Performance Testing of Combinatorial Solvers With Isomorph Class Instances

Franc Brglez
Dept. of Computer Science
NC State University
Raleigh, NC, USA
brglez@ncsu.edu

Jason A. Osborne
Dept. of Statistics
NC State University
Raleigh, NC, USA
osborne@stat.ncsu.edu

ABSTRACT

Combinatorial optimization problems expressed as Boolean constraint satisfaction problems (BCSPs) arise in several contexts, ranging from the classical unate set-packing problems to the binary minimum cover problems, including the Haplotype Inference by Pure Parsimony (HIPP) problem. These problems are being solved under different formulations and in different formats. Results of experiments that are reported can be seldom compared and replicated.

This paper is not about ‘the best BCSP solver’. Rather, it is a case study of how the scientific method can be applied to comparing the performance of not only BCSP solvers but also other solvers that address NP-hard problems. The approach is founded on two premises: (1) the introduction of instance isomorphs as families of equivalence classes, based on randomized replicas of a given reference instance, and (2) the use of isomorph classes for the design of reproducible experiments with BCSP solvers that includes performance testing hypotheses. We introduce a number of BCSP reference instances from different domains, generate isomorph classes and use various versions of cplex to characterize the solver performance and the isomorph classes themselves. This methodology may make it easier to (1) reliably improve the performance of combinatorial solvers and, (2) report results of experiments under the proposed schema.

Categories and Subject Descriptors:
G.3 [Probability and Statistics]: Experimental design

General Terms: Algorithms, Scientific Method, Reliability

1. INTRODUCTION

A number of efforts have been made to formalize the experiments and experimental analysis of combinatorial problems, ranging from guidelines to pitfalls [1, 2, 3, 4, 5, 6, 7, 8].

Reproducibility is one of the main principles of the scientific method, and refers to the ability of a test or experiment to be accurately reproduced, or replicated, by someone else working independently. Our approach is analogous to testing the lifetime of hardware components: an equivalence class of N isomorphs, all derived from the same reference instance represents a batch of N replicated hardware components, a combinatorial solver X that reads and solves each problem instance represents a controlled operating environment Y maintained for the lifetime of each hardware component, and the empirical cumulative distribution function (ECDF) represents the solvability function $S^X(x)$ while the reliability or survival function $R^X(y)$ represents the complement of ECDF. Whereas $x$ represents RunTime, $y$ represents LifeTime. Without loss of generality, we present our approach on representative instances from the well-known category of Boolean constraint satisfaction problems (BCSPs) [9] that clearly push the limits of the state-of-the-art combinatorial solver cplex [10]. Typically, such problems are being solved under different formulations and in different formats and the results of experiments that are reported can be seldom compared and replicated.

An instance of a Boolean constraint satisfaction problem is given by $m$ constraints applied to $n$ Boolean variables. The well-known conjunctive-normal-form format (.cnf) captures such constraints very concisely. However, different computational problems arise not only from the nature of constraints but also depend on the goals of the optimization task – a feature that is not supported by the .cnf format. We reconcile these issues by using the familiar 0/1 integer program (IP) formulation that naturally expresses the constraints as well as the goals of the optimization task when formulating an optimization instance. In the Appendix we show example instances in a simple-to-read .lpx format, a subset of the cplex format [10] that is also readable by the public-domain solver lp_solve [11, 12].

For years, publications on special purpose BCSP solvers have been comparing their performance to cplex whose performance was usually dominated by the new special-purpose solver being published. However, our recent work and comparisons with cplex reveals cases where cplex appears to dominate on a number of instances [13]. It is a given that the developer of a special purpose BCSP solver expects to design it in a way that will outperform a general purpose LP solver such as cplex which may only handle BCSPs on the side. One of the most important goals of this paper is to initiate a methodology of performance testing that will reliably measure and improve the performance of any and all BCSP solvers, thereby extending the work initiated in [14]. The paper is organized as follows:

Section 2 introduces several classes of the Boolean constraint satisfaction problem (BCSP) under the 0/1 integer pro-
gram (IP) formulation, including examples of transformations between related unate and binate minimization and maximization instances.

Section 3 formalizes the construction of isomorph classes from a single reference instance and concludes with a preview of examples of isomorphs that induce significant variability in RunTime performance of cplex.

Section 4 outlines the main elements of the experimental environment we use, the isomorph classes, and the solvers to design and to execute a number of experiments on these classes. This section also includes a table and a brief characterization of hard-to-solve reference instances from different domains, assembled and translated into the .lp format, including “block instances” of increasing size, each with a ‘hidden solution’. A subset of these instances is used to induce a number isomorph classes for the experiments reported in the next section.

Section 5 defines five experimental designs and reports on results of experiments for each design. In particular, the report for each design has three components: (1) a design goal, linked to a test of hypothesis, (2) discussion of results, and (3) resolution of hypothesis.

Section 6 and Appendix conclude the paper.

2. INSTANCE FORMULATIONS

We start with basic notation and definitions and conclude with examples that illustrate them.

Notation and Definitions. Unlike textbooks [15], we represent constraints in both the maximization and the minimization BCSP instance with the ‘>=’ relation, i.e.

\[ \max w^T x \text{ subject to } Ax \geq b, \ x \in \{0, 1\} \]

and

\[ \min w^T x \text{ subject to } Ax \geq b, \ x \in \{0, 1\} \]

where \(w\) is an \(n\)-vector in \(R^n\) or \(Z^n\), \(A\) is an \(m \times n\) constraint matrix with entries from \{0, 1, –1\}, and \(b\) is an \(n\)-dimensional vector whose entries are no longer 1’s by default. The entries in \(b\) depend on the context of the constraint and also on the distribution of the ± signs within the constraint, as we explain next.

Denoting \(I_p\) and \(I_n\) as subsets of \{1, 2, ..., \(n\)\} we distinguish between three classes of constraints:

**unate-positive**, equivalent to the set cover constraint:

\[ \sum_{i \in I_p} (+x_i) \geq +1 \]

i.e. at least one \(x_i\) must be set to 1.

