Gomory cuts in a hybrid constraint programming approach

Andrea Lodi[1,2], Maria Silvia Pini[3], Francesca Rossi[3]

[1] DEIS, University of Bologna, viale Risorgimento 2 - 40136 Bologna - Italy
[2] IBM T.J. Watson Research, P.O. Box 218, Yorktown Heights, NY 10598 - USA
alodi@us.ibm.com
[3] Department of Pure and Applied Mathematics, University of Padova, Italy.
{frossi,mpini}@math.unipd.it.

Abstract. Recently, many successful hybrid approaches which use both constraint programming (CP) and operations research (OR) techniques to solve a problem have been proposed. They usually use OR techniques, like cutting planes, which are specific to the problem. In this paper we investigate the use of generic cutting planes (more specifically, Gomory cuts) in a typical CP-OR hybrid approach. This allows us to have a general algorithm to be applied to all problem classes. We will show that our approach is indeed promising: on the class of partial latin square completion problems, the performance of our algorithm is comparable to that of a pure CP algorithmand better in several cases. In particular we will show that the hybrid approach solves many problem instances that the CP approach is not able to solve in a reasonable amount of time.

1 Introduction

Constraint Programming (CP) [5] and Operations Research (OR) [13] techniques are both suitable for solving combinatorial search problems. However, they have complementary advantages and drawbacks. On one hand, CP offers more sophisticated modelling tools like global constraints [15], and effective search tree pruning techniques like constraint propagation [5]. On the other hand, OR provides better tools for reasoning about optimality via the possibility of considering relaxed problems, and other pruning facilities like the cutting planes within a branch and cut strategy.

To overcome the limitations and maintain the advantages of both kinds of techniques, recently many hybrid approaches have been proposed [7–10, 12] for solving combinatorial optimization problems. Some of them are based on interleaving a CP depth-first search algorithm with problem relaxation and cutting planes. More precisely, at each step in a depth-first search, the current problem is relaxed and solved optimally via the cutting planes. The optimal solution of the relaxed problem is then used to go back to the CP side with useful information for the strategy to choose the next step (that is, the next variable-value selection).

Some existing methods which employ this approach use cutting planes which are specific to the classes of problems considered. For example, in [8] Travelling Salesmen Problems are relaxed via cutting planes which achieve subtour elimination.

In this paper, we investigate the use of generic cuts in a hybrid setting. The reason for doing this is that the resulting algorithm would be general and thus applicable to all classes of problems. In particular, we consider Gomory cuts to investigate their convenience in the hybrid approach. It was folklore knowledge that Gomory cuts were not useful. In [2] it is shown that the way they have been always applied was wrong. Our goal is to see whether they can be useful in a hybrid CP-OR environment.

To achieve our goal, we develop an experimental setting where we consider the completion of the partial latin square problems and we compare the performance of a hybrid CP-OR algorithm with Gomory cuts, a hybrid CP-OR algorithm without Gomory cuts, and a pure CP algorithm. We perform several types of experiments by varying both the parameters of the problems and the features of the hybrid algorithm.

We will show that the hybrid CP-OR algorithm we propose in this paper, which uses Gomory cuts in the OR part, is always better than the hybrid approach without Gomory cuts and for several problem instances outperforms the pure CP approach. Moreover, it provides the high-level modelling facilities typical of CP, like global constraints. Such modelling and performance considerations thus make this hybrid approach more attractive than the pure CP approach.

More precisely, while there is no algorithm which is uniformly better on all problem instances, our hybrid algorithm with Gomory cuts looks convenient for critically constrained problems (where it is better than CP in several instances) or for problems whose constrainedness cannot be known in advance. This algorithm thus combines the modelling capabilities of CP (with compactness given by global constraints), together with the convenient branch and cut strategies, typical of OR approaches. Moreover, it does not need to use domain-specific cuts.

We therefore claim that it can be feasible and convenient to use suitable generic cutting planes, like Gomory cuts, within a CP-OR hybrid approach. It remains to be checked whether other generic classes of cutting planes can be useful in a hybrid CP-OR approach, and whether the same behavior holds also for other classes of problems.

