Alcom-FT Technical Report Series
ALCOMFT-TR-02-109

Proceedings CPAIOR'02

Heuristic Constraint Propagation

Using Local Search for Incomplete Pruning and Domain Filtering of Redundant Constraints for the Social Golfer Problem*

Meinolf Sellmann
University of Paderborn
Department of Mathematics and Computer Science
Fürstenallee 11, D-33102 Paderborn, Germany
tel: +49-5251-606727
fax: +49-5251-606697
email: sello@uni-paderborn.de

Warwick Harvey
IC-Parc
Imperial College
Exhibition Road, London SW7 2AZ, UK
tel: +44-20-7594-8425
fax: +44-20-7594-8432
email: wh@icparc.ic.ac.uk

Abstract

For NP-hard constraint satisfaction problems the existence of a feasible solution cannot be decided efficiently. Applying a tree search often results in the exploration of parts of the search space that do not contain feasible solutions at all. Redundant constraints can help to detect inconsistencies of partial assignments higher up in the search tree. Using the social golfer problem as an example we show how redundant constraints that are potentially NP-hard themselves can be propagated incompletely using local search heuristics.

Keywords: redundant constraints, local search, incomplete propagation, social golfer problem, hybrid methods

*This work was partly supported by the German Science Foundation (DFG) project SFB-376, and by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

1 Introduction

Assume we are given an NP-hard constraint satisfaction problem (CSP). Even though there is no proof that we cannot solve the problem efficiently, there is strong empirical evidence that we cannot compute a solution in polynomial time. The common approach is to explore the search space in some sophisticated manner that tries to consider huge parts implicitly. For CSPs, that means that we try to cut off relatively large regions that do not contain any feasible solutions. In constraint programming, particularly when performing some kind of tree search, constraint propagation and arc consistency algorithms are used for that purpose. Basically, the model and the degree of propagation determine how the work is partitioned between the choice points and the search tree as a whole. That is, we can aim at reducing the number of choice points by spending additional effort in each of the choice points, or we can choose to keep the work done per choice point small, resulting in a bigger search tree.

Thus, we face a trade off between the time spent per choice point and the total number of choice points. To take the alternatives to extremes, on one hand we can explore the entire domain space, and on the other hand we can compute a solution or prove that none exists in the first node visited, without making any choices which might need to be backtracked. For most applications, the optimal balance will lie somewhere in between the two extremes.

If we find that we expand too many choice points we may want to give more burden to the individual choice points. Revising the model by adding redundant constraints is a common way to achieve that goal. In general, we expect the redundant constraints to detect inconsistencies higher up in the search tree. But since checking whether a given partial assignment is extendible to a full solution is usually of the same computational complexity as the original problem, redundant constraints typically still only enforce a relaxation of the actual problem.

We propose adding tight redundant constraints that may be hard to be verified exactly, but that can be checked by applying some heuristic. That is, when formulating additional constraints, we do not wish to restrict ourselves to considering only those constraints which are (relatively) easy to propagate completely. Instead, we perform an incomplete check of hard redundant constraints using the rich heuristic machinery that was developed in the operations research community.

The remaining paper is structured as follows: In Section 2 we review some of the research that has been performed on the integration of constraint programming and local search. To familiarize the reader with our example application, we proceed with an introduction of the social golfer problem in Section 3. Using this example we develop the idea of heuristic propagation of redundant constraints by means of local search in Section 4. Finally, in Section 5 we confirm our discussion by giving numerical results.

2 Literature on the Integration of CP and Local Search

In recent years, a substantial number of different approaches regarding the integration of constraint programming (CP) and local search (LS) have been developed. The main problem when constructing CP-LS hybrids is caused by the fact that CP uses monotonic reasoning whereas LS does not.

Fairly balanced hybrids result from sequential applications of the two methods [2, 11]. Other balanced hybrids can also be achieved by applying decomposition methods (like Lagrangian relaxation, column generation or Benders decomposition), where sub- and master problem can be solved by different solution methods. But there also exist many developments that favor one of the methods and just use the other one to overcome certain weaknesses. Constrained local search, for example, uses LS as the predominant approach, with CP used to find neighbors in a sparse and/or large neighborhood [1, 10]. Focussing on constraint programming instead has yielded hybrids that use local search to adapt the variable and/or value ordering in the search tree.

