a CSP

A CSP is defined by a tuple \((V, D, C)\) such that:

\[
\text{CSP} = \begin{cases} 
    V : & \text{Set of variables.} \\
    D : & \text{Domain (set of values of variables).} \\
    C : & \text{Set of constraints.}
\end{cases}
\]
Intuitive definition

**Definition of a CSP**

A CSP is defined by a tuple \((V, D, C)\) such that:

\[
\text{CSP} = \begin{cases} 
V : & \text{Set of variables.} \\
D : & \text{Domain (set of values of variables).} \\
C : & \text{Set of constraints.}
\end{cases}
\]

- \(V = \{x, y, z\}\)
- \(D = \{0, 1\}\)
- \(C = \{=, \neq\}\)
A CSP is defined by a tuple \((V, D, C)\) such that:

\[
\text{CSP} = \begin{cases} 
V & \text{Set of variables.} \\
D & \text{Domain (set of values of variables).} \\
C & \text{Set of constraints.}
\end{cases}
\]

\[
\text{CSP} \implies V = \{x, y, z\} \\
D = \{0, 1\} \\
C = \{\,=, \neq\,\}
\]

CSP formula (aka CSP instance)

\((z = y) \land (y \neq x) \land (x = z)\)
1. A short introduction to CSP/ COP

2. GHOST

3. Our models and results

4. Conclusion and discussion
GHOST is a templated C++ solver aiming two users:

- The **casual user**, using GHOST to solve an already encoded problem.
- The **developer user**, who wants to use GHOST to implement and solve his/her problem.
Uses of GHOST

// Variables
vector< Unit > variables = { ... }

// Specific to the target selection problem
vector< UnitEnemy > enemies = { ... }

// Domain
TargetDomain domain( variables.size(), &enemies );

// Constraints
vector< shared_ptr<TargetConstraint> > constraints
{ make_shared<TargetConstraint>( &variables, &domain ) };

// Objective
shared_ptr<TargetObjective> objective = make_shared<MaxDamage>();

// Solver
Solver<Unit, TargetDomain, TargetConstraint>
  solver( &variables, &domain, constraints, objective );

// Call Solver::solve
solver.solve();
Uses of GHOST

// Define its own variables
class Unit : public Variable {...}

// Define its own domain
class TargetDomain : public Domain<Unit> {...}

// Define its own constraints
class TargetConstraint : public Constraints<Unit, TargetDomain> {...}

// Define its own objectives
class TargetObjective : public Objective<Unit, TargetDomain> {...}
Architecture of GHOST

Variables
Domain
Constraints

heuristics

SAT post-process

OPTI Objectives

OPTI post-process

Solution

Source code
https://github.com/richoux/GHOST
Outline

1. A short introduction to CSP/COP
2. GHOST
3. Our models and results
4. Conclusion and discussion
RTS problems

Problem families

Strategy
Tactic
Reactive Control

Example of problems

Wall-in
Target selection
Build order planning
RTS problems

Problem families
- Strategy
- Tactic
- Reactive Control

Example of problems
- Build order planning
- Wall-in
- Target selection
Reactive control:
Target selection problem
Target selection problem: Statement

- Range
- Cool down
- Damage efficiency

<table>
<thead>
<tr>
<th>Damage type</th>
<th>Concussive</th>
<th>Normal</th>
<th>Explosive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>100%</td>
<td>100%</td>
<td>50%</td>
</tr>
<tr>
<td>Medium</td>
<td>50%</td>
<td>100%</td>
<td>75%</td>
</tr>
<tr>
<td>Large</td>
<td>25%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Florian Richoux  GHOST 15 / 35
Target selection problem: Statement

- Range,
- Cooldown,
- Damage efficiency.
Target selection problem: Statement

- Range,
- Cooldown,
- Damage efficiency.