The article is organized as follows: Section 2 gives the main concepts of CP, OR, and hybrid CP-OR approaches, Section 3 introduces the algorithms we will consider in this paper, Section 4 defines the experimental setting, Section 5 gives the experimental results and discusses them, and Section 6 presents some related works. Finally, Section 7 concludes the paper by summarizing its main results.

2 Background

In this section we will introduce the basic notions about Constraint Programming and Operations Research that will be used in our algorithms.

2.1 Constraint programming

In our context, a constraint is a relation among some variables with a finite domain. A constraint problem is just a set of variables, plus a set of constraints over subsets of them.

Constraint programming [5] is a body of techniques to solve constraint problems. For the purpose of this paper, we only consider CP techniques based on depth-first search and constraint propagation.

Depth-first search orders the variables and instantiates them one at a time in its domain, in a way that is compatible with the constraints and the already instantiated variables. If a variable cannot be assigned any value, backtracking occurs and the algorithm tries to instantiate the previous variable with another value. When all variables are instantiated, we have a solution of the given problem. If not all variables are instantiated and no more backtracking is possible, the problem has no solution.

Constraint propagation propagates information from one constraint to another one. Each propagation step considers a small part of the constraint problem (most of the times just one constraint), and eliminates from the variable domains of such part some domain elements, which are recognized to be incompatible to the considered constraints. Then, the new domains (possibly smaller) are used when considering adjacent constraints. This is repeated until quiescence. For example, arc-consistency [5] considers one constraint at a time, and eliminates from the domains of its variables those values which do not have any support in the domains of the other variables according to this constraint. The new problem obtained at the end of this process is equivalent to the original one (that is, it has the same set of solutions) but with possibly smaller domains. Achieving arc-consistency requires time $O(ed^2)$ where e is the number of constraints and d is the size of the domains. In general, performing constraint propagation where each step considers a subpart with at most k variables requires $O(n^k)$ time.

Since depth-first search in the worst case has to perform as many tries for each variable as the cardinality of its domain, the smaller the domains the better it is for such search technique. Thus all depth-first search constraint programming algorithms use constraint propagation after every variable instantiation. This usually allows for much pruning within the search tree. The overall complexity of the search algorithm remains exponential in the size of the problem, but the average complexity can be improved significantly by the use of constraint propagation.

Global constraints [3, 15] are non-binary constraints which occur very often in real-life problems. For this reason, they have been equipped with a compact way to express them and specialized constraint propagation techniques which exploit the semantics of the global constraint and have a higher pruning power with

respect to arc-consistency. A typical example is `alldifferent`$(X_1,\ldots,X_n)$ which states that variables X_1,..., X_n must have different values. If this constraint were modelled by a set of $n(n-1)/2$ inequalities of the form $X_i \neq X_j$, and arc-consistency would be applied, no domain pruning would be achieved, except in the very special case in which some domain has just one element. On the contrary, by modelling it as a global constraint, and applying its specialized propagation algorithm, we can perform a global reasoning on all the n variables, for example by comparing the cardinality of the domains to n, and thus achieve more pruning.

Constraint problems can contain both global constraints and other kinds of constraints. During search, arc-consistency (or another generic constraint propagation algorithm) will be applied to all the constraints, except to global constraints for which their specialized propagation algorithm will be used.

2.2 Cutting planes and relaxed problems in OR

An Integer Linear Programming (ILP) problem is a set of linear equalities and inequalities where variables must have integer values, plus a linear objective function, to be minimized.

The usual solving technique for ILP problems is called branch and bound [13]. As in CP, this technique involves visiting the nodes of a search tree. At each node, the branch strategy is decided on the basis of an optimal solution of a relaxed version of the current problem. More precisely, the current problem is relaxed by eliminating the requirement that variables have to be integer, such relaxed problem is solved optimally, and the solution found is used to decide the next search step. Since the relaxed problem does not contain all constraints of the original problem, the value of its optimal solutions is a lower bound of the value of any optimal solution of the original problem.

Usually a non-integer variable is chosen in the optimal solution of the relaxed problem, and it is made integer by imposing additional constraints. For example, if the optimal solution of the relaxed problem contains $X = 4.3$, we add $X \leq 4$ and $X \geq 5$ to the original problem and then we follow one of these two paths according to heuristics. When we find a solution, we continue visiting the search tree but first we add the constraint that the objective function has to be better than the value of the current solution.

