Grundlagen der Wissensverarbeitung
Computer Science
Universität Hamburg

Tutorial 7: Constraint Satisfaction

Rafael Ruz Gómez
Miguel Robles Urquiza

6 de diciembre de 2017

Universität Hamburg

Exercise 7.1

Formalize this riddle in the form of a constraint network, with the constraints being reasonably small. (I.e. writing a single constraint is not a good solution!) In the following pattern each letter stands for a digit so that the resulting sum is correct.

```
    S E N D
+ M O R E
=======
M O N E Y
```

Let's call S_i the sum done in the last step. Then the constraints are:

- Different letter have different values: $S \neq E, E \neq N, N \neq D...$
- $(D + E)\,\%10 == Y \qquad (S_0 = (D + E))$
- $(((S_0 - S_0)\,\%10)/10 + N + R)\,\%10 == E \qquad (S_1 = ((S_0 - S_0)\,\%10)/10 + N + R)$
- $(((S_1 - S_1)\,\%10)/10 + E + O)\,\%10 == N \qquad (S_2 = ((S_1 - S_1)\,\%10)/10 + E + O)$
- $(((S_2 - S_2)\,\%10)/10 + S + M)\,\%10 == O \qquad (S_3 = ((S_2 - S_2)\,\%10)/10 + S + M)$
- $(((S_3 - S_3)\,\%10)/10 == M$

Manual constraint solving.

Crossword puzzles are often used in newspapers because they provide joy in solving semi-complex problems by combining logics and human experience. For the crossword above we want to find 6 words of length 3 that fit into the 3x3 table in a way that 3 words can be read horizontal from left to right and 3 words can be read vertically from top to bottom. Choose the words from the following list:

add, ado, age, ago, aid, ail, aim, air,
and, any, ape, apt, arc, are, ark, arm,
art, ash, ask, auk, awe, awl, aye, bad,
bag, ban, bat, bee, boa, ear, eel, eft,
far, fat, fit, lee, oaf, rat, tar, tie.

First, try to solve the problem without any formal methods or tools. How do you approach this problem as a human? (It is not necessary to give a full solution to the problem at this point, but you should report on the strategies you employ as a human and the problems you encounter.)

Basically, the constraint is that the letters where a pair of words intersect must be the same.
So we take the first word and see if the next word on the list mach with the constrain. If it works, we add the world correctly and continue checking the constrain from the begginin, but without reppeating words that we have already use.
If not, we discard that word from this attemp and we continue checking.
We tried but we have not found any solution manually for this text, there is always 1 word left.

Solve the problem by hand using domain consistency as a first step and as a second step the arc consistency. Document this process thoroughly.

- Domain consistency:

  We can solve the problem by treating it as a CSP. The set of variables is S = "X,Y,Z", that represents X as the first letter of the word, Y as the second and Z as the last. Dom(X) = Dom(Y) = Dom(Z) (I.e All the letters of the alphabet)

  Now, we take the first letter of the alphabet and check if it's appears in any of the words given. If not, we delete that letter from the domain. Taken all the letters of the alphabet, we have now a domain consistent.

- Arc consistency:

  A good way to begin is seeing which letter appearing more. We take one of this words, let's think the word is "XYZ".

  After that, it's time to check if there is other word that starts with "X". If that happens, we should put the word in the first row or in the first colum and add the other word in the correct position.
  If don't, we shoul check if there is other word that starts with "Y". If that happens, we should put the word in the first row or in the first colum and add the other word in the correct position.
  If don't, we should check if there is other word that starts with "Z". If that happens, we should put the word in the first row or in the first colum and add the other word in the correct position.
  If don't, we should check if there is other word that has a "X" in second position. If that happens, we should put the word in the second row or second coulm and add the other word in the correct position.
  If don't, we should check if there is other word that has a "Y" in second position. If that happens, we should put the word in the second row or second coulm and add the other word in the correct position.
  If don't, we should check if there is other word that has a "Z" in second position. If thath happens, we should put the word in the second row or second coulm.
  If don't, we should check if there is other word that has a "X" in third position. If thath happens, we should put the word in the third row or third coulm and add the other word in the correct position.
  If don't, we should check if there is other word that has a "Y" in third position. If thath happens, we should put the word in the third row or third coulm and add the other word in the correct position.
  If don't, we should check if there is other word that has a "Z" in third position. If thath happens, we should put the word in the third row or third coulm and add the other word in the correct position.

• **Implement the arc consistency algorithm (found in sect. 4.5 of Poole and Mackworth (2010)) along with a suitable representation of the problem to solve this puzzle.**

```
void GraphArcConsistency( vector< vector<string> > &variables ){
  list<int> to_do;

  to_do.push_back(row0);
  to_do.push_back(row1);
  to_do.push_back(row2);
  to_do.push_back(col0);
  to_do.push_back(col1);
  to_do.push_back(col2);
```

First of all, we create the "to do"list and we instert all the variables on it.

```
  while( !to_do.empty() ){
    int actualVariable = to_do.front();
    to_do.pop_front();

    bool deleteWord = true;
    bool deleteWordFinal = false;
    bool insertedDependencies = false;
```

Then the main loop is while we have something to do(i.e. any arc to check) then we remove it from the "to do"list and initialize some booleans for later.

```
if( actualVariable < 3 ){     // If we are checking a row
  for( std::vector<string>::iterator it = variables[actualVariable].begin() ; it != variables[actualVariable].end() ; )
    deleteWordFinal = false;
    for (size_t i = col0; i <= col2; i++) {    // We check the 3 columns
      deleteWord = true;
      for( vector<string>::iterator it2 = variables[i].begin() ; it2 != variables[i].end() ; it2++ ){
        if( (*it)[i-3] == (*it2)[actualVariable] ){
          deleteWord = false;
        }
      }
      if( deleteWord )
        deleteWordFinal = true;
    }
    if( deleteWordFinal ){
      it = variables[actualVariable].erase(it);
      if( !insertedDependencies ){
        to_do.push_back(col0);
        to_do.push_back(col1);
        to_do.push_back(col2);
        to_do.sort();
        to_do.unique();
        insertedDependencies = true;
      }
    }
    else it++;
  }
}
```

After that we check if we are processing an arc involving a row word or a column word. If it's the row 'n', we check the position 'n' of the words we can put on the columns.

If there's no word for any of the columns that satisfies the constraint then we delete the word from the domain of the row 'n' and we add the columns connected to that constraint (in this case the 3 columns) to check them later.

```
    else{              // If we are checking a column
      for( vector<string>::iterator it = variables[actualVariable].begin() ; it != variables[actualVariable].end() ; ){
        deleteWordFinal = false;
        for (size_t i = row0; i <= row2; i++) {    // We check the 3 rows
          deleteWord = true;
          for( vector<string>::iterator it2 = variables[i].begin() ; it2 != variables[i].end() ; it2++ ){
            if( (*it)[i] == (*it2)[actualVariable-3] ){
              deleteWord = false;
            }
          }
          if( deleteWord )
            deleteWordFinal = true;
        }
        if( deleteWordFinal ){
          it = variables[actualVariable].erase(it);
          if( !insertedDependencies ){
            to_do.push_back(row0);
            to_do.push_back(row1);
            to_do.push_back(row2);
            to_do.sort();
            to_do.unique();
            insertedDependencies = true;
          }
        }
        else it++;
      }
    }
  }
}
```

If we are processing an arc involving a column word, we do the same but checking and adding the rows connected.