### Damage efficiency

<table>
<thead>
<tr>
<th></th>
<th>Concussive</th>
<th>Normal</th>
<th>Explosive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>size</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>100%</td>
<td>100%</td>
<td>50%</td>
</tr>
<tr>
<td>Medium</td>
<td>50%</td>
<td>100%</td>
<td>75%</td>
</tr>
<tr>
<td>Large</td>
<td>25%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Target selection problem: Model

**CSP/ COP model**

- \( V = (\text{Sub})\text{set of your units.} \)
- \( D = \text{A group of enemy units.} \)
- \( C = \text{Each living, ready-to-shoot unit must aim a living enemy unit within its range, if any.} \)
Target selection problem: Model

CSP/ COP model

- $V = (\text{Sub})\text{set of your units.}$
- $D = \text{A group of enemy units.}$
- $C = \text{Each living, ready-to-shoot unit must aim a living enemy unit within its range, if any.}$

COP objectives

- Max damage, where our group tries to deal as much damage as possible within the current frame.
- Max kill, where our group tries to kill as much enemy units as possible within the current frame.
Target selection problem: Results

Experiments

Mean of 100 runs for each objectives. SAT-Opti Timeouts: 1-3ms and 2-5ms.

<table>
<thead>
<tr>
<th>Objective</th>
<th>Win %</th>
<th># Draws</th>
<th># GHOST living units</th>
<th>GHOST HP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Damage</td>
<td>86.0</td>
<td>3</td>
<td>2.8</td>
<td>153.4</td>
</tr>
<tr>
<td>Max Kill</td>
<td>80.0</td>
<td>2</td>
<td>3.0</td>
<td>163.2</td>
</tr>
<tr>
<td>5ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Damage</td>
<td>91.0</td>
<td>1</td>
<td>2.8</td>
<td>149.6</td>
</tr>
<tr>
<td>Max Kill</td>
<td>82.0</td>
<td>3</td>
<td>3.0</td>
<td>169.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective</th>
<th># Opponent living units</th>
<th>Opponent HP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Damage</td>
<td>1.0</td>
<td>37.1</td>
</tr>
<tr>
<td>Max Kill</td>
<td>1.1</td>
<td>62.8</td>
</tr>
<tr>
<td>5ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Damage</td>
<td>1.1</td>
<td>58.7</td>
</tr>
<tr>
<td>Max Kill</td>
<td>1.0</td>
<td>37.2</td>
</tr>
</tbody>
</table>
Tactic:
Wall-in problem
Wall-in problem: Statement

Florian Richoux
Wall-in problem: Statement
Wall-in problem: Model

CSP/ COP model

- $V = \text{Set of your buildings.}$
- $D = \text{Positions around a choke (in a } 16 \times 12\text{ rectangle).}$
Wall-in problem: Model

Overlap
Wall-in problem: Model

Overlap

Buildable
Wall-in problem: Model

Overlap

Buildable

NoHoles
<table>
<thead>
<tr>
<th>Wall-in problem: Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overlap</strong></td>
</tr>
<tr>
<td>![Overlap Diagram]</td>
</tr>
<tr>
<td><strong>Buildable</strong></td>
</tr>
<tr>
<td>![Buildable Diagram]</td>
</tr>
<tr>
<td><strong>NoHoles</strong></td>
</tr>
<tr>
<td>![NoHoles Diagram]</td>
</tr>
<tr>
<td><strong>Starting Target Tile</strong></td>
</tr>
<tr>
<td>![Starting Target Tile Diagram]</td>
</tr>
</tbody>
</table>
Wall-in problem: Model

CSP Wall-in question
What is the position of my buildings to make a wall? (and what are the buildings I use?)
Wall-in problem: Model

CSP Wall-in question
What is the position of my buildings to make a wall? (and what are the buildings I use?)

COP Wall-in question
Optimize an objective function for my wall:
- Make a wall using the smallest number of buildings.
- Make a wall using the lowest technology.
- Make a wall with the fewest number of gaps.
100 tests over 48 chokes from 7 StarCraft classic maps (so 4800 tests).
100 tests over 48 chokes from 7 StarCraft classic maps (so 4800 tests).

### CSP results: making a wall with 20ms runs

<table>
<thead>
<tr>
<th>#Runs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Cost</td>
<td>1.59</td>
<td>0.58</td>
<td>0.33</td>
<td>0.18</td>
</tr>
<tr>
<td>Solved %</td>
<td>45.83%</td>
<td>71.10%</td>
<td>80.70%</td>
<td>88.45%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Runs</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Cost</td>
<td>0.13</td>
<td>0.03</td>
<td>0.01</td>
<td>0.0006</td>
</tr>
<tr>
<td>Solved %</td>
<td>91.58%</td>
<td>97.23%</td>
<td>99.18%</td>
<td>99.95%</td>
</tr>
</tbody>
</table>
Wall-in problem: Results

100 tests over 48 chokes from 7 StarCraft classic maps (so 4800 tests).

