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 binate 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* $\mathcal{S}^X(x)$ while the *reliability* or *survival function* $\mathcal{R}^Y(y)$ 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:

Section2 introduces several classes of the Boolean constraint satisfaction problem (BCSP) under the 0/1 integer pro-

ExpCS '07, 13-14 June 2007, San Diego, CA

gram (IP) formulation, including examples of transformations between related unate and binate minimization and maximization instances.

Section3 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*.

Section4 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 .lpx 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.

Section5 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 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.

$$\max \ \mathrm{w^T x} \quad \text{subject to} \quad \mathrm{Ax} \geq \mathrm{b}, \quad \mathrm{x} \in \{0,1\}$$

and

$$\min \ \mathrm{w^T x} \quad \text{subject to} \quad \mathrm{Ax} \geq \mathrm{b}, \quad \mathrm{x} \in \{0,1\}$$

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 $\pm$ signs within the constraint, as we explain next.

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

unate-positive, equivalent to the set cover constraint:

$$\sum_{i \in I_p} (+x_i) >= +1$$

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

unate-negative, equivalent to the set packing constraint:

$$\sum_{j \in I_n} (-x_j) >= -1$$

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 $-x_1 - x_2 - x_3 >= -1$ is equivalent to the following pair-wise constraints:
$-x_1 - x_2 >= -1$, $-x_1 - x_3 >= -1$, $-x_2 - x_3 >= -1$.

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

$$\sum_{i \in I_p} (+x_i) + \sum_{j \in I_n} (-x_j) >= +1 - |I_n|$$

If $I_p \in \emptyset$, the constraint $\sum_{j \in I_n} (-x_j) >= 1 - |I_n|$ 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

$$\begin{array}{lccccccccc}
c1: & & +x_2 & +x_3 & +x_4 & & & & >= & +1 \\
c2: & & +x_2 & & & +x_5 & +x_6 & & >= & +1 \\
c3: & & & & & +x_5 & +x_6 & +x_7 & >= & +1 \\
c4: & & & +x_3 & & & & +x_7 & >= & +1 \\
c5: & +x_1 & & & +x_4 & & & +x_7 & >= & +1 \\
c6: & +x_1 & & +x_3 & & & +x_6 & & >= & +1
\end{array}$$

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

$$\begin{array}{lrrrrrrrrr}
c1: & & & -x_3 & & -x_5 & & >= & -1 \\
c2: & & & & -x_4 & -x_5 & & >= & -1 \\
c3: & & -x_2 & & & & -x_7 & >= & -1 \\
c4: & & & & -x_4 & & -x_6 & >= & -1 \\
c5: & -x_1 & & & & -x_5 & & >= & -1 \\
c6: & -x_1 & -x_2 & & & & & >= & -1
\end{array}$$

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

$$\begin{array}{lrrrrrrrrr}
c1: & & +x_2 & +x_3 & +x_4 & & & >= & +1 \\
c2: & & -x_2 & & & -x_5 & -x_6 & >= & -2 \\
c3: & & & & & +x_5 & +x_6 & -x_7 & >= & 0 \\
c4: & & & -x_3 & & & & +x_7 & >= & 0 \\
c5: & -x_1 & & & -x_4 & & & -x_7 & >= & -1 \\
c6: & -x_1 & & -x_3 & & & -x_6 & & >= & -1
\end{array}$$

A weighted binate instance (obj=min).
ObjectiveOpt 22 ; Solution 0100000
This instance is generated from the binate instance above by simply changing the objective from 'max' to 'min'.

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
@VariablePermutationPairs (isomorph,reference)
1,1 2,2 3,3 4,4 5,5 6,6 7,7 0,0

Max

$+21x_1 + 22x_2 + 23x_3 + 25x_4 + 26x_5 + 27x_6 + 29x_7$

st

$$\begin{array}{rrrrr}
-x_3 & -x_1 & -x_6 & >= & -1 \\
-x_1 & -x_4 & -x_7 & >= & -1 \\
-x_5 & -x_2 & -x_6 & >= & -2 \\
+x_3 & +x_2 & +x_4 & >= & +1 \\
+x_7 & -x_3 & & >= & 0 \\
-x_7 & +x_6 & +x_5 & >= & 0
\end{array}$$

It is clear by inspection that no permutation of variables took place in the isomorph_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.

On the other hand, column or variable permutation also took place in the isomorph_CLR below: if we know the permutation, the effort to verify that new new instance is in fact the isomorph of the reference is relatively simple.