Cutting planes [13] are linear inequalities of the form $\alpha x \leq \alpha_0$ which eliminate some non-optimal solutions. They are added to a relaxed version of the problem after an optimal solution is found. Thus they remove the optimal solution of the relaxed problem while not eliminating any optimal solution of the original problem. The optimal solution of the relaxed problem is used to compute the cutting planes. Adding the cutting planes allows us to obtain a new relaxed problem which is closer to the original problem and thus whose optimal solutions have values which are closer to the value of the optimal solutions of the original problem. Cutting planes can be iteratively added until the improvement of the lower bound given by the optimal solution of the relaxed problem is not cost-effective. At this point, a branching step is performed.

Gomory cuts are a class of cutting planes which can be computed by taking into consideration the values of the variables in a solution and their offset with respect to an integer. Gomory cuts are generic in the sense that they are applicable to all integer linear programming problems. Other cutting planes, which are specific to the considered problems, can be used as well. For some classes of these cuts, there exist polynomial algorithms to find them.

2.3 Hybrid CP-OR approaches

Many hybrid approaches which try to combine the techniques which are typical of CP with those of OR have been proposed in the literature [7–10, 12]. For the purpose of this paper, we consider one which relates CP and ILP and can be described as follows:

1. The problem is modelled using CP tools. This allows to take advantage of the more sophisticated CP modelling environment. For example, global constraints can be used to easily model many real-time scenarios.
2. Constraint propagation is performed to reduce the variable domains. If we have a solution (that is, one value for each variable), we stop and return the solution.
3. The current problem is transformed into an ILP problem. Of course this can be done only if the CP problem does not contain constraints which cannot be modelled by an ILP. Many global constraints can be modelled within an ILP [14], and also many other classes of constraints. However, a CP problem could also contain, for example, quadratic constraints, which cannot be modelled in an ILP.
4. Given the current ILP, the integer constraint is relaxed, and the relaxed problem is solved with OR techniques, possibly using also cutting planes, that can be specific for the considered problem, to tighten the relaxation.
5. The optimal solution of the relaxed problem is used to decide the next variable to be instantiated in the CP side, and the value to which it has to be instantiated. The instantiation is then performed.
6. Go back to step 2.

Summarizing, this algorithm can also be seen as a CP search algorithm where the next variable-value instantiation is decided using information coming from an ILP environment.

3 Gomory cuts in a hybrid approach: our algorithms

In this section we will describe the algorithms for solving constraint optimization problems which we will consider in the following of the paper. In particular, we will use a pure CP algorithm, a pure ILP algorithm with and without Gomory cuts, and a hybrid CP-OR algorithm with and without Gomory cuts.

The pure CP algorithm we consider in this paper is a classical depth-first search algorithm with backtracking, which applies constraint propagation at

each search step. At each node of the search tree, arc-consistency is performed and then an uninstantiated variable among those with the smallest domain is chosen and instantiated to its smallest value. In the following, this algorithm will be called `CP`.

The pure OR algorithm we use is a branch-and-cut algorithm where cutting planes are used to improve the solution of the relaxed problem. In the following, this algorithm will be called `ILP` if it does not use Gomory cuts, and `ILP-Gomory` if it uses them.

The hybrid algorithm with Gomory cuts follows the scheme described in Section 2.3, where arc-consistency is used to propagate constraints. The ILP version of the CP problem is obtained via the methodology in [14] (for the class of considered problems this step is always possible), it is solved via the primal (for the tree root) or dual (for the other nodes) Simplex algorithm, and Gomory cuts are applied to refine its relaxed optimal solution. The next branch step is perfomed by considering the variable whose decimal part is closer to 0.5, and by instantiating it to the closer integer. In the following, this algorithm will be called `H-Gomory`. The hybrid algorithm without Gomory cuts is like the one just explained above, except that it doesn't apply Gomory cuts. In the following, this algorithm will be called `H`.

4 Experimental setting

To compare the behavior of the above described algorithms, we have considered a specific class of problems: partial Latin Square completion [9]. These problems are easy to model but occur in a wide variety of real-life scenarios, from conflict-free wavelength routing in wide-area network, to statistical design, to error-correcting codes.