For a more complete overview on the field we refer to the recent tutorial by Focacci et al. [6].

As with other methods, constraint programming forms the basis of our approach. However, our use of local search is quite different to any of the methods mentioned above. For a given model for a problem, we suggest considering the addition of hard (with respect to computational complexity) redundant constraints, and then using local search to perform (incomplete) propagation of these constraints.

The idea of using a stochastic search method to prove unsatisfiability is not new; it was Challenge 5 in [12]. However, it appears that little work has been done on this, and to the best of our knowledge the work presented here is the first to do it, albeit on a fairly specific set of constraints that comprise subproblems of the real problem we are trying to solve.

3 The Social Golfer Problem

We introduce the idea of heuristic constraint propagation using the *Social Golfer Problem* as an example:

> 32 golfers want to play in 8 groups of 4 each week, in such way that any two golfers play in the same group at most once. How many weeks can they do this for? [1]

The problem can be generalized by parameterizing it to g groups of s players each, playing for w weeks, which we write as g-s-w from now on. When $(s-1)w = gs-1$, we have a configuration where every player must play with every other exactly once. This corresponds to a *resolvable Balanced Incomplete Block Design*. Perhaps the most well-known of these is *Kirkman's Schoolgirl Problem*, posed (and solved) by Thomas Kirkman in 1850. This instance, which is equivalent to the golfer 5-3-7 problem, can be stated as follows:

> How can 15 schoolgirls walk in 5 rows of 3 each for 7 days so that no girl walks with any other girl in the same triplet more than once?

Even though the description of the social golfer problem appears fairly easy, computational approaches have great difficulties solving even small instances in a reasonable amount of time. In our view, there are two main aspects to the problem that cause its big computational complexity:

[1] Problem 10 in CSPLib [3].

- The social golfer problem is highly symmetric.
- The clique structure of the constraints ensuring that any two golfers do not play together more than once makes it hard to judge the feasibility of a partial assignment.

Before we show how we can overcome some of the difficulties caused by the constraint structure of the problem by adding redundant constraints, we briefly review some previous work on symmetry breaking and the golfer problem.

3.1 Symmetries in the Golfer Problem

The social golfer problem contains a remarkable number of symmetries. Players can be placed at any position within a group, groups can be rearranged within their week, and the weeks can be ordered arbitrarily. Furthermore, the player names can be permuted in any way desired. To give an example: even the best models (in terms of symmetry reduction) for the original schoolgirl instance still contain more than 10^{12} symmetries.

Because of its highly symmetric structure, the social golfer problem has been used as a playground for research on the systematic breaking of symmetries in recent years.

In [14], Barbara Smith applied an approach that breaks symmetries during the search (SBDS [8]) to the social golfer problem. In combination with careful model selection she was able to efficiently break most of the symmetries, but still found non-unique solutions for the instances studied. Note that work is in progress which removes SBDS's need for an explicit list of symmetries [9]; eventually this should allow SBDS to be used to eliminate all symmetries from the problem.

In [7], Filippo Focacci and Michela Milano presented another generic method for breaking symmetries, based on *global cut seeds*, generating *symmetry removal cuts*. With this approach it should be possible to eliminate all the symmetries of the social golfer problem, but at the time of writing this has not been done.

Independently, Torsten Fahle, Stefan Schamberger and Meinolf Sellmann developed a very similar technique, SBDD [4]. It is based on the detection of dominance relations between choice points and works particularly well for highly symmetric problems. To date, this is the only technique which has been used to completely eliminate all symmetries from non-trivial instances of the social golfer problem.

We have chosen to apply symmetry breaking during search (SBDD) in combination with a straightforward model for the social golfer problem that can be implemented with very little effort using the ILOG Solver environment. The groups are modeled as sets of players with the cardinality of each set fixed to s. Each week contains g such sets, and the full pattern covers w weeks. Initially, we fix all the players in the first week in increasing order. Additionally, we insert the first s players into the first s groups for all weeks thereafter. Finally, the first group of the second week can be filled with the smallest indexed players possible. None of these initial labellings exclude any unique solutions.

