Solving constraint satisfaction problems using arc consistency and path consistency techniques

December 30, 2019

1 Local consistency algorithms

1.1 Arc consistency

For the arc consistency we use AC1 algorithm.

Algorithm 1 AC1 algorithm

1: **procedure** REVISE(v_i, v_j) ▷ a network with two variables v_i,v_j, domains D_i and D_j, and constraint R_{ij}
2: **for** each $a_i \in D_i$ **do**
3: **if** there is no $a_j \in D_j$ with $(a_i, a_j) \in R_{ij}$ **then**
4: remove a_i from D_i

procedure AC1(N) ▷ a constraint network $N =< V, D, C >$
2: **repeat**
for each arc (v_i, v_j) with $R_{ij} \in C$ **do**
4: Revise(v_i, v_j)
Revise(v_j, v_i)
6: **until** no domain is changed

1.2 Path consistency

For the path consistency we use PC1 algorithm.

Algorithm 2 PC1 algorithm

1: **procedure** PATH_REVISE($\{v_i, v_j\}, v_k$) ▷ a binary network $< V, D, C >$ with variables v_i,v_j,v_k
2: **for** each pair $(a_i, a_j) \in R_{ij}$ **do**
3: **if** there is no $a_k \in D_k$ such that $(a_i, a_k) \in R_{ik}$ and $(a_j, a_k) \in R_{jk}$ **then**
4: remove (a_i, a_j) from R_{ij}

procedure PC1(N) ▷ a constraint network $N =< V, D, C >$
2: **repeat**
for each (ordered) triple of variables v_i, v_j, v_k **do**
4: Path_Revise(v_i, v_j, v_k)
until no constraint is changed

2 Python implementation

For the implementation we use modified **python_constraint** module.

2.1 Installation

- Clone the repository
  `git clone https://github.com/asergeenko/python-constraint.git`
- Build and setup the module
  `python setup.py build`
  `python setup.py install`

2.2 Module overview

Python implementations of the AC1 and PC1 algorithms are located in `consistency.py` file. There are four methods:

arc_revise*(var1,var2,problem,constraints_for_variable)*
#Implements REVISE procedure from AC1 algorithm

Input:
var1 and *var2* - variables to revise,
problem - **python_constraint**'s *Problem* instance,
constraints_for_variable - dictionary with constraints for variable *var1*
(returned by *getArcs* function from **constraint** module).

Output:
Returns *True* if arc *(var1,var2)* is not arc-consistent
and some values are removed from the domain.

ac1*(arcs, problem)*
#Implements AC1 procedure

Input:
arcs - arcs (returned by *getArcs* function from **constraint** module),
problem - **python_constraint**'s *Problem* instance

path_revise*(var1, var2, var3,problem)*
#Implements PATH_REVISE procedure from PC1 algorithm

Input:
var1,*var2.var3* - variables to revise,
problem - **python_constraint**'s *Problem* instance

Output:
Returns *True* if the pair *(var1,var2)* is not path-consistent relative to *var3*
and some values are removed.

pc1*(problem)*

#Implements PC1 procedure

Input:
problem - **python_constraint**'s *Problem* instance

3 Problem solving examples

3.1 Examples overview

These algorithms are suitable for CSP problems with binary constraints. So we have *N-Queens problem* (can be found in `examples/queens/queens.py`) and *Map coloring problem* (can be found in `examples/map_coloring/map_coloring.py`). N-queens example is the part of the original **python_constraint** module and has just one parameter:

```
size = 8 # size of the chessboard
```

Map coloring example is new and has the following parameters:

```
countries = ['A','B','C'] # countries
num_colors = 2 # number of colors
neighbors = ['AB','BC','CA'] # countries that border each other
```

3.2 Command-line parameters

```
python queens.py [-h] [-s][1] [-ac] [-pc]
python map_coloring.py [-h] [-ac] [-pc]
```

`-h` - show help message
`-s` - show solutions on the chessboard (doesn't show by default)
`-ac` - use arc consistency algorithm (not used by default)
`-pc` - use path consistency algorithm (not used by default)

[1] This parameter is used only with `queens.py` example.