A weighted binate instance (obj=max) – isomorph_CLR.
ObjectiveOpt 100
Solution 1100011
@VariablePermutationPairs (isomorph,reference)
1,3 2,1 3,2 4,5 5,6 6,4 7,7 0,0

Max

$+22x_1 + 23x_2 + 21x_3 + 27x_4 + 25x_5 + 26x_6 + 29x_7$

st

$$\begin{array}{rrrrr}
-x_3 & -x_7 & -x_5 & >= & -1 \\
+x_1 & +x_2 & +x_5 & >= & +1 \\
-x_2 & +x_7 & & >= & 0 \\
-x_3 & -x_2 & -x_4 & >= & -1 \\
-x_6 & -x_1 & -x_4 & >= & -2 \\
+x_4 & +x_6 & -x_7 & >= & 0
\end{array}$$

Since one may be tempted to dismiss LR-isomorphs as trivial, we bring forward a 350-variables example described in more detail later. The name of the isomorph class is f51mb_350_B_40v_20_20_LR, and its reference instance is in cnf-format, i00.cnf. Since *cplex* takes files in .lpx format, we must translate it. The act of translation alone can induce instances in LR-class, depending on the implementation of the translator program. Let the first translator produce an instance in the 'reference order' given by the instance in the .cnf format and let two more translators rely on some hashing schemes that result in instances having row orders that are both different from the row order of the reference instance. Also, the order in which the variable appear in each row may be different. Such instances can be found in the class of 1+32 instances in the web-archive under the directory f51mb_350_B_40v_20_20_LR, say i00.lpx, i06.lpx, and i17.lpx. Upon invoking *cplex* 9.0 on each of these instance, we get a solution and a proof of optimality, however runtimes differ dramatically, despite running on the same dedicated CPU:

translator	instance	Obj_opt	RunTime (secs)
T1	i00.lpx	24	114.91
T2	i06.lpx	24	82.55
T3	i17.lpx	24	1801.86

These instances under f51mb_350_B_40v_20_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 re-ordered with relative ease into an almost equivalent if not equivalent order[1]. 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 .lpx format (with translators to/from .lpx), (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) hierarchical archives of 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 *ObjectiveBest*. 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 hypothesees 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 .lpx 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*_cliq that have been derived from from these instances will have complement graphs that are much less 'internconnected' – 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 test4.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 (binate): Instances rot.b, alu4, e64.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 set 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 *cliq and *cliq1 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) *f51mb_350_CLR*, based on a binate set cover block composition reference with 350 variables and 413 constraints. The name *f51mb_350_CLR* is an alias for the class

f51mb_0350_B_0040_20_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 *ObjectiveBest* as reported by different BCSP solvers when applied to several instance classes.

[1]Such 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 *cplex*090.

Legend:

ObjBest:	values of objective function reported for each instance by *cplex*090
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 completness of the underlying graph (100 * number_of_edges/$(n * (n-1))$ (number of unique edges is counted after expanding each constraint into a clique)

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) (experiments with options may produce results that better of worse as the ones shown)
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].

