The Mcf-Separator – Detecting and Exploiting Multi-Commodity Flow Structures in MIPs*

Tobias Achterberg† Christian Raack‡

Abstract

Given a general mixed integer program (MIP), we automatically detect block structures in the constraint matrix together with the coupling by capacity constraints arising from multi-commodity flow formulations. We identify the underlying graph and generate cutting planes based on cuts in the detected network. Our implementation adds a separator to the branch-and-cut libraries of Scip and Cplex. We make use of the complemented mixed integer rounding framework (c-MIR) but provide a special purpose aggregation heuristic that exploits the network structure. Our separation scheme speeds-up the computation for a large set of MIPs coming from network design problems by a factor of two on average.

Keywords: mixed integer programming, network detection, cut-based inequalities

1 Introduction

In this paper we present a novel separation heuristic for general mixed integer programs (MIPs), which we call multi-commodity flow cut separator (Mcf), now available in Scip 1.2 [60] and Cplex 12.1 [33].

The Mcf separator identifies a coupled multi-commodity arc-flow formulation in the constraint matrix and constructs the corresponding network. It then generates inequalities based on cuts in the detected network. Our separation scheme makes use of the *complemented mixed integer rounding* approach (c-MIR) introduced by Marchand [39] and Marchand and Wolsey [40]. Instead of using the default aggregation heuristic we aggregate inequalities in such a way that the resulting base inequalities correspond to cuts of the detected network. In this context our approach can be considered as being an *alternative c-MIR aggregation heuristic* which exploits combinatorial structure. If the considered MIP instance contains a network structure, e.g., if it corresponds to a network design problem, our implementation is able to identify it and to produce strong valid special-purpose cuts which help to improve the dual bound and to accelerate the branch-and-cut solver. On the other hand, our implementation is able to decide whether the detected structure is consistent or not. In particular, we are not generating cutting planes if the structure is not consistent. This way we introduce almost no overhead for instances that do not fit into our framework, following a remark from Bixby and Rothberg [17]:

> It may also be tempting to consider *a new method* in the context of a *single problem class*. While an idea that provides a *big benefit for one problem class* can be quite useful, both for solving problems of that class and for developing insights into generalizations of such methods, one practical difficulty is that MIP practitioners are typically unaware that they are confronting a problem of that class. At a minimum, *a method should be able to recognize models to which it can be applied*, ideally introducing little or *no overhead when the model does not fit the mold*.

*This research has been supported by the DFG research Center Matheon

†IBM, achterberg@de.ibm.com

‡Zuse Institute Berlin (ZIB), Takustr. 7, D-14195 Berlin, raack@zib.de

Let $\mathcal{A} = (\alpha_{ij})_{i\in M, j\in N}$ be a rational matrix with m rows and n columns. We denote by M and N the row and column row indices of $\mathcal{A}$. The set of integer variables is given by $I \subseteq N$. We consider the mixed integer program (MIP)

$$\begin{aligned} \min\ & \kappa^T x \\ \mathcal{A}x\ & \leqq b \\ x_j & \in \mathbb{Z}, \quad \forall j \in I \end{aligned} \tag{1}$$

where the linear constraints of the system (1) are given either as equations or as $\leq$-inequalities. Upper and lower bounds on variables are already included in the constraint system. Associated with (1) we define $X := \{x \in \mathbb{R}^{N\setminus I} \times \mathbb{Z}^I \,:\, \mathcal{A}x \leqq b\}$ to be the mixed integer set containing all feasible solutions.

Based on the observation that many known strong valid inequalities for different problems can be obtained by MIR, Marchand and Wolsey [40] (also see [39, 41]) proposed a c-MIR procedure that is nowadays one of the most successful separation schemes in state-of-the-art MIP-solvers such as Scip and Cplex, see [1, 17, 59]. The idea is to generate MIR inequalities from single (base) constraints that are valid for X. Marchand and Wolsey [40] employ four different operations to obtain such a base inequality. In the first *aggregation* step a conic combination $u^T\mathcal{A}x \leq u^T b$ of the system (1) using weights $u \geq 0$ is considered. This is followed by *bound substitution* trying to substitute continuous variables by variable upper or lower bounds of the form $x_j \leq cx_{j'}$ or $cx_{j'} \leq x_j$ with $j \in N\setminus I, j' \in I$. In the third step a subset $C \subseteq I$ of bounded integer variables is *complemented* using simple upper or lower bounds of the form $x_j \leq u_j$ or $l_j \leq x_j$ with $j \in C$. Eventually, the resulting mixed integer knapsack inequality is *scaled* using an appropriate multiplier $\delta > 0$. Each of these four operations is carried out by heuristics using information from the (current) solution of the linear programming relaxation. For an introduction to MIR the reader is referred to [43, 57]. The c-MIR framework is introduced in [39, 41] while some new insights on c-MIR and flow-cover inequalities are given in [34]. The implementation of c-MIR in Scip together with computational tests is described in [1, 59]. A similar computational c-MIR study concerning the Cut Generation Library (CGL) of the COIN-OR-initiative [22] is provided by [26]. Results on the performance of c-MIR in Cplex can be found in [17].