A Latin Square (LS) of order n is an $n \times n$ matrix such that each cell contains exactly one of the integers from 1 to n, and each of such integers appears at most once in any row and any column. A partial Latin Square (PLS) of order n is an LS of order n where some cells can be empty. A PLS can be completed if it is possible to fill the empty cells with integers so to obtain a LS.

A PLS completion problem is the problem of starting from a PLS and obtaining, if possible, one of its completions. The problem of deciding if a PLS can be completed is NP-complete [4].

In this section we briefly describe the problem encodings we considered.

In the CP framework, a PLS completion problem is modelled as a constraint satisfiability problem (CSP). More precisely, a PLS is modelled by using n^2 variables x_{ij}, corresponding to all the cells c_{ij} of the matrix $n \times n$. If a variable corresponds to an empty cell, its domain is $\{1, \ldots, n\}$, otherwise it is a singleton domain containing only one integer (the integer chosen for that cell). Moreover, we have an `alldifferent` constraint involving the variables in each row and each column. That is:

$$x_{ij} \in \{1, \ldots, n\} \; \forall i, j \in \{1, \ldots, n\}$$

$$x_{ij} = k \ \forall i, j \text{ such that } c_{ij} = k$$
$$\texttt{alldifferent}(x_{i1}, x_{i2}, \ldots, x_{in}) \ \forall i = 1, \ldots, n$$
$$\texttt{alldifferent}(x_{1j}, x_{2j}, \ldots, x_{nj}) \ \forall j = 1, \ldots, n$$

In the OR framework, the PLS completion is modelled as an Integer Linear Programming (ILP) problem, with an objective function to be maximized. The ILP model contains n^3 variables x_{ijk}, each corresponding to a cell c_{ij} with value k. A variable x_{ijk} has domain $\{0, 1\}$ if c_{ij} is empty. If c_{ij} has value k, then $x_{ijk} = 1$. The constraints model the row and column restrictions, and make sure that each cell has at most one value. Finally, the objective function, to be maximized, computes the number of variables which have a value different from 0, which is equal to the number of filled cells. That is:

$$\max \ \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} x_{ijk}$$

subject to

$$\sum_{i=1}^{n} x_{ijk} \leq 1 \ \forall j, k$$
$$\sum_{j=1}^{n} x_{ijk} \leq 1 \ \forall i, k$$
$$\sum_{k=1}^{n} x_{ijk} \leq 1 \ \forall i, j$$
$$x_{ijk} \in \{0, 1\} \ \forall i, j, k$$
$$i, j, k = 1, 2, ..., n$$

If in the optimal solution the number of filled cells is equal to n^2, then the PLS can be completed, i.e., a feasible solution exists.

In the algorithm `CP` we have used the CSP formulation to model the completion PLS, whereas in the hybrid algorithms, at each search step the current CSP problem is transformed into an ILP problem by following the same lines as in [14].

To test our algorithms over PLS completion problems, we have developed a generator of such problems, which is based on the following parameters: n, the size of the matrix, and k, the number of filled cells. Starting from this parameters, we generate a matrix $n \times n$ and then we choose randomly the cells to fill, and the value (over {1,. . . ,n}) to put in each of such cells. Moreover, we make sure that such partially filled matrix satisfies the definition of a PLS.

5 Experimental results

We have implemented all our algorithms using the ILOG libraries [11]. In particular, we used Solver **5.6** for algorithm `CP` and for solving the CSP formulation in algorithms `H` and `H-Gomory`, and we used CPLEX **8.1** for solving the continuous relaxation of the ILP formulation in the hybrid approaches `H` and `H-Gomory`. To test the influence of Gomory cuts in a pure OR environment, we have also used CPLEX with two different settings, one requiring the separation of Gomory cuts (in algorithm `ILP-Gomory`) and one forbidding Gomory cuts (in algorithm `ILP`).

All the experiments have been performed on an AMD Athlon 1250 MHz.

n	k	ILP	ILP-Gomory
10	2	0.69	2.00
10	5	0.63	3.50
10	7	0.52	4.60
15	3	11.41	73.71
15	7	11.32	95.33
15	11	11.00	96.60

Table 1. Time in seconds to find an optimal solution of PLS completion problems using ILP and ILP-Gomory. All PLS problems generated are solvable with objective function at the maximum value, i.e., $n^2 - k$. This means that all the cells have been filled. Times are averaged over 10 PLS completion problems.

