Evaluating ASP and commercial solvers on the CSPLib

Marco Cadoli, Toni Mancini, Davide Micaletto and **Fabio Patrizi**[1]

Abstract. This paper deals with three solvers for combinatorial problems: the commercial state-of-the-art solver Ilog OPL, and the research ASP systems DLV and SMODELS. The first goal of this research is to evaluate the relative performance of such systems, using a reproducible and extensible experimental methodology. In particular, we consider a third-party problem library, i.e., the CSPLib, and uniform rules for modelling and selecting instances. The second goal is to analyze the effects of a popular reformulation technique, i.e., symmetry breaking, and the impact of other modelling aspects, like global constraints and auxiliary predicates. Results show that there is not a single solver winning on all problems, and that reformulation is almost always beneficial: symmetry-breaking may be a good choice, but its complexity has to be carefully chosen, by taking into account also the particular solver used. Global constraints often, but not always, help OPL, and the addition of auxiliary predicates is usually worth, especially when dealing with ASP solvers. Moreover, interesting synergies among the various modelling techniques exist.

1 Introduction

The last decade has witnessed a large effort in the development of solvers for combinatorial problems. The traditional approach based on writing *ad hoc* algorithms, complete or incomplete, or translating in a format suitable for Integer Programming solvers[2], has been challenged by the use of libraries for Constraint Programming (CP), such as Ilog SOLVER[3], interfaced through classical programming languages, e.g. C++ or Prolog. At the same time, the need for a higher level of abstraction led to the design and development of 1) General purpose languages for constraint modelling/programming –e.g. OPL [16], XPRESS^{MP}[4] or GAMS [1]– and 2) Languages based on specific solvers, such as AMPL [7], DLV [8], SMODELS [11] or ASSAT [9].

This paper focuses on the last class of solvers, which are highly declarative, and characterized by the possibility of decoupling the specification of a problem from the instance, and by having optional procedural information. In particular, we consider one commercial state-of-the-art solver, i.e., Ilog OPL and some ASP solvers, namely DLV and SMODELS. The latter has the interesting property of sharing the specification language with several other solvers through the common parser LPARSE[5]. As a matter of fact, such systems exhibit interesting differences, including availability (OPL and ASP are, respectively, payware and freeware systems, the latter often being open source), algorithm used by the solver (resp. backtracking- and fixpoint-based), expressiveness of the modelling language (e.g., availability of arrays of finite domain variables vs. boolean matrices), compactness of constraint representation (e.g., availability of global constraints), possibility of specifying a separate search procedure.

The first goal of this research is to evaluate the relative performance of such systems, using a reproducible and extensible experimental methodology. In particular, we consider a third-party problem library, i.e. the CSPLib[6], and uniform rules for modelling and instance selection. *The second goal is to analyze the effects of* a popular reformulation technique, i.e. *symmetry breaking*, and the *impact of* other modelling aspects, like the use of *global constraints and auxiliary predicates*.

As for symmetry-breaking, given the high abstraction level of the languages, an immediately usable form of reformulation is through the addition of new constraints (cf., e.g., [3, 6]). Since previous studies [13] showed that this technique is effective when simple formulae are added, it is interesting to know –for each class of solvers– what is the amount of symmetry breaking that can be added to the model, and still improving performances. As a side-effect, we also aim to advance the state of knowledge on the good practices in modelling for some important classes of solvers.

Comparison among different solvers for CP has already been addressed in the literature: in particular, we recall [5] and [17] where SOLVER is compared to other CP languages such as, e.g., OZ [15], CLAIRE[7], and various Prolog-based systems. Moreover, some benchmark suites have been proposed, cf., e.g., the COCONUT one [14]. Also on the ASP side, which has been the subject of much research in the recent years, benchmark suites have been built in order to facilitate the task of evaluating improvements of their latest implementations, the most well-known being ASPARAGUS[8] and ASPLib[9]. However, less research has been done in comparing solvers based on different formalisms and technologies, and in evaluating the relative impact of different features and modelling techniques. In particular, very few papers compare ASP solvers to state-of-the-art systems for CP. To this end, we cite [4], where two ASP solvers are compared to a CLP(FD) Prolog library on six problems: Graph coloring, Hamiltonian path, Protein folding, Schur numbers, Blocks world, and Knapsack, and [12], where ASP and Abductive Logic Programming systems, as well as a first-order finite model finder, are compared in terms of modelling languages and relative performances on three problems: Graph coloring, N-queens, and a scheduling problem.

