Documentation for the RandomCsp library — version 1.8.x

J.I. van Hemert
jvhemert@cs.leidenuniv.nl

February 16, 2004

1 Introduction

RandomCsp is a program and a C++ library for generating random constraint satisfaction problems (CSPs). This file contains information about how to use the program and its output and how to make use of the library in your own code.

The RandomCsp project has three goals. To facilitate as a suit of programs to create and analyse randomly created binary constraint satisfaction problems, for those who do not like to work with libraries for any reason a suite of programs is included that provides a simple interface to the library. With these programs it is easy to create a set of random problem instances, to verify solutions and to analyse instances on a number of features.

To be used as a library to implement and test new or existing constraint satisfaction solving techniques, the library part has an extended documentation of its class hierarchy that helps new developers on their way creating new tools or solving techniques for binary constraint satisfaction. At the same time the library allows testing using a number of theoretical models.

To be freely available for anyone, the library and programs that come with it are all licensed under the GNU General Public License, ensuring free use forever. You can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but **without any warranty**; without even the implied warranty of **merchantability** or **fitness for a particular purpose**. See the GNU General Public License for more details. You should have received a copy (see the file `COPYING`) of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

This program has been developed and tested on a Linux machine using a Debian distribution. It has been compiled and tested with the GNU C++ compiler.

```
00.ma_v20_d20_c_t0.5_c0.5_s1.csp
-- -- --- --- - ---- ---- --
|   |  |    |  |  |     |    |
1   2  3    4  5  6     7    8
```

Figure 1: Example filename output by RandomCsp

2 Using RandomCsp programs

After installing (see the file '`INSTALL`') you can have a quick look at the output format by running 'randomcsp 'without any parameters. If you would like to change a parameter in the algorithm used for generating the CSPs start the program with '-h' as argument to get a list of possible options.

RandomCsp will default to generating a filename from the parameters used in generating the CSP. Below is explained how to interpret the filename, the numbers correspond to those in Figure 1

1. a follow-up number, each instance generated in one go gets a unique number
2. the model that was used to generate this instance
3. an integer determining the amount of variables in the CSP
4. an integer determining the size of the domain of each variable *or* determining the size of the set from which a domain will be randomly chosen for each variable
5. a boolean that chooses between the two options from 3. 'c' Is used for a constant domain size for each variable and 'r' for a randomly chosen domain size for each variable
6. a double (real) 0 or higher, that represents the tightness of the constraints. This determines the amount of conflicts in the constraints
7. a double (real) between 0 and 1, that represents the connectivity in the network of connected variables. This determines the amount of arcs between variables, not that this is only suitable for models A–D that use four parameters. In case of model G this represents the p parameter of the geometric distribution
8. an integer used as the seed for the random generator

Note that this output can differ when using different models or special options. For instance, Model E (`-me`) only uses the tightness parameter. Also when using `-T` or `-S` the program will test if the instance is solvable and add either solvable or unsolvable to the filename.

RandomCsp outputs a file that starts with an identifying line reading something similar to

```
CSP generated with RandomCsp 1 7 0
```

The CSP entry is used to determine if the file is an actual CSP suitable for reading. The rest of the line are ignored and then we get the version used for generating this file. It is not used at this time. The next line determines which format has been used in the saving process. Two options currently exist: listformat and matrixformat.

The loading function of the library automatically finds out which format a file uses and then loads it into its internal structure. This makes it completely transparent to the user and developer.

listformat We start with a keyword that denotes what the meaning of the entries are, this is either `savingconflicts` or `savingnonconflicts`. Then we have one number on a line denoting the number of variables, followed by a line that shows the domain sizes of each variable. After one empty line the actual CSP is stated. Every line contains one non-empty constraint, starting with two numbers that represent the variables this constraint corresponds to. The rest of the line consists of pairs of domain values. The meaning of these pairs is determined by the keyword mentioned above. If it is `savingconflicts` then every pair represents one conflict, but if it is `savingnonconflicts` it represents the opposite. That is all the domain value pairs in this constraint are conflicts (1's in the next format), except for the pairs on this line. This way of saving lets us save disc space in the case where we have a high ratio (¿ 0.5) of conflicts.

matrixformat We start with a single number on one line. This number represent the amount of variables in the constraint satisfaction problem. Then on the second line the domain sizes of all the variables are given. The third line is left blank. The remainder of the file consists of all the constraints between the variables. This should be interpreted as follows. The first line (of the remainder) shows which conflicts exist between the first and second variable when the first variable has a value of one. The second line does the same for the first variable having a value of two, and so on until the domain size of the first variable. The next block will show the constraints for the first and third variable. In other words if $v_1, v_2, \ldots v_n$ are the n variables in the CSP, and $C(v_1, v_2)$ denotes a set of constraints between the two variables v_1 and v_2, we have the following output (for n variables):

$$
\begin{gathered}
C(v_1, v_2) \\
C(v_1, v_3) \\
\vdots \\
C(v_1, v_n) \\
C(v_2, v_3) \\
\vdots \\
C(v_2, v_n) \\
C(v_3, v_4) \\
\vdots \\
C(v_{n-1}, v_n)
\end{gathered}
$$

For example two variables v_1 and v_2 both with a domain size of five:

$$
C(v_1, v_2) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}
$$

In this matrix a one means that the two variables are not allowed to have these values at the same time. This is called a conflict.