Table 1 shows the CPU time needed to find an optimal solution for ILP and ILP-Gomory, for some values of n and k. All experimental results are obtained by running the algorithms on 10 instances of PLS completion problems, and by taking the average of the times.

It is easy to see that forcing the separation of Gomory cuts (in ILP-Gomory) is not convenient for PLS, at least in the standard setting of CPLEX. In fact, for any n and any k in Table 1, as well as in many other experimental results non provided here, the performance of ILP is always better than that of ILP-Gomory.

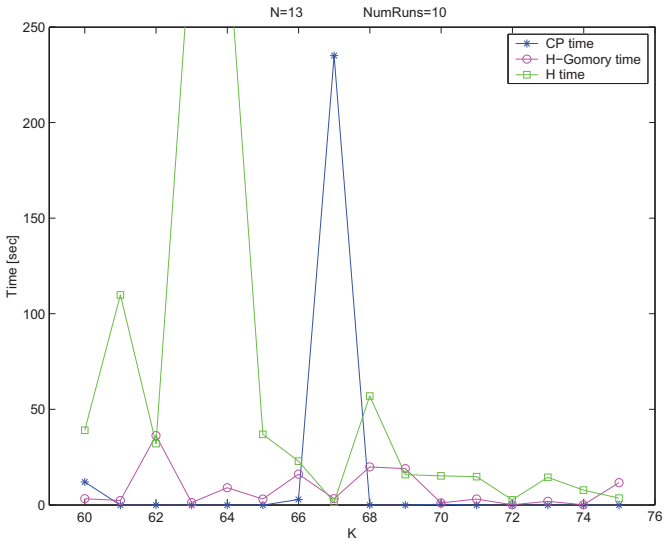

Fig. 1. Computing times (averaged over 10 PLSs) of CP, H-Gomory and H to find a solution for a PLS completion problem, or to discover that there is no completion.

Figure 1 compares the behavior of algorithms CP, H-Gomory and H, over PLS completion problems with $n = 13$ and k which varies in a way such that PLS in input has between nearly 35% and 40% of filled cells, that is the critical phase for algorithm CP. In fact, when we have a PLS with less than 35% or more than 45% filled cells, CP finds a solution very quickly, whereas in this region it sometimes takes a very long time. Notice that all the PLS problems represented in Figure 1 are completable, i.e., they always produce a success in the search tree.

Algorithm CP is usually much more convenient than any of the two hybrid algorithms (H and H-Gomory) for many values of k. However, CP has also very high peaks, which cannot be found in H-Gomory.

If we compare H and H-Gomory, H-Gomory is always more convenient than algorithm H which does not use Gomory cuts, since in some cases H needs much more memory than the available one and so it is not able to find a solution even if the PLS is completable. Therefore H-Gomory is always better than H, which means that applying Gomory cuts in the hybrid CP-OR approach is always useful.

We can conclude that CP is almost always better, but that by using CP we may run the risk of incurring in a very high peak. Thus we may use CP for underconstrained PLS's (filled up to 30-35%) and overconstrained ones (filled for more than 40-45% of the cells). In the middle region, it is safer to use H-Gomory, which shows a better behavior offering also a higher-level modelling capabilities, and a tollerable slow-down with respect to CP while avoiding its high peaks.

Table 2 compares algorithms CP, H and H-Gomory over PLS's with various values of n, while Figure 1 considered only $n = 13$.

It is easy to note again that algorithm H-Gomory is always better than algorithm H. This allows us to claim that applying Gomory cuts in a hybrid CP-OR approach is better than not applying them. With respect to algorithm CP, the behavior is rather clear: either CP is able to solve the problem very fast or it is unable to solve it in a reasonable amount of time (20 minutes of CPU time in the table), while algorithm H-Gomory is a very reliable approach which consistently gives quite quick solutions, thus denoting a very robust behavior.