In this research we consider the CSPLib problem library for our experiments. CSPLib is a collection of 45 problems, classified into 7 areas, and is widely known in the CP community. Since many of them are described only in natural language, this work also provides, as a side-effect, formal specifications of such problems in the modelling languages adopted by some solvers.

[1] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza". E-mail: cadoli|tmancini|micaletto|patrizi@dis.uniroma1.it

[2] e.g. Ilog CPLEX, cf. `http://www.ilog.com/products/cplex`.

[3] cf. `http://www.ilog.com/products/solver`.

[4] cf. `http://www.dashoptimization.com`.

[5] cf. `http://www.tcs.hut.fi/Software/smodels`.

[6] cf. `http://www.csplib.org`.

[7] cf. `http://claire3.free.fr`.

[8] cf. `http://asparagus.cs.uni-potsdam.de`.

[9] cf. `http://dit.unitn.it/~wasp`.

2 Methodology

In this section we present the methodology adopted in order to achieve the two goals mentioned in Section 1. For each problem, we define a number of different formulations, a *base specification*, obtained by a straightforward and intuitive "translation" of the CSPLib problem description into the target language, and several *reformulated* ones, obtained by using different techniques proposed in the literature: *(i)* symmetry-breaking, *(ii)* addition of global constraints and *(iii)* addition of auxiliary predicates. Moreover, in order to establish whether merging different reformulations, which are proven to improve performances when used alone, speeds-up even more the computation, we considered additional specifications for the same problems, obtained by *combining* the aforementioned techniques, and exploring the existence of synergies among them. Finally, a preliminary evaluation of the impact of numbers and arithmetic constraints in all the languages involved in the experimentation has been performed on one problem.

We tried to do the modelling task in a way as systematic as possible, by requiring the specifications of the various solvers to be similar to each other. The criteria followed during the modelling task are discussed in Section 3.2. As for the instances, in this paper we opted for problems which input data is made of few integer parameters (with two exceptions, which are discussed in Section 3.1). In order to have a synthetic qualitative measure of the various solvers performance, for each problem we fix all the input parameters but one. Hence in our results we report for each problem and solver the largest instance (denoted by the value given to the selected parameter) that is solvable in a given time limit (one hour).

3 The experimental framework

3.1 Selecting the problems

So far we have formulated and solved 7 problems, which are listed along with their identification number in CSPLib and the parameter that defines problem instances. Such problems cover the 7 areas of the collection. For space reasons, we omit their descriptions, which can be found at `www.csplib.org`.

Id	Problem name	Instances defined by
017	Ramsey	# of graph nodes
010	Social golfer (decisional version) (32 players, 8 groups of 4)	Schedule length
006	Golomb rulers	# of marks
001	Car sequencing	Benchmark instances
018	Water buckets	Benchmark instances
032	Maximum density still life	Board size
033	Word design (decisional version)	# of words

Some comments are in order. First of all, we considered the decisional versions of two problems, Social golfer and Word design, since our solvers were unable to achieve the optimal solutions. Secondly, for two problems, Car sequencing and Water buckets, instances could not be naturally encoded by a single parameter. Hence, they have been derived from benchmarks taken from the CSPLib. In particular, as for Car sequencing, we considered benchmarks "4/72", "6/76" and "10/93". Unfortunately, they were too hard for our solvers. Hence, from any of them, we generated a set of instances by reducing the number of car classes (cf. the CSPLib problem description) to all possible smaller values, and consequently resizing station capacities in order to avoid an undesirable overconstraining that would make instances unfeasible. Thus, instances derived from the same benchmark could be ordered according to the value for the (reduced) number of classes, which can be regarded as a measure for their size. As for Water buckets instead, we observe that the CSPLib formulation actually fixes the capacities of the three buckets and both the start and goal states. Hence, we built a specification parametric wrt the start and goal states, and designed 11 instances from that considered in the original problem description, which have been proved to be non-trivial by preliminary experiments. Since such instances could not be denoted by a single parameter, solvers' performance for this problem and the different specifications have been compared by considering the overall time needed to solve them.