**unate-negative**, equivalent to the set packing constraint:

\[ \sum_{j \in I_n} (-x_j) \geq -1 \]

i.e. at most one \(x_j\) can be set to 1. Whenever \(|I_n| > 2\), it defines a clique constraint [15] and can be decomposed into \(|I_n|(|I_n| - 1)/2\) equivalent constraints. For example, the single constraint \(-x_1 - x_2 - x_3 \geq -1\) is equivalent to the following pair-wise constraints:

\[ -x_1 - x_2 \geq -1, -x_1 - x_3 \geq -1, -x_2 - x_3 \geq -1. \]

A **binate**, a combination of set cover and packing constraints with a relaxed right-hand-side:

\[ \sum_{i \in I_p} (+x_i) + \sum_{j \in I_n} (-x_j) \geq +1 - |I_n| \]

If \(I_p = \emptyset\), the constraint \(\sum_{j \in I_n} (-x_j) \geq 1 - |I_n|\) is satisfied for all combinations of values of \(x_j\), except for all \(x_j = 1\).

If all constraints are unate-positive, the solution of the maximization instance is trivial, similarly for the minimization of the instance where all constraints are unate-negative. However, for the general case, both the maximization and the minimization can be equally hard.

**Remark:** An instance of a Boolean constraint satisfaction problem (BCSP) is a maximization or a minimization problem with any combination of unate-positive, unate-negative, and binate constraints. Minimum (weighted) binate set cover, maximum (weighted) unate set packing, minimum (weighted) vertex cover, (weighted vertex) maximum clique, etc. are all BCSPs. Min Ones and Max Ones problems are special cases of unit-weighted BCSPs. Classes of Max CSP (Min CSP) problems as defined in [9] are also included in this formulation of BCSP. The next few example illustrate the structure of some such instances.

**Instance examples.** We show small examples and solutions of a weighted minimum set cover instance, a weighted vertex maximum clique instance that is derived directly from the structure of the set cover instance, and a weighted binate instance with a maximization objective. We also show solutions of related instances with the same structure: a weighted maximum set packing instance and a weighted binate instance with a minimization objective. Examples of additional instance transformations (and how they may relate) will be introduced in the full-length paper.

A weighted minimum set cover instance.

**ObjectiveOpt 70**

Solution 1010100

\[ Min \]

\[ +21x_1 + 22x_2 + 23x_3 + 25x_4 + 26x_5 + 27x_6 + 29x_7 \]

\[ st \]

\[ c1 : \quad +x_2 +x_3 +x_4 \quad \geq \quad +1 \]

\[ c2 : \quad +x_2 +x_5 +x_6 \quad \geq \quad +1 \]

\[ c3 : \quad +x_5 +x_6 +x_7 \quad \geq \quad +1 \]

\[ c4 : \quad +x_3 \quad \geq \quad +1 \]

\[ c5 : \quad +x_1 +x_4 +x_7 \quad \geq \quad +1 \]

\[ c6 : \quad +x_1 +x_3 +x_6 \quad \geq \quad +1 \]

A weighted maximum set packing instance.

This instance is generated from the set packing instance by (1) flipping the ‘+’ variable signs in each row to ‘-’, (2) replacing the right-hand-side with values of -1, and (3) changing the objective from ‘min’ to ‘max’.

**ObjectiveOpt 52**

Solution 0001010

A weighted vertex maximum clique instance.

This instance is generated from the set packing instance by (1) expanding all clique constraints into pair constraints (one pair on each row), (2) flipping the ‘+’ variable signs in each row to ‘-’, (3) replacing the right-hand-side with values of -1, and (4) changing the objective from ‘min’ to ‘max’.

**ObjectiveOpt 100**

Solution 1010011
Max
+21x_1 + 22x_2 + 23x_3 + 25x_4 + 26x_5 + 27x_6 + 29x_7
st
  c_1: \quad -x_3 -x_5 \quad >= -1
  c_2: \quad -x_4 -x_5 \quad >= -1
  c_3: \quad -x_2 \quad >= -1
  c_4: \quad -x_4 -x_6 \quad >= -1
  c_5: \quad -x_1 -x_2 \quad >= -1
A weighted binate instance (obj=max).
ObjectiveOpt 100
Solution 0110101

Max
+21x_1 + 22x_2 + 23x_3 + 25x_4 + 26x_5 + 27x_6 + 29x_7
st
  -x_3 -x_1 -x_6 \quad >= -1
  -x_1 -x_4 -x_7 \quad >= -1
  -x_5 -x_2 -x_6 \quad >= -2
  +x_3 +x_2 +x_4 \quad >= +1
  +x_7 -x_3 \quad >= 0
  -x_7 +x_6 +x_5 \quad >= 0
It is clear by inspection that no permutation of variables took place in the isomorp LR, while rows have been permuted (row 1 in the reference instance is now row 4 in the isomorph). Furthermore, the order of variable positions in the row 4 in the isomorph is different from the order of variable positions in the row 1 in the reference instance.

3. CLASSES OF INSTANCE ISOMORPHS

Isomorphs of sat instances have been shown to induce significant variability in SAT solvers [14]. In this paper, we demonstrate that instance isomorphs of BCSP’s (Boolean constraint satisfaction problems) as defined in the preceding section are also fundamental to exploring performance variability of combinatorial solvers that take them as input.

Given a (sparse) matrix formulation of the reference instance, an isomorph is generated by applying to the reference any subset of four primitive operations:

C: random permutation of variables – effectively a permutation of columns in the matrix;
L: random permutation of the variable order in any row of the matrix;
R: random permutation of rows in the matrix, followed by permutation of the weight vector (not needed if all weights have the value of 1);
X: random sign flipping (from positive to negative and vice versa) of any variable – while maintaining consistency of the right-hand-side value so that the instance remains a BCSP and the value of its objective function invariant.

The operation of flipping the variable sign (X) has intrinsic merits with SAT solvers and can only be applied to instances of BCSP in special situations. In this paper, we shall consider isomorphs in two equivalence classes only: LR and CLR. Two isomorphs from each of the two classes are shown below, based on LR operations and CLR operations applied to the same reference instance: the weighted binate instance in the previous section.