4 Heuristic Propagation

After having introduced the social golfer problem, we now come to the actual focus of this paper, the heuristic propagation of additional redundant constraints by means of

	group 1			*group 2*			*group 3*			*group 4*			*group 5*		
week 1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
week 2	1	4	7	2	5	8	3	*	*	*	*	*	*	*	*

Table 1: A partial instantiation of the 5-3-2 social golfer problem.

local search.

We already mentioned that the constraints requiring that every golfer must not play with any other more than at most once makes it very hard to judge the extendibility of a partial assignment. That is, the subtree rooted by the current choice point may not contain any feasible solution, but due to the fact that the different constraints in the problem are propagated independently from each other and only interact via domain reductions, we cannot detect infeasibilities high up in the search tree. As a matter of fact, when using the model we described above, many searches will only backtrack when the assignments for an entire week are almost complete or even after having started to do assignments in the last week only.

Obviously, to check whether a partial assignment can still be extended to a feasible solution is of the same computational complexity as the original problem itself. Therefore, the pruning and filtering algorithm applied in the search nodes cannot be expected to be exact. Rather, it must be looked at as a heuristic to tighten the problem formulation and to shrink the search space. And of course, there is a trade off between the time needed to apply that heuristic and the time needed to explore the remaining subtree.

Now, to improve the situation for the social golfer problem as well as for many other CSPs, we can try to formulate necessary constraints for partial assignments to be extendible to complete, feasible solutions. For the social golfer problem (and, we believe, for many other problems as well), we are left with the decision to choose a weak redundant constraint that can be propagated efficiently, or to pick a condition that is more accurate but much harder to verify. For the latter, we may consider applying a heuristic to perform incomplete domain filtering or pruning. Note that since the added constraints are redundant, the incompleteness of this filtering does not affect either the soundness or the completeness of the search; if some opportunity for pruning is missed, it just means that the tree searched may be larger than strictly necessary.

In the following, we describe two different types of additional constraints that we would like to add to our model to be able to detect inconsistencies quickly and as early as possible. The first type of redundant constraint is defined with respect to the possibility of completing the assignments in a given week. Since we represent weeks by rows in our schedule under construction, we use the term *horizontal constraints* to refer to these constraints. Correspondingly, by *vertical constraints* we mean those used to check necessary conditions for a partial assignment to be extendible to a w-week solution.

4.1 Horizontal Constraints

Consider the example in Table 1. We are searching for a two-week schedule for 15 golfers, to be arranged in 5 groups of 3 in each week.

For the given partial assignment, suppose we add player 6 to group 3. Next observe that players 10, 11 and 12 must be separated in week 2. As there are only three more

	group 1			*group 2*			*group 3*			*group 4*			*group 5*		
week 1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
week 2	1	4	7	2	5	8	3	6	10	11	*	*	12	*	*

Table 2: A more complete partial instantiation of the 5-3-2 social golfer problem.

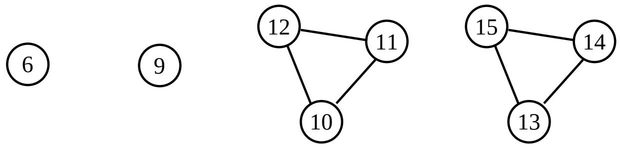

Figure 1: The residual graph of week 2 from Table 1.

groups that have not yet been filled completely, the third player in group three must be one of players 10, 11 and 12. A possible continuation is given in Table 2.

Now there are only two groups left that have not been completed yet, but players 13, 14 and 15 must still be separated. Therefore, the current partial assignment is inconsistent, and we can backtrack.

We can generalize this observation. For a given incomplete week, we define a residual graph R that consists of a node for each unassigned player and an edge for each pair of such players that have already been assigned to play together in some other week. An example of such a graph, corresponding to week two from Table 1, is shown in Figure 1. Then, for the given week we count the number of groups that are not closed yet and compare that number with the size of the biggest clique in R. If the first number is smaller than the latter, then there is no way to extend the current assignment to the rest of the week, and the assignment is inconsistent.

