Constraints on Set Variables for Constraint-based Local Search
(MSc thesis at Uppsala University)

Rodrigo Gumucio

SAIS yearly workshop 2012

May 14, Örebro, Sweden
(Video link from La Paz, Bolivia)
Motivating problem: The Social Golfer problem

- Imagine that in a golf club, \( g \cdot s \) players meet once a week in order to play golf in \( g \) groups of size \( s \).

- The challenge is to schedule a tournament over \( w \) weeks such that any two players meet in at most one week.

- An instance of this problem is denoted by
  
  \[ \text{golf-}g\cdot s\cdot w \]

  where

  - \( g \) is the number of groups
  - \( s \) is the size of each group
  - \( w \) is the number of weeks
  - \( g \cdot s \) is the total number of players

- The figure above shows a solution to \( \text{golf-}3\cdot3\cdot4 \).
The social golfer problem can be modelled with constraints:
- Each of the $g \cdot s$ players plays in exactly one group each week.
- All $g$ groups of a week are of the same size $s$.
- Any two players meet in at most one week.

It can be modelled with either integer or set variables, and hence with either integer or set constraints, respectively.

A set model is given by:
- A 2d matrix of set variables: $Players_{gw} \equiv$ the set of players meeting in group $g$ of week $w$.
- A new $\text{AtMost1}(Players)$ set constraint to ensure any two players meet at most once.

An integer model is given by:
- A 3d matrix of int variables: $Player_{gs} \equiv$ the player of slot $s$ in group $g$ of week $w$.
- A $\text{SocialTournament}(Player)$ integer constraint to ensure any two players meet at most once.
Consider again the golf-3·3·4 instance:
Model with set variables:

- $Players_{gw}$ has $3 \cdot 4$ set vars.
- A single set constraint: $\text{ATMOST}1(Players)$.
- No need to introduce a concept outside the problem formulation.

Model with int variables:

- $Player_{gsw}$ has $3 \cdot 3 \cdot 4$ int vars.
- A single integer constraint: $\text{SOCIALTOURNAMENT}(Player)$.
- Needs to introduce the concept of player slot within a group.
Constraint-based local search

- Constraint-based local search is a useful technique to find solutions to constraint problems using stochastic local search.
- It trades the completeness and quality of a systematic search technique (like constraint programming) for speed and scalability.

![Diagram showing the application of constraint-based local search with violation count]

violation = 13
Constraint-based local search

- Constraint-based local search is a useful technique to find solutions to constraint problems using stochastic local search.
- It trades the completeness and quality of a systematic search technique (like constraint programming) for speed and scalability.
Constraint-based local search

- Constraint-based local search is a useful technique to find solutions to constraint problems using stochastic local search.
- It trades the completeness and quality of a systematic search technique (like constraint programming) for speed and scalability.

\[
\begin{array}{cccc}
1 & 2 & 7 & 4 \\
4 & 5 & 6 & 3 \\
3 & 8 & 9 & 1 \\
1 & 4 & 7 & 2 \\
2 & 5 & 8 & 5 \\
5 & 6 & 9 & 1 \\
6 & 1 & 5 & 2 \\
2 & 4 & 9 & 6 \\
9 & 3 & 5 & 7 \\
1 & 5 & 9 & 2 \\
2 & 6 & 7 & 3 \\
3 & 4 & 8 & 1 \\
\end{array}
\]

violation = 8
Constraint-based local search is a useful technique to find solutions to constraint problems using stochastic local search. It trades the completeness and quality of a systematic search technique (like constraint programming) for speed and scalability.
Constraint-based local search

Constraint-based local search is a useful technique to find solutions to constraint problems using stochastic local search. It trades the completeness and quality of a systematic search technique (like constraint programming) for speed and scalability.

To find solutions using local search:
1. (Randomly) initialise all the variables.
2. Re-assign a few variables: local move.
3. If the new assignment is not good enough, then go to step 2.
Constraints in constraint-based local search

- Constraints are used mainly to:
  - Guide the local search to promising regions in the search space.
  - Determine when a given assignment is regarded as a solution.

- Constraints are implemented by a set of functions:
  - Violation functions help to select a promising variable (of a promising constraint) to re-assign in a move.
  - Differentiation functions help to make a move in a good direction for a constraint or variable.

The violation functions for ATMOST1 basically count the number of times two players meet after the first allowed time.

ATMOST1 needs a differentiation function for swap moves.

