CSPLIB: a benchmark library for constraints*

Ian P. Gent and Toby Walsh

Department of Computer Science, University of Strathclyde, Glasgow, United Kingdom. Email {ipg,tw}@cs.strath.ac.uk

Abstract. We introduce CSPLIB, a benchmark library for constraints. We discuss the advantages and disadvantages of building such a library. Unlike many other domains (for example, theorem proving, or machine learning), representation remains a major issue in the success or failure to solve constraint satisfaction problems. All problems are therefore specified in natural language as this allows users to find good representations for themselves. We illustrate the format of a typical entry in the library by means of an example due to B. Smith, the template design problem. We desribe the current release of the library (which is available from `http://csplib.cs.strath.ac.uk`). Most importantly, we appeal to the community to become active users of and contributors to CSPLIB.

1 Introduction

In recent years, new constraint satisfaction algorithms have often been benchmarked on hard, random problems. Indeed, the study of random constraint satisfaction problems has itself become part of mainstream research (see, for example, [1, 3]). There are, however, many reasons for wanting a larger class of problems in our benchmark suites. In this paper, we motivate the need for a benchmark library for constraints. We identify why this is a more difficult task than in other areas like machine learning or theorem proving. Finally, we describe CSPLIB, the first release of a benchmark library which is available over the World Wide Web at `http://csplib.cs.strath.ac.uk`.

2 Motivation

There are many motivations for developing a benchmark library for constraints.

Benchmarking: One of the main applications of a library is to provide a common set of problems on which different groups can quickly benchmark their algorithms.

* Supported by EPSRC award GR/K/65706. The authors are members of the the APES research group, http://www.cs.strath.ac.uk/~apes, and thank the other members at Leeds and Strathclyde. We wish to thank the many colleagues we have discussed CSPLib with, and especially those who have already helped in its construction.

Competitions: Many benchmark libraries are developed as part of a system competition. For example, a theorem proving competition has been held alongside the last four Conferences on Automated Deduction based on the highly successful TPTP benchmark library. These competitions have been responsible for focusing more research on heterogenous approaches to theorem proving.

Modelling: A library of constraint problems may concentrate the research community on important issues concerning modelling, and problem reformulation.

Realism: Current experimental methodology can be criticized for using problems that are not realistic as they are randomly generated and contain just binary constraints. A benchmark library can contain problems that are both realistic and non-binary.

New applications: A library can provide rapid entry into new domains; in particular, it may offer a short cut past the usual bottleneck of extracting problems from industry.

Showcase: The existence of a problem in the library may suggest that constraints is a suitable technology for this type of problem; the entry may then provide clues to help a user model a related problem.

Challenges: A benchmark library can be a forum for open problems and other challenges that can push constraint technology onto new heights.

Whilst there are many constructive benefits of a benchmark library, there are also several potential pitfalls.

Over-fitting: If the library is small, we run the risk of over-fitting our algorithms.

Representativeness: Even if the library is large, certain problem features may be rare or absent.

Modelling: Much of the success in solving a constraint problem may be how we model the problem. If the benchmark library is too prescriptive in how problems are modelled, it may be of little value.

Effort: The construction and maintenance of a benchmark library requires many people to donate time and effort with few direct or immediate rewards.

3 A cautionary tale

The machine learning community has had a long established benchmark library, the UCI Machine Learning Repository[1]. This library contains some 111 data sets, and these are widely used to compare machine learning algorithms. In 1993, Holte published results which raise interesting questions about this library [2]. He showed that very simple classification rules (1-level decision trees which classify examples on the basis of just a single attribute) compare very favourably

[1] See `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

in accuracy with the much more complex rules learnt by the (then) state of the art C4 algorithm on 16 data sets commonly used in machine learning research, 14 of which are taken from the UCI Machine Learning Repository.