Despite the fact that MIR inequalities based on different heuristics are generated by Scip and Cplex, computational results suggest that important cut-based MIR inequalities for certain network design problems are rarely found, see for instance [49]. This has mainly two reasons. First, the network structure is not known to these solvers. Secondly, the corresponding aggregation simply involves too many rows of the original system (1). It is natural to impose a conservative limit on the maximal number of inequalities considered for aggregation in general purpose MIR or Chvátal-Gomory based procedures since this limit appears in the exponent of the running time function. Moreover, it is very likely that (without additional information) the generated inequalities become very dense if too many inequalities are aggregated. For these reasons, the default c-MIR aggregation limit in Scip has been set to 7 inequalities. A similar value is used in Cplex.

Our implementation makes use of the c-MIR separation schemes implemented in Scip and Cplex based on [40]. We skip the default aggregation heuristic and instead construct the vector u using information from the network detection. The aggregation as described in the following sections can involve a huge number of flow conservation and capacity constraints. Already for a medium sized network with 50 nodes and 100 commodities we potentially aggregate more than 2500 constraints (assuming a cut with two equally sized shores). Nevertheless, since the support of the resulting base constraint corresponds to a cut of the detected network, the aggregated constraint tends to be very sparse. It is important to understand that our framework is not explicitly generating cutting planes. It only calculates weight vectors u. The remaining steps, in particular bound substitution and complementing, are carried out by the c-MIR functions of Scip, see [1, 59]. Notice that scaling the base constraint with different values $\delta > 0$ before MIR can be seen as using weight vectors δu^T for aggregation. Also certain bound substitutions can be done already by aggregation as explained in the next section.

This paper is organized as follows. Section 2 introduces the type of models and matrix structures which our detection algorithm tries to identify. We also introduce different strong cut-based inequalities, and we show that these can be obtained using the same aggregation and c-MIR procedure. In Section 3, the network detection algorithm is explained in detail. We evaluate the quality of the algorithm using a large set of publicly available network design instances. Section 4 describes our aggregation and separation scheme. In particular, we explain how to find promosing cuts in the detected network. Some important extensions and model variants are considered in Section 5. In Section 6, we report on our computational experiments with the Mcf separator of Scip and Cplex. The experiments are carried out with the mentioned network design instances and in addition using the Miplib 3.0 [15], Miplib 2003 [2], the MIP instances of Hans Mittelmann [42], and the Cplex-internal testset. We conclude with some remarks in Section 7.

2 Network Design

Combinatorial optimization problems arising for instance from applications in telecommunication and public transportion very often involve the problem of designing a network [20, 36, 47, 51]. This task can be roughly described as follows. Given a potential network topology (a graph), network links (connections, streets, bus-lines) and nodes (locations, intersections, stations) have to be dimensioned to allow for the *flow* of commodities (data, passengers, goods) corresponding to certain user demands. If different commodities have to be routed independently through the network, we speak of *multi-commodity* flows. Dimensioning in this context means to assign *capacity* to the network elements (links and nodes). In practice, the set of possible capacity assignments typically has a discrete structure. Link capacities in telecommunication applications, for instance, can be composed of integer multiples of a certain base bandwith. In public transportation they typically are chosen from a finite set of possible vehicle frequencies and types. But also the routing of demands might be discrete in the sense that it is restricted to single-path or integer flows.

In the mathematical literature there is a vast variety of approaches to model and solve network design problems depending on the requirements to incorporate. With our network detection and cutting plane approach we focus on rather general mixed integer programming models, so-called *arc-flow formulations*, which allow to dimension the links of a network such that a multi-commodity flow of given demands can be accommodated. (Notice that node dimensioning can always be broken down to link dimensioning by introducing artificial network links.) In the following we will introduce such models in more detail. In particular, we aim at working out the structure of the corresponding matrix which our network detection and separation algorithms rely on.

Moreover, we will introduce the concepts of network cuts and cut-based inequalities. It is well known that cutting planes defined on network cuts are among the most effective when used within branch-and-cut frameworks to solve network design problems, see [5, 13, 14, 24, 30, 38, 45, 49] for computational studies. Cut-based inequalities define facets of the corresponding polyhedra under very mild conditions and they are usually sparse. We will show how important classes of cut-based inequalities can be obtained with the c-MIR-approach using an appropriate aggregation of flow-conservation and capacity constraints. It will be emphasized that the same aggregation and MIR procedure can be used for many of the model variations used in practice.

2.1 Model and matrix structure

First, we consider a basic network flow model, which is described in the following. A discussion on more general model types can be found in Section 5. We are given a connected directed graph $G = (V, A)$ with nodes V and arcs A and a set of commodities K. With every $k \in K$ a vector $d^k \in \mathbb{Q}^V$ of demand (supply) values is associated. We call v a supply node with respect to commodity k if $d_v^k > 0$ and a demand node if $d_v^k < 0$. For every commodity $k \in K$ we have to construct a flow in G, which can be considered as being a vector $f^k \in \mathbb{R}_+^A$ with the property that for every node $v \in V$ the flow leaving v on all outgoing arcs minus the flow entering v on all incoming arcs equals the value d_v^k. We assume that $\sum_{v \in V} d_v^k = 0$ for all $k \in K$, i.e., there is no flow leaving the network or entering it from "outside". Very often a commodity corresponds to a single point-to-point demand, that is, there is exactly one (source) node $s \in V$ with $d_s^k > 0$ and one (target) node $t \in V$ with $d_t^k < 0$.