The Water buckets problem has been used also to evaluate the impact of numbers and arithmetic constraints in problem models. In fact, it is well known that such issues may greatly degrade solvers' performance, and that this behavior strongly depends on the underlying solving algorithm. In order to understand how negatively numbers and arithmetic constraints affect the behavior of the various solvers, we built a new set of instances, obtained by the original one by duplicating the capacities of the buckets (originally 3, 5, and 8), and the amount of water in each of them both in the start and goal states. Of course, the new instances are equivalent to the original ones, having the same set of solutions, but force all solvers to deal with larger domains for numbers.

3.2 Selecting the problem models

As claimed in Section 2, in order to build an extensible experimental framework, we followed the approach of being as uniform and systematic as possible during the modelling phase, by requiring the specifications of the various solvers to be similar to each other. Hence, even if not always identical because of the intrinsic differences among the languages that could not be overcome, all of the models share the same ideas behind the search space and constraints definitions, independently of the language. Below, we discuss the general criteria followed during the modelling phase, and the different formulations considered for each problem[10].

General modelling criteria. The first obvious difference between OPL and the ASP solvers concerns the search space declaration. The former relies on the notion of *function* from a finite domain to a finite domain, while the latter ones have just *relations*, which must be restricted to functions through specific constraints. Domains of relations can be implicit in DLV, since the system infers them by contextual information. For each language, we used the most natural declaration construct, i.e, functions for OPL, and untyped relations for DLV. Secondly, since the domain itself can play a major role in efficiency, sometimes it has been inferred through some *a posteriori* consideration. As an example, in Golomb rulers the maximum position for marks is upper-bounded by 2^m (m being the number of marks), but choosing such a large number can advantage OPL, which has powerful arc-consistency algorithms for domain reduction. As a consequence, we used the upper bound $3L/2$ for all solvers, L being the maximum mark value for the optimum of the specific instance.

Finally, since the performance typically depends on the instances being positive or negative, we considered the *optimization* versions of Ramsey, Golomb rulers, Water buckets, and Maximum density still life problems. As a matter of fact, for proving that a solution is optimal, solvers have to solve both positive and negative instances.

Base specifications. The first formulation considered for each problem is the so called *base specification*. This has been obtained by a straightforward translation of the CSPLib problem description into

[10] cf. `http://www.dis.uniroma1.it/~patrizi/ecai06/encodings.html` for their encodings.

the target language, by taking into account the general criteria discussed above, and, arguably, is the most natural and declarative.

Reformulation by symmetry-breaking. The first kind of reformulation considered aims at evaluating the impact of performing symmetry-breaking. Since we deal with highly declarative languages that exhibit a neat separation of the problem specification from the instances, we adopted the approach of adding further constraints to the base specification that break the structural symmetries of the problems. Symmetry-breaking is dealt with in a systematic way, by considering the general and uniform schemes for symmetry-breaking constraints presented in [10]. These are briefly recalled in what follows (examples below are given in the simple case where all permutations of values are symmetries; generalizations exist):

- **Selective assignment (SA):** A subset of the variables are assigned to precise domain values: a good example is in the Social golfer problem, where, in order to break the permutation symmetries among groups, we can fix the group assignment for the first and partially for the second week.
- **Selective ordering (SO):** Values assigned to a subset of the variables are forced to be ordered: an example is given by the Golomb rulers problem, where, in order to break the symmetry that "reverses" the ruler, we can force the distance between the first two marks to be less than the difference between the last two.
- **Lowest-index ordering (LI):** Linear orders are fixed among domain values and variables, and assignments of variables $(x_1, \ldots, x_n)$ are required to be such that, for any pair of values (d, d'), if $d < d'$ then $min\{i|x_i = d\} < min\{i|x_i = d'\}$. An example is given by the Ramsey problem: once orders are fixed over colors, e.g. red $<$ green $<$ blue, and over edges, we can force the assignments to be such that the least index of red colored edges is lower than the least index of green colored ones, and analogously for green and blue edges.
- **Size-based ordering (SB):** After fixing a linear order on values, we force assignments to be such that $|\{x \in V|x = d\}| \leq |\{x \in V|x = d'\}|$, for any pair of values $d \leq d'$, V being the set of variables. For example, in the Ramsey problem we could require the number of blue colored edges to be greater than or equal to that of green ones, in turn forcing the latter to be greater than or equal to the number of red colored edges. Generalizations of this schema do exist, depending on the way the partition of the variables set into size-ordered sets is defined.
- **Lexicographic ordering (LX):** This schema is widely applied in case of search spaces defined by matrices, where all permutations of rows (or columns) are symmetries. It consists in forcing the assignments to be such that all rows (resp. columns) are lexicographically ordered.
- **Double-lex (lex^2) ordering (L2):** A generalization of the previous schema, applicable where the matrix has both rows and columns symmetries. It consists in forcing assignments to be such that both rows and columns are lexicographically ordered (cf., e.g., [6]). A good example is Social golfer, in which the search space can be defined as a 2D matrix that assigns a group to every combination player/week. Such a matrix has all rows and columns symmetries (we can swap the schedules of any two players, and the group assignments of any two weeks).