In this way we can define a sufficient condition for a witness which proves that the current partial assignment is inconsistent: a clique exceeding a certain cardinality. Finding a maximum cardinality clique is known to be NP-hard for arbitrary graphs. Even though the residual graphs we are dealing with have a special structure, we do not know how to exploit it to compute maximum cliques efficiently. Thus, rather than solving this problem exactly, we instead apply a heuristic search to find such a witness, and if we succeed, we can backtrack immediately.

But we can do even more: Reconsider the situation given in Table 1. We can add neither player 6 nor player 9 to group 3 for the same reason: the members of groups 4 and 5 of week 1 must be separated, and to do so we require the two open positions in group 3 of week 2.

When checking the redundant constraint described above, assume we set up the residual graph and the heuristic we apply finds the two disjoint cliques of size three ($\{10, 11, 12\}$ and $\{13, 14, 15\}$). Since the sizes of these cliques are equal to the number of incomplete groups, we have not found any witnesses showing that the current partial assignment cannot be extended to a full schedule — indeed, the schedule can still be completed. However, since group 3 has only two open positions left, we can conclude that group 3 must be a subset of $\{3\} \cup \{10, 11, 12\} \cup \{13, 14, 15\}$. That is, we can use heuristic information for domain filtering.

	group 1				*group 2*				*group 3*				*group 4*				*group 5*			
week 1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
week 2	1	5	9	13	2	6	10	*	3	7	*	*	4	8	*	*	*	*	*	*

Table 3: A partial instantiation of the 5-4-2 social golfer problem.

We conclude that finding a witness for unsatisfiability is a hard task, but it can be looked for by applying a heuristic. Moreover, even if we do not find such a witness, we may find other "good" witnesses, namely some fairly large cliques. Their information can be combined and used for domain filtering.

Therefore, it is advantageous to use a heuristic that not only provides us with good solutions quickly, but that also gives us several solutions achieving almost optimal objective values. Local search heuristics seem perfectly suited for this purpose.

4.1.1 Heuristic Clique Search

To find large cliques in the residual graph, we perform a randomized local search that works in the following way: We initialize our current clique C with a random node and set $C_{\text{best}} \leftarrow C$. Next we intensify C by repeatedly searching for a random node that is adjacent to all nodes in C. If no such node exists anymore, we compare the cardinalities $|C|$ and $|C_{\text{best}}|$ and update C_{best} if necessary. We then move on with a diversification step by adding a random node $v \in V \setminus C$ to C and removing all nodes in C that are not adjacent to v. These nodes shall not be considered in the next diversification step. Now, the loop is complete and we return to the intensification phase. The process stops after having found a clique that exceeds the crucial cardinality or after a given iteration limit.

Obviously, the approach as sketched above produces a sequence of cliques that we can use for pruning and domain filtering. We do not claim that the clique search procedure we use is very sophisticated for finding maximum cardinality cliques in a given graph. In fact, many other heuristic and exact approaches can be thought of, and there has certainly been a lot of relevant research done, including on the approximation of maximum cardinality cliques. However, we did not aim to develop a special method that produces one clique of large cardinality, but rather that finds a large number of fairly big cliques. In any event, since the residual graphs we are dealing with are rather small, we found that our local search procedure works satisfactorily in practice.

Before we continue by presenting another type of redundant constraint developed for the social golfer problem, we give a more complete example of how horizontal constraints can be used for pruning and domain filtering.

Consider the partial assignment for the social golfer instance 5-4-2 given in Table 3. The associated residual graph for week 2 is given in Figure 2.

