Using Global Constraints for Frequency Allocation

ASTEC Technical Report 97/07

Greger Ottosson
Computing Science Dept.
Uppsala University
`greger@csd.uu.se`

Mats Carlsson
Swedish Institute of
Computer Science
`matsc@sics.se`

July 2, 1997

1 Introduction

This report describes an experiment in using a global constraint to improve the performance of the Freplan frequency allocation algorithm for mobile telephone networks [CG93, CO96]. Global constraints have arisen as a part of the recent trend of integration between traditional operations research techniques (OR) and constraint programming, and their usefulness has been shown in e.g. [PL95, BC94, AB93].

A mobile telephone network is partitioned into *cells*. Each cell is served by a unique *base station*. The problem consists in assigning a set of frequencies to the cells, while minimizing the potential interference. Thus, the problem constraints are *soft*: each constraint prescribes a minimal separation between two frequencies and has a weight denoting the interference that will be incurred if the constraint is violated. Problem instances are usually over-constrained; that is, not all constraints can be satisfied, and the objective then becomes to minimize the interference, according to some criteria. The criteria used in this work is that of minimizing the maximal interference. For a complete description of the problem, see [CG93, CO96].

Roughly, the search for the optimal solution to the allocation problem is done as follows. The set of constraints is sorted with respect to weight, and then partitioned wrt. a *threshold* value into two sets of *passive* and *active* constraints, such that the threshold equals the maximal weight of the passive constraints and all active constraints have weight greater than the threshold. The problem is then solved treating the active constraints as *hard* constraints and ignoring the passive ones. If a solution is found, the threshold is lowered (more constraints are made active) and the search for a solution repeated. If no solution is found for a set of active constraints, the threshold is raised (some constraints are made passive) and the process continues this way until the optimal solution is found for the problem as a whole, at which time the threshold equals the maximal interference of the solution.

The work described here kicks in after the partitioning, where the weight of constraints have disappeared, that is, all constraints are hard and must be satisfied. The problem can then be viewed as a graph, where nodes in the graph correspond to cells and edges in the graph to constraints. In a graph, a *clique* is a complete subgraph, i.e. a set of nodes which are all adjacent to each other, and a *maximal clique* is a complete subgraph not contained as a subset of a larger complete subgraph [Rob84, Tsa93]. In our graph we search for cliques (not necessarily maximal), i.e. sets of nodes where all pairs of nodes are connected through a constraint, and cover this set of constraints by a single new constraint. This new constraint is called `all_distinct/1` and expresses the fact that a set of variables should all have distinct values. This constraint is in effect equivalent to that of the binary constraints it covers, but is through its implementation able to propagate information between nodes more eagerly, and may thus reduce the search needed for finding a solution. In this report we

[0] Available from `http://www.docs.uu.se/astec/Reports/Reports/9707.ps.gz`

describe the algorithm used for implementing the constraint and how it is applied to Freplan. We also evaluate its efficiency.

This report is structured as follows. Section 2 describes how cliques are found in the graph, how the constraint itself is implemented and how it is applied to the propagation machinery of Freplan. Section 3 provides the result of applying this technique using different heuristics and also compared to standard Freplan. We conclude in Section 4 by discussing the impact of the results obtained.

2 Implementation

This section describes how we identify portions (cliques) of the constraint graph to which we can apply the new constraint (Sec. 2.1), the implementation of the global constraint itself (Sec. 2.2) and how it is incorporated in the propagation machinery of Freplan (Sec. 2.3).

2.1 The clique detection algorithm

As mentioned above, the idea is to cover a complete subgraph (the binary constraints of a clique) in the constraint graph by a single constraint. Figure 1 illustrates how this can be done. Binary constraints

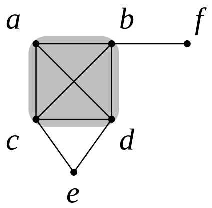


Figure 1: Clique Detection

between nodes a, b, c and d are identified as a clique and a new constraint `all_distinct([`a, b, c, d`])` is added. This new constraint enforces the same restrictions on frequencies assigned to cells a, b, c and d, but is able to rule out frequencies more eagerly since it covers more than two variables and uses a specialized algorithm. More on how this is done can be found in Sec. 2.2.

This section will describe different techniques or heuristics for clique detection.

The Bucket Packing Heuristic

This algorithm is based on a simple "bucket technique", where nodes are dropped into buckets, each forming a clique. We check each node N with each existing bucket B, and add N to B iff N is actively constrained to each node N' in B. The heuristic controls in which order we try nodes, and in how many buckets a single node may be added. The latter gives some control of the overlap between cliques. The intuition is that increased overlap means increased propagation strength, but also increased cost.