These functions must be very fast.
Main contributions

- Solid evidence that, using constraint-based local search, solving problems modelled with sets has the following advantages:
  - It can reduce the solution time.
  - It can even be a necessity in terms of memory.
- The design and implementation of an extension of a constraint-based local search solver (namely Comet) by:
  - Adding the notion of set constraint.
  - Providing the notion of set constraint system (i.e., a constraint combinator for constraints on set variables).
Comet’s local search architecture

- Comet is a language and tool for modelling and solving constraint problems, using systematic or local search.
- It represents the state-of-the art in constraint-based local search.

- Unfortunately, it does not support set constraints for local search.
- Fortunately, it does support user-defined invariants on set incremental variables.

An extension is possible: The architecture is open, and set constraints can be built on top of set invariants!

Two new features are needed at the constraint layer:
- Support for user-defined set constraints.
- Support for a mechanism to combine such constraints, that is at least one constraint combinator.
Extension of Comet’s local search

The extension consists of:

- A `SetConstraint<LS>` interface together with an abstract class that provides a mechanism to define set constraints.
- A set constraint system that provides a mechanism to combine set constraints.
- To define a set constraint, extend and specialise the abstract class (named `UserSetConstraint<LS>`).
- The provided constraint combinator (i.e., the set constraint system) could be done only through a tricky implementation.
- The full source code is in my MSc thesis.

```java
interface SetConstraint<LS> {
    ... 
    var{set{int}}[] getSetVariables();
    var{int} violations();
    var{int} violations(var{set{int}} s);
    int getSwapDelta(var{set{int}} s, int u, int v, var{set{int}} t);
    ... 
}
```
My ATMOST1 constraint is the set version of the SOCIALTOURNAMENT integer constraint of [Dynamic Decision Technologies, 2010].

The essence of the integer version: count the number of times players $a$ and $b$ meet, denote it by $\#(a, b)$, and maintain it incrementally.

The same can be done with set variables: keep the set of groups where $a$ and $b$ meet, denote it by $m(a, b)$, and maintain it incrementally.

Note: $|m(a, b)| = \#(a, b)$.

The violation and differentiation functions are based on these values.

The constraint is satisfied whenever $|m(a, b)| \leq 1$ for all $a$ and $b$. 
The Social Golfer problem: search

The tabu search algorithm of [Dynamic Decision Technologies, 2010] (based on [Dotú and Van Hentenryck, 2007]) is adapted for the set approach:

```java
while (violations > 0 && (System.getCPUPTime() - t0) < timeout)
    selectMin(w in Weeks,
        g1 in Groups, s1 in Slots: conflict[w, g1, s1] > 0,
        g2 in Groups: g2 != g1, s2 in Slots,
        delta = tourn.getSwapDelta(golfer[w, g1, s1], golfer[w, g2, s2]) : tabu[w, golfer[w, g1, s1], golfer[w, g2, s2]] < it || violations + delta < best) (delta) {
        golfer[w, g1, s1] := golfer[w, g2, s2]; ...
    }
```

```java
while (violations > 0 && (System.getCPUPTime() - t0) < timeout)
    selectMin(w in Weeks,
        g1 in violatedGroups[w], g2 in Groups: g2 != g1,
        s1 in golfersInConflict[w, g1],
        s2 in group[w, g2],
        delta = tourn.getSwapDelta(group[w, g1], s1, s2, group[w, g2]):
        ((tabu[w, s1, s2] < it) || violations + delta < best)) (delta) {
        group[w, g1].delete(s1); group[w, g2].insert(s1);
        group[w, g2].delete(s2); group[w, g1].insert(s2); ...
    }
```
## The Social Golfer problem: results over 25 runs