The local search procedure we apply returns three disjoint cliques, namely $\{11, 12\}$, $\{14, 15, 16\}$, and $\{17, 18, 19, 20\}$. Since there are still four groups that have not been filled up yet and the largest clique is of size four, we cannot prune right away. But we do know that the final element in group 2 must come from the clique of size four, and so we can shrink the set of possible elements appearing in this group to $\{2, 6, 10, 17, 18, 19, 20\}$. This leaves just three open groups left, and we have a remaining clique of size 3 at hand. Again we cannot prune here, but we know that the remaining two elements of

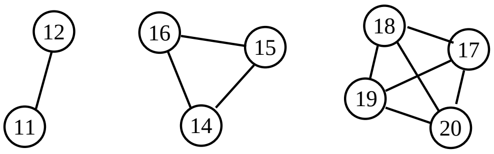

Figure 2: The residual graph of week 2 from Table 3.

	group 1			*group 2*			*group 3*			*group 4*			*group 5*		
week 1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
week 2	1	6	7	2	5	10	3	4	13	8	11	14	9	12	15
week 3	1	5	14	2	4	12	3	7	11	9	10	13	6	8	15
week 4	1	4	*	2	*	*	3	5	*	*	*	*	*	*	15
week 5	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
week 6	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
week 7	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Table 4: A partial instantiation of the 5-3-7 social golfer problem.

groups 3 and 4 must come from the cliques of size three and four, so we can shrink the set of possible elements of those groups to be $\{3, 7, 14, 15, 16, 17, 18, 19, 20\}$ and $\{4, 8, 14, 15, 16, 17, 18, 19, 20\}$, respectively. Now there is only one open group left, but we still have to separate players 11 and 12; as a result, the current assignment is inconsistent and we can backtrack.

The example shows that it can even be advantageous to have several cliques at hand rather than one big clique only: all cliques together allowed us to prune the search at the current choice point. But if we had known the largest clique of size four only, we would have had to expand the subtree below.

4.2 Vertical Constraints

Horizontal constraints are very helpful to judge the extendibility of the week currently under construction. But they do not help much in getting a clearer view of whether a partial assignment can still be extended to a full w-week solution. Therefore, we added another type of redundant constraint to our model, so-called vertical constraints.

Again, we start our discussion with an example (see Table 4). In the current partial assignment, week 4 can still be completed consistently. But is there still a continuation of the given schedule to a full 7-week solution?

Looking at player 15 (let us assume she is female), we find that she has played with all players in $\{6, 8, 9, 12, 13, 14\}$ already. As there are 4 weeks left that she has not yet been assigned partners for, she must still play with all players in $\{1, 2, 3, 4, 5, 7, 10, 11\}$. To be able to do so, there must be four independent pairs of players in this set that still have not been assigned to play with each other.

To check this, we define a residual graph again (this time for each player, in contrast to each week for horizontal constraints) that consists of one node for each player that player 15 has not been assigned to play with yet, and an edge between all such players

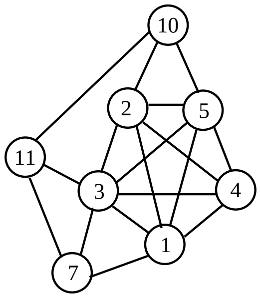

Figure 3: The residual graph of player 15 from Table 4.

	group 1	*group 2*	*group 3*	*group 4*
week 1	1 2 3	4 5 6	7 8 9	10 11 12
week 2	1 4 7	2 8 10	3 5 11	6 9 12
week 3	1 8 11	2 4 9	3 6 10	5 7 12
week 4	* * *	* * *	* * *	* * *
week 5	* * *	* * *	* * *	* * *

Table 5: A partial instantiation of the 4-3-5 social golfer problem.

that have already played with each other. The residual graph corresponding to player 15 from Table 4 is given in Figure 3.

When trying to find four disjoint stable sets (the term *independent set* is also used in the literature) of size two heuristically, we find that we do not succeed. But of course, this does not prove that there is none. That is, for the vertical constraints we need a witness that not enough disjoint stable sets of a given size exist anymore. The given example already gives an idea of what such a witness might look like. Looking at Figure 3, we find that the clique $\{1, 2, 3, 4, 5\}$ prevents us from finding four disjoint stable sets of size two. That is, a clique that exceeds a certain cardinality is a witness that the current assignment is inconsistent.