Similar conclusions has been drawn in [9], where a pure CP algorithm and a hybrid CP-OR algorithm have been compared over PLS problems. However, their hybrid algorithm did not explore the use of cuts in general nor of Gomory cuts in particular.

Thus, the use of general purpose cuts like Gomory cuts in hybrid CP-OR approaches (which was totally disregarded until now in the literature) seems promising. Indeed, such cuts can be separated rather quickly from current commercial and non-commercial ILP solvers which are also able to manage them in an efficient way avoiding numerical issues that are sometimes related to their use. Their effect in our context is twofold: first they improve each continuous relaxation by allowing a better selection of the branching variable (and value), second they act as a primal heuristic procedure to obtain feasible solutions for optimization problems.

		CP		H			H-Gomory			
					choice			choice	Gomory	
n	k	fails	time	fails	points	time	fails	points	time	runs
13	69	8.4e+06	328.54	0	7	9.45	0	2	3.60	10
14	68	1.7e+06	68.78	0	42	75.15	0	1	4.79	10
14	80	> 2.8e+07	>20m	0	24	33.55	0	6	15.12	10
15	46	1.4e+07	574.84	≥ 0	> 488	>10m	0	11	90.04	10
15	72	>3.0e+07	>20m	0	> 384	> 10m	0	7.4	27.79	10
15	75	>3.1e+07	>20m	0	70	155.02	0	0.1	1.972	10
15	76	>3.1e+07	>20m	0	73	163.85	0	47.9	144.78	10
15	77	>3.1e+07	>20m	0	69.3	157.70	0	13	49.01	10
15	85	>3.0e+07	>20m	0	43.8	86.37	0	9.2	34.75	10
15	89	>2.9e+07	>20m	0	50	90.43	0	5	15.78	10
16	82	> 2.9e+07	>20m	0	99	274.18	13	24	130.04	10
16	95	1.9e+07	863.44	≥ 0	> 467	>10m	0	4.5	39.81	10
16	102	> 2.9e+07	>20m	≥ 0	>516	>10m	0	35	115.79	10
16	109	>8.1e+07	> 1h	0	42	71.25	0	3	12.24	10
16	110	>2.7e+07	>20m	0	4.8	78.57	0	4.8	15.87	10
16	113	>2.6e+07	>20m	> 263	>323	> 312(mem)	0	19	54.42	10
16	114	>2.5e+07	>20m	0	46	90.5	0	14	38.35	10
16	115	>2.5e+07	>20m	> 50	>260	> 264(mem)	0	0	0.5	10
16	116	>2.6e+07	>20m	>1	>50	> 90.85(mem)	0		63.35	10
14	70	246.8	0.013	0	23	42.70	0	6	16.72	10
15	40	2.4	0.06	≥ 0	> 36	> 185 (mem)	0	18	156.25	10
15	55	30.8	0.009	≥ 0	>35	> 115.5 (mem)	0	8	48.95	10
15	70	3.4	0.03	0	72	312.34	0	20	72.93	10
16	70	106.6	0.07	≥ 0	>27	>135.5 (mem)	0	15	120.05	10
16	75	86655	3.56	0	50	203.35	0	21	132.34	10
16	85	65	0.07	>0	> 25	> 81 (mem)	0	2	15.57	10

Table 2. Comparison among algorithms `CP`, `H` and `H-Gomory`. "Mem" means memory limit exceeded.

6 Related Work

The work which is closest to ours is the one presented in [9], where different methods to find a solution of PLS completion problems are proposed and experimentally tested. However, their hybrid approach does not focus on forcing the use of cutting planes to integrate the CP and the OR appraoches. Moreover, the heuristics to choose the variables and their values are different from ours, and are probabilistically based on an LP rounding. As in our paper, also in [9] the conclusion is that none of the methods uniformly dominates the others.

In [6] they tackle the same class of problems but they just use a pure CP approach to solve them. Experimental results show that their CP approach can solve some PLS completion problems of order 45 in the phase transition (with 60% of preassigned cells), which [9], as well as our approach, can't solve with

hybrid approach. However, they use specific constraints which are tailored to the class of considered problems, while we employ generic Gomory cuts.