A weighted binate instance (obj=max) – isomorph_LR.
ObjectiveOpt 100
Solution 0110101

<table>
<thead>
<tr>
<th>translator</th>
<th>instance</th>
<th>Obj_opt</th>
<th>RunTime (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>i00.lpx</td>
<td>24</td>
<td>114.91</td>
</tr>
<tr>
<td>T2</td>
<td>i06.lpx</td>
<td>24</td>
<td>82.55</td>
</tr>
<tr>
<td>T3</td>
<td>i17.lpx</td>
<td>24</td>
<td>1801.86</td>
</tr>
</tbody>
</table>

These instances under f51mb_350_B_40v_20_LR do not represent the extreme cases: instance i12 is solved for the same optimum in 60.37 seconds, while instance i30 times out.
at 2115.28 seconds without proving that the best objective reported at 24 is indeed the optimum.

As shown in sections that follow, such solver sensitivity to the order of data in the instance file is not unusual – which explains why researchers may report vastly different performance results with the same instance, on the same platform, and with the same version of the solver!

Two questions arise: (1) do instances from a CLR-class induce solver variability that is equivalent to the variability induced by instance in the LR-class, and (2) is a CLR-isomorph class needed and why. The answer to the second question is affirmative – and is based on a few years of ‘lessons-learned’ experience [16, 17].

We do need to perform most if not all experiments with instances from the CLR-class because we cannot anticipate when we may encounter a ‘smart solver’ that will attempt to re-order input data in some predetermined fashion, so that most if not all instances from the LR-class may be reordered with relative ease into an almost equivalent if not equivalent order1. While this is apparently not the case (yet) with the cplex solver, we have had the experience with ‘smart’ BDD variable-ordering solvers where the only way to expose their sensitivity to order requires that we also rename and permute the variables in each input file instance [17].

4. EXPERIMENTAL ENVIRONMENT

The environment for the series of experiments reported in this paper is still evolving. The main components include a schema and utilities to maintain: (1) hierarchies of BCSP reference instances in a common .lp format (with translators to/from .lp Format). (2) hierarchies of BCSP isomorphs generated from each reference instance, (3) BCSP solver encapsulators that also process any combination of solver options and platform specifics into a unique solver ID, (4) hierarchically-archived BCSP-specific experimental results tagged by instance ID, instance class, and solver ID. The leaves of experimental results are directories that contain files with raw results in a form specific to each solver and each isomorph class. This includes files with distributions of observed variables, extracted from raw results and now in a simple tabular format.

Standard statistical technique are applied to analyze the distributions of observed variables such as RunTime and Objective Best. These techniques include resolution of hypothesis tests that have been formulated as the part of the experimental design, outlined in the section that follows. For example, we examine hypotheses which address the branch-and-bound BCSP performance of two solvers, with and without options, cplex (version 9.0) and cplex (version 10.1).

A substantial number of BCSP instances has been collected, translated into the .lp format, and run in cplex. A subset of these instances and runs is summarized as reference instances in Table 1. A larger set and similar results are being prepared for a technical report and a web-posting under http://www.cbl.ncsu.edu/xBed/

Table 1 summarizes instance categories and current status vis-a-vis cplex (version 9.0). As shown in the table, most instance have not been solved optimally and represent an on-going challenge for cplex and other BCSP solvers. It may be of some interest to observe, not only the column on the sparsity measure (sp) but also the column on the measure of completeness of the underlying instance graph. For example, instances in*sc have constraint matrices that are sparse, but the underlying structure of the graph is highly ‘interconnected’ and hard to solve to optimality. Now, the maximum clique instances in*ccliq that have been derived from these instances will have complement graphs that are much less ‘interconnected’ – and these instance have been solved to optimality in a reasonable time frame. Additional highlights from the table follow.

min set cover (unate): Instances ex5.pi and test.4.pi represent column-row reduced versions of the most challenging unate instances from the LogicSyn91 set [18]. Instances in*sc have been transformed into set cover instances from the set packing instances described below.

min set cover (bineate): Instances rot.b, alt4, ef64.b represent column-row reduced versions of the most challenging binate instances from the LogicSyn91 set [18].

max set packing (unate): Instances in*sp are translated versions of set packing instances kindly submitted by Y. Guo, as a follow-up on a publication request [19], now updated in [20]. This a set of 500 random instances in five size categories, from 500 variables to 1500 variables. We adopted the first instance in each category as the reference instance for our experiments with isomorphs. Also, we adopted instance in413_sp as a reference instance of special interest.

max independent set: Instances fr30* are translations of a subset of unit-weighted independent reference instances with hidden solution, from http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/setbenchmarks.htm. The instance dsjc125* is1 a useful test instance floating on the Web, with comments that point to the original publications [21].

max clique: Instances ccliq and ccliq1 are weighted and unit-weighted instance of maximum clique problems. They have been derived from the instances fr30*, dsjc125*, and in*sp described earlier.

blocks (min vertex cover): Instances in this set represent block compositions of increasing size (and a hidden solution) of the minimum vertex cover problem.

blocks (min binate cover): Instances in this set represent block compositions of increasing size (and a hidden solution) of the minimum binate cover problem.

A description of instance block composition with hidden solution and controlled overlap used to create instance above will be provided elsewhere. Some aspects of the method are available in [23]. Due to space constraints, the report on results with five experimental designs in next the section concentrates only on two very different classes, each containing 32 instances: (1) in401_sp_CLR, based on a set packing reference with 500 variables and 1000 constraints, and (2) f51mb350_CLR, based on a binate set cover block composition reference with 350 variables and 413 constraints. The name f51mb350_CLR is an alias for the class f51mb0350B_004020_20_CLR as it is listed in Table 1 and also posted on the Web.

5. EXPERIMENTAL DESIGNS

We executed five experimental designs to gather observations of RunTime and Objective Best as reported by different BCSP solvers when applied to several instance classes.

1Such strategy has also been demonstrated to backfire since it prevents the solver from ‘seeing’ many input orders that could improve its average performance.
Table 1: Introducing a subset of reference instances and basic experiments with cplex090.