Above schemes for symmetry-breaking can be qualitatively classified in terms of "how much" they reduce the search space (i.e., their *effectiveness*), and in terms of "how complex" is their evaluation. In particular they can be partially ordered as follows: SA $<$ SO $<$ LI $<$ LX $<$ L2, and LI $<$ SB, where $s_1 < s_2$ means that schema s_2 better reduces the search space. However, s_2 typically requires more complex constraints than s_1.

In many cases, more than a single schema is applicable for breaking the symmetries of a given specification and the problem of choosing what is the "right" amount of symmetry-breaking that is worth adding for a given solver arises. In what follows, we give a partial answer to this question.

Reformulation by adding global constraints. Global constraints (GC) encapsulate, and are logically equivalent to, a set of other constraints. Despite this equivalence, global constraints come with more powerful filtering algorithms, and a specification exhibiting them is likely to be much more efficiently evaluable. One of the most well-known global constraints supported by constraint solvers is `alldifferent(x1,...,xn)` that forces the labeling algorithm to assign different values to all its input variables. Of course, such a constraint can be replaced by a set of binary inequalities `xi` $\neq$ `xj` (for all `i` $\neq$ `j`), but such a substitution will result in poorer propagation, hence in less efficiency. Several global constraints are supported by OPL, e.g., `alldifferent` and `distribute`. According to the problems structure, the former has been applied to Golomb Rulers and the latter to Social golfer, Car sequencing and Word design. As for Ramsey, Water bucket and Maximum density still life none of such reformulations applies.

On the other hand, ASP solvers do not offer such a feature, hence no comparison can be made on this issue.

Reformulation by adding auxiliary predicates. A predicate in the search space is called *auxiliary* if its extensions functionally depend on those of the other ones. The use of auxiliary guessed predicates is very common in declarative programming, especially when the user needs to store partial results, to maintain intermediate states. Although the use of auxiliary predicates increases the size of the search space, in some cases this results in a simplification of complex constraints and in a reduction of the number of their variables, and hence may lead to appreciable time savings.

We consider equivalent specifications, obtained by using auxiliary predicates, for all of the selected problems. However, the bottom-up evaluation algorithms of DLV and SMODELS may significantly advantage ASP solvers over OPL on such specifications, since auxiliary predicates are usually defined in rule-heads. To this end, when adding auxiliary predicates to OPL specifications, we also added simple search procedures instructing the labelling algorithm to not branch on auxiliary variables, while maintaining the default behavior on the others.

Reformulation by combining different techniques. In many cases, more than one single reformulation strategy improves performances on a given problem. Hence, the question arises whether synergies exist among them, and what techniques are likely to work well together, for each solver. To this end, for each problem we consider two additional formulations: the first one has been obtained, according to the aforementioned uniformity criteria, by merging the two reformulations (among symmetry-breaking, addition of global constraints and of auxiliary predicates) that, for each solver, resulted to be the most performant. Finally, in order to understand whether there exist better, undiscovered synergies, we relaxed the uniformity hypothesis, and considered some of the other possible combinations of reformulation strategies, with the goal to further boost performances.

4 Experimental results

Our experiments have been performed by using the following solvers: *i)* Ilog SOLVER v. 5.3, invoked through OPLSTUDIO 3.61,

ii) SMODELS v. 2.28, by using LPARSE 1.0.17 for grounding, *iii)* DLV v. 2005-02-23, on a 2 CPU Intel Xeon 2.4 Ghz computer, with a 2.5 GB RAM and Linux v. 2.4.18-64GB-SMP.