Dir	Instance	ObjBest	Proof	Ones	RunTime	n	m	cdMax	rdMax	sp(%)	gc(%)
	in101_sc	189316	no	57	2112.85	1000	500	50	77	5.55	68.82
min	in201_sc	547921	no	56	2114.91	1000	1000	100	79	5.59	84.99
(unate)	in401_sc	593034	no	68	2112.52	500	1000	100	45	5.72	85.57
set	in501_sc	589992	no	54	2116.38	1500	1000	150	157	7.85	91.84
cover	in601_sc	954508	no	72	2118.01	1500	1500	150	111	5.60	90.88
	in101_sp	64408	no	19	2116.68	1000	500	50	77	5.55	68.82
max	in201_sp	77596	no	13	2117.8	1000	1000	100	79	5.59	84.99
(unate)	in401_sp	77418	yes	12	866.87	500	1000	100	45	5.72	85.57
set	in413_sp	74435	no	12	1057.95	500	1000	100	46	5.55	83.65
packing	in501_sp	76906	no	15	2118.39	1500	1000	150	157	7.85	91.84
	in601_sp	98805	no	15	2119.45	1500	1500	150	111	5.60	90.88
	dsjc125_is1	34	yes	34	17.7	125	736	23	2	1.60	9.50
max	frb30-15-1	27	no	27	2118.49	450	17827	122	2	0.44	17.65
indep.	frb30-15-2	27	no	27	2118.08	450	17874	116	2	0.44	17.69
set	frb30-15-3	28	no	28	2118.05	450	17809	122	2	0.44	17.63
	frb30-15-4	28	no	28	2118.68	450	17831	110	2	0.44	17.65
	frb30-15-5	28	no	28	2119.11	450	17794	128	2	0.44	17.61
	dsjc125_cliq1	4	yes	4	0.53	125	7014	119	2	1.60	90.50
	frb30-15-1_cliq1	15	no	15	2120.41	450	83198	407	2	0.44	82.35
max	frb30-15-2_cliq1	15	no	15	2120.02	450	83151	404	2	0.44	82.31
clique	frb30-15-3_cliq1	15	no	15	2118.98	450	83216	400	2	0.44	82.37
	frb30-15-4_cliq1	15	no	15	2118.5	450	83194	401	2	0.44	82.35
	frb30-15-5_cliq1	15	no	15	2120.63	450	83231	403	2	0.44	82.39
	in201_cliq	7265040	yes	361	3.56	1000	74959	572	2	0.20	15.01
	in201_cliq1	361	yes	361	235	1000	74959	572	2	0.20	15.01
unate	ex5.pi	36	yes	36	19.44	974	686	71	74	2.85	16.79
cover	test4.pi	105	no	105	2117.77	5117	1435	54	159	1.36	10.07
min	rot.b	84	yes	84	6.34	887	1257	158	79	1.23	7.29
binate	alu4	32	yes	32	38.5	481	592	165	74	3.46	20.16
cover	e64.b	47	no	47	2117.97	571	920	35	14	1.29	6.08
	dsjc_0125	91	yes	91	20.97	125	736	23	2	1.60	9.50
	dsjc_0250	182	no	182	2113.14	250	1472	23	2	0.80	4.73
min	dsjc_0250_0100	183	no	183	2112.98	250	1572	24	2	0.80	5.05
vertex	dsjc_0500	366	no	366	2111.15	500	2944	23	2	0.40	2.36
cover	dsjc_0500_0200	368	no	368	2112.45	500	3344	26	2	0.40	2.68
blocks	dsjc_1000	736	no	736	2126.75	1000	5888	23	2	0.20	1.18
	dsjc_1000_0400	754	no	754	2118.36	1000	7088	29	2	0.20	1.42
	dsjc_2000	1480	no	1480	2132.11	2000	11776	23	2	0.10	0.59
	dsjc_2000_0800	1511	no	1511	2116.64	2000	14976	30	2	0.10	0.75
	f51mb	12	yes	12	0.26	175	187	49	33	7.62	29.37
	f51mb_0350	24	yes	24	73.54	350	374	49	33	3.81	14.64
min	f51mb_0350_B_0040_20_20	24	yes	24	114.89	350	413	73	33	4.34	26.67
binate	f51mb_0525	36	no	36	2119.42	525	561	49	33	2.54	9.75
cover	f51mb_0525_B_0060_40_20	36	no	36	2118.11	525	660	94	53	3.45	37.82
blocks	f51mb_0700	48	no	48	2120.5	700	748	49	33	1.91	7.31
	f51mb_0700_B_0080_60_20	48	no	48	2118.25	700	925	112	73	3.11	50.13
	f51mb_1400	96	no	96	2120.55	1400	1496	49	33	0.95	3.65
	f51mb_1400_B_0160_80_80	96	no	96	2117.76	1400	2009	271	129	2.16	40.50

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_sp_CLR* 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 cplex090and 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_CLR*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 cplex090and cplex101, without options, are equivalent.* Inferences are based on observations of *ObjectiveBest* with solvers *cplex090* and *cplex101*, applied at 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 χ^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 barcharts 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!

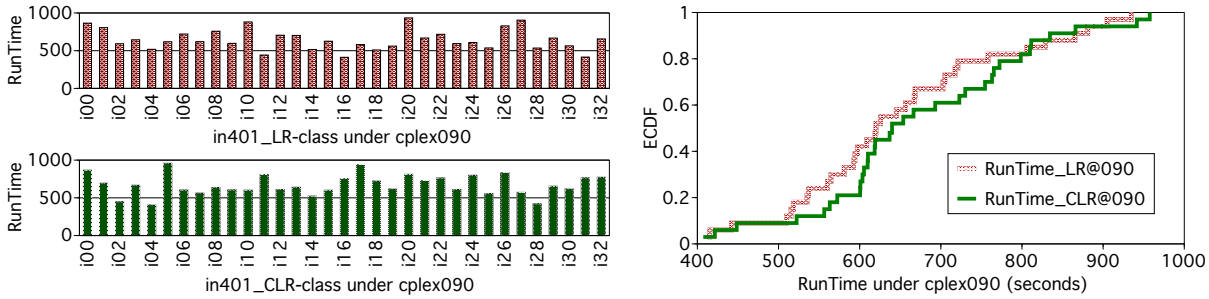
Design1/Figure1: Solvers *cplex090*, *cplex101*are applied to instances from *in401_sp_LR*, *in401_sp_CLR*, *f51mb_350_LR*, and *f51mb_350_CLR*. There are notable differences between statistics of *in401_sp_LR*and *f51mb_350_LR* regardless of the solver, and the differences between *in401_sp_CLR* and *f51mb_350_CLR* are similarly notable. Both *f51mb_350_LR* and *f51mb_350_CLR* exhibit heavy-tail distribution. However, differences between *in401_sp_LR*, *in401_sp_CLR* under the same solver are smaller than the differences between the solvers themselves, whether both solvers are applied to *in401_sp_LR* or *in401_sp_CLR*. Differences between solvers will be analyzed in subsequent designs.