<table>
<thead>
<tr>
<th>Dir</th>
<th>Instance</th>
<th>ObjBest</th>
<th>Proof</th>
<th>Ones</th>
<th>RunTime</th>
<th>n</th>
<th>m</th>
<th>cdMax</th>
<th>rdMax</th>
<th>sp(%)</th>
<th>gc(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>in101_ac</td>
<td>189016</td>
<td>no</td>
<td>54</td>
<td>2112.85</td>
<td>1000</td>
<td>500</td>
<td>50</td>
<td>77</td>
<td>5.55</td>
<td>68.82</td>
</tr>
<tr>
<td>(unate)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>in201_ac</td>
<td>547921</td>
<td>no</td>
<td>56</td>
<td>2114.91</td>
<td>1000</td>
<td>1000</td>
<td>100</td>
<td>79</td>
<td>5.59</td>
<td>84.99</td>
</tr>
<tr>
<td></td>
<td>in401_ac</td>
<td>593034</td>
<td>no</td>
<td>68</td>
<td>2112.52</td>
<td>500</td>
<td>1000</td>
<td>100</td>
<td>45</td>
<td>5.72</td>
<td>85.57</td>
</tr>
<tr>
<td></td>
<td>in501_ac</td>
<td>589992</td>
<td>no</td>
<td>54</td>
<td>2116.38</td>
<td>1500</td>
<td>1000</td>
<td>150</td>
<td>157</td>
<td>7.85</td>
<td>91.84</td>
</tr>
<tr>
<td></td>
<td>in600_ac</td>
<td>954508</td>
<td>no</td>
<td>72</td>
<td>2118.01</td>
<td>1500</td>
<td>1500</td>
<td>150</td>
<td>151</td>
<td>5.50</td>
<td>90.88</td>
</tr>
<tr>
<td>max</td>
<td>in201_ac</td>
<td>77596</td>
<td>no</td>
<td>13</td>
<td>2117.8</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>79</td>
<td>5.59</td>
<td>84.99</td>
</tr>
<tr>
<td>(unate)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>in401_ac</td>
<td>77418</td>
<td>yes</td>
<td>12</td>
<td>866.87</td>
<td>500</td>
<td>1000</td>
<td>100</td>
<td>45</td>
<td>5.72</td>
<td>85.57</td>
</tr>
<tr>
<td></td>
<td>in413_ac</td>
<td>74435</td>
<td>no</td>
<td>12</td>
<td>1057.95</td>
<td>500</td>
<td>1000</td>
<td>100</td>
<td>46</td>
<td>5.55</td>
<td>83.65</td>
</tr>
<tr>
<td></td>
<td>in501_ac</td>
<td>76906</td>
<td>no</td>
<td>15</td>
<td>2118.39</td>
<td>1500</td>
<td>1000</td>
<td>150</td>
<td>157</td>
<td>7.85</td>
<td>91.84</td>
</tr>
<tr>
<td></td>
<td>in600_ac</td>
<td>98805</td>
<td>no</td>
<td>15</td>
<td>2119.45</td>
<td>1500</td>
<td>1500</td>
<td>150</td>
<td>151</td>
<td>5.50</td>
<td>90.88</td>
</tr>
<tr>
<td>max</td>
<td>dsjc125_s1</td>
<td>34</td>
<td>yes</td>
<td>34</td>
<td>17.7</td>
<td>125</td>
<td>736</td>
<td>23</td>
<td>2</td>
<td>1.60</td>
<td>9.50</td>
</tr>
<tr>
<td>indep.</td>
<td>frb30-15-1</td>
<td>27</td>
<td>no</td>
<td>27</td>
<td>2118.49</td>
<td>450</td>
<td>1787</td>
<td>122</td>
<td>2</td>
<td>0.44</td>
<td>17.65</td>
</tr>
<tr>
<td></td>
<td>frb30-15-2</td>
<td>27</td>
<td>no</td>
<td>27</td>
<td>2118.08</td>
<td>450</td>
<td>1787</td>
<td>116</td>
<td>2</td>
<td>0.44</td>
<td>17.69</td>
</tr>
<tr>
<td></td>
<td>frb30-15-3</td>
<td>28</td>
<td>no</td>
<td>28</td>
<td>2118.05</td>
<td>450</td>
<td>1789</td>
<td>122</td>
<td>2</td>
<td>0.44</td>
<td>17.63</td>
</tr>
<tr>
<td></td>
<td>frb30-15-4</td>
<td>28</td>
<td>no</td>
<td>28</td>
<td>2118.68</td>
<td>450</td>
<td>1783</td>
<td>110</td>
<td>2</td>
<td>0.44</td>
<td>17.65</td>
</tr>
<tr>
<td></td>
<td>frb30-15-5</td>
<td>28</td>
<td>no</td>
<td>28</td>
<td>2119.11</td>
<td>450</td>
<td>1779</td>
<td>128</td>
<td>2</td>
<td>0.44</td>
<td>17.61</td>
</tr>
<tr>
<td>max</td>
<td>dsjc125.cliq</td>
<td>4</td>
<td>yes</td>
<td>4</td>
<td>0.53</td>
<td>125</td>
<td>7014</td>
<td>119</td>
<td>2</td>
<td>1.60</td>
<td>90.50</td>
</tr>
<tr>
<td>clique</td>
<td>frb30-15-1.cliq</td>
<td>15</td>
<td>no</td>
<td>15</td>
<td>2120.41</td>
<td>450</td>
<td>8319</td>
<td>407</td>
<td>2</td>
<td>0.44</td>
<td>82.35</td>
</tr>
<tr>
<td></td>
<td>frb30-15-2.cliq</td>
<td>15</td>
<td>no</td>
<td>15</td>
<td>2120.02</td>
<td>450</td>
<td>8315</td>
<td>404</td>
<td>2</td>
<td>0.44</td>
<td>82.31</td>
</tr>
<tr>
<td></td>
<td>frb30-15-3.cliq</td>
<td>15</td>
<td>no</td>
<td>15</td>
<td>2118.98</td>
<td>450</td>
<td>8321</td>
<td>400</td>
<td>2</td>
<td>0.44</td>
<td>82.37</td>
</tr>
<tr>
<td></td>
<td>frb30-15-4.cliq</td>
<td>15</td>
<td>no</td>
<td>15</td>
<td>2118.5</td>
<td>450</td>
<td>8319</td>
<td>401</td>
<td>2</td>
<td>0.44</td>
<td>82.35</td>
</tr>
<tr>
<td></td>
<td>frb30-15-5.cliq</td>
<td>15</td>
<td>no</td>
<td>15</td>
<td>2120.63</td>
<td>450</td>
<td>8323</td>
<td>403</td>
<td>2</td>
<td>0.44</td>
<td>82.39</td>
</tr>
<tr>
<td></td>
<td>in201.cliq</td>
<td>7265040</td>
<td>yes</td>
<td>361</td>
<td>3.56</td>
<td>1000</td>
<td>7495</td>
<td>572</td>
<td>2</td>
<td>0.20</td>
<td>15.01</td>
</tr>
<tr>
<td></td>
<td>in201.cliq</td>
<td>361</td>
<td>yes</td>
<td>361</td>
<td>3.56</td>
<td>1000</td>
<td>7495</td>
<td>572</td>
<td>2</td>
<td>0.20</td>
<td>15.01</td>
</tr>
<tr>
<td>in501</td>
<td>36</td>
<td>yes</td>
<td>36</td>
<td>19.44</td>
<td>974</td>
<td>686</td>
<td>71</td>
<td>74</td>
<td>2.85</td>
<td>16.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>no</td>
<td>105</td>
<td>2117.77</td>
<td>5117</td>
<td>1435</td>
<td>54</td>
<td>150</td>
<td>1.36</td>
<td>10.07</td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>rot.b</td>
<td>84</td>
<td>yes</td>
<td>84</td>
<td>6.34</td>
<td>887</td>
<td>1257</td>
<td>158</td>
<td>79</td>
<td>1.23</td>
<td>7.29</td>
</tr>
<tr>
<td>binate</td>
<td>alu4</td>
<td>32</td>
<td>yes</td>
<td>32</td>
<td>38.5</td>
<td>481</td>
<td>592</td>
<td>165</td>
<td>74</td>
<td>3.46</td>
<td>20.16</td>
</tr>
<tr>
<td></td>
<td>e64.b</td>
<td>47</td>
<td>no</td>
<td>47</td>
<td>2117.97</td>
<td>571</td>
<td>920</td>
<td>35</td>
<td>14</td>
<td>1.29</td>
<td>6.08</td>
</tr>
</tbody>
</table>