To find large cliques in the residual graph of a player, we can apply the local search procedure developed in Section 4.1.1. However, we are facing a slight drawback when using cliques as witnesses. For the schoolgirl problem, i.e. when we know that every player must play with every other player exactly once, the bound on the maximum cardinality clique in the residual graph can be chosen to be rather tight. But if we look at the social golfer problem instance 4-3-5 for example, every golfer does not play every other golfer; they play every other golfer except one, and a priori we cannot say which one that is. As a result, if we look for a clique large enough to guarantee that there will be too few disjoint sets of the appropriate size, that condition is stronger than we would like. This means that it does not become satisfied until later in the computation, and we are not able to prune as early as we would like.

To see an example of this, consider the partial schedule in Table 5. There are five players which player 12 has so far not been assigned to play with, and to complete the schedule we need two disjoint stable sets of size two. To prove that this is not possible

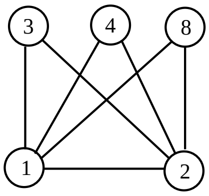

Figure 4: The residual graph of player 12 from Table 5.

	group 1	*group 2*	*group 3*	*group 4*
week 1	1 2 3	4 5 6	7 8 9	10 11 12
week 2	1 4 7	2 5 12	3 8 11	6 9 10
week 3	1 6 11	2 4 9	3 7 12	5 8 10
week 4	* * *	* * *	* * *	* * *
week 5	* * *	* * *	* * *	* * *

Table 6: A partial instantiation of the 4-3-5 social golfer problem.

with a single clique, we would need a clique of size four. Looking at the residual graph for player 12 (see Figure 4), there are three cliques of size three, namely $\{1, 2, 3\}$, $\{1, 2, 4\}$ and $\{1, 2, 8\}$, but no cliques of size four. However, there are no pairs of disjoint stable sets of size two either, so the schedule cannot be completed but this is not detected. (Note that all the residual graphs for the other players have exactly the same structure.)

As with the horizontal constraints, we can obtain better results by considering more than one clique at once. To illustrate this, consider the slightly different example in Table 6. Looking at the residual graph of player 12 (see Figure 5), we find two cliques of size three, namely $\{1, 4, 6\}$ and $\{4, 6, 9\}$. As before, there are no cliques of size four, but this time there is a pair of disjoint stable sets of size two: $\{1, 9\}$ and $\{4, 8\}$. So we cannot tell that the schedule cannot be extended simply by looking at this residual graph. However, we can draw inferences about which player is the one that player 12 will not be playing with: it must be an element of the intersection of all cliques of size three. This is because all cliques of size three have to be broken by the removal of a node; otherwise we are guaranteed that there is no way to partition the remaining four nodes into a pair of disjoint stable sets. Thus we can deduce that the player that player 12 must not play with is either player 4 or player 6.

We now look at the residual graphs for player 4 and player 6 (see Figure 5). For

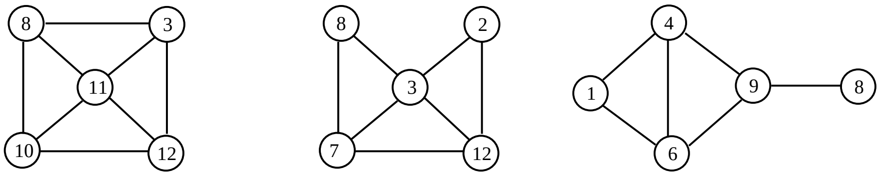

Figure 5: The residual graphs of players 4 (left), 6 (middle) and 12 (right) from Table 6.

player 4 we find that the intersection of cliques $\{3, 8, 11\}$ and $\{10, 11, 12\}$ requires that player 4 will certainly not play with player 11. Similarly, player 6 must not play with player 3, since, for example, $\{3, 7, 8\} \cap \{2, 3, 12\} = \{3\}$. This means both of these players must play with player 12, which contradicts the fact that player 12 must not play with one of them. Hence we can prune the search.

5 Numerical Results

To confirm our theoretical discussion, we implemented the model as described in Section 3.1 in C++, compiled by gcc 2.95 with maximal optimization (O3). All experiments were performed on a PC with a Pentium III/933MHz-processor and 512 MB RAM running Linux 2.4.