This heuristic *covers* or puts a layer of global constraints on the constraint graph, generally increasing propagation strength. High values for the parameter (i.e. how many cliques a single node may occur in), will usually result in a large number of strongly overlapping cliques.

The k-clique Heuristic

As opposed to the previous bucket packing technique, where a set of global constraints are spread over the constraint graph, one may consider trying to isolate a few places where the extra propagation strength might do more use than it would on average.

The k-clique heuristic determines, before optimization begins, all cliques of size k, where k is the number of available frequencies, and saves them. For each clique C we annotate a weight $w = min(\{w_i | c_i \text{ is a constraint in } C\})$, indicating at which time in the optimization procedure this clique should become active. During optimization, this clique is made into a global constraint when the current threshold becomes less that w.

As we will see later, for many benchmarks, most, if not all, k-cliques have a very low weight w, thus they will become active very late into the optimization, if at all. A natural adaptation of this heuristic for that kind of situation is to collect not only k-cliques, but also $(k-d)$-cliques, where d is set appropriately. We will thus collect a set of cliques larger than or equal to $k-d$, but only cliques that are not subsumed by any other clique in the set. For example, say that C_1 is a clique of size 7 and C_2 a clique of size 6, $C_2 \subset C_1$. C_2 will not be in the set, and neither will any other clique that is a subset of C_1. But if C_2 has a larger weight than C_1, then it should, for some threshold during the optimization be active when C_1 is not. The annotated weight should be generalized to be an interval in which the clique should be active. This is currently not implemented, so some $(k-d)$-cliques may be lost.

2.2 The Global Constraint

The constraint `all_different(`L`)` states that a list of variables L should all have distinct values. This can be expressed as a set of binary constraints between each pair of variables x and y in L, i.e.

$$\forall x, y \in L \; x \neq y$$

E.g., consider an instance of the constraint, `all_different([`a, b, c, d`])`, where domains $a, b \in \{1, 2, 3\}, c \in \{2, 3, 4, 5\}$ and $d \in \{3, 5\}$. Say 5 is removed from the domain of d. That makes d determined to 3, and the pairwise constraints representing `all_different/1` will remove 3 from the domains of a, b and c. Thus, the domains are, after the propagation: $a, b \in \{1, 2\}, c \in \{2, 4, 5\}$ and $d \in \{3\}$.

Studying this example, one will notice that a and b will jointly occupy 1 and 2, regardless which one takes which value. So, 1 and 2 are not possible values for c and 2 could then be removed from the domain of c. This will not be done if the `all_different/1` is implemented by the pairwise disequality constraints. A constraint that does not remove all values that can be safely removed is called *incomplete*. The rest of this section will describe an algorithm due to Régin [Reg94], which implements a *complete* version of this constraint, i.e. one that would remove 2 from the domain of c as described above. This is the algorithm used in our extension to Freplan.

Given variables $x_1, \ldots, x_n$ and corresponding domains $d_1, \ldots, d_n$, let G be the bipartite graph where the variables form the left hand side nodes, the union of the domains the right hand side nodes and edges represent domain inclusion. That is, there's an edge from node x_i to node v_j iff $v_j \in d_i$. Figure 2(a) depicts this for our previous example. For example, node a has an edge to node $1, 2$ and 3 because it has value $1, 2$ and 3 in its domain. The edge that is about to be removed is the dashed line at the bottom.

The question now is: what edges (domain elements) can be in some satisfiable solution to this constraint? To answer this, and to reveal the algorithm for computing it, we will first introduce some terminology.

A *matching* in a bipartite graph is a set of edges with no nodes in common. A *maximal matching* in a bipartite graph is a matching with as many edges as possible [HR90]. In our case we will require that every left hand side node is covered by exactly one such edge and that every right hand side node are covered by at most one such edge. These edges are called *matched*. Nodes that are covered are also called *matched* and those that are not matched are called *exposed*. Figure 2(b) shows a maximal matching for our example. Node 4 is matched and node 5 is exposed. A *strongly connected component* (SCC) in a graph is a set of nodes such that from every node we can get to any other node in the SCC by following edges. A *path* in a graph is a set of consecutive edges, i.e. two consecutive edges in the path share a common node. An *alternating path* is a path starting in an exposed node, following an even number of edges and ending in a matched node.

Now, the algorithm for finding which edges that can, and that can not be in a solution to the constraint is as follows:

1. Compute a maximal matching to the bipartite graph, and mark all matched edges as *used*, i.e. they are trivially in a solution.