To accommodate the multi-commodity flow we have to dimension the network arcs. Every arc

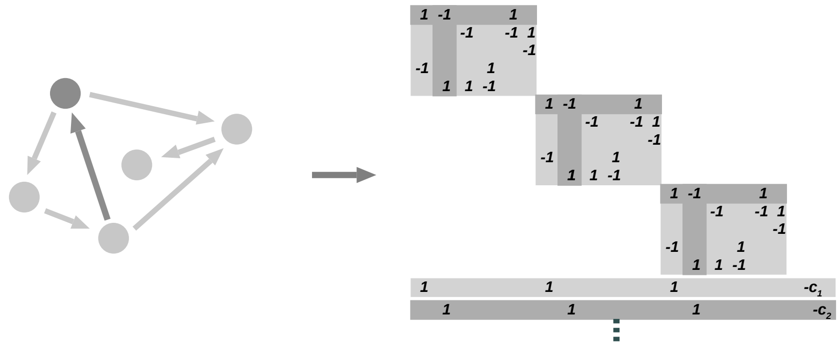


Figure 1: A digraph and the corresponding matrix representing a coupled multi-commodity flow. A node has a flow-row in every commodity. An arc has a column in every commodity and corresponds to one coupling capacity row.

$a \in A$ can be equipped with integer multiples of the capacity value $c_a \in \mathbb{Q}, c_a > 0$, while the number of capacity modules installable on a is bounded by $u_a \in \mathbb{Z} \cup \{\infty\}, u_a \geq 1$. A flow $f \in \mathbb{R}_+^{A \times K}$ is said to be feasible if for every arc a the total flow (over all commodities) is not exceeding the arc capacity. The capacitated network design problem now asks for a capacity assignment to the arcs plus a feasible network flow for all commodities that minimizes a given linear (flow and/or capacity) cost function [5, 13, 37].

Let f_a^k be the flow of commodity $k \in K$ on arc $a \in A$. Variables $y_a \in \mathbb{Z}_+$ state how often capacity c_a is provided on arc $a \in A$. A natural way to describe all feasible multi-commodity flows together with all feasible capacity assignments is to use an arc-flow formulation of the form:

$$\sum_{a \in \delta^+(v)} f_a^k - \sum_{a \in \delta^-(v)} f_a^k = d_v^k \quad \forall v \in V,\, k \in K \tag{2a}$$

$$\sum_{k \in K} f_a^k - c_a y_a \leq 0 \quad \forall a \in A \tag{2b}$$

$$y_a \leq u_a \quad \forall a \in A \tag{2c}$$

$$f, y \geq 0, \tag{2d}$$

where $\delta^+(v)$ and $\delta^-(v)$ denote all arcs in A having v as source and target node, respectively. The flow conservation equations (2a) describe the flow for each individual commodity. The capacity constraints (2b) ensure that the flows are feasible by providing sufficient capacity on the network arcs.

The constraint matrix corresponding to the system (2), as visualized in Figure 1, consists of $|K|$ blocks, which all correspond to the same $|V| \times |A|$ node-arc incidence matrix of the graph $G = (V, A)$. Such a matrix has the property to contain one $+1$ and one -1 entry in every column which correspond to the source and target node of the arc represented by the column. The $|K|$ blocks are coupled by the capacity constraints that, for each arc, sum up the flow-variables of all commodities and limit this total flow by the arc capacity.

2.2 Cuts and cut-based inequalities

Let S be a nonempty proper subset of the nodes V and let $\delta(S) := \delta^+(S) \cup \delta^-(S)$ be the corresponding dicut, where $\delta^+(S)$ denotes all arcs in a with source in S and target in $V \backslash S$ and $\delta^-(S)$ subsumes all arcs with target in S and source in $V \backslash S$. For the ease of exposition we stick to the single-commodity case here with flow-variables f_a for every $a \in A$ and node demands (supplies) d_v. The multi-commodity case is covered by considering single-commodity relaxations of (2) obtained by aggregating all flow-rows (2a) corresponding to a subset of the commodities $Q \subseteq K$ and setting $f_a := \sum_{k \in Q} f_a^k$ for every $a \in A$ as well as $d_v := \sum_{k \in Q} d_v^k$ for every $v \in V$, see for instance [5, 50].

To develop cut-based inequalities we regard the structure $G_S = (\{S, V \backslash S\}, \delta(S))$ as a two-node network and restrict ourselves to consider the flow across the cut (the flow between S and $V \backslash S$)

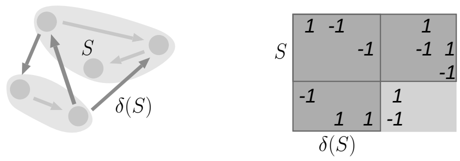


Figure 2: The cut $\delta(S)$ obtained by aggregating the flow-rows of a nodeset S

and the capacity provided on the dicut $\delta(S)$. We denote by $d_S := \sum_{v \in S} d_v =: -d_{V\setminus S}$ the total cut demand (supply) of the artificial node S and assume that $d_S < 0$, that is, S is a (single-commodity) demand node. Notice that in the multi-commodity case the sign of d_S depends on the choice of the subset Q considered for the single-commodity relaxation. Now it obviously holds that the cut demand is bounded by the cut capacity, that is, informally:

$$\text{capacity}(\delta^-(S)) \geq \text{demand}(V\setminus S \to S) \tag{3}$$

Similarly the supply of S (which is the demand of $V\setminus S$) cannot exceed the capacity on $\delta^+(S)$. Observation (3) is crucial both from the theoretical and practical point of view. In practice, if inequality (3) is tight the network cut $\delta(S)$ can be considered as being a bottleneck. The observation also has theoretical consequences, especially for network flow theory and the max-flow-min-cut theorem [3]. When solving network design problems using branch-and-cut frameworks, inequality (3) can be used to derive cutting planes, which is our main motivation here.