In the results reported here, we build up the schedule week by week, choosing as branching variable the group with the smallest domain or as branching value the player with the fewest possible groups he or she can be assigned to, depending on what leaves us with fewer choices.

To apply SBDD, we need to define a *symmetry detection function* ϕ that for two given choice points c_1 and c_2, represented as *patterns* reflecting the branching decisions taken so far, returns *true* if and only if there exists a symmetry showing that c_1 defines a subtree of c_2 under that symmetry. Then, at every choice point we check whether it is dominated in this fashion by some previously expanded choice point, and if so, we prune the search.

To find a symmetry that proves a dominance relation between choice points, we iterate over all week permutations in c_2 and search for a player permutation that is feasible with respect to the currently required matching of the weeks. That is, we set up a nested CSP to find a suitable symmetry or prove that none exists.

As this full dominance check is very expensive for the golfer problem (due to the large number of symmetries), we only perform it when a week is being filled completely. For the remaining nodes, we fix the player permutation to the identity and search for feasible orderings of the weeks, the groups and the players within the groups. This less expensive check is implemented as a pairwise dominance check between weeks, followed by the computation of a maximum cardinality matching on a bipartite graph $(V_1 \cup V_2, E)$ where the weeks in c_1 and c_2 define the nodes in V_1 and V_2, respectively, and $(v_i^1, v_j^2) \in E$ iff week i in c_1 is dominated by week j in c_2. For further details we refer to [4].

Regarding the additional redundant constraints, we present a comparison of four different parameter settings:

1. The plain implementation without redundant constraints (PI),
2. plus horizontal constraints only (H),
3. plus vertical constraints only (V), and
4. plus horizontal and vertical constraints (HV).

In Tables 7 and 8, the variants are evaluated on several social golfer instances. To make the comparison fair and reduce the impact of other choices such as the variable and

	4-3-3	4-3-5	5-4-3	5-4-6	5-3-3	5-3-7
(PI)	102	227	7430	6409	17129	
(H)	76	90	6205	2818	16396	266268
(V)	100	116	7415	3300	16973	153697
(HV)	76	74	6205	1770	16396	85790

Table 7: The number of choice points visited to find all unique solutions.

	4-3-3	4-3-5	5-4-3	5-4-6	5-3-3	5-3-7
(PI)	0.32	0.98	46.63	142.51	109.78	
(H)	0.23	0.31	40.19	33.8	109.88	13310.8
(V)	0.36	0.47	57.3	46.69	117.68	1815.27
(HV)	0.25	0.26	47.78	23.21	115.47	393.96

Table 8: The CPU time needed to compute all unique solutions (seconds).

value orderings used, we compute all unique solutions of an instance (or prove that there are none), counting the number of choice points and measuring the CPU time required.

Clearly, using both types of additional constraints results in the biggest reduction of choice points. Whether or not that gain also results in a reduction of CPU time is determined by the trade off between the time needed to apply the incomplete propagation algorithm and the time saved by the reduction of choice points. Generally, the CPU time of (HV) is very good, but sometimes it is outperformed by (H). This occurs when the number of weeks is small, and is because for small numbers of weeks, the vertical constraints do not prune the search tree very often. Then, their application is ineffective and therefore a waste of time.

But also for instances with many weeks, the algorithm version (H) still yields surprisingly good results. This can be understood by considering that the horizontal constraints will prune the search when a partial week cannot be extended to a full week. When there are many weeks, there is less flexibility available when deciding which players should play with which (and hence a partial week assignment is much less likely to be extendible to a full week). The more weeks there are and the closer the instance is to requiring every player to play with every other player, the more important the horizontal constraints become: for example, finding all solutions to the 5-3-7 instance (which requires every player to play with every other) cannot be tackled in a reasonable amount of time using (PI), but with the horizontal constraints, we find all seven unique solutions in less than four hours.