### Run time (milliseconds)

<table>
<thead>
<tr>
<th>instance g-s-w</th>
<th>integer model</th>
<th>set model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n average</td>
<td>min.</td>
</tr>
<tr>
<td>6-3-8</td>
<td>25 166367.08</td>
<td>7447</td>
</tr>
<tr>
<td>6-4-6</td>
<td>25 72048.84</td>
<td>2958</td>
</tr>
<tr>
<td>6-5-6</td>
<td>0 &gt; timeout</td>
<td>-</td>
</tr>
<tr>
<td>7-3-9</td>
<td>25 7847.88</td>
<td>780</td>
</tr>
<tr>
<td>7-4-7</td>
<td>24 352799.88</td>
<td>1907</td>
</tr>
<tr>
<td>7-6-4</td>
<td>25 85.80</td>
<td>65</td>
</tr>
<tr>
<td>8-3-10</td>
<td>25 1100.08</td>
<td>444</td>
</tr>
<tr>
<td>8-4-8</td>
<td>25 694801.16</td>
<td>17162</td>
</tr>
<tr>
<td>8-5-6</td>
<td>25 357.84</td>
<td>166</td>
</tr>
<tr>
<td>8-6-5</td>
<td>25 2622.16</td>
<td>678</td>
</tr>
<tr>
<td>8-7-4</td>
<td>25 443.28</td>
<td>197</td>
</tr>
<tr>
<td>8-8-5</td>
<td>0 &gt; timeout</td>
<td>-</td>
</tr>
<tr>
<td>9-3-12</td>
<td>5 815324.00</td>
<td>28589</td>
</tr>
<tr>
<td>9-4-9</td>
<td>25 347040.68</td>
<td>111125</td>
</tr>
<tr>
<td>9-6-6</td>
<td>25 245999.64</td>
<td>20040</td>
</tr>
<tr>
<td>9-7-5</td>
<td>25 23233.52</td>
<td>20090</td>
</tr>
<tr>
<td>9-8-4</td>
<td>25 2325.52</td>
<td>1857</td>
</tr>
<tr>
<td>9-9-5</td>
<td>3 496988.00</td>
<td>44865</td>
</tr>
<tr>
<td>10-3-13</td>
<td>25 35562.04</td>
<td>4894</td>
</tr>
<tr>
<td>10-4-10</td>
<td>25 960130.00</td>
<td>42674</td>
</tr>
<tr>
<td>10-5-8</td>
<td>25 125212.08</td>
<td>9853</td>
</tr>
<tr>
<td>10-6-7</td>
<td>0 &gt; timeout</td>
<td>-</td>
</tr>
<tr>
<td>10-7-5</td>
<td>25 435.32</td>
<td>429</td>
</tr>
<tr>
<td>10-8-5</td>
<td>2 950746.50</td>
<td>616029</td>
</tr>
<tr>
<td>10-9-4</td>
<td>25 26255.20</td>
<td>1860</td>
</tr>
</tbody>
</table>
The Social Golfer problem: solved instances

<table>
<thead>
<tr>
<th></th>
<th>s = 3</th>
<th>s = 4</th>
<th>s = 5</th>
<th>s = 6</th>
<th>s = 7</th>
<th>s = 8</th>
<th>s = 9</th>
<th>s = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
<td>w δ(w)</td>
</tr>
<tr>
<td>6</td>
<td>8 0</td>
<td>6 0</td>
<td>6 0</td>
<td>3 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>9 0</td>
<td>7 0</td>
<td>6 -1</td>
<td>4 -1</td>
<td>8 0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>10 0</td>
<td>8 0</td>
<td>6 0</td>
<td>5 -3</td>
<td>4 0</td>
<td>5 -4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>12 0</td>
<td>9 0</td>
<td>7 0</td>
<td>6 -3</td>
<td>5 0</td>
<td>4 0</td>
<td>5 -5</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>13 0</td>
<td>10 0</td>
<td>8 0</td>
<td>7 +1</td>
<td>5 0</td>
<td>5 +1</td>
<td>4 0</td>
<td>3 0</td>
</tr>
</tbody>
</table>

The $\delta(w) \geq 0$ values are relative [Dotú and Van Hentenryck, 2007], which uses the same meta-heuristic; the negative ones are relative the state of the art.

Two instances not solved by [Dotú and Van Hentenryck, 2007] were solved (as by [Cotta et al., 2006] and [Harvey and Winterer, 2005]):

- golf-10-6-7
- golf-10-8-5
Schur’s problem

- A set $T$ of integers is sum-free if $a, b \in T \rightarrow a + b \not\in T$. Example: $\{1, 3, 5\}$. Counterexamples: $\{1, 3, 4\}$ and $\{1, 2\}$.

- Schur’s problem, denoted schur-k-n, is about finding a partition of the set $\{1, \ldots, n\}$ into $k$ sum-free sets. Let $S(k)$ denote the largest such $n$.

- $S(1) = 1$, $S(2) = 4$, $S(3) = 13$, $S(4) = 44$, but $S(5)$ is unknown. Example: $S(2) = 4$ as $\{1, 2, 3, 4\} = \{1, 4\} \cup \{2, 3\}$.