As Holte himself argues, the practical significance of this result depends on whether or not these data sets are representative of those that arise in practice. Since we might not expect "real" classification problems to be solved by very simple rules, we might doubt if the data sets used in this study are representative of the data sets that actually arise in practice. By extension, we might also doubt the representativeness of the whole repository. Holte accepts that many classification problems that arise in practice do not have simple solutions (for example, the prediction of the secondary structure of protein, or the classification of won and lost positions in chess) and that machine learning techniques optimized for easy problems will perform poorly on such "hard" problems. However, he argues that many real classification problems are not so hard. He notes that many of the data sets in the UCI repository are drawn from a real-life domain and have not been specially "doctored" to make them easier to classify. Although these data sets cannot represent the entire range of "real" problems (for example, they do not represent any of the hard problems mentioned earlier), he argues that the number and diversity of them suggests that they represent a class of problems that often arises. Whilst we accept much of his argument, we must nevertheless treat parts of it with caution. For instance, as many of the data sets are taken from studies of machine learning algorithms, there may be a bias to the sort of easy problems which current machine learning algorithms are able to solve.

4 A model library

Having taken on board lessons from this cautionary tale, what should a model constraints library look like? Some (though not all) of the desiderata of such a library include:

Location: It should be easy to find. The World Wide Web solves this problem easily.

Ease of use: It should be easy to use. Software tools need to be provided to make benchmarking new algorithms as painless as possible.

Diversity: It should contain as diverse a set of problems as possible. This will help prevent over-fitting.

Size: It should be as large as possible. Ideally, it should grow continually so that unrepresentative problems become increasingly less important.

Extensibility: So that the library can grow, the library should should be easy to extend.

Completeness: It should be comprehensive as possible. Users of the library should not need to go anywhere else to find their constraint benchmarks.

Topicality: It should be as up to date as possible. For example, if a more optimal solution to a problem is found, the library should quickly contain this information to prevent duplication of effort.

Independence: It should be independent of any particular constraint solver. Our aim is not to promote any one system but to provide a mechanism for open and fair comparisons of many different systems.

Difficulty: It should not contain just hard or just easy problems. As problems met in practice are often of varying difficulty, a benchmark library should reflect this mix.

Accuracy: The library should be as free of errors as possible. Any errors found in the library should be corrected promptly.

We might look to domains other than machine learning to see how to develop a benchmark library which meets these desiderata. In theorem proving, for example, the TPTP (Thousands of Problems for Theorem Provers) library was started by Suttner and Sutcliffe in 1993 [5] and has since proved highly successful and influential. It is a valued resource in the theorem proving community, and there is an annual system competition based on problems drawn at random from TPTP. The library contains over four thousand different problems, in 28 different domains, as well as comprehensive references on their source and a variety of problem generators. A software tool converts the problems into input formats used by the major theorem proving systems. In addition, the library provides guidelines on how to evaluate theorem proving systems.

Whilst we can learn much from the success of TPTP, there is one major and very significant difference between theorem proving and constraint satisfaction. In theorem proving, there is little debate about representation. Users of TPTP are happy to accept a simple skolemized first-order language. In constraint satisfaction, representation is much more of an issue. Our ability to solve a constraint satisfaction problem may hinge upon exactly how we decide to represent it. A benchmark library for constraints cannot therefore be as prescriptive as in theorem proving about the representation of the problems in it. At the 1998 DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimization, we discussed this problem and suggested that problems should be presented in natural language[2]. Any other representation may make assumptions about representation that are unwanted.