For every problem, we wrote the specifications described in Sections 2 and 3 in the different languages. We then ran the different specifications for each solver on the same set of instances, with a timeout of 1 hour. Table 1 shows a summary of the results concerning base specifications and their various reformulations. In particular, for each problem and solver, we report the largest instance the various systems were able to solve (in the given time-limit) for the base specifications, and the improvements obtained by the different reformulations. A 0^+ (resp. 0^-) means that the size of the largest instance solved within the time limit was unchanged, but absolute solving times were significantly lower (resp. higher). As for Car sequencing, we report, for each set of instances generated from CSPLib benchmarks "4/72", "6/76" and "10/93", the largest one solved, i.e., the one with the largest number of classes (cf. the discussion about instance selection for this problems in Section 3.1), and, as for Water buckets, the overall time needed to solve the whole set of instances (instances that could not be solved contributed with 3,600 seconds to the overall time).

Impact of symmetry-breaking. From the experiments, it can be observed that symmetry-breaking may be beneficial, although the complexity of the adopted symmetry-breaking constraints needs to be carefully chosen. As an example, DLV performs much better on the Ramsey problem with LI symmetry-breaking constraints, but it is slowed down when the more complex SB schema is adopted. A similar behavior can be observed on SMODELS.

As for Social golfer, Table 1 does not show significant performance improvements when symmetry-breaking is applied, with the ASP solvers (especially SMODELS) being significantly slowed down when adopting the most complex schemas (LX and L2). However, it is worth noting that, on smaller negative (non-benchmark) instances, impressive speed-ups have been obtained for all systems, especially when using SA. As for LX, we also observe that it can be applied in two different ways, i.e., forcing either players' schedulings or weekly groupings to be lexicographically ordered. Values reported in Table 1 are obtained by lexicographically ordering weekly groupings: as a matter of fact, ordering players' schedulings is even less performant on SMODELS, being comparable for the other solvers. General rules for determining the right "amount" of symmetry breaking for any given solver on different problems are currently still unknown, but it seems that the simplest ones, e.g., SA, have to be preferred when using ASP solvers.

Impact of global constraints. Experiments confirm that OPL may benefit from the use of global constraints. As an example, the base specification of the Golomb rulers problem encodes the constraint that forces the differences between pairs of marks to be all different by a set of binary inequalities. By replacing them with an `alldifferent` constraint, OPL was able to solve the instance with 11 marks in the time-limit, and time required to solve smaller instances significantly decreases. Also the Social golfer specification can be restated by using global constraints, in particular the `distribute` constraint. However, in this case our results show that OPL does not benefit from such a reformulation, in that it was not able to solve even the 4-weeks instance (solved in about 11 seconds with the base specification). Global constraints help OPL also on other problems, i.e., Car sequencing, where `distribute` can be used, even if the performance improvements don't make it able to solve larger instances. Finally, Word Design seems not to be affected by the introduction of `distribute`.

Impact of auxiliary predicates. ASP solvers seem to benefit from the use of auxiliary predicates (often not really needed by OPL, which allows to express more elaborate constraints), especially when they are defined relying on the minimal model semantics of ASP (hence, in rule heads). As an example, SMODELS solves the 6-weeks instance of the Social golfer problem in 6 seconds, when the auxiliary `meet/3` predicate is used, while solving the base specification requires 41 minutes. Even if not as much remarkable, a similar behavior is observed in DLV. Similar results have been obtained for Ramsey, where the auxiliary predicate `color_used/1` is added.

Using the `meet` auxiliary predicate helps also OPL (but only after a simple search procedure that excludes branches on its variables has been added, cf. Section 3.2). In particular, the 5-weeks instance has been solved in just 8 seconds (with respect to the 80 seconds of the base specification). It is interesting to note that, by adding a smarter search procedure, solving time dropped down to less than a second. This is a good evidence that exploiting the peculiarities of the solver at hand may significantly increase the performance.

Synergic reformulations. Specifications obtained by combining, for each problem and solver, the most two performant techniques, in many cases further boost performances, or at least do not affect them negatively. This is the case, e.g., of Social Golfer, Golomb Rulers, and Word design problems when solved by OPL, that proved to be able to deal with larger instances, and Car sequencing, where solving times significantly decreased. Few exceptions do exist, i.e., Ramsey and Maximum density still life, where solving times were a bit higher, but did not prevent OPL to deal with the largest instances previously solved. Similar results can be observed for SMODELS and DLV. This gives evidence that combining "good" reformulations is in general a promising strategy to further boost performance of all solvers. Table 1 also shows some of the results obtained by other possible combinations, without considering any uniformity criteria. It can be observed that in few cases even better results could be achieved (cf., e.g., OPL on Social golfer and the specification with auxiliary predicates and global constraints), but in several others, only worse performances were obtained.