2. Make the matched edges directional, going from left to right.

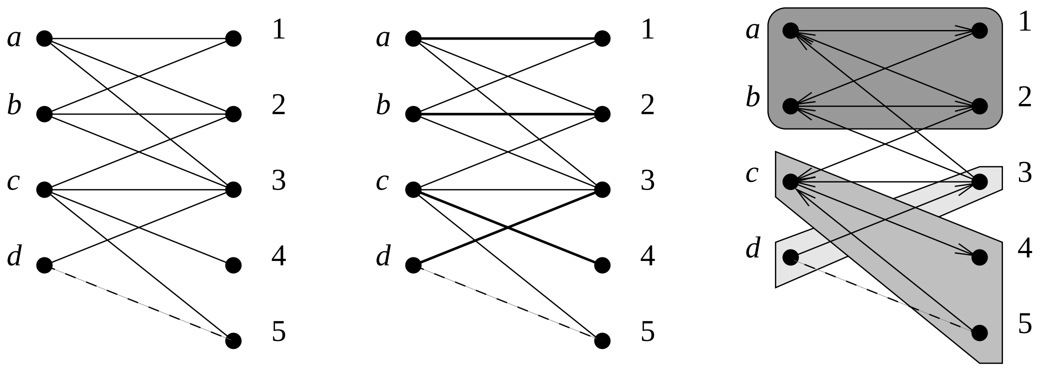
3. Make all remaining edges directional and go from right to left.

4. Find all alternating paths and mark all involved edges as *used*.

5. Find all strongly connected components, and mark all edges between involved nodes as *used*.

6. All edges not marked as *used* cannot be in any true instance of this constraint, and the corresponding values can be removed from the corresponding domains.

For a proof that this algorithm is correct, see [Reg94]. The intuition is to consider all possible



(a) Bipartite Graph (b) Maximal Matching (c) Processed Graph

Figure 2: Illustration of the complete `all_distinct/1` constraint

maximal matchings and how those can be derived from a single maximal matching. Consider our example again. The result of applying the algorithm is shows in Fig. 2(c). The matched edges from 2(b) are directed going from left to right, and the opposite is true for all other edges. The darkest gray section shows the single SCC this graph contains, the middle darkest section show an alternating path $5 \to c \to 4$ (remember that 5 is the only exposed node), and the lightest gray region shows an edge that is *used* since it's a part of the current matching. Following the algorithm above, the values 2 and 3 will be removed from the domain of node c.

Why can all edges in an alternating path be in a solution? Well, given the current maximal matching, all matched edges in the path can be swapped with all the non-matched edges and we have a new maximal matching, which trivially is a solution. The same is true for the edges going between nodes in a strongly connected component.

Having understood the constraint, we may also construct a test that can determine—before the algorithm is executed—whether or not the algorithm *might* result in the pruning of a domain. An edge is removed because it points to an value node that is already spoken for, and a value node can be spoken for by either an SCC or a determined variable node (which has a single outgoing edge, like d in Fig. 2(c)). In both cases, the following criteria much be fulfilled for that to happen. Let $d_1, \ldots, d_n$ be the domains of the nodes.

$$\{d'_1, \ldots, d'_m\} \subseteq \{d_1, \ldots, d_n\}$$
$$m \leq |d_1 \cup \ldots \cup d_m|$$

In words, there must exist a subset of the nodes whose domains coincide enough to form an excluding SCC. This criteria is tested before executing the algorithm of the constraint.

2.3 The Propagation Machinery

As outlined above and more thoroughly described in [CG93, CO96], solving a frequency assignment problem is in Freplan done by making increasingly more constraints active, and then solving for a solution to the active constraints using search and constraint propagation. While searching, constraint propagation is done to avoid search in futile branches of the search tree. Constraints are actively used to prune domains, and thus reduce search. After the clique detection has been done (see Sec. 2.1), and some (usually not all) of the binary constraints have been covered with a new `all_distinct/1` constraint, we have a set of constraints of two types; binary and global.

Binary constraints connect two variables, global constraints connect an arbitrary amount of variables. Each constraints is *suspended* on all the variables it connects; this is to enable all relevant constraints to be checked when the domain of a variable is pruned.

The propagation algorithm is essentially centered around a queue Q of variables whose domains have been pruned. For each step in the propagation procedure, a variable v is removed from Q, and the constraints associated with v is checked. When checking a constraint infeasible values of the associated variables are removed from the domains. For binary constraints this is done on a per value basis, maintaining *arc-consistency* [Mac77, Van89], while the complete `all_distinct/1` algorithm (see Sec. 2.2) is used for the global constraints. Variables whose domains have been pruned are added to Q, unless they are already there.