Design2/Figure2: Solvers *cplex090* and *cplex101*are applied to instances from *in401_sp_CLR* and *f51mb_350_CLR*. A mere inspection of the respective barcharts for *in401_sp_CLR* class reveals non-trivial differences between the two solvers, with *cplex090* emerging as the dominating solver. The dominance of *cplex090* is suggested also by inspection of the barcharts and statistics for *f51mb_350_CLR* class.

Design3/Figure3: Solvers *cplex090*, *cplex090-dfs*, *cplex090-feas*, *cplex101*, *cplex101-dfs*, and *cplex101-feas2*, are applied to instances from *in401_sp_CLR* and *f51mb_350_CLR*. The differences between solvers, with and without options are striking, for both the *in401_sp_CLR* and *f51mb_350_CLR* class. Perhaps remarkably, the same solver, *cplex090-feas*, appears to dominate all other solver on *both* classes. We postpone the discussion whether this domination has statistical significance until the next section.

***RunTime* statistics for isomorph classes in401_sp_LR and in401_sp_CLR.**

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090	in401_sp_LR	865	412	935	620	639	133	32	uniform
cplex090	in401_sp_CLR	865	407	957	638	666	133	32	uniform
cplex101	in401_sp_LR	839	608	1236	858	883	158	32	uniform
cplex101	in401_sp_CLR	841	625	1316	816	843	142	32	normal



***RunTime* statistics for isomorph classes f51mb_350_LR and f51mb_350_CLR.**

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090	f51mb_350_LR	115	60.4	2115	110	256	458	32	heavy-tail
cplex090	f51mb_350_CLR	115	71.3	2118	127	232	393	32	heavy-tail
cplex101	f51mb_350_LR	86.6	51.1	2116	113	313	520	32	heavy-tail
cplex101	f51mb_350_CLR	87.3	53.5	2117	159	408	619	32	heavy-tail

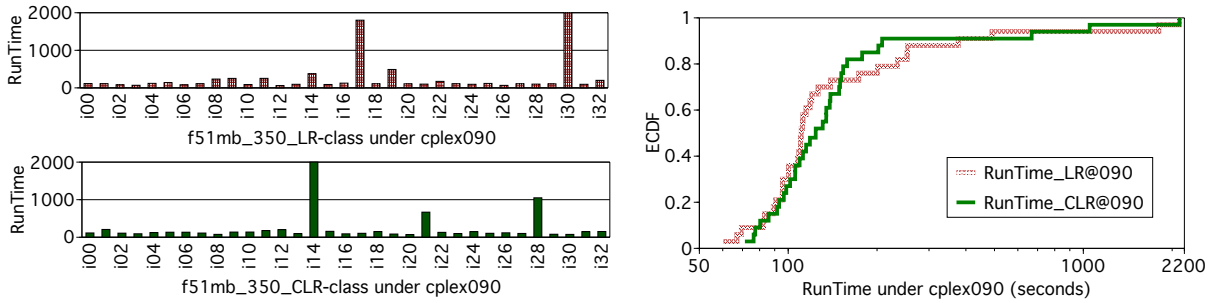


Figure 1: *Branch&bound* experiments with LR and CLR classes of isomorphs.

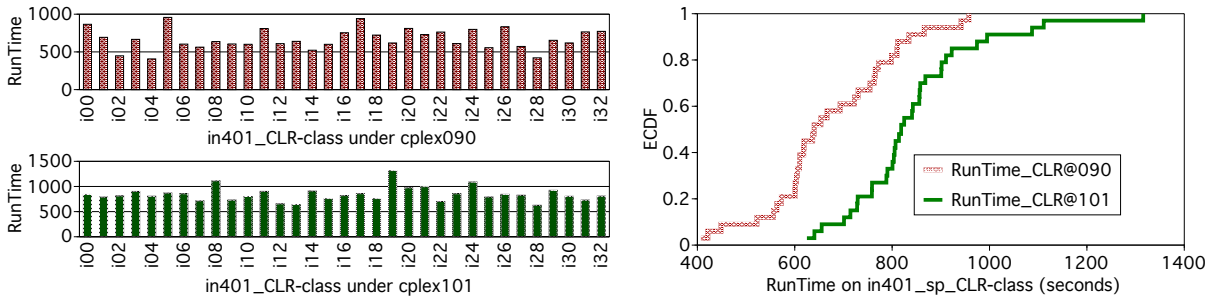
Design4/Figure4: The purpose of this design is to produce a distribution of *ObjectiveBest* at predetermined timeout intervals with solvers *cplex090* and *cplex101*, applied at timeout intervals of 16, 32, and 64 seconds, to instances from the class *in401_sp_CLR*. The most noticeable feature of these 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: on two isomorphs with *cplex090*, and one isomorph with *cplex101*. Hoever, for the branch-and-bound solver to *prove* the value of 77418 is indeed an optimum, *cplex090* takes on an average of 666 seconds, while *cplex101* takes on an average of 843 seconds (see statistics in Figure 2).