Legend:  
- **ObjBest**: values of objective function reported for each instance by cplex090  
- **Proof**: an indicator variable whether cplex has proven ‘ObjBest’ as optimal  
- **Ones**: total number of ‘ones’ in the solution vector  
- **RunTime**: runtime in seconds, reported by cplex  
- **n, m**: number of variables, number of constraints  
- **cdMax, rdMax**: maximum number of non-zero entries in a column, maximum number of non-zero entries in a row  
- **sp(%)**: a sparsity measure for the constraint matrix (100 * number_of_non-zeros/(n * m))  
- **gc(%)**: a measure completeness of the underlying graph (100 * number_of_edges/(n * (n – 1)))  

Notes:  
- **platform**: Intel-based processor, 3.2 GHz, 2 GB cache, under RedHat Linux  
- **cplex options**: the only option used is the value of timeout (set at 2112 seconds for all instances below)  
- **reductions**: all matrices that represent the benchmarks in the list below have been reduced to the extent possible, using standard column and row reduction techniques [22].
The two versions of cplex (versions 9.0 and 10.1), each with two options, -dfs as an alias for depth-first-search option, and -feas2 as an alias for an option that emphasizes optimality over feasibility give rise to six solver IDs: cplex090, cplex090-dfs, cplex090-feas, cplex101, cplex101-dfs, and cplex101-feas2. We report the results on four classes of isomorphs: in401_sp_LR, in401_sp_CLR, f51mb_350_LR, and f51mb_350_CLR. In addition, we also contrast the isomorph class in401_spCLR to a class of random instances in401_sp_RND.

Unless stated explicitly, each version of cplex is run on each instance in these classes without a timeout restriction; i.e. branch-and-bound solver has sufficient resources to prove that the returned value of ObjectiveBest is indeed the global optimum.

**Design Goals.** We articulate the goals of the five designs by first linking them to hypotheses that are to be addressed and resolved. We discuss the results in the subsection that follows.

**Design1 Hypothesis:** For the same reference instance, the isomorph class CLR is equivalent to the isomorph class LR. Inferences are based on observations of RunTime with solvers cplex090 and cplex101, applied to instances from the classes in401_sp_LR, in401_sp_CLR, f51mb_350_LR, and f51mb_350_CLR. For a preview of statistics summary, see Figure 1.

**Design2 Hypothesis:** The branch-and-bound performance of solvers cplex090 and cplex101, without options, are equivalent. Inferences are based on observations of RunTime with solvers cplex090 and cplex101, applied to instances from the classes in401_sp_LR, in401_sp_CLR, f51mb_350_LR, and f51mb_350_CLR. For a preview of statistics summary, see Figure 2.

**Design3 Hypothesis:** The branch-and-bound performance of any two solvers, formed from the list of six solvers above, are equivalent. Inferences are based on observations of RunTime with solvers cplex090, cplex090-dfs, cplex090-feas, cplex101, cplex101-dfs, and cplex101-feas2, applied to instances from in401_sp_CLR and f51mb_350_CLR. For a preview of statistics summary, see Figure 3.

**Design4 Hypothesis:** The fixed timeout performance of solvers cplex090 and cplex101, without options, are equivalent. Inferences are based on observations of ObjectiveBest with solvers cplex090 and cplex101, applied to timeout intervals of 16, 32, and 64 seconds, to instances from the class in401_sp_CLR. For a preview of statistics summary, see Figure 4.