Since global constraints are more expensive to check, we have experimented with delaying their check until there are no more binary constraints in Q. This has been done using a second queue G, to which global constraints are added instead of being checked. The propagation procedure runs as normal (checking only binary constraints) until Q is empty. Then a global constraint from G is checked, and the process iterates. This scheme will ensure that the binary constraints will handle efficiently the propagation it *can* handle, while the global constraints will trigger and execute if the condition for possible pruning is fulfilled (see Sec. 2.2).

3 Evaluation

This section will try to evaluate the usefulness of the described global constraint in Freplan. A comparison is made for a set of benchmarks between ordinary Freplan (Freplan Standard) with only binary constraints, Freplan with global constraints using the bucket packing heuristic (Freplan Cover) and Freplan with global constraints using the k-clique heuristic (Freplan KClique). The criteria for comparison will first and foremost be that of performance, i.e. execution time related to quality of solution, but we will also try to see the potential of stronger consistency methods by comparing the quality of solution with search performed instead of time elapsed.

All benchmarks have been run on a SUN SPARCstation 4 with a 85MHz microSPARC CPU. Characteristics of the benchmarks are shown in Fig. 3.

3.1 Performance

Figure 5 shows the performance of Freplan Standard, Freplan Cover and Freplan KClique with respect to time. The x-axis is time in minutes on a logarithmic scale, and the y-axis is interference. A dot in the plot corresponds to a solution found by the anytime algorithm with interference (solution quality) y at time x.

The following conclusions can be drawn. The global constraints are quite expensive computationally (not surprising), which is especially clear when looking at the curve for Freplan Cover, which for most benchmarks lags behind Freplan Standard in finding the initial solutions.

Freplan KClique diverges less from Freplan Standard than Freplan Cover does. This is because it is more restrictive with creating global constraints, and because these constraints do not kick in initially, but rather at a certain threshold. This is shown clearly when studying A and D among others. Although, when it *do* kick in, it is generally not improving the performance.

Figure 4 shows for benchmark A a comparison between the two heuristics. There are three bars for each heuristics; the first corresponds to the number of invocations made, the second to how many invocations passed the viability test (see Sec. 2.2), and the last to the number of invocations that resulted in the pruning of some domain. We can, for these heuristics and parameters see that bucket packet heuristics cause more invocations, and that a larger portion of these invocations is useless.

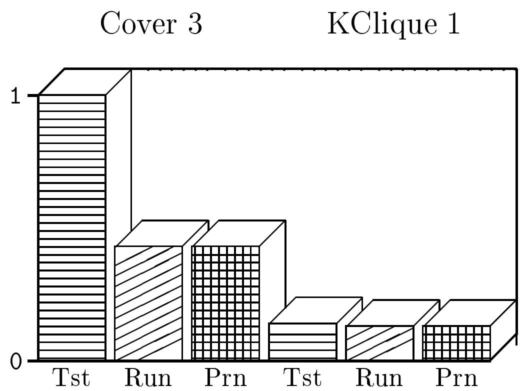


Figure 4: The effectiveness of the global constraint

Figure 6 shows the propagation strength of Fre-

Instance	#Cells	#Constraints	Density	#Frequencies
A	903	42474	10%	21
B	87	3221	86%	48
C	105	2109	39%	12
D	158	3333	27%	24

Figure 3: The characteristics of the problem instances. Density denotes the ratio of constraints above optimal threshold to the total amount of possible constraints.

plan Standard, Freplan Cover and Freplan KClique on a set of different benchmarks. Here, the x-axis is a value corresponding to the search depth, and a dot in the graph thus represents a solution with interference y found at a search depth x. The value for the x-axis is calculated as a function of the width of the iterative broadening [CO96] and the backtrack limit [CO96].

As shown in the previous section, there is not a general performance improvement of using the global constraint. Only in some special cases is it beneficial. On the other hand, we can from these figures see that when compared not to time, but rather to the search depth, the global constraint does improve Freplan. So there is potential for an performance improvement by using stronger propagation techniques — as the global constraint — but the cost in this case outweighs the benefits.

3.2 Clique Detection Heuristics

In this section we will try to see how the two different clique detection heuristics we have used can be tailored for maximum performance.

The Bucket Packing Heuristic

The bucket packing technique (Sec. 2.1) used for clique detection has one parameter that can be altered, the maximal number of cliques a single variable can occur in. The results for the benchmarks with the parameter ranging from 1 to 9 is shown in Fig. 7. From the graph one can conclude that the heuristic is not stable; there seems to be no single best value for the overlap heuristic.