Impact of numbers and arithmetic constraints. Experiments performed on Water buckets, with the most performant specification (i.e., that with auxiliary predicates) on instances obtained by doubling both the buckets capacities and the water contents in the start and goal states, confirm that numbers and arithmetic constraints are a major obstacle for all solvers. In particular, the time needed by OPL to solve the whole set of instances is almost 13 times higher, while for DLV and SMODELS the solving time increases, respectively, of about 130 and 45 times.

5 Conclusions

In this paper we reported results about an experimental investigation which aims at comparing the relative efficiency of a commercial backtracking-based and two academic ASP solvers. In particular, we modelled a number of problems from the CSPLib into the languages used by the different solvers, in a way as systematic as possible, by also applying symmetry-breaking, using global constraints and auxiliary predicates, and evaluating synergies among such techniques. Results show that there is not a single solver winning on all problems, with ASP being comparable to OPL for many of them, and that reformulating the specification almost always improves performances. However, even if our experiments suggest some good modelling practices, an exact understanding of which reformulations lead to the best performances for a given problem and solver remains a

Problems	OPL				DLV				SMODELS			
Ramsey	Base	LI	SB	Aux	Base	LI	SB	Aux	Base/Aux*	LI	SB	
	16	-3	0^-	0	9	$+5$	-1	0^+	9	**+7**	-1	
(# of nodes)	Aux-SB	Aux-LI			Aux-LI	Aux-SB						
	0^-	0^-			**+7**	-1						
Social Golfer	Base	SA	L2	LX	Base	SA	L2	LX	Base	SA	L2	LX
	5	0	0	0	**6**	0	-2	0	**6**	0^+	-6	-6
(# of weeks)	Aux	GC	Aux-SA	Aux-GC	Aux	Aux-SA	Aux-LX		Aux	Aux+SA	Aux+LX	
	0^+	-2	0^+	**+1**	0^+	0^+	0		0^+	0^+	-6	
Golomb Rulers	Base	SO	Aux	GC	Base	SO	Aux		Base	SO	Aux	
	10	0^+	$+1$	$+1$	9	0	0		6	0	$+2$	
(# of marks)	Aux-GC	Aux-SO-GC			Aux-SO				Aux-SO			
	$+1$	**+2**			0				$+2$			
Car Sequencing	Base	SO	Aux		Base	SO	Aux		Base	SO	Aux	
bench. 4/72	10	0	0^+		**13**	0	0		**13**	0	0	
(# of classes)	Aux-GC	Aux-SO-GC			Aux-SO				Aux-SO			
	0^+	0^+			0^+		0		0^+			
Car Sequencing	Base	SO	Aux		Base	SO	Aux		Base	SO	Aux	
bench. 6/76	6	0	0		**9**	0	0		**9**	0	0	
(# of classes)	Aux-GC	Aux-SO-GC			Aux-SO				Aux-SO			
	0	0			0^+				0^+			
Car Sequencing	Base	SO	Aux		Base	SO	Aux		Base	SO	Aux	
bench. 10/93	**12**	0	0		**12**	0	0		**12**	0	0	
(# of classes)	Aux-GC	Aux-SO-GC			Aux-SO				Aux-SO			
	0	0			0^+				0^+			
Water Bucket	Base	Aux			Base	Aux			Base	Aux		
(total time)	54.77 s	**18.22** s			1332.62 s	202.00 s			9765.8 s	173.43 s		
Maximum Density Still Life	Base	SO	SB	Aux	Base	SO	SB	Aux	Base	SO	SB	Aux
	8	0	0	0^-	7	0	0	-1	7	**+1**	0	**+1**
(board size)	Aux-SO	Aux-SB			Aux-SO	Aux-SB			Aux-SO	Aux-SB		
	0^-	0^-			0	0			**+1**	0		
Word Design	Base	SO	SB	LX	Base	SO	SB	LX	Base	SO	SB	LX
DNA	86	0	-30	**+1**	5	0	0	0	11	0^+	-1	$+3$
(# of words)	Aux	GC	Aux-LX	Aux-GC	Aux	Aux-SO	Aux-LX		Aux	Aux-LX	Aux-SO	
	0	0	**+1**	0	0	0	0		$+10$	$+1$	$+35$	

* Since for Ramsey problem the use of auxiliary predicates seems to be unavoidable in SMODELS, the Base and Aux specifications coincide.