5 Community

To be successful, the research community must become active users of the library and contributors to it. This is in fact the most important desideratum of all: if CSPLIB is not used and contributed to, the time spent in setting it up will have been wasted. However it is one aspect of the library beyond the control of the maintainers. We hope that researchers in constraints will come to view CSPLIB as a one stop shop for finding constraints benchmarks, and disseminating new benchmarks. We hope that when you read a paper at CP-2000 solving a new kind of problem with constraints methods, you will find the problem already

[2] We first suggested English, but the non-native speakers of English quickly and correctly pointed out the bias in such a choice.

posted at CSPLIB. More than that, we hope that CSPLIB will become a rich enough source of benchmarks that when you invent a new technique, you will surf to CSPLIB in the expectation of finding a challenge problem there which has the features which your technique can exploit. And more even than that, we hope that as you browse CSPLIB you will be inspired by the problems that remain open to invent new ways of using constraints to solve difficult problems. None of these possibilities can happen unless the community becomes active in submitting problems to the library. Our experience is that writing an entry takes less than an hour, and the service to the community is invaluable.

What kind of problems would now be found at CSPLIB if authors of papers at CP-98 had been able to contribute to it? Just from test instances reported in the proceedings [4] but not available in other benchmark libraries, CSPLIB would now contain instances from protein structure prediction, due date constraints in scheduling, ordinary differential equations, flow shop scheduling, temporal reasoning problems, key/lock configuration, handling aerial images, integration with a graphics editor, hoist scheduling problems, and vehicle routing problems. Some of these instances are available either in the web or detailed in published papers, but many are not easily available in any form.

From this list of problems, it is clear that we intend to be liberal in the variety of problems in CSPLIB. Here is a non-exhaustive list of some of the other ways in which we will be liberal:

- There is no requirement that you present a solution to a benchmark problem. Test instances you have *failed* to solve will be just as valuable as those that you can. Much research is driven by open problems.
- There is no requirement that constraint programming is currently the best technique for solving the benchmark. We encourage such problems because they may suggest new techniques in constraint solving, or simply aid our understanding of its limits. Examples of this already in CSPLIB are Golomb Rulers (prob006) and Low Autocorrelation Binary Sequences (prob005).
- There is no requirement that an encoding be given in a particular constraint solver. It is unlikely that constraint programs for all of the CP-98 list could easily be written in any current system.
- Beyond this, there is not even a requirement that an encoding be given for *any* constraint solver. The only requirement is a natural language specification.
- There is no requirement that specific test instances be given. Where possible, we hope that contributors will submit benchmark instances, but there will sometimes be good grounds for not doing so. For example, where random benchmark instances are used, a generating program may be more useful to the community than particular instances. In other cases detailed data may be confidential, but the community would still benefit from a description of the problem.
- There is no requirement that problems be toy problems. For example, prob011 is of scheduling a very real basketball tournament.
- There is no requirement that the problem be real. For example prob014 is 'solitaire battleships', a puzzle from Games Magazine.

6 Submission Guidelines

To help users submit benchmark problems to CSPLIB, we have prepared some simple guidelines. We repeat these in full:

> "We welcome submission of all your constraint problems. We hope that the library will thereby grow to become a valued resource within the constraints community.
>
> A key factor in solving many constraints problems is the modelling of the constraints. We therefore specify all problems in CSPLIB using natural language. We also provide hints about modelling these problems.
>
> As we want to help people benchmark their algorithms on problems in CSPLIB with minimum effort, we encourage users of the library to send us any tools that might be useful to others (e.g. C parsers for data files, AMPL specification of problems, ILOG Solver code, ...). All such code is placed in the library through the generosity of the authors, and comes with all the usual disclaimers.
>
> To make comparison with previous work easier, we provide links to papers that use these benchmarks. If you use a problem in CSPLIB, please send us the URL to your paper so we can add it to the references section.
>
> Finally, to make it easy to compare your results with others, we will provide a record of results. Please help us keep these records up-to-date by sending in your results promptly. "

7 An example entry

Entries in the benchmark library typically consist of an index page, a specification page, a reference page, a results page, one or more data files (where applicable), and one or more software tools. The specification page is a simple natural language description of the problem. Numerous links are provided to outside resources, including online version of papers which report results on the benchmark in question. Problems are divided into different subject categories. They range from simple combinatorial puzzles like the existence or non-existence of a Golomb ruler of certain size, to more realistic examples like scheduling the 1997/98 Atlantic Coast Conference basketball tournament. It is our intention that no problem will be too small or too big for inclusion in the library.