For the network design model (2) observation (3) breaks down to

$$\sum_{a \in \delta^-(S)} c_a y_a \geq d_{V\setminus S} \; . \tag{4}$$

We will now generalize this base inequality, and we will show how it can be obtained by aggregating original constraints of the system (2), see also Figure 2. First, summing up (and relaxing) all flow equations (2a) corresponding to S and restricting the capacity constraints (2b) to the dicut $\delta(S)$ results in the following (two-node, single-commodity) cutset relaxation of the formulation (2):

$$\sum_{a \in \delta^+(S)} f_a - \sum_{a \in \delta^-(S)} f_a \leq d_S \tag{5a}$$

$$f_a - c_a y_a \leq 0 \quad \forall a \in \delta(S) \tag{5b}$$

$$y_a \leq u_a \quad \forall a \in \delta(S) \tag{5c}$$

The key to derive strong valid cut-based inequalities for network design problems is to study the convex hull of the solution space defined by (5) and the integrality of y_a. This structure is known as a *single node flow set* and has been studied extensively in the literature [4, 7, 11, 28, 29, 41, 46, 52, 53, 55, 56] in particular for the case that y_a is a binary variable, i.e., $u_a = 1$ for all $a \in A$. Note that this has been done mainly for the fact that single node flow sets arise as natural relaxations of general MIPs and not because they correspond to network cuts. In contrast, related structures with unbounded integer variables have been studied in [5, 50] as *cutset polyhedra* motivated by network design.

We now add all capacity constraints (5b) corresponding to a subset A^- of the arcs $\delta^-(S)$ to inequality (5a) which gives the mixed integer knapsack base inequality

$$\sum_{a \in \delta^+(S)} f_a - \sum_{a \in \bar{A}^-} f_a - \sum_{a \in A^-} c_a y_a \leq d_S, \tag{6}$$

where $\bar{A}^- := \delta^-(S)\setminus A^-$. We observe that setting $A^- := \delta^-(S)$ and relaxing yields (4). The solution space corresponding to (6) is known as a *mixed (integer) knapsack set*, see [6, 41] and the references therein. By applying MIR to (6) we will recover some well-known strong valid inequalities for network design problems in the sequel. Since (6) is an aggregation of original constraints all presented MIR inequalities can, in principle, be obtained by using the c-MIR heuristic of Marchand and Wolsey [40, 41].

Let us first consider the unbounded case, that is, $u_a = \infty$ for all $a \in \delta(S)$. For simplicity we assume that the installable capacity c_a is independent from the arcs a, i.e., $c_a = c > 0$ for all $a \in A$. In this case, dividing (6) by c and applying MIR gives the well-known *flow-cutset inequalities* [5, 13, 21].

$$\sum_{a \in \bar{A}^-} f_a + \sum_{a \in A^-} r y_a \geq r \left\lceil \frac{d_{V\setminus S}}{c} \right\rceil, \tag{7}$$

where r denotes the remainder of the division of $d_{V\setminus S}$ by c. By setting $A^- := \delta^-(S)$ we obtain the *cutset inequalities*

$$\sum_{a \in \delta^-(S)} y_a \geq \left\lceil \frac{d_{V\setminus S}}{c} \right\rceil. \tag{8}$$

Inequality (8) is crucial since it provides a lower bound on the number of capacity modules that has to be provided on the cut to allow for feasible flows. Atamtürk [5] proves that flow-cutset inequalities together with all trivial inequalities yield a complete description of the cutset polyhedron for S. On the other hand, the cutset inequalities (8) turn out to be the most effective cuts in practice, see for instance [5, 14, 21] (directed models), [35, 37, 38, 49, 50] (undirected models), and [13, 30, 49, 50] (so-called bidirected models). If the capacities are not arc-independent (or similarly if there is more than one arc facility), the base inequality (4) can be divided by one of the given capacities c_a before applying MIR. In fact, the facet defining cutset inequalities and flow-cutset inequalities in [5, 13, 35, 38, 50] are of this type, see also [48].

Cutset inequalities and flow-cutset inequalities clearly remain valid if we impose upper bounds on the capacity variables. In many applications the capacity variables are binary, modeling the decision wether or not to install a certain arc facility. For uncapacitated network design problems with $c > d_{V\setminus S}$ for all nonempty $S \subset V$, the inequalities (8) and (7) reduce to the *Steiner-cut* (or dicut) and *mixed dicut inequalities*

$$\sum_{a \in \delta^-(S)} y_a \geq 1 \quad \text{and} \quad \sum_{a \in \bar{A}^-} f_a + \sum_{a \in A^-} d_{V\setminus S} y_a \geq d_{V\setminus S},$$

see for instance [45]. Notice that $\lceil \frac{d_{V\setminus S}}{c} \rceil = 1$ and $r = d_{V\setminus S}$ in this case.