The k-clique Heuristic

The $(k-d)$-clique heuristic (Sec. 2.1) used for clique detection also has one parameter that can be altered, the size of cliques detected. The results for the benchmarks with the minimum clique size d ranging from 0 to 2 is shown in Fig. 8. For most of the benchmarks, the cliques do not become active at all, and in cases when they do (e.g. A and C), they are too costly to be of benefit.

4 Conclusion

We have added a global constraint for expressing disequalities between variables to the frequency allocation program Freplan, with the aim of improving performance through reduced search. In short, *the propagation is improved, the search is reduced, but the cost is too high to make an overall performance improvement.*

We have put reasonable effort into this attempt, i.e. the implementation of the global constraint is efficient and the heuristics tried for clique detection are not naive. This does not mean that global constraints for frequency allocation in general — and Freplan in particular — is a dead end. An even more optimized implementation of the constraint, and an even more clever and selective heuristic for clique detection *could* result in an improved frequency planner. The fact that problem instances still grow bigger might also be on our side, since reduced search then become increasingly important.

On the other hand, one might also consider the result of this attempt as a sign that more radical changes in the approach is of need.

Finally, a note on how the extensions described in this paper relates to the objective function used for optimizing. Freplan itself is inherently subject to minimizing the *maximal* interference, due to its way of partitioning constraints into passive and active. Since our extension here is on a lower level, and does not alter how the search on a higher level is performed, we are subject to the same limitation.

Should the search of Freplan be changed to allow, for example, to minimize the sum of the interferences of violated constraints, our extension would thus carry over without changes.

References

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.

[BC94] Nicolas Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.

[CG93] M. Carlsson and M. Grindal. Automatic frequency assignment for cellular telephones using constraint satisfaction techniques. In David S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 647–665, Budapest, Hungary, 1993. The MIT Press.

[CO96] Mats Carlsson and Greger Ottosson. Anytime frequency allocation with soft constraints. In *CP96 Pre-Conference Workshop on Applications*, 1996.

[HR90] Nora Hartsfield and Gerhard Ringel. *Pearls in Graph Theory. A Comprehensive Introduction.* Academic Press, Boston, 1990.

[Mac77] Alan Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[PL95] Jean-Francois Puget and Michel Leconte. Beyond the glass box: Constraints as objects. In John Lloyd, editor, *Proceedings of the International Logic Programming Symposium (ILPS-95), Portland, Oregon*, pages 513–527. MIT Press, 1995.

[Reg94] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367. Amer. Assoc. Artificial Intelligence, 1994.

[Rob84] F.S. Roberts. *Applied Combinatorics.* Prentice-Hall, 1984.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[Van89] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, Cambridge, MA, 1989.