As an example of a typical entry, Barbara Smith has proposed template design, a problem extracted from the printing industry, as a possible benchmark for the constraints community. The problem appears as `prob002` in CSPLIB, We reproduce the pages from CSPLIB concerned with this problem. One factor missing from the reproduction of these pages are the many hyperlinks to online sources of information. We believe this may be one of the most useful aspects of the library. We therefore encourage users to send us any relevant URLs.

prob002: template design

proposed by **Barbara Smith**
bms@scs.leeds.ac.uk

Overview

The problem description consists of:

- a problem specification
- a data file
- a list of results
- a reference

Back to CSPLib home page.

prob002: template design

proposed by **Barbara Smith**
bms@scs.leeds.ac.uk

Specification

This problem arises from a colour printing firm which produces a variety of products from thin board, including cartons for human and animal food and magazine inserts. Food products, for example, are often marketed as a basic brand with several variations (typically flavours). Packaging for such variations usually has the same overall design, in particular the same size and shape, but differs in a small proportion of the text displayed and/or in colour. For instance, two variations of a cat food carton may differ only in that on one is printed ‘Chicken Flavour’ on a blue background whereas the other has ‘Rabbit Flavour’ printed on a green background. A typical order is for a variety of quantities of several design variations. Because each variation is identical in dimension, we know in advance exactly how many items can be printed on each mother sheet of board, whose dimensions are largely determined by the dimensions of the printing machinery. Each mother sheet is printed from a template, consisting of a thin aluminium sheet on which the design for several of the variations is etched. The problem is to decide, firstly, how many distinct templates to produce, and secondly, which variations, and how many copies of each, to include on each template.

The following example is based on data from an order for cartons for different varieties of dry cat-food.

Variation	Order Quantity
Liver	250,000
Rabbit	255,000
Tuna	260,000
Chicken Twin	500,000
Pilchard Twin	500,000
Chicken	800,000
Pilchard	1,100,000
Total	3,665,000

Each design of carton is made from an identically sized and shaped piece of board. Nine cartons can be printed on each mother sheet, and several different designs can be printed at once, on the same mother sheet. (Hence, at least 407,223 sheets of card will be required to satisfy these order quantities.)

Because in this example there are more slots in each template (9) than there are variations (7), it would be possible to fulfil the order using just one template. This creates an enormous amount of waste card, however. We can reduce the amount of waste by using more templates; with three templates, the amount of waste produced is negligible. The problem is therefore to produce template plans which will minimize the amount of waste produced, for 1 template, 2 templates,... and so on.

It is permissible to work in units of say 1000 cartons, so that the order quantities become 250, 255, etc.

A variant is to allow up to 10% under-production of some designs, if this allows the overall over-production to be reduced. This is not a sensible option for the catfood problem, because it leads to under-production of all the designs.

The optimal solutions for the catfood problem are shown below. For each template, the table gives a list of the number of slots allocated to each design, e.g. [1,1,1,1,1,2,2,] means that 1 slot is allocated to each of the first five designs and two each to the last two.

No. of templates	Layouts	No. of Pressings of each template	Total pressings
1	[1,1,1,1,1,2,2]	550,000	550,000
2	[0,0,0,0,0,2,7]	158,000	
	[1,1,1,2,2,2,0]	260,000	418,000
3	[0,5,3,0,0,1,0]	51,000	
	[0,0,1,0,0,7,1]	107,000	
	[1,0,0,2,2,0,4]	250,000	408,000

Back to CSPLib home page.

prob002: template design

proposed by **Barbara Smith**
bms@scs.leeds.ac.uk

Data file

The format is:

Number of designs

Number of designs which can be printed on one sheet of card (= no. of slots in each template)