In the bounded case ($u_a < \infty$ for all $a \in A$) with arc-dependent capacities (c_a for all $a \in A$) a large class of valid inequalities for (5), which incorporate the bounds on the capacity variables, is given by *flow-cover inequalities* [28, 46, 52]. Marchand and Wolsey [40, 41] and recently Louveaux and Wolsey [34] observed that strong valid lifted flow cover inequalities can be obtained by MIR. (Among others, this observation led to the development of the c-MIR framework.) Starting from the relaxation (6) they allow to *complement* a subset C of the capacity variables using the upper bounds u_a. The resulting base inequalities are divided by some constant $c > 0$ and MIR is applied. If C is a flow-cover and $c = c_a$ for an appropriate $a \in A$ this leads to flow-cover inequalities dominating those introduced for instance in [52]. In this context, strong flow cover inequalities (simultaneously lifted by using the superadditive MIR function) can be derived in the same way as flow-cutset and mixed dicut inequalities with the additional feature of complementing simple bounds. Notice that the more general c-MIR approach of Marchand and Wolsey [40] and its implementation in SCIP [1, 59] does not explicitly choose a (flow)-cover for complementing but uses a simple heuristic to determine C which is based on the solution of the LP-relaxation.

In the light of Marchand and Wolsey [40] and Louveaux and Wolsey [34], it is important to understand that using the capacity constraints (5b) of A^- in the aggregation for (6) is equivalent to a bound substitution setting $f_a := c_a y_a - \bar{f}_a$ for $a \in A^-$ with $\bar{f}_a \geq 0$ being the slack of the capacity constraint. The artificial variables $\bar{f}_a$ are deleted anyway by MIR (if they are considered as continuous), that is, resubstitution is not necessary. Note that the capacity constraints we use for aggregation may extend the concept of simple variable upper bounds $f_a \leq c_a y_a$. First, in the multi-commodity case the term f_a subsumes all flow-variables f_a^k for $k \in K$, and second, the arc capacity formulation might involve more variables than only a single arc facility y_a, see Section 5.

Hence the implicit bound substitution we carry out here is more general than the one proposed in [34, 40]. Flow-cutset inequalities as well as flow cover inequalities can contain both outflow and inflow-variables [5, 52]. In this case one explicitly has to substitute the variable upper bounds by introducing the slacks of the capacity constraints, apply MIR, and resubstitute the original variables. The inequalities considered by our separation scheme contain only inflow-variables (or by switching to $V \backslash S$ only outflow-variables). This allows us to include the bound substitution in the aggregation step (although SCIP might do additional bound substitution using simple variable upper bounds in its default heuristic).

2.3 Summary

In our implementation we generate cut-based inequalities using the c-MIR approach based on arc-flow formulations hidden in the constraint system $\mathcal{A}x \leqq b$ of the given MIP. We first identify a subsystem of the from (2) and construct the corresponding network as described in Section 3, that is, we resolve the structure given in Figure 1 backwards. In the detected network we identify interesting cuts, see Section 4. For every such cut we call the c-MIR procedure of SCIP with a weight vector u that results in a base inequality of the form (6). Given a cut and the corresponding nodeset S, the corresponding aggregation can be summarized as follows:

- Aggregate all flow-rows (2a) for nodes in S and commodities in a subset Q of K.
- Add all capacity constraints (2b) corresponding to a subset A^- of the arcs $\delta^-(S)$.
- For every capacity c, that is, for every coefficient c of an integral variable in one of the used capacity constraints, use $\delta = 1/|c|$ as a multiplier to scale the base inequality.

MIR is applied to all these scaled base inequalities. In the first step we restrict our attention to the commodity subset consisting of all demand commodities with respect to S, that is, $Q = K_S^- := \{k \in K : d_S^k < 0\}$, where $d_S^k := \sum_{v \in S} d_v^k$. In the second step we consider the subset $A^- = \delta^-(S)$ and additionally the subset A^- that gives the most violated inequality among all possible subsets A^-. The reverse direction with $Q = K_S^+ := \{k \in K : d_S^k > 0\}$ and subsets of $\delta^+(S)$ is considered by repeating the same procedure for $V \backslash S$. The separation scheme is described in more detail in Section 4. Clearly, the corresponding aggregation involves a large number of the original constraints (2a) and (2b) already for small sized networks.

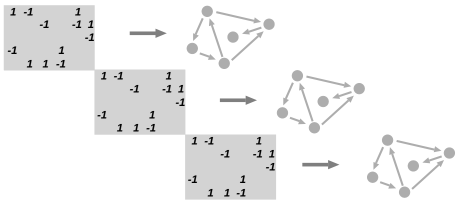
Allthough the network design model (2) and the corresponding c-MIR aggregation are already rather general, we have to consider model variations which are frequently used in practice. These are in particular *multi-facility* problems, *undirected* capacity models, and *single-path-flow* formulations. In Section 5 we explain how these extensions and variants are incorporated into our framework.