Table 1. Sizes of the largest instances solved by OPL, DLV, and SMODELS in 1 hour, using the base specification and their reformulations. Variations are given wrt the base specifications. Bold numbers denote the best results.

challenge. Finally, we also started an investigation about the impact of numbers and arithmetic constraints in problem specifications.

Our current efforts are aimed at covering a larger part of the CSPLib, extending the set of reformulations involved (e.g., using implied constraints) and considering other systems (e.g. ASP or SAT based).

6 Acknowledgements

The authors would like to thank the anonymous reviewers, in particular for suggesting a better Water Bucket SMODELS encoding.

REFERENCES

[1] E. Castillo, A. J. Conejo, P. Pedregal, R. Garca, and N. Alguacil, *Building and Solving Mathematical Programming Models in Engineering and Science*, John Wiley & Sons, 2001.

[2] P. Codognet and D. Diaz, 'Compiling constraints in clp(FD)', *J. of Logic Programming*, **27**, 185–226, (1996).

[3] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy, 'Symmetry-breaking predicates for search problems', in *Proc. of KR'96*, pp. 148–159, Cambridge, MA, USA, (1996). Morgan Kaufmann, Los Altos.

[4] A. Dovier, A. Formisano, and E. Pontelli, 'A comparison of CLP(FD) and ASP solutions to NP-complete problems', in *Proc. of ICLP 2005*, volume 3668 of *LNCS*, pp. 67–82, Sitges, Spain, (2005). Springer.

[5] A. J. Fernández and P. M. Hill, 'A comparative study of eight constraint programming languages over the Boolean and Finite Domains', *Constraints*, **5**(3), 275–301, (2000).

[6] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh, 'Breaking row and column symmetries in matrix models', in *Proc. of CP 2002*, volume 2470 of *LNCS*, p. 462 ff., Ithaca, NY, USA, (2002). Springer.

[7] R. Fourer, D. M. Gay, and B. W. Kernigham, *AMPL: A Modeling Language for Mathematical Programming*, Intl. Thomson Publ., 1993.

[8] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, 'The DLV System for Knowledge Representation and Reasoning', *ACM Trans. on Comp. Logic*. To appear.

[9] F. Lin and Z. Yuting, 'ASSAT: Computing answer sets of a logic program by SAT solvers', *Artif. Intell.*, **157**(1–2), 115–137, (2004).

[10] T. Mancini and M. Cadoli, 'Detecting and breaking symmetries by reasoning on problem specifications', in *Proc. of SARA 2005*, volume 3607 of *LNAI*, pp. 165–181, Airth Castle, Scotland, UK, (2005). Springer.

[11] I. Niemelä, 'Logic programs with stable model semantics as a constraint programming paradigm', *Annals of Math. and Artif. Intell.*, **25**(3,4), 241–273, (1999).

[12] N. Pelov, E. De Mot, and M. Denecker, 'Logic Programming approaches for representing and solving Constraint Satisfaction Problems: A comparison', in *Proc. of LPAR 2000*, volume 1955 of *LNCS*, pp. 225–239, Reunion Island, FR, (2000). Springer.

[13] A. Ramani, F. A. Aloul, I. L. Markov, and K. A. Sakallak, 'Breaking instance-independent symmetries in exact graph coloring', in *Proc. of DATE 2004*, pp. 324–331, Paris, France, (2004). IEEE Comp. Society Press.

[14] O Shcherbina, A. Neumaier, D. Sam-Haroud, X.-H. Vu, and T.-V. Nguyen, 'Benchmarking global optimization and constraint satisfaction codes', in *Proc. of COCOS 2002)*, volume 2861 of *LNCS*, pp. 211–222, Valbonne-Sophia Antipolis, France, (2003). Springer.

[15] G. Smolka, 'The Oz programming model', in *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, 324–343, Springer, (1995).

[16] P. Van Hentenryck, *The OPL Optimization Programming Language*, The MIT Press, 1999.

[17] M. Wallace, J. Schimpf, K. Shen, and W. Harvey, 'On benchmarking constraint logic programming platforms. response to [5]', *Constraints*, **9**(1), 5–34, (2004).