Constructing Constraint Solvers Using Monte Carlo Tree Search

Arūnas Prokopas

Alan Frisch², Ian P. Gent¹, Christopher Jefferson¹, Lars Kotthoff³, Ian Miguel¹, Peter Nightingale¹

¹University of St Andrews. {ap637,lan.Gent,caj21,ijm,pwn1}@st-andrews.ac.uk
²University of York alan.frisch@york.ac.uk; ³University College Cork larsko@4c.ucc.ie
Constraint Programming

A programming paradigm, where you specify the properties of the desired solution and not the steps that are required to reach it.

Constraint Problems are specified as:
- Variables and Variable Domains
- Constraints between these Variables.
Constraint Solvers

- Can handle wide range of problems
- Complex and sophisticated
- Often require extensive tuning by an expert
Dominion

- Constructs solvers for specific problems
- Hard to choose the best components
Monte Carlo Tree Search

- Relatively new (c. 2006)
- Best-first search algorithm
- Does not rely on domain knowledge
- Can handle large search spaces
- Relies on simulations to sample data set
MCTS repeats 4 basic steps:

1. Selection - select a sequence of decisions that are already explored.
2. Expansion – add another decision to the current sequence
3. Simulation – test if the decision is beneficial (leads to better outcomes).
4. Back-propagation – update the value of all decisions that lead to the current outcome.
MCTS in Dominion

Two data-structures are maintained: PartialTree and OpenNodes list.

1. Most interesting node from OpenNodes is selected
2. All choices for that node are expanded
3. Simulations are run for each of them
4. Best of the choices is added to the Partial Tree
## Preliminary Results

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Worst</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominion</td>
<td>Worst</td>
<td>Best</td>
<td>Worst</td>
<td>Best</td>
</tr>
<tr>
<td>Magic Squares</td>
<td>9999.9</td>
<td>9999.9</td>
<td>9999.9</td>
<td>9999.9</td>
</tr>
<tr>
<td>n-Queens</td>
<td>1044.5</td>
<td>9999.9</td>
<td>1044.1</td>
<td>9999.9</td>
</tr>
<tr>
<td>Disc.Tmography</td>
<td>0.7058</td>
<td>100.00</td>
<td>0.1935</td>
<td>95.005</td>
</tr>
<tr>
<td>Knapsack</td>
<td>0.0127</td>
<td>5012.8</td>
<td>0.0178</td>
<td>4022.7</td>
</tr>
<tr>
<td>Graph Coloring</td>
<td>0.2578</td>
<td>251.23</td>
<td>0.0428</td>
<td>0.7318</td>
</tr>
<tr>
<td>Sonet</td>
<td>0.4545</td>
<td>4.2712</td>
<td>0.1010</td>
<td>0.4246</td>
</tr>
<tr>
<td>Golomb Ruler</td>
<td>0.2777</td>
<td>0.8569</td>
<td>0.0760</td>
<td>0.2777</td>
</tr>
</tbody>
</table>

Ratios between Dominion and Minion run times for best and worst instances.
Future Work

- Counter bad decisions early on in the search tree
- Establishing component relations
- Use machine learning to predict good choices