3 Network Detection

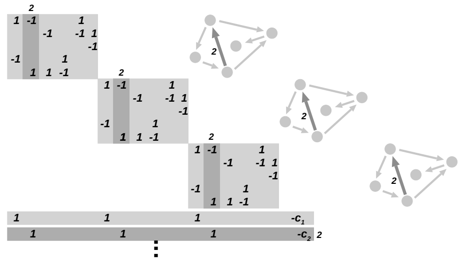
3.1 Introduction

We start with a high-level presentation of our network detection algorithm. Thereafter we will explain the corresponding sub-procedures in more detail. Notice that we have not implemented our algorithms in the way we present them here. Our aim is to describe the core idea of our implementation. To obtain a fast and stable algorithm one has to introduce more involved data structures. We will point out necessary improvements and implementational issues in the detailed description of the four sub-procedures **Flow Detection**, **Arc Detection**, **Node Detection**, and **Network Construction** whenever possible.

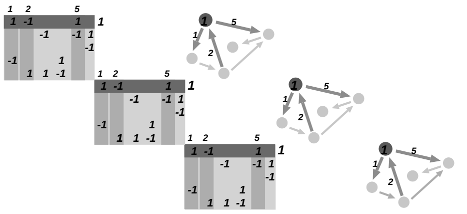
We will start by explaining the main ingredients of the detection. The idea is to start with flow conservation constraints or *flow-rows* of the form (2a). The flow structure of the network is characterized by a $\{0, +1, -1\}$-matrix such that each column has at most one $+1$ and at most one -1 entry as it can be seen in Figure 1. A subset of the rows of $\mathcal{A}$ will be called an *embedded network* if it has this property, up to scaling of individual rows. Our **Flow Detection** procedure is based on a *Row Scanning Addition Algorithm* introduced by Bixby and Fourer [16], see also Brown and Wright



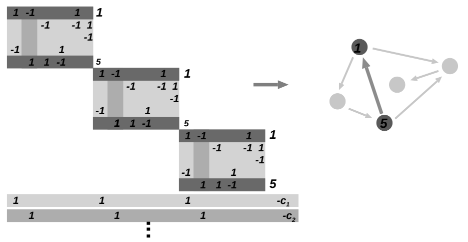
(a) Flow Detection resulting in a disconnected graph with one component per commodity



(b) Arc Detection: using capacity constraints to assign arc-ids



(c) Node Detection: Compare arc-id patterns in the different commodities and assign the same node-id to (almost) identical patterns



(d) Network Construction: Ask flow-variables for source and target node, construct incidence function according to majority vote, minority votes are inconsistencies

Figure 3: The network detection algorithm

[19]. It identifies an embedded network by consecutively adding flow-rows to the system starting with an empty set of rows. Each flow-row in the matrix represents one node in the flow network, a $+1$ coefficient corresponds to an outgoing flow, a -1 coefficient corresponds to an incoming flow. An equation can be multiplied by -1 in order to fit to the flow structure. In addition, if the current row is an inequality but all previous rows are equations, we can also multiply all previous rows by -1 to make the current row fit. This operation is called *reflection* [16, 19].

The flow structure could result in a flow network with multiple independent components, see Figure 3(a). In the perfect case, these components are isomorphic and represent the different commodities of the problem. Now the task is to find these isomorphisms. Notice that the problem of deciding whether two graphs are isomorphic has not yet been proven to belong to $\mathcal{P}$ or to be $\mathcal{NP}$-complete [25]. Since in practice the different components are usually not identical due to user and solver preprocessing as explained below, it is more important in our context to decide wether one graph is contained in another or alternatively to maximize the largest common subgraph. Both problems are $\mathcal{NP}$-complete ($\mathcal{NP}$-hard) [25].

The main idea to solve the graph isomorphism problems in the network detection is to find capacity coupling constraints of the form (2b) defined on the arcs of the network. We identify the capacity constraints and the corresponding arcs in the **Arc Detection** procedure. In the perfect (directed) case, a capacity constraint contains one flow-variable of each commodity and one or more capacity-variables. The structure of the capacity constraints, however, depends on the formulation, see Section 5. The Arc Detection procedure assigns arcs to the coupling capacity constraints and all corresponding flow-variables, see Figure 3(b).

To determine the nodes of the graph G we compare the arc-patterns of the flow-rows in the different commodities in the **Node Detection** procedure. The arc-pattern of a flow-row is given by the arc-ids of the involved flow-variables. If two flow-rows of two different commodities have a similar arc-pattern we decide to map them to the same node, see Figure 3(c). Eventually, we determine the source and target incidence functions for the network arcs in the **Network Construction** procedure. In a perfect network, the flow-variables of an arc (of a capacity constraint) should point to the same source and target node in the different commodities. Then, the source and target node assignment means to just use these two uniquely determined nodes, see Figure 3(d). In reality, however, flow-variables of the same arc might have different source or target nodes in the different commodities, that is, the detected network matrices are not isomorphic or arcs and nodes have been assigned incorrectly. For every arc $a \in A$ we use the majority vote of the flow-variables across the commodities and assign source and target accordingly. Additionally, we record the minority votes as inconsistencies in the network data structure. The number of inconsistencies divided by the number of commodities gives the arc inconsistency ratio $\Psi(a) \in [0, 1)$, which is used to discard individual arcs. The average inconsistency ratio over all network arcs is called the network inconsistency $\Psi(G) \in [0, 1)$, which is used to decide whether or not our network detection was successful and whether the separation scheme should be applied.