Design5/Figure5: 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.

***RunTime* statistics for isomorph class in401_sp_CLR.**

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090	in401_sp_CLR	865	407	957	638	666	133	32	uniform
cplex101	in401_sp_CLR	841	625	1316	816	843	142	32	normal



***RunTime* statistics for isomorph class f51mb_350_CLR.**

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090	f51mb_350_CLR	115	71.3	2118	127	232	393	32	heavy-tail
cplex101	f51mb_350_CLR	87.3	53.5	2117	159	408	619	32	heavy-tail

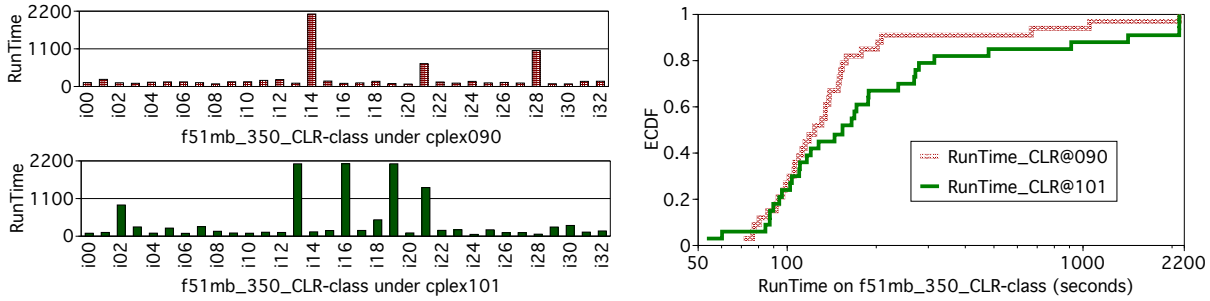


Figure 2: *Branch&bound* experiments with two CLR isomorph classes and two solvers.

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, 2×2 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*.

Source	DF	Sum of Squares	Mean Square	F	p-value
rule	1	1474	1474	0.1	0.7875
solver	1	1416808	1416808	70.1	$< .0001$
rule*solver	1	36565	36565	1.8	0.1810
Error	124	2505820	20208		
Total	127	3960669			

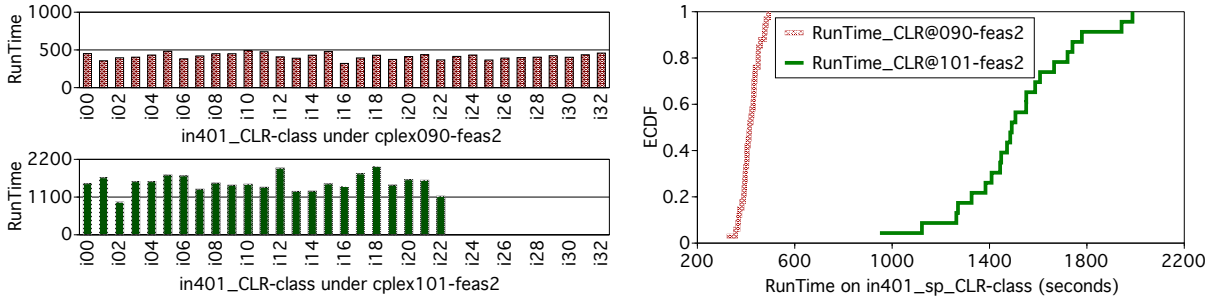
The analysis above pertains to instance class *in401_sp_LR*. A similar experiment involving another 128 runtimes was carried out with the instance class *f51mb_350_LR*. The distributions of these four samples are decidedly non-normal, displaying a long right tail, with some observations truncated at 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 distribution are similar for the two rules: $\text{MedV}_{LR} = 110.4$, $\text{MedV}_{CLR} = 127.4$ 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 32 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-

***RunTime* statistics for isomorph class in401_sp_CLR.**

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090	in401_sp_CLR	865	407	957	638	666	133	32	uniform
cplex101	in401_sp_CLR	841	625	1316	816	843	142	32	normal
cplex090-dfs	in401_sp_CLR	798	411	748	574	576	85.7	32	uniform
cplex101-dfs	in401_sp_CLR	678	592	1200	904	925	149	32	normal
cplex090-feas2	in401_sp_CLR	451	321	493	413	416	38.9	32	uniform
cplex101-feas2	in401_sp_CLR	1491	950	1987	1496	1510	247	22	normal



***RunTime* statistics for isomorph class f51mb_350_CLR.**

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090	f51mb_350_CLR	115	71.3	2118	127	232	393	32	heavy-tail
cplex101	f51mb_350_CLR	87.3	53.5	2117	159	408	619	32	heavy-tail
cplex090-dfs	f51mb_350_CLR	102	60.1	2117	225	388	526	32	near-exponential
cplex101-dfs	f51mb_350_CLR	179	89.2	2116	227	441	585	32	exponential
cplex090-feas2	f51mb_350_CLR	115	49.2	446	94.2	113	69.8	32	near-exponential
cplex101-feas2	f51mb_350_CLR	99.1	58.1	2118	127	316	591	32	heavy-tail