2.1 Using the CheckCsp program

The directory `checkcsp` contains a program that can be used for checking solutions of constraint satisfaction problems. It needs two filenames as arguments; the first must be the file containing the problem as generated by the RandomCsp library, the second is a file containing the solution that is will be tested. The solutionfile should be one line containing integers representing the values of the variables with spaces between them. Example:

```
11 23 6 7 9 2 16 2 8
-- -- - - - - -- - -
|  |      ...      |
v1 v2     ...      vn
```

CheckCsp will output the number of conflicts that the solution violates in the given CSP. The solution and CSP should have the same number of variables, otherwise the program will abort with an error.

what	how
Domain sizes for each variable	one variable per line
Tightness of each constraint	a 2-dimensional table
Number of conflicts between every pair of variables	a 2-dimensional table
Number of variables	n
Total domain size	$\sum_{i=1}^{n} D_i$
Domain size (m) averaged	$\sum_{i=1}^{n}(D_i)/n$
Number of constraints	$\|C\|$
Maximum number of constraints	$0.5 * n * (n-1)$
Ratio of constraints	ratio of above two (should approx p_1)
Number of conflicts	sum of number of conflicts over all constraints
Maximum number of conflicts	maximum number possible
Ratio of conflicts	ratio of above to (should approx p_2)
Constrainedness (kappa)	$(n-1) * 0.5 * p_1 * (\log(\frac{1}{1-p_2})/\log(m))$
Number of components:	number of separate graphs
Ratio of components:	$(\|\text{components}\| - 1)/(n-1$

Table 1: Description of the information output by AnalyzeCsp

2.2 Using the AnalyzeCsp program

The directory `checkcsp` contains a program for analysing `csp` instances from files. It requires one filename on the commandline and outputs the information displayed in Table 1.

3 Using the RandomCspC library

Instead of generating a lot of problems that consume a lot of disc space it is also possible to use the code of the generator in your own source code. It is written in `C++`, and should compile cleanly on most machines. The code basically falls apart into these parts (classes):

ErrorC is used for generating error messages to the user and for generating message when the source is being debugged.

CspC is a class for storing and using binary constraint satisfaction problems.

RandomCspA is an abstract base class to start the branch of random models to create CSPs.

RandomCsp3C is a base class to branch models with three parameters.

RandomCsp4C is a base class to branch models with four parameters.

ModelARandomCspC implements Model A [Palmer, 1985] (four parameters).

ModelBRandomCspC implements the model used most often [Palmer, 1985, MacIntyre et al., 1998] (four parameters).

ModelCRandomCspC implements the Model C, a combination of A & B, not frequently used[Palmer, 1985, MacIntyre et al., 1998] (four parameters).

ModelDRandomCspC implements the Model D, also a combination of A & B, not frequently used [Palmer, 1985, MacIntyre et al., 1998] (four parameters).

ModelERandomCspC implements Model E, an improvement over model A–D [Achlioptas et al., 1997] (three parameters) .

ModelFRandomCspC implements Model F, an improvement over Model E [MacIntyre et al., 1998] (four parameters).

ModelGRandomCspC implements a model that generates deceptive instances for evolutionary algorithms. Uses the tightness as in Model E and uses the connectivity parameter as p for the geometric distribution (four parameters).

ModelRBRandomCspC implements the model from [Xu & Li, 2000], which should ease the creation of solvable problem instances.

StaticCspC is used for loading in instances from disc.

The documentation for using the source code consists of all the comments in it. Most functions, no matter how intuitive the name, have a small block of comment above them stating what they do and what parameters they require. For a thorough description of every function and class we point to [Hemert, 2004].

4 About the author

RandomCsp was written by J.I. van Hemert in 1998–2004. RandomCsp is copyright of J.I. van Hemert. You can reach me by e-mail: `jvhemert@cwi.nl` or have a look at my home page: `http://www.cwi.nl/~jvhemert`. My mailing address:

J.I. van Hemert
Centrum voor Wiskunde en Informatica
P.O. Box 94079
NL-1090 GB Amsterdam
The Netherlands

References

[Achlioptas et al., 1997] Achlioptas, D., Kirousis, L. M., Kranakis, E., Krizanc, D., Molloy, M. S., and Stamatiou, Y. C. (1997). Random constraint satisfaction a more accurate picture. In Smolka, G., editor, *Principles and Practice of Constraint Programming — CP97*, pages 107–120. Springer-Verlag.

[Hemert, 2004] Hemert, J.I. (2004). Technical specifications of the RandomCsp library — version 1.8.0

[MacIntyre et al., 1998] MacIntyre, E., Prosser, P., Smith, B., and Walsh, T. (1998). Random constraint satisfaction: theory meets practice. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, pages 325–339. Springer-Verlag.

[Palmer, 1985] Palmer, E. M. (1985). *Graphical Evolution.* John-Wiley & Sons, New York.
An introduction to the theory of random graphs - an important tool in modeling networks of interactions.

[Xu & Li, 2000] Ke Xu and Wei Li (2000). Exact Phase Transitions in Random Constraint Satisfaction Problems *In Journal of Artificial Intelligence Research 12 (2000)*, pages 93-102.