3.2 Inconsistency and presolving

If $\Psi(G) = 0$ we detected a consistent coupled multi-commodity flow network. The commodity network matrices can be considered being isomorphic and we correctly assigned arcs and nodes to rows and columns. If, however, $\Psi(G)$ is close to 1 our detection failed or there is no consistent embedded network in the constraint matrix. In our implementation we fixed the maximum inconsistency ratio $\Psi^{\max}$ to 0.02. If $\Psi(G) > \Psi^{\max}$, then all network data structures are released and it is not tried to generate cutting planes. In addition we do not allow for arcs with individual inconsistency ratio $\Psi(a)$ greater than $\Psi_a^{\max} = 0.5$. The influence of the inconsistency parameters $\Psi^{\max}$ and $\Psi_a^{\max}$ is tested in Section 6.1.

There are several reasons for potential inconsistencies. First, our detection is a heuristic. Its success largely depends on a proper identification and ordering of flow and capacity-rows, see the detailed description below. But already the formulation of the concrete MIP instance can be "corrupted" even if it corresponds to a coupled multi-commodity flow. As a consequence, the detection procedures cannot expect pure and isomorphic network matrices. The same node or the same arc do not need to be present in every commodity and an arc does not necessarily have both a source and a target node.

User presolving It is known that the rank of a network matrix corresponding to a directed network $G = (V, A)$ is exactly $|V| - 1$. For every commodity, an arbitrary row in (2a) can be omitted. To save constraints, this preprocessing is sometimes already carried out by the modeler and results in deleting a node from G for every commodity, see Figure 4. Moreover, the node that is deleted typically differs from commodity to commodity. For example, if each commodity has a single source node, it is common to omit the flow conservation constraint of this source node from the formulation.

Another common presolving technique is to discard all flow-variables that correspond to arcs pointing into source-nodes or pointing away from target-nodes. This is done to avoid cycle-flows in the solutions. Deleting flow-variables corresponds to deleting arcs in the network matrix. Again the omitted arcs differ from commodity to commodity. It turns out that, in practice, our detection has to face multi-commodity formulations with blocks for individual commodities that are not isomorphic, although they originally correspond to the same network. However, our detection procedures is still correctly identifying most of the underlying graphs even if the formulations have passed these user presolving techniques, see Table 2.

Solver presolving In order to decrease the size of the formulation, state-of-the-art MIP-solvers carry out a series of preprocessing steps before starting the actual branch-and-cut procedure. The

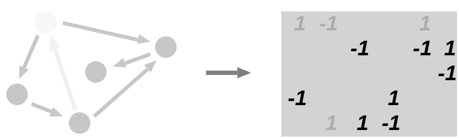


Figure 4: A node and all incoming arcs deleted by user presolving. Notice that two of the remaining arcs have no source.

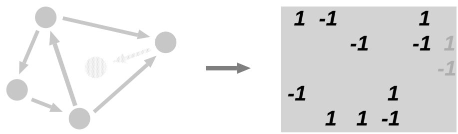


Figure 5: Loosely connected nodes and arcs deleted by solver presolving.

model is transformed by deleting redundant constraints and by fixing, substituting, and deleting variables. We refer to [1] for a description of the presolving methods used in Scip. We observed that by preprocessing, in particular loosely connected nodes and arcs are deleted from the original graph, see Figure 5. If for instance node v has only one outgoing arc a and no incoming arc, the resulting flow-row has the form $f_a = d_v$. Hence f_a can be fixed and removed from the system. If, alternatively, v has only one outgoing arc a and only one incoming arc a', one of the corresponding flow-variables can be substituted by the other since $f_a - f_{a'} = d_v$.

As shown in Table 2, the number of nodes and arcs deleted by preprocessing may amount to more than 20% even for pure network design instances of type (2). But also if the network size is strongly reduced (as for the instance sets *avub, arc.set, fc*) the inconsistency ratio $\Psi(G)$ is not necessarily increasing. Also our separator performs very well, compare with Table 3. The remaining graphs after presolving seem to reflect the core of the network such that generated cut-inequalities still capture important structural information.

3.3 Notation

Before explaining the four sub-procedures of our network detection strategy in more detail, we introduce some useful notation. Our detection algorithm identifies potential flow-row and capacity-row candidates, which are subsets $M_F \subseteq M$ and $M_C \subseteq M$ of the rows of $\mathcal{A}$. During the course of the algorithm, these sets are reduced in order to obtain two disjoint subsets, which correspond to the nodes and arcs in the final multi-commodity flow network structure. Constructing the network means to map the rows and columns of the matrix $\mathcal{A}$ to network elements and commodities. The mappings are

$$\begin{aligned} rowcom &: \quad M \to K \cup \{0\}, \quad i \mapsto rowcom(i) \\ colcom &: \quad N \to K \cup \{0\}, \quad j \mapsto colcom(j) \\ rowarc &: \quad M \to A \cup \{0\}, \quad i \mapsto rowarc(i) \\ colarc &: \quad N \to A \cup \{0\}, \quad j \mapsto colarc(j) \\ rownode &: \quad M \to V \cup \{0\}, \quad i \mapsto rownode(i), \end{aligned}$$

where *rowcom* and *colcom* map rows and columns of the flow-system to commodities, the functions *rowarc* and *colarc* map the coupling (capacity) constraints and the flow-variables to arcs of the network, and *rownode* assigns a node to every flow-row. A mapping to 0 means that the corresponding row or column has not been assigned. To construct the graph $G = (V, A)$ we use the source and target incidence functions

$$\begin{aligned} s &: \quad A \to V, \quad a \mapsto s(a) \\ t &: \quad A \to V, \quad a \mapsto t(a) \end{aligned}$$