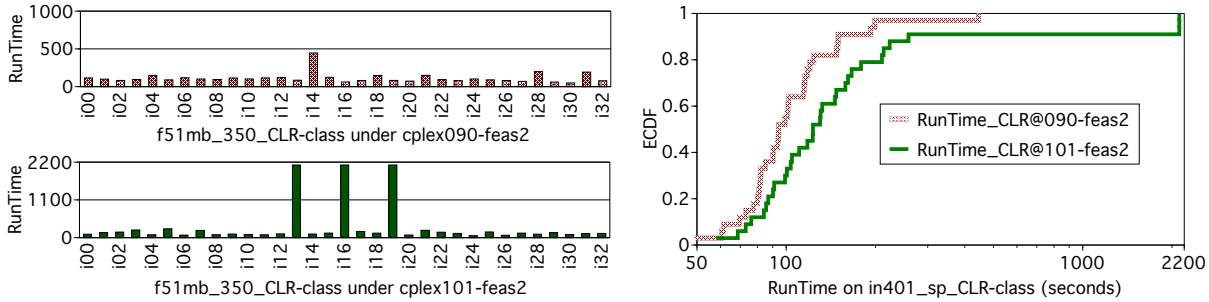


Figure 3: *Branch&bound* experiments with two CLR isomorph classes and six solvers.

sults indicate a highly significant difference ($\chi^2 = 19.2, p < 0.0001$) for instance class *in401_sp_CLR*, but nearly significant difference for instance class *f51mb_350_CLR* ($\chi^2 = 3.32, p = 0.0683$). In both cases, solver *cplex090* runtimes are generally lower, as seen in the table below, which gives 95% confidence intervals for the median runtime among the four populations of instances.

Instance class	*cplex* solver	sample median	approx. 95% confidence interval
in401_sp_CLR	090	638.4	(605.3, 730.1)
in401_sp_CLR	101	816.3	(790.3, 857.7)
f51mb_350_CLR	090	127.43	(105.6, 148.8)
f51mb_350_CLR	101	159.35	(110.4, 237.3)

Design3 Resolution: 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 wth cplex090 and cplex101 on the same class, executed without time-out constraint.

Solver	Class	RefV	MinV	MaxV	MedV	MeanV	StdV	N	Distribution
cplex090@16	in401_sp_CLR	65086	59852	71797	65992	66080	3721	32	uniform
cplex101@16	in401_sp_CLR	65662	59196	73626	66676	65964	3782	32	uniform
cplex090@32	in401_sp_CLR	66826	60658	75114	69240	68548	3351	32	uniform
cplex101@32	in401_sp_CLR	73626	59196	75114	68946	68217	3593	32	uniform
cplex090@64	in401_sp_CLR	66826	64451	77418	70260	69829	3450	32	uniform
cplex101@64	in401_sp_CLR	73626	64377	77418	69193	69219	2813	32	normal