Using the (HV) setting, we have been able to compute all unique solutions for all instances with at most 5 groups and 5 players per group (see Table 9). To the best of our knowledge, this is the first computational approach that has been able to compute these numbers for instances of this size. Moreover, we found solutions for many previously unsolved (at least by constraint programming) larger instances, such as the 10-6-6 and the 9-4-6 instances.

	Number of Groups – Number of Players per Group									
Number of Weeks	2-2	3-2	3-3	4-2	4-3	4-4	5-2	5-3	5-4	5-5
2	1	1	1	2	1	1	2	2	1	1
3	1	2	1	8	4	2	23	251	40	2
4	0	1	1	16	3	1	310	13933	20	1
5		1	0	19	0	1	3468	9719	10	1
6		0		13		0	13277	49	0	1
7				6			14241	7		0
8				0			3192	0		
9							396			
10							0			

Table 9: The number of unique solutions for several social golfer instances.

6 Conclusions

Using as an example the social golfer problem, we have introduced the idea of heuristic constraint propagation for hard redundant constraints. We proposed two different types of additional constraints, so called horizontal and vertical constraints. Propagating both types of constraints exactly would require the computation of maximum cardinality cliques in residual graphs with certain structural properties. Instead, we perform an incomplete propagation using a local search method to find a number of large cliques instead of one maximal one. We have shown how such sets of fairly good solutions can be used for domain filtering and pruning. The experiments clearly show that adding tight redundant constraints to the problem can be of benefit, even when they can only be propagated incompletely.

Our ability to use local search for domain filtering and proving unsatisfiability relies on the fact that we have identified sufficient conditions for proving that a partial schedule cannot be extended to a complete one, and that a local search procedure can be defined such that a solution returned is a proof that the conditions have been met. It would be interesting to see what other kinds of constraints this can be done for, and whether it can be generalized far enough to pass Challenge 5 from [12].

References

[1] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem, *ORSA*, Journal on Computing, 3:149–156, 1991.

[2] Y. Caseau and F. Laburthe. Heuristics for large constrained routing problems, *Journal of Heuristics*, 5:281–303, 1999.

[3] *CSPLib: a problem library for constraints*, maintained by I.P. Gent, T. Walsh, B. Selman, `http://www-users.cs.york.ac.uk/~tw/csplib/`

[4] T. Fahle, S. Schamberger, M. Sellmann. Symmetry Breaking, *Proc. of CP'01*, LNCS 2239, pp. 93–107, 2001.

[5] T. Feo and M. Resende. Greedy randomized adaptive search procedures, *Journal of Global Optimization*, 6:109–133, 1995.

[6] F. Focacci, F. Laburthe, A. Lodi. Local Search and Constraint Programming, *Handbook of Metaheuristic*, Kluwer Academic Publishers, to appear.

[7] F. Focacci and M. Milano. Global Cut Framework for Removing Symmetries, *Proc. of CP'01*, LNCS 2239, pp. 77–92, 2001.

[8] I.P. Gent and B. Smith. Symmetry Breaking During Search in Constraint Programming, *Report 99.02, University of Leeds*, Jan. 1999.

[9] I. McDonald. Unique Symmetry Breaking in CSPs Using Group Theory, *Proc. of Workshop on Symmetry in Constraints (SymCon'01)*, Dec. 2001.

[10] G. Pesant and M. Gendreau. A view of local search in constrained programming, *Principle and Practice of Constrained Programming (CP'96)*, LNCS 1118, pp. 353–366, 1996.

[11] S. Prestwich. A hybrid search architecture applied to hard random 3-SAT and low-autocorrelation binary sequences, *Principle and Practice of Constrained Programming (CP 2000)*, LNCS 1894, pp. 337–352, 2000.

[12] B. Selman, H. Kautz and D. McAllester. Ten Challenges in Propositional Reasoning and Search, *Proc. IJCAI'97*, pp. 50–54, Aug. 1997.

[13] ILOG. ILOG SOLVER. Reference manual and user manual. V5.0, ILOG, 2000.

[14] B. Smith. Reducing Symmetry in a Combinatorial Design Problem, *Proc. of CPAIOR'01*, Wye, UK, pp. 351–360, April 2001.