CSP results: making a wall with 20ms runs

<table>
<thead>
<tr>
<th>#Runs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Cost</td>
<td>1.59</td>
<td>0.58</td>
<td>0.33</td>
<td>0.18</td>
</tr>
<tr>
<td>Solved %</td>
<td>45.83%</td>
<td>71.10%</td>
<td>80.70%</td>
<td>88.45%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Runs</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Cost</td>
<td>0.13</td>
<td>0.03</td>
<td>0.01</td>
<td>0.0006</td>
</tr>
<tr>
<td>Solved %</td>
<td>91.58%</td>
<td>97.23%</td>
<td>99.18%</td>
<td>99.95%</td>
</tr>
</tbody>
</table>

COP results: optimize a wall within 150ms (with a SAT timeout of 20ms)

<table>
<thead>
<tr>
<th></th>
<th>Satisfaction run</th>
<th>Optimization run</th>
<th>Optimization run solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Buildings</td>
<td>3.12</td>
<td>2.65</td>
<td>96.83%</td>
</tr>
<tr>
<td>#Gaps</td>
<td>1.19</td>
<td>0.05</td>
<td>96.79%</td>
</tr>
<tr>
<td>Low tech</td>
<td>1.95</td>
<td>1.56</td>
<td>95.87%</td>
</tr>
</tbody>
</table>
Wall-in problem: Results

More results for #Gaps

Over 4800 tests:
- the solver found 4646 walls (96.8%) within 150ms,
- whose 4400 were perfect, i.e., gap-free. (94.7% of walls).
Strategy:
Build order planning problem
Build order problem: Statement

BO: A series of actions following a specific timing, in order to achieve a goal.

Dependencies
Actions may have dependencies.

Example
To build a Factory, you need first a Barracks.
BO can be modeled as a permutation problem: Huge search space reduction.
BO can be modeled as a **permutation** problem: Huge search space reduction.

**CSP/ COP model**

- \( V \) = All actions you need to reach your goal.
- \( D \) = Order of actions.
- \( C \) = Each dependency of an action \( \alpha \) must occurs before \( \alpha \).
Build order problem: Model

BO can be modeled as a permutation problem: Huge search space reduction.

CSP/ COP model

- V = All actions you need to reach your goal.
- D = Order of actions.
- C = Each dependency of an action $\alpha$ must occurs before $\alpha$.

COP objective

Minimize the makespan.
StarCraft simulator inside GHOST

We code a SC simulator to emulate resources gathering, units producing (including workers), supply capacity, constructions, etc.
Build order problem: Simulator

StarCraft simulator inside GHOST

We code a SC simulator to emulate resources gathering, units producing (including workers), supply capacity, constructions, etc.

- Time to go build something: 5t
- Time to go back gathering minerals after building something: 4t
- Time to go from the base to mineral patches to start mining: 5t
- Time for a worker to switch from mineral to gas: 5t
- Mineral gathering rate: 0.68 mineral per worker per t
- Gas gathering rate: 1.15 gas per worker per t
# Build order problem: Results

## Two test sets

1. 3,647 BOs from TeamLiquid, GosuGamers and ICCup.
2. 8 BOs from top Korean pro-gamers.

Mean of 10 runs for each BO.  

SAT-Opti Timeouts: 20-30ms.
Build order problem: Results

Two test sets

1. 3,647 BOs from TeamLiquid, GosuGamers and ICCup.
2. 8 BOs from top Korean pro-gamers.

Mean of 10 runs for each BO.

SAT-Opti Timeouts: 20-30ms.

<table>
<thead>
<tr>
<th>Games till 10,000 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match-up</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>All</td>
</tr>
<tr>
<td>PvP</td>
</tr>
<tr>
<td>PvT</td>
</tr>
<tr>
<td>PvZ</td>
</tr>
<tr>
<td>All pro</td>
</tr>
</tbody>
</table>
Two test sets

1. 3,647 BOs from TeamLiquid, GosuGamers and ICCup.
2. 8 BOs from top Korean pro-gamers.