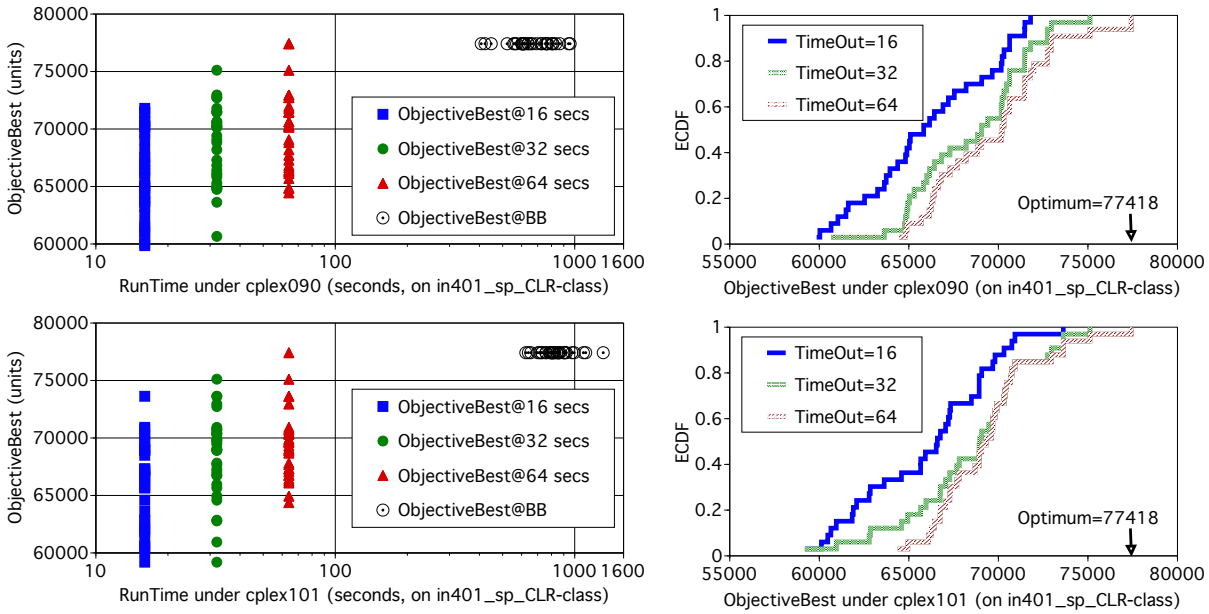


Figure 4: *Timeout* and *branch&bound* experiments with isomorph class in401_sp_CLR and two solvers.

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.

Instance class *in401_sp_CLR*

cplex	Factor2 (solver options)		
solver	none	dfs	feas2
090	666.1	576.2	416.0
101	843.3	925.2	1510.2
Diff	177.2	349.0	1094.2

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 factor 2 for both instance classes, *in401_sp_CLR* and *f51mb_350_CLR*.

Instance class *f51mb_350_CLR*

cplex	Factor2 (solver options)		
solver	none	dfs	feas2
090	231.6	388.2	113.2
101	408.3	440.8	315.8
Diff	176.7	52.6	202.6

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.**

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

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

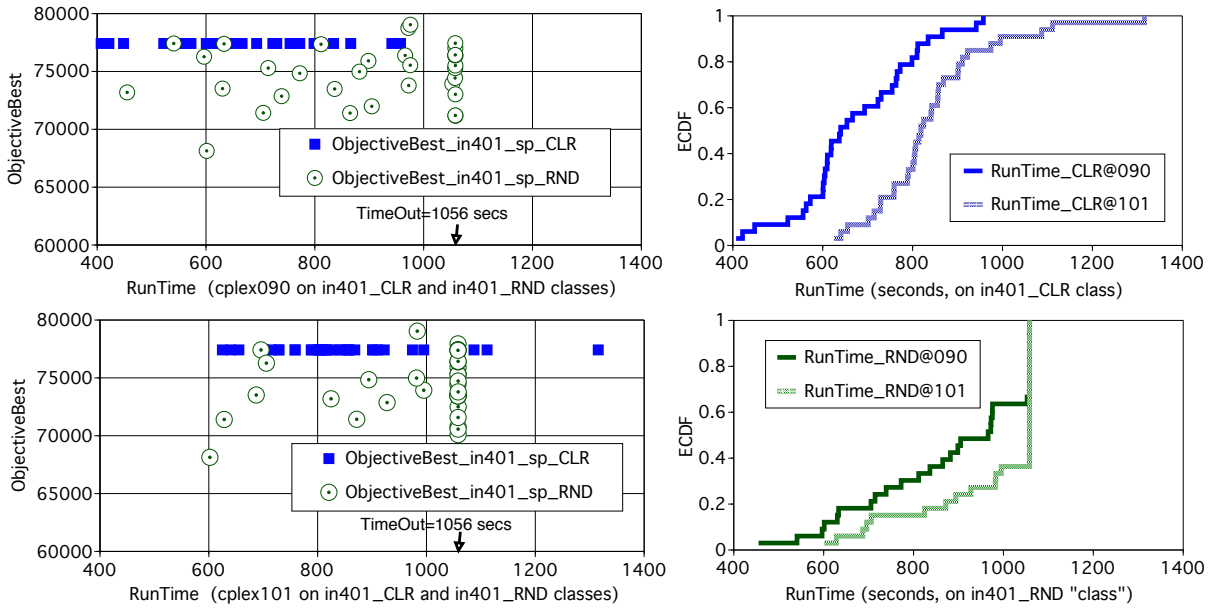


Figure 5: Contrasting *branch&bound* two-solver experiments with an isomorph class and a 'random class'.

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

[1] J.N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, pages 42(2):201–212, 1994.

[2] R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. Designing and reporting on computational experiments with heuristic methods. *J. of Heuristics*, 1(1):9–32, 1995.

[3] J. Hooker. Testing heuristics: We have it all wrong. *J. of Heuristics*, pages 1:33–42, 1996.

[4] F. Brglez. Design of Experiments to Evaluate CAD Algorithms: Which Improvements Are Due to Improved Heuristic and Which Are Merely Due to Chance? Technical Report 1998-TR@CBL-04-Brglez, Computer Science, NCSU, Raleigh, NC 27695, April 1998.

[5] H. H. Hoos and T. Stuetzle. Evaluating Las Vegas Algorithms – Pitfalls and Remedies. In *UAI-98*, pages 238–245. Morgan Kaufmann Publishers, 1998.