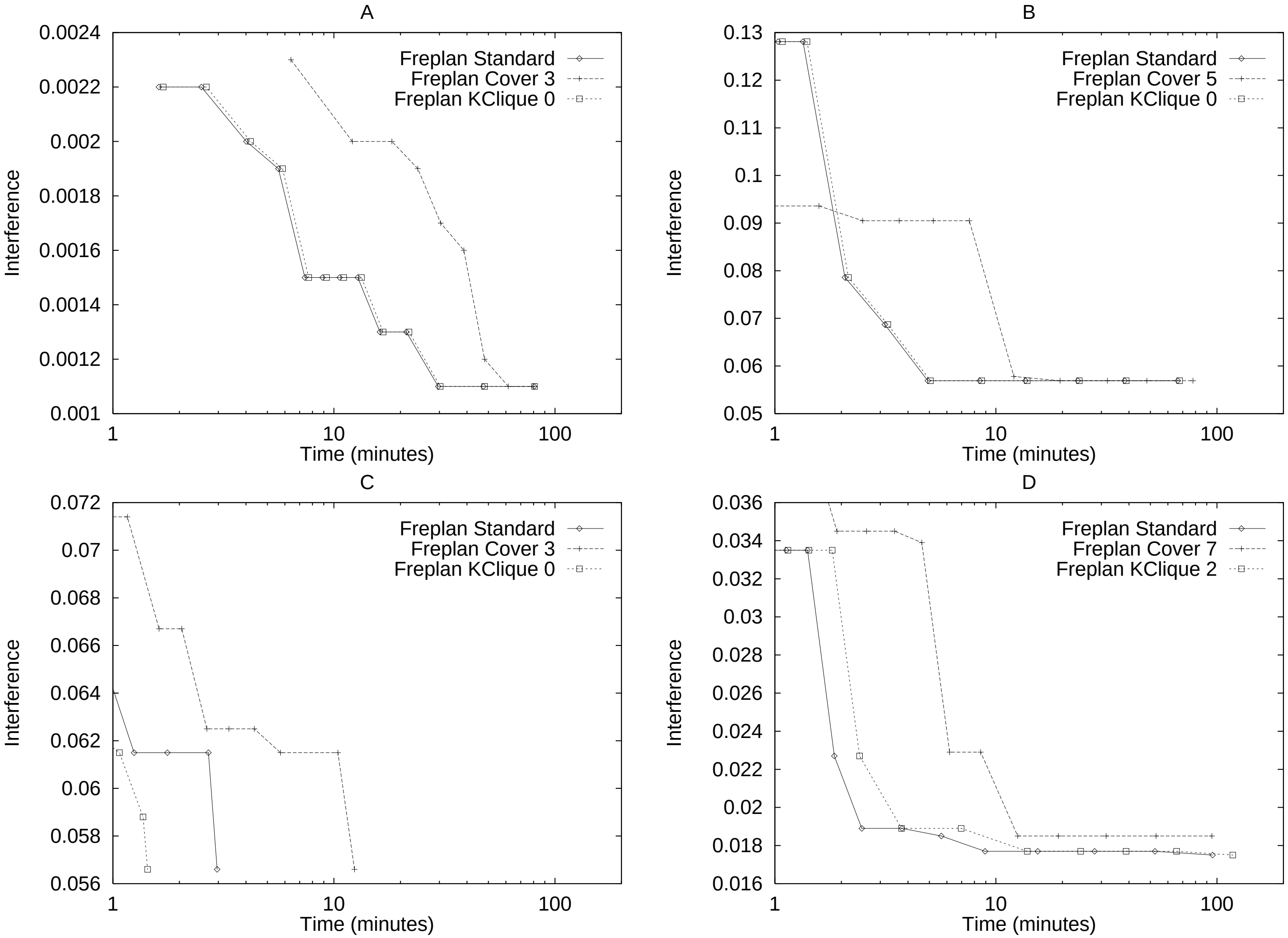


Figure 5: Freplan with binary constraints versus Freplan with global constraints

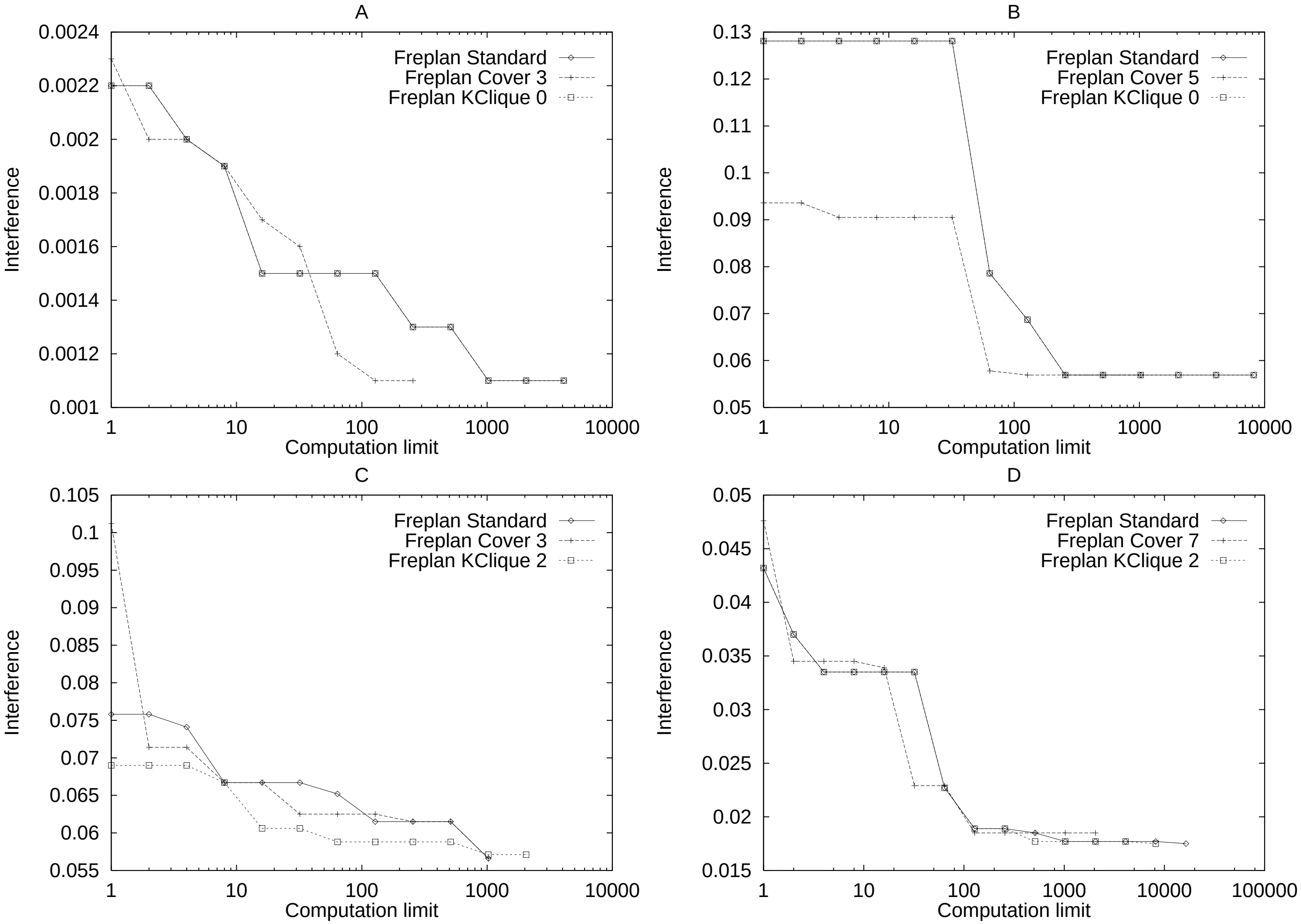