**Design5:** Instances from the ‘random class’ in401_sp_RND induce variability in both RunTime and ObjectiveBest even when cplex is run on each instance without a timeout restriction. As a consequence, we cannot articulate a simple hypothesis as we did for instances in the isomorph classes. Also, due to large variability in ‘difficulty’ of solving a number of instances from the ‘random class’ in401_sp_RND, our computational resources are insufficient to resolve them. For a preview of statistics summary with cplex090 and cplex101, see Figure 5.

**Discussion of Results.** We first informally discuss the statistics summaries of five designs in Figures 1 – 5. A section that follows addresses the resolution of the hypothesis tests as formulated earlier for each of these designs.

In Designs 1 – 3 (in Figures 1 – 3), we run cplex as a branch-and-bound solver that reports the same optimum value for each instance in its class – what is being observed is the RunTime to find this optimum. The RunTime statistics for each class and each solver includes minimum (MinV), maximum (MaxV), median (MedV), mean (MeanV), standard deviation (StdV), number of samples (N), and Distribution. The runtime for each reference instance is listed in a separate column (RefV). We determine the reported distribution by running a combination of tests on the observed data: ranging from Cramer-Von Mises, Kolmogorov-Smirnov to $\chi^2$ goodness-of-fit-tests [24, 25]. We also plot empirical cumulative distribution functions (ECDFs) for classes of most interest (LR vs CLR), and a subset of all possible solver pairs (e.g. cplex090 vs. cplex101) on the CLR class. The bar charts illustrate values of RunTime values reported by specific solvers on instances from a given isomorph class.

Designs 1 – 3 emphasize the view of cplex as a branch-and-bound solver that terminates by proving an optimum before an externally imposed timeout. However, note that most instances shown in Table 1 time out within 5% of the externally imposed limit of 2112 seconds – and all we have to show for it is a single value of the variable ObjectiveBest. The purpose of the experimental Design 4 is to produce a distribution of ObjectiveBest at predetermined timeout intervals. To get a distribution of ObjectiveBest on such instances, at a cost no greater than the cost of a single run with timeout value of 2112, we now consider instances from the classes in401_sp_CLR and f51mb_350_CLR, pick a timeout value $T_{out}$ from a set of {16, 32, 64} seconds, and run cplex with a timeout of $T_{out}$ on the reference and all 32 instances. The random variable we observe in this design is the value of ObjectiveBest. Note that for value of $T_{out} = 64$, the total runtime of the experiments with (1+32) instances is 2112 seconds – however, we now may have 33 distinct values of ObjectiveBest in its distribution!
Design 4/Figure 4: The purpose of this design is to produce a distribution of ObjectiveBest at predetermined timeout intervals with solvers `cplex090` and `cplex101`, applied at timeout intervals of 16, 32, and 64 seconds, to instances from the class `in401_sp_CLR`. The most noticeable feature of these results is that there is no appreciable difference between the two solvers, even at the timeout of 64 seconds. The most interesting part is the fact that an optimum value of 77418 has been reached by both solvers already in 64 seconds.

Design 5/Figure 5: Currently, one of the most common approaches to evaluate the runtime performance of algorithms (by computer scientists) is to test them on a large number of ‘random instances’. The purpose of this design is to illustrate some of the shortcomings for this approach. All instances from the designated ‘random class’ `in401_sp_RND` have 500 variables, 1000 constraints, and as ‘similar’ distributions of variables over constraints as the generator that produced them can support – a non-trivial problem in itself. In contrast, instances from the isomorph class `in401_sp_CLR` also have 500 variables, 1000 constraints – but all are isomorphs of the same reference instance. The striking difference between the two classes is demonstrated in the two Runtime-vs-ObjectiveBest diagrams in Figure 5: with `in401_sp_CLR`, the only random variable we can observe is RunTime, whereas with `in401_sp_RND`, both RunTime and ObjectiveBest are random variables. Moreover, due to large variability in ‘difficulty’ of solving a number of instances from the ‘random class’ `in401_sp_RND`, the resources we need to solve them are much more unpredictable than for the instances from the class `in401_sp_CLR`. In summary, to test the performance of two or more solvers on a class of ‘random instances’, we cannot use the relatively simple hypothesis tests we proposed for classes of ‘isomorph instances’.

Resolution of Hypotheses Tests. Statistics summarized in Figures 1 – 4 provide an initial basis for comparisons of instance classes and solvers. We now proceed to resolve the four hypotheses stated initially in this section for each of the four designs.
**Design1 Resolution:** With solver `cplex090`, RunTime values are observed on 32 instances generated using rule LR, and on 32 instances generated using rule CLR. Similarly, with solver `cplex101`, RunTime values are observed for another set of 64 instances, generated using the two rules. Such an arrangement of solvers and rules constitutes a balanced, $2 \times 2$ factorial experiment. Since diagnostic plots indicate that RunTime distributions are roughly normally distributed with constant variance, an analysis of variance (ANOVA) is carried out to investigate the effects of solver and rule. The ANOVA table below indicates that while there is a highly significant solver effect, there is no evidence of any difference in RunTime mean due rule.

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>Mean Square</th>
<th>F</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule</td>
<td>1</td>
<td>1474</td>
<td>1474</td>
<td>0.1</td>
<td>0.7875</td>
</tr>
<tr>
<td>solver</td>
<td>1</td>
<td>1416808</td>
<td>1416808</td>
<td>70.1</td>
<td>&lt; .0001</td>
</tr>
<tr>
<td>rule*solver</td>
<td>1</td>
<td>36565</td>
<td>36565</td>
<td>1.8</td>
<td>0.1810</td>
</tr>
<tr>
<td>Error</td>
<td>124</td>
<td>2505820</td>
<td>20208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>127</td>
<td>3960669</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The analysis above pertains to instance class `in401_sp_LR`. A similar experiment involving another 128 runtimes was carried out with the instance class `f51mb_350_LR`. The distributions of these four samples are decidedly non-normal, displaying a long right tail, with some observations truncated at the timeout of 2112 seconds so that the $F$-test from ANOVA is not appropriate. The log-rank test, a non-parametric statistical procedure commonly used for reliability or survival analysis, may be used to investigate the hypothesis that RunTime distributions, under the LR and CLR, are the same. For both solvers, we find no significant difference in RunTime between the two rules: $\chi^2 = 0.0187, p = 0.8913, df = 1$ for `cplex090`, $\chi^2 = 1.08, p = 0.2976, df = 1$ for `cplex101`. The medians from the two distributions are similar for the two rules: $\text{MedV}_{LR} = 112$, $\text{MedV}_{CLR} = 112$, with `cplex090` and $\text{MedV}_{LR} = 112.6$, $\text{MedV}_{CLR} = 159.5$ with `cplex101`. According to the log-rank test, these differences are consistent with chance variability among instances and are not due to the rule used to generate them. Other non-parametric statistical procedures such as the Wilcoxon test for comparing distributions under truncated sampling, lead to the same conclusion regarding no differences due to rule.