Mean of 10 runs for each BO.

SAT-Opti Timeouts: 20-30ms.

<table>
<thead>
<tr>
<th>Match-up</th>
<th>Humans</th>
<th>GHOST</th>
<th>% solved</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>522.50</td>
<td>491.11</td>
<td>98.8</td>
<td>31.39</td>
</tr>
<tr>
<td>PvP</td>
<td>516.08</td>
<td>485.56</td>
<td>99.3</td>
<td>30.52</td>
</tr>
<tr>
<td>PvT</td>
<td>527.66</td>
<td>506.65</td>
<td>98.3</td>
<td>21.01</td>
</tr>
<tr>
<td>PvZ</td>
<td>515.81</td>
<td>458.23</td>
<td>99.7</td>
<td>57.58</td>
</tr>
<tr>
<td>All pro</td>
<td>506.38</td>
<td>480.88</td>
<td>100</td>
<td>25.50</td>
</tr>
</tbody>
</table>
Why searching for walls takes more time than build orders?

Wall-in CSP search space

39
42
104
Why searching for walls takes more time than build orders?

Wall-in CSP search space

Wall-in COP search space
1. A short introduction to CSP/ COP

2. GHOST

3. Our models and results

4. Conclusion and discussion
Conclusion

GHOST is:

▶ **General**: Can deal with any CSP/COP models without parameter tuning or code optimization needed.

▶ **Easy**: Both for casual and developer users.

▶ **Efficient**: We obtained good results on different RTS problems.

▶ **Open**: Contributions welcome!
Future works

- GHOST will be a library using BWAPI 4.
- Implement a pause/resume system for long computations.
- Parallelize the solver exploiting the available cores.
- CSP/COP cannot deal with uncertainty. Think to a new formalism.
Questions?

florian.richoux@univ-nantes.fr
@FloRicx
Complete Vs. Incomplete solvers

Complete solvers

[Diagram showing a tree structure with nodes labeled X, Y, and Z, and numbers 1, 2, 3 at various levels.]
Complete Vs. Incomplete solvers

Complete solvers

Meta-heuristics

(2,3,1)
Complete Vs. Incomplete solvers

**Complete solvers**

- Way faster
- Can deal with big size problems
- Not sure to find a solution
- Lost if no solutions

**Meta-heuristics**

- Florin Richoux
Local Search Meta-heuristics

- Search space
- Neighborhood
- Current config.
- Configurations
- Frozen config.
Local Search Meta-heuristics

- **Search space**
- **Neighborhood**
- **Current config.**
- **Configurations**
- **Frozen config.**
Main idea

Constraint \( x = y \) \[\Rightarrow \left( x = y \right) \wedge \left( y = z \right) \]
Adaptive Search [Codognet Diaz 2001]

Main idea

**Constraint** $x = y$

$\Rightarrow (x = y) \land (y = z)$

$C_1 \land C_2$

$\Rightarrow$

Assume we have $x = 3$, $y = 6$, $z = 1$

$\Rightarrow$

$\begin{align*}
\text{Error}(C_1) &= 3 \\
\text{Error}(C_2) &= 5
\end{align*}$

Error function which measures how much the constraint is satisfied $| x - y |$
Main idea

Constraint $x = y$

$\Rightarrow \left\{ \begin{array}{c} (x = y) \wedge (y = z) \\ C_1 \\ C_2 \end{array} \right.$

Error function which measures how much the constraint is satisfied

$|x - y|$

$\Rightarrow \left\{ \begin{array}{c} \text{Assume we have} \\ x = 3, \ y = 6, \ z = 1 \\ \text{Error}(C_1) = 3 \\ \text{Error}(C_2) = 5 \end{array} \right.$

Projection (usually addition of errors) on each variables

$\Rightarrow \left\{ \begin{array}{c} \text{Error}(x) = 3 \\ \text{Error}(y) = 3 + 5 = 8 \\ \text{Error}(z) = 5 \end{array} \right.$