[6] C. C. McGeoch. Experimental Analysis of Algorithms. In P. Pardalos and E. Romeijn, editor, *Handbook of Global Optimization, Volume 2: Heuristic Approaches*. Kluwer Academic Publishers, 2001.

[7] D. S. Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. In M. H. Goldwasser and D. S. Johnson and C. C. McGeoch, editor, *Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. Am. Math. Society, 2002.

[8] D. G. Feitelson. Experimental Computer Science: The Need for a Cultural Change, 2005. Manuscript, from http://www.cs.huji.ac.il/ feit/pub.html.

[9] S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000.

[10] Home page for cplex, 2007. http://www.ilog.com/products/cplex/.

[11] Home page for lp_solve, 2007. http://lpsolve.sourceforge.net/5.5/.

[12] About cplex and lp_solve file formats, 2007. http://lpsolve.sourceforge.net/5.5/CPLEX-format.htm.

[13] X. Y. Li, M. F. M. Stallmann, and F. Brglez. Effective bounding techniques for solving unate and binate covering problems. In *DAC*, pages 385–390, 2005.

[14] F. Brglez, X. Y. Li, and M. F. M. Stallmann. On SAT instance classes and a method for reliable performance experiments with SAT solvers. *Ann. Math. Artif. Intell.*, 43(1):1–34, 2005.

[15] G. L. Nemhauser and L,A, Wolsey. *Integer and Combinatorial Optimization*. John Wiley, 1988.

[16] J. E. Harlow and F. Brglez. Design of Experiments in BDD Variable Ordering: Lessons Learned. In *Proceedings of the International Conference on Computer Aided Design*. ACM, November 1998.

[17] J. E. Harlow III and F. Brglez. Design of experiments and evaluation of BDD ordering heuristics. *International Journal on Software Tools for Technology Transfer (STTT)*, 3(2):193–206, May 2001. Springer-Verlag Heidelberg. http://springerlink.metapress.com/, ISSN: 1433-2779 (Paper) 1433-2787 (Online).

[18] S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991.

[19] Y. Guo, A. Lim, B. Rodrigues, and Y. Zhu. Heuristics for a brokering set packing problem. In *Eighth International Symposium on Artificial Intelligence and Mathematics, January 4-6, 2004, Fort Lauderdale, Florida, USA*. ACM, January 2004.

[20] Y. Guo, A. Lim, B. Rodrigues, and Y. Zhu. Heuristics for a bidding problem. *Comput. Oper. Res.*, 33(8):2179–2188, 2006.

[21] D. S. Johnson, R. Aragon C, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39:378–406, 1991.

[22] G.D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[23] M. F. M. Stallmann and F. Brglez. High-contrast algorithm behavior: Observation, conjecture, and experimental design. In *ACM-FCRC*, 2007. Proceedings of Workshop on Experimental Computer Science, Part of ACM FCRC, San Diego, 13-14 June 2007.

[24] K. A. Brownlee. *Statistical Theory and Methodology In Science and Engineering*. Krieger Publishing, 1984. Reprinted, with revisons, from second edition, 1965.

[25] L. J. Bain and M. EngelHardt. *Introduction to Probability and Mathematical Statistics*. Duxbury, 1987.

APPENDIX

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 way of 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 explicity, 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).

```
\ @file   exA_spb_max.lpx
\ @date   2007-02-01-20-26-19
\
\ ObjectiveBest   100 ;   SolutionProvedOptimal   1
\ SolutionCoordinates     0110101
\
Max
  obj: +21x1  +22x2  +23x3  +25x4  +26x5
       +27x6  +29x7
st
  c1:     +x2 +x3 +x4              >= +1
  c2:     -x2         -x5 -x6      >= -2
  c3:                 +x5 +x6 -x7  >=  0
  c4:         -x3             +x7  >=  0
  c5: -x1         -x4         -x7  >= -1
  c6: -x1     -x3         -x6      >= -1
Binary
  x1  x2  x3  x4  x5
  x6  x7
End
```

```
\ @file   exA_spb_max_morph_CLR.lpx
\ @date   2007-02-14-16-39-47
\ @remark see comments about the origin of this file
\         ----------------------------------------
\ @VariablePermutationPairs (isomorph,reference --
\  terminated with 0,0)
\ 1,3 2,1 3,2 4,5 5,6 6,4 7,7 0,0
\
\ ObjectiveBest   100 ;   SolutionProvedOptimal   1
\ SolutionCoordinates     1100011
\
Max
  obj: +22x1  +23x2  +21x3  +27x4  +25x5  +26x6  +29x7
st
  -x3 -x7 -x5 >= -1
  +x1 +x2 +x5 >= +1
  -x2 +x7 >= 0
  -x3 -x2 -x4 >= -1
  -x6 -x1 -x4 >= -2
  +x4 +x6 -x7 >= 0
Binary
  x1  x2  x3  x4  x5  x6  x7
End
```