Another related work integrating constraint programming and integer programming is [1], where two algorithms to integrate CP and IP are compared to a pure CP algorithm and a pure IP one. While one of their two hybrid algorithms is similar to ours, they use generic cuts, whereas we use Gomory cuts. Moreover, we consider Latin Square Problems whereas they consider Mutually Orthogonal Latin Squares problem, which are tuples of Latin Squares where all pairs of corresponding elements in two of the problems are different. Their experiments show that integrating CP and IP is convenient as the problem size grows.

7 Conclusions

In this paper we considered the class of Partial Latin Square completion problems and we have studied the behavior of three complete search algorithms on several instances of such problems. One algorithm follows a pure CP search strategy, another one is a hybrid CP-ILP algorithm with no Gomory cuts, and a third one is a hybrid CP-ILP algorithm with Gomory cuts.

Other hybrid CP-OR approaches have been proposed in the past, and several works have considered Partial Latin Square Problems. However, in this paper we studied in particular the influence of Gomory cuts in the OR part of the algorithm.

Gomory cuts are not useful in our pure ILP approach. However, they are convenient in our hybrid CP-ILP algorithm. In fact, by comparing the algorithm `CP`, `H`, and `H-Gomory`, our experimental results show that:

- `H-Gomory` is always better than `H`;
- `H-Gomory` is better than `CP` in several instances in the critically constrained area, that is, when the problems have from 10% to 40% of the constraints (filled cells for PLS problems);
- `H-Gomory` is comparable to `CP` in the over-constrained region.

Thus, while there is no algorithm which is uniformly better on all problem instances, the hybrid algorithm with Gomory cuts looks convenient in terms of solving time especially for critically constrained problems (where it is better than `CP`) or for problems whose constrainedness cannot be known in advance. This algorithm thus combines the modelling capabilities of CP (with compactness given by global constraints) and its constraint propagation techniques, together with the convenient branch and cut strategies, typical of OR approaches.

Acknowledgments

We would like to thank Michela Milano and Alessio Guerri for many useful suggestions and comments on this work.

References

1. G. Appa, I. Mourtos and D. Magos. Integrating constraint and integer programming for the orthogonal latin squares problem. *CP 2002*, 17–32, 2002.
2. E. Balas, S. Ceria, G. Cornuejols and N.Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1-9, 1995.
3. N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Math. Comput. Model ling*, 20(12), 1994.
4. C. J. Colbourn. The complexity of completing partial latin squares. *Dicrete Applied Mathematics*, 8:25-30, 1984.
5. R. Dechter. *Constraint processing*. Morgan Kauffmann, 2003.
6. I. Dotú, A. del Val and M. Cebrián. Redundant Modeling for the Quasigroup Completion Problem. *CP 2003*, 288–302, 2003.
7. F. Focacci, A. Lodi and M. Milano. Cost-based domain filtering. *Proc. CP 99*, 189–203. Springer Verlag, LNCS 1713, 1999.
8. F. Focacci, A. Lodi and M. Milano. Cutting Planes in Constraint Programming: an hybrid approach. *Proc. CP 2000*, 187–201. Springer Verlag, LNCS 1894, 2000.
9. C. Gomes and D. Shmoys. Completing Quasigroups or Latin Square: A structured Graph Coloring Problem. *Proc. Computational Symposium on Graph Coloring and Generalizations*, 2002.
10. J. Hooker. *Constraint Programming and Integer Programming*. Tutorial at CP 2002. URL: `http://ba.gsia.cmu.edu/jnh/cornell.ppt`.
11. ILOG Solver 5.3 User's Manual, ILOG Hybrid Cooperating Optimizers 1.3.1 User's Guide and Reference, CPLEX 8.1 User's Manual, 2002.
12. M. Milano. Integration of OR and AI constraint-based techniques for combinatorial optimization. Tutorial at IJCAI 2001, Seattle. URL: `http://www-lia.deis.unibo.it/Staff/MichelaMilano/tutorial1IJCAI2001.pdf`.
13. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982.
14. P. Refalo. Linear Formulation of Constraint Programming Models and Hybrid So lver. *Proc. CP 2000*, 369–383. Springer Verlag, LNCS 1894, 2000.
15. J.C. Regin. A filtering algorithm for constraints of difference in CSPs. *Proc. AAAI-94*, 362–367, 1994.