Figure 6: Freplan with binary constraints versus Freplan with global constraints

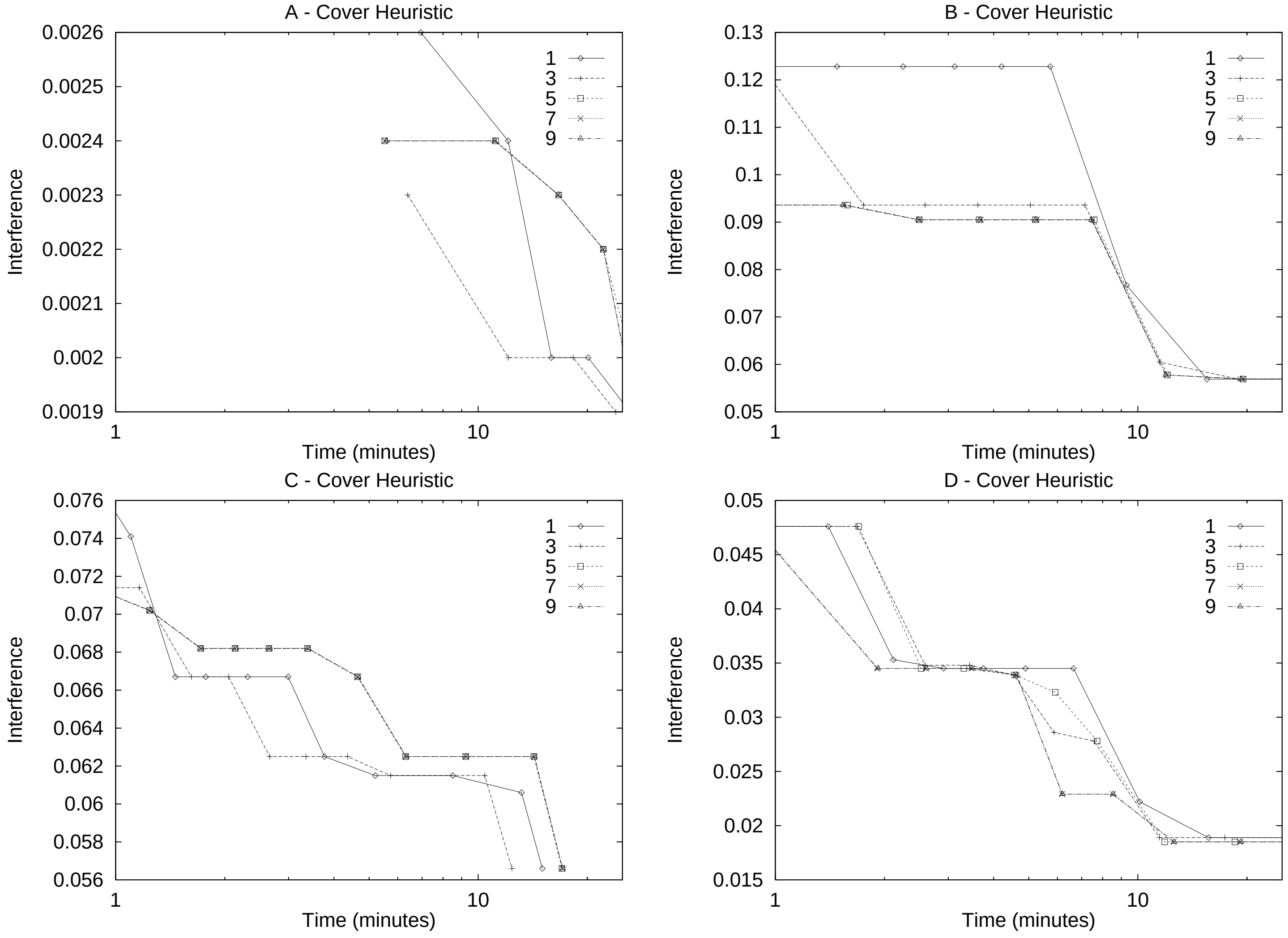