Order quantities for each design, in thousands

```
Cat Food

7 9

250 255 260 500 500 800 1100

Herbs & Spices

30 42

280 280 230 230 230 230 150 100 100 100

100 90 90 90 90 90 90 80 80 80

80 70 70 70 70 70 70 70 60 60

Magazine inserts

50 40

50 53 55 60 85 90 100 100 105 110

137 140 140 140 150 150 150 150 150 150

150 150 168 170 170 195 195 200 200 200

210 210 225 230 230 230 250 250 250 250

250 250 250 250 265 270 270 375 375 405
```

Back to CSPLib home page.

prob002: template design

proposed by **Barbara Smith**
bms@scs.leeds.ac.uk

References

ILP and Constraint Programming Approaches to a Template Design Problem . Les Proll and Barbara Smith, Research Report 97.16, May 1997. (To appear in INFORMS Journal of Computing.) *(Abstract)*

Back to CSPLib home page.

prob002: template design

proposed by **Barbara Smith**
bms@scs.leeds.ac.uk

Results

Solutions for the problems in the data files are given in Proll and Smith (1997) . Most solutions given there for the magazine inserts problem are not known to be optimal.

This problem appears to be difficult for both ILP and CP. For ILP, this is because the problem cannot easily be expressed in terms of linear constraints. Whilst CP can in principle handle non-linear constraints, a straightforward formulation with only sufficient constraints to ensure that solutions are correct appears to be practical only for the smallest problems. For the larger problems, whilst it is possible to find solutions which cannot be far from optimal using a more sophisticated approach, proving optimality or finding better solutions has so far proved impossible.

Back to CSPLib home page.

8 Release 1.0

CSPlib was first released on March 4th, 1999. The release version contained fourteen problems, divided into five, overlapping subject areas: scheduling and related problems, bin packing and related problems, frequency assignment and related problems, combinatorial mathematics, and games and puzzles. The total size of the library is just over 1 MByte. In its first month of operation, the library has received more than 100 visitors each week. A low-volume mailing list is maintained to make announcements about significant changes to the library. In addition, users can email `csplib@cs.strath.ac.uk` to propose new entries, or to extend or correct existing ones. The site is located at `http://csplib.cs.strath.ac.uk` and is mirrored at `http://www.cs.cornell.edu/home/selman/csplib` in the United States to help improve the speed of access. We are very grateful to Bart Selman for his assistance with the mirror.

9 Conclusions

We have introduced CSPlib, a benchmark library for constraints. With the support of the research community, we believe that this library can grow to be a very valued resource. Unlike many other domains (for example, theorem proving, or machine learning), representation remains a major issue in our ability to solve constraint satisfaction problems. All problems in the library are therefore specified in natural language so that users can devise their own representations. In addition, many entries contain example representations (e.g. CHIP or Solver code). The success or failure of the library depends on its uptake by the constraints community. We encourage you therefore to benchmark your algorithms on the problems contained within CSPlib, to send us results and any other data of interest that can be included in the entries, to notify us of any errors, but most of all, to propose new problems. Without your help, we cannot succeed.

References

1. D. Achlioptas, L.M. Kirousis, E. Kranakis, D. Krizanc, M.S.O. Molloy, and Y.C. Stamatiou. Random constraint satisfaction: A more accurate picture. In G. Smolka, editor, *Proceedings of Third International Conference on Principles and Practice of Constraint Programming (CP97)*, pages 107–120. Springer, 1997.
2. R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 3:63–91, 1993.
3. E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: Theory meets practice. In *4th International Conference on Principles and Practices of Constraint Programming (CP-98)*, pages 325–339. Springer, 1998.
4. M. Maher and J.-F. Puget, editors. *Principles and Practice of Contraint Programming – CP98*. Springer, 1998. LNCS 1520.
5. G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In A. Bundy, editor, *12th International Conference on Automated Deduction*, pages 252–266. Springer Verlag, 1994. LNAI 814.