**Design2 Resolution:** With two solvers, `cplex090` and `cplex101`, RunTime values are again observed independently and with truncation at $t = 2112$ seconds on 62 randomly selected instances from two classes: first on `in401_sp_CLR`, followed by `f51mb_350_CLR`. To investigate the hypothesis that the two solvers have the same RunTime distributions for the conceptual population of instances, the log-rank test is used again. The re-
Design3 observes RunTime values from 32 instances within each of six solver classes, arranged in a 2 × 3 factorial layout, with the two-level factor solver and a second factor, factor2 taking three values: none, dfs and feas2. Here, the values none, dfs and feas2 refer to solver configuration with no options (default), and options -dfs, -feas2 as explained in the earlier section. The solver combinations are observed under two instance classes of isomorphs; in401_sp_CLR and f51mb_350_CLR, which are analyzed separately.

In the experiment with in401_sp_CLR, an ANOVA indicates a highly significant interaction between solver and factor2 ($F = 169.1, p < 0.0001, df = 2, 176$). The six solver means are given in the table below. Using the Tukey-Kramer adjustment to control the experimentwise error rate at .05 in all pairwise comparisons among the means, all 15 pairs differ significantly, with one minor exception, the difference between none and dfs using solver cplex090, which is nearly significant. These significant differences indicate that there are both solver effects and factor2 effects.
ObjectiveBest statistics for isomorph class in401_sp_CLR.

Here, branch&bound times out at 16, 32, 64 seconds and returns the best objective value for each instance. See Fig. 2 for RunTime statistics observed with cplex090 and cplex101 on the same class, executed without time-out constraint.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Class</th>
<th>RefV</th>
<th>MinV</th>
<th>MaxV</th>
<th>MedV</th>
<th>MeanV</th>
<th>StdV</th>
<th>N</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplex090@16 in401_sp_CLR</td>
<td>65086</td>
<td>59852</td>
<td>71797</td>
<td>65992</td>
<td>66080</td>
<td>3721</td>
<td>32</td>
<td>uniform</td>
<td></td>
</tr>
<tr>
<td>cplex101@16 in401_sp_CLR</td>
<td>65662</td>
<td>59196</td>
<td>73626</td>
<td>65964</td>
<td>65946</td>
<td>3782</td>
<td>32</td>
<td>uniform</td>
<td></td>
</tr>
<tr>
<td>cplex090@32 in401_sp_CLR</td>
<td>66826</td>
<td>60658</td>
<td>75114</td>
<td>69240</td>
<td>68548</td>
<td>3782</td>
<td>32</td>
<td>uniform</td>
<td></td>
</tr>
<tr>
<td>cplex101@32 in401_sp_CLR</td>
<td>73626</td>
<td>59196</td>
<td>75114</td>
<td>68946</td>
<td>68217</td>
<td>3593</td>
<td>32</td>
<td>uniform</td>
<td></td>
</tr>
<tr>
<td>cplex090@64 in401_sp_CLR</td>
<td>66826</td>
<td>64451</td>
<td>77418</td>
<td>70260</td>
<td>69829</td>
<td>3450</td>
<td>32</td>
<td>uniform</td>
<td></td>
</tr>
<tr>
<td>cplex101@64 in401_sp_CLR</td>
<td>73626</td>
<td>64377</td>
<td>77418</td>
<td>69193</td>
<td>69219</td>
<td>2813</td>
<td>32</td>
<td>normal</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Timeout and branch&bound experiments with isomorph class in401_sp_CLR and two solvers.

and the effects of one factor depend on the level of the other factor. One characterization of the interaction of these factors is that the solver effect varies across levels of factor2; it is more pronounced for the feas2 level of factor2 than for dfs or none, as may be seen by inspection of difference Diff shown in the bottom row of the table below.

<table>
<thead>
<tr>
<th>Instance class</th>
<th>in401_sp_CLR</th>
<th>Factor2 (solver options)</th>
<th>cplex solver</th>
<th>none</th>
<th>dfs</th>
<th>feas2</th>
</tr>
</thead>
<tbody>
<tr>
<td>090</td>
<td>666.1</td>
<td>376.2</td>
<td>416.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>843.3</td>
<td>925.2</td>
<td>1510.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diff</td>
<td>177.2</td>
<td>349.0</td>
<td>1094.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inspection of diagnostic plots of residuals versus predicted values, not included here, indicates inhomogeneity of variance in RunTime values; the larger the RunTime, the more variability, with the variance increasing linearly with the mean. A square root transformation stabilizes the variance and the statistics above are computed from an analysis of the transformed data.

A similar analysis may be carried out for the data observed from the f51mb_350_CLR class, though some accomodation would have to be made to accomodate for the truncation due to timeout. Descriptively, the table of RunTime means suggests a different interaction between solver and factor2 for the f51mb_350_CLR class than was observed for the in401_sp_CLR class. In particular, the solver effect is most pronounced for the feas2 level of factor2 for both instance classes, in401_sp_CLR and f51mb_350_CLR.

<table>
<thead>
<tr>
<th>Instance class</th>
<th>f51mb_350_CLR</th>
<th>Factor2 (solver options)</th>
<th>cplex solver</th>
<th>none</th>
<th>dfs</th>
<th>feas2</th>
</tr>
</thead>
<tbody>
<tr>
<td>090</td>
<td>231.6</td>
<td>388.2</td>
<td>113.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>408.3</td>
<td>440.8</td>
<td>315.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diff</td>
<td>176.7</td>
<td>52.6</td>
<td>202.6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In an analysis of all twelve combinations of solver, factor2 and instance class, this would be classified as a three-factor interaction, though, for simplicity of exposition, such a three-factor analysis is not undertaken here.