- Modelling this problem with integer variables will not scale: A set model requires $k$ SUM-FREE constraints, while an integer model requires $O(k \cdot n^2)$ SUM-FREE constraints. To solve this problem with constraint-based local search:
  - A SUM-FREE set constraint is needed.
  - A tabu-search meta-heuristic is used for simplicity.
### Schur’s problem: results

#### GC memory usage (KB)

<table>
<thead>
<tr>
<th>instance</th>
<th>3-13</th>
<th>4-37</th>
<th>4-38</th>
<th>4-39</th>
<th>4-40</th>
<th>4-41</th>
<th>4-42</th>
<th>4-43</th>
<th>4-44</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>21</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>31015.56</td>
<td>244420.28</td>
<td>258275.08</td>
<td>272206.86</td>
<td>277904.29</td>
<td>286657.71</td>
<td>327303.50</td>
<td>300813.00</td>
<td>396890.00</td>
</tr>
<tr>
<td>min.</td>
<td>21224</td>
<td>161948</td>
<td>178221</td>
<td>200513</td>
<td>200637</td>
<td>210572</td>
<td>255207</td>
<td>239060</td>
<td>396890</td>
</tr>
<tr>
<td>max.</td>
<td>37010</td>
<td>304502</td>
<td>319811</td>
<td>336613</td>
<td>331583</td>
<td>346756</td>
<td>364786</td>
<td>362566</td>
<td>396890</td>
</tr>
<tr>
<td>std.dev.</td>
<td>5209.47</td>
<td>46818.22</td>
<td>45790.06</td>
<td>49212.87</td>
<td>50458.18</td>
<td>56444.48</td>
<td>45433.86</td>
<td>87331.93</td>
<td>-</td>
</tr>
</tbody>
</table>

#### VM memory usage (KB)

<table>
<thead>
<tr>
<th>instance</th>
<th>3-13</th>
<th>4-37</th>
<th>4-38</th>
<th>4-39</th>
<th>4-40</th>
<th>4-41</th>
<th>4-42</th>
<th>4-43</th>
<th>4-44</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>21</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>62914.56</td>
<td>492830.72</td>
<td>513802.24</td>
<td>524288.00</td>
<td>524288.00</td>
<td>524288.00</td>
<td>524288.00</td>
<td>524288.00</td>
<td>524288.00</td>
</tr>
<tr>
<td>min.</td>
<td>32768</td>
<td>262144</td>
<td>262144</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
</tr>
<tr>
<td>max.</td>
<td>65536</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
</tr>
<tr>
<td>std.dev.</td>
<td>9073.05</td>
<td>86943.33</td>
<td>52428.80</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

#### set model

<table>
<thead>
<tr>
<th>instance</th>
<th>3-13</th>
<th>4-37</th>
<th>4-38</th>
<th>4-39</th>
<th>4-40</th>
<th>4-41</th>
<th>4-42</th>
<th>4-43</th>
<th>4-44</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>25</td>
<td>23</td>
<td>18</td>
<td>17</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>18371</td>
<td>18295</td>
<td>18039</td>
<td>18777</td>
<td>17745</td>
<td>19155</td>
<td>17817</td>
<td>17401</td>
<td>18214</td>
</tr>
<tr>
<td>min.</td>
<td>16955</td>
<td>17020</td>
<td>17078</td>
<td>17622</td>
<td>17275</td>
<td>18902</td>
<td>17817</td>
<td>17401</td>
<td>18214</td>
</tr>
<tr>
<td>max.</td>
<td>18550</td>
<td>19334</td>
<td>19690</td>
<td>19685</td>
<td>18714</td>
<td>19408</td>
<td>17817</td>
<td>17401</td>
<td>18214</td>
</tr>
<tr>
<td>std.dev.</td>
<td>487.42</td>
<td>559.48</td>
<td>846.86</td>
<td>493.68</td>
<td>582.44</td>
<td>357.80</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### integer model

#### set model

<table>
<thead>
<tr>
<th>instance</th>
<th>3-13</th>
<th>4-37</th>
<th>4-38</th>
<th>4-39</th>
<th>4-40</th>
<th>4-41</th>
<th>4-42</th>
<th>4-43</th>
<th>4-44</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>25</td>
<td>23</td>
<td>18</td>
<td>17</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
</tr>
<tr>
<td>min.</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
</tr>
<tr>
<td>max.</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
</tr>
<tr>
<td>std.dev.</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Problem instances require less memory using the set model.

Both the integer and set models find the best solutions to the closed instances, that is Schur numbers up to $S(4) = 44$.

Unfortunately, the advantage in memory consumption is not enough to find $S(5)$, which thus remains open.
A similar experiment was done for Steiner triple systems. As expected, the set approach required much less memory. Instances much larger than with an integer model are solved. Check my MSc thesis for details.
I have demonstrated that set variables for constraint-based local search are not only a convenience for faster & higher-level modelling. Set variables, and hence set constraints, can be necessary because solutions to problem instances with integer variables:

- may not be found otherwise,
- would not fit into memory, or
- take much more time to be solved.

I have also contributed an extension of the constraint-based local search back-end of Comet to support set constraints.

My MSc thesis is at http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-159180.