Figure 7: The bucket packing heuristic with different overlap limits

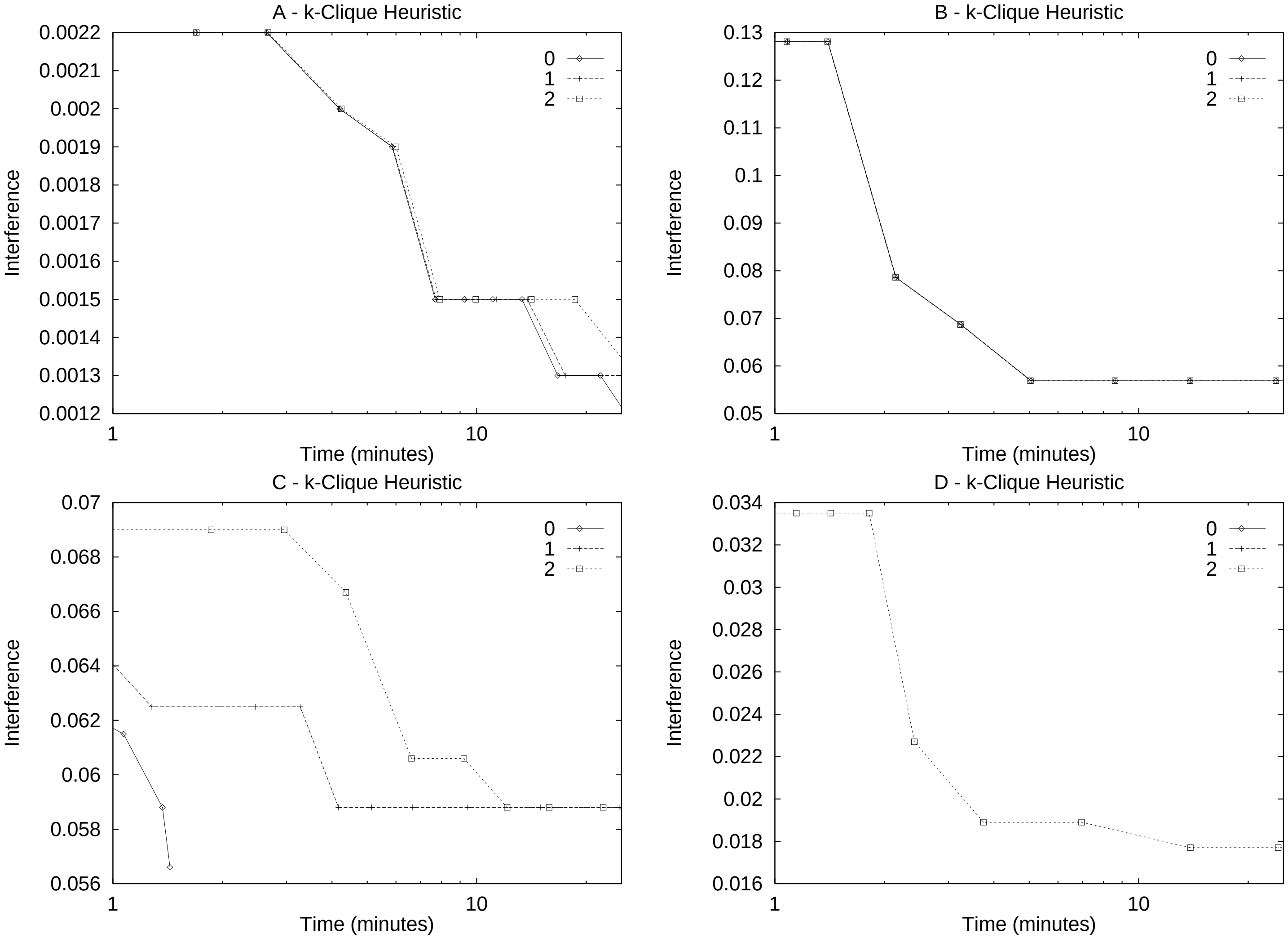


Figure 8: The bucket packing heuristic with different values for d