Design4 Resolution: As shown in Figure 2, the main purpose of Design4 is to produce a distribution of ObjectiveBest at predetermined timeout intervals with solvers cplex090 and cplex101. An independent samples t-statistics of ObjectiveBest reveals no significant difference between the two solvers, even at the timeout of 64 seconds. This is in marked contrast to the resolution of Design2, where we report a significant difference between the two solvers. The
RunTime statistics for an isomorph class in401_sp_CLR and a ‘random class’ in401_sp_RND.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Class</th>
<th>RefV</th>
<th>MinV</th>
<th>MaxV</th>
<th>MedV</th>
<th>MeanV</th>
<th>StdV</th>
<th>N</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplex090</td>
<td>in401_sp_CLR</td>
<td>865</td>
<td>407</td>
<td>957</td>
<td>638</td>
<td>666</td>
<td>133</td>
<td>32</td>
<td>uniform</td>
</tr>
<tr>
<td>cplex101</td>
<td>in401_sp_CLR</td>
<td>841</td>
<td>625</td>
<td>1316</td>
<td>816</td>
<td>843</td>
<td>142</td>
<td>32</td>
<td>normal</td>
</tr>
<tr>
<td>cplex090**</td>
<td>in401_sp_RND</td>
<td>541</td>
<td>455</td>
<td>1058</td>
<td>969</td>
<td>894</td>
<td>177</td>
<td>32</td>
<td>incomplete</td>
</tr>
<tr>
<td>cplex101**</td>
<td>in401_sp_RND</td>
<td>696</td>
<td>602</td>
<td>1058</td>
<td>1058</td>
<td>979</td>
<td>139</td>
<td>32</td>
<td>incomplete</td>
</tr>
</tbody>
</table>

**Due to system constraints, a timeout of 1056 seconds must be imposed to complete branch&bound runs with the ‘random class’.

Figure 5: Contrasting branch&bound two-solver experiments with an isomorph class and a ‘random class’.

message is thus clear: we need to allow for a larger timeout value for each solver if we are to detect a significant difference between solvers by only observing values of ObjectiveBest at fixed timeout intervals.

6. CONCLUSIONS

This paper is not about ‘the best BCSP solver’. Rather, it is a case study of how the scientific method can be applied to comparing the performance of not only BCSP solvers but also other solvers that address NP-hard problems. Reproducibility is one of the main principles of the scientific method, and refers to the ability of a test or experiment to be accurately reproduced, or replicated, by someone else working independently.

This paper demonstrates that a class of instance isomorphs can induce solver RunTime variability that may span orders of magnitude. We may thus experimentally observe RunTime distributions, produced by different solvers on the same instance class, that may range from uniform, normal, exponential, to heavy-tail. Such observations not only provide a reliable mechanism to measure, with statistical significance, differences between two or more solvers, they also provide a method to reliably design and improve a new generation of combinatorial solvers.

See http://www.cbl.ncsu.edu/xBed/ for more information.

Acknowledgments. This work benefited a great deal from discussions, over the years, with Matt Stallmann and Xiao Yu Li. In particular, Matt Stallmann helped with the scripts that facilitated invocations of cplex. Eric Sills, from the NCSU High Performance Computing (HPC) facility with fast dedicated processors, assisted in a number of ways to maintain continuous access to computing resources and its environment. We also thank Peter Notebaert for providing the background on the origins and citations related to the .lpx format, and Y. Guo for readily sharing reprints of his papers and the 500-instance benchmark set that now has a new life in a number of settings, all in the .lpx format.

7. REFERENCES

In order to capture any instance of a BCSP in an easy to read and an easy to understand form, we advocate the familiar 0/1 integer program (IP) formulation that naturally expresses the constraints as well as the goals of the optimization task. The .lpx format as illustrated by the two small examples below is a subset of the cplex format [10] that is also readable by the public-domain solver lp_solve [11, 12]. However, also note that the lp formats of these two solvers are not equivalent in general!

We keep the emphasis on keeping the extension .lpx as a reminder that all variable names are always prefixed with ‘x’, followed by a number in range [1, n] – a feature we rely on to post-process the respective solver outputs. Unfortunately, the acronym ‘lpx’ is overloaded, and the number of hits from a web search engine, in response to a query about lpx, is huge and none of the current listing have the context that is relevant. May be with time, a search on ‘.lpx’ will point to examples such as the ones shown below.

A. SMALL EXAMPLES IN .LPX FORMAT

The two small examples in the .lpx format below illustrate all constraint categories we may find in a BCSP instance. Both examples will be read by both lp_solve as well as by cplex and both solvers will produce correct results.

In the first file, the constraint lines are labeled explicitly, a feature that is useful for a reference instance. However, as the second example shows, the constraint lines need not be labeled – a feature we find convenient when writing out an isomorph instance (in which rows and variables are randomly permuted by a morphing tool).

A. SMALL EXAMPLES IN .LPX FORMAT

The two small examples in the .lpx format below illustrate all constraint categories we may find in a BCSP instance. Both examples will be read by both lp_solve as well as by cplex and both solvers will produce correct results.

In the first file, the constraint lines are labeled explicitly, a feature that is useful for a reference instance. However, as the second example shows, the constraint lines need not be labeled – a feature we find convenient when writing out an isomorph instance (in which rows and variables are randomly permuted by a morphing tool).

A. SMALL EXAMPLES IN .LPX FORMAT

The two small examples in the .lpx format below illustrate all constraint categories we may find in a BCSP instance. Both examples will be read by both lp_solve as well as by cplex and both solvers will produce correct results.

In the first file, the constraint lines are labeled explicitly, a feature that is useful for a reference instance. However, as the second example shows, the constraint lines need not be labeled – a feature we find convenient when writing out an isomorph instance (in which rows and variables are randomly permuted by a morphing tool).