All of our algorithms, based on data structures provided by Scip, rely on sparse array representations of the rows and columns of the matrix $\mathcal{A}$. Whenever iterating rows or columns, we in fact iterate all corresponding non-zeroes. For a subset $N' \subseteq N$ of the column indices, the set $M[N'] := \{i \in M : \exists j \in N' \text{ with } \alpha_{ij} \neq 0\}$ contains all row indices with a non-zero entry in one of the columns of N'. Similarly, for a row index set $M' \subseteq M$, the set $N[M'] := \{j \in N : \exists i \in M' \text{ with } \alpha_{ij} \neq 0\}$ corresponds to all columns with a non-zero entry in one of the rows of M'. We abbreviate $M[j] := M[\{j\}]$ and call $M[j]$ the support of column j. Similarly, $N[i] := N[\{i\}]$ denotes the support of row i.

3.4 Flow Detection

The goal of the flow detection Algorithm 1 is to find an embedded network in $\mathcal{A}$ that is inclusion-wise maximal with respect to the rows. For finding the embedded network we use a modified row-scanning-addition algorithm [16]. Roughly speaking, this algorithm starts with an empty set of flow-rows and adds rows until a maximal embedded network has been built.

Prior to calling Algorithm 1 we identify a potential set of flow-row candidates M_F among all rows M. Initially, the set M_F contains all rows in $\mathcal{A}$ that have, up to scaling, entries in the set $\{0, +1, -1\}$. Note that in contrast to Bixby and Fourer [16] we do not allow for scaling of columns in order to obtain a $\{0, -1, 1\}$ system. All nonzero coefficients of a row in M_F have the same absolute value. We do not explicitly scale rows but keep track of the scaling factors. The actual scaling is carried out by the weighted aggregation in the c-MIR procedure. Since in practice the degree of network nodes is relatively small and for efficiency reasons we do not allow for flow-rows with more than 10% non-zeroes, that is, we limit the node degree to $0.1|A|$ (in the single-commodity case).

Our flow detection algorithm is strongly driven by a scoring of the flow-row candidates. To every row i in M_F we assign a score $s_F(i) \in \mathbb{R}_+$. The larger $s_F(i)$, the more we trust row i to be part of a flow-system. The following properties of row i (in decreasing order of their importance) increase its score $s_F(i)$ in our implementation:

- Row i does not need to be scaled, i.e., its coefficients are among $\{0, -1, +1\}$.
- All variables (with non-zero coefficient) are continuous (corresponding to splittable flows).
- All variables in row i (with non-zero coefficient) are integer or all variables in row i (with non-zero coefficient) are binary (corresponding to integer splittable or single-path flows).
- Row i has both positive and negative coefficients. (There are both inflow and outflow variables)
- Row i is an equation.

We use the number of non-zeroes and the absolute dual value of row i in the initial solution of the LP relaxation for tie-breaking. The larger these values are the earlier the flow-row candidate is considered: scanning and evaluation of the flow-row candidates in Steps 4 and 10 of Algorithm 1 are carried out in non-increasing order of s_F.

The submatrix defined by the flow-rows might contain independent blocks, i.e., the corresponding network is not necessarily connected. These different blocks, that is, the corresponding rows and columns, are assigned to different commodities. In contrast to the row-scanning-addition algorithm of Bixby and Fourer [16] our procedure constructs the flow-system of every commodity one by one (Steps 5-13). We denote by $M_F(k)$ all flow-rows that are assigned to commodity k, $M_F(k) := \{i \in M_F : \mathit{rowcom}(i) = k\}$. Similarly, the set $N_F(k) := N[M_F(k)]$ contains all flow-variables assigned to commodity k. If $M_F(k)$ cannot be increased, a new commodity is created until all potential flow-row candidates have been considered. In Step 14 of Algorithm 1 we say that a finished commodity k is too small if $|M_F(k)| < 3$ or there exists a commodity k' such that $|M_F(k)| < 0.5|M_F(k')|$. In this case the commodity mappings for the corresponding rows and columns are released. These rows can then indeed be used again for new commodities. But notice that every row is considered at most once as the starting row of a commodity in Step 4, which guarantees the termination of the algorithm.

Every step of the addition method results in a feasible flow-system (an embedded and connected sub-network) for the current commodity k. Given a flow-row candidate $i \in M_F \setminus M_F(k)$, we say that i *fits* to $M_F(k)$, if

- i is adjacent to $M_F(k)$, i.e., the intersection of $N_F(k)$ and $N[i]$ is non-empty,
- the augmented system $M_F(k) \cup \{i\}$ is an embedded network, i.e., it has at most one $+1$ and at most one -1 entry in every column (up to scaling and reflection).

In Steps 9-13 we scan all columns of $M_F(k)$ for adjacent flow-rows. For efficiency reasons we take the first row that fits. But note that this row has the largest score w.r.t. the current column. To accelerate the loop 9-13 we consider only those columns j in Step 9 that have exactly one entry in

$M_F(k)$. This is achieved by introducing arrays that count the number of $+1$ and -1 entries in the current commodity for every column. In our implementation these arrays are also used for testing if a row fits to $M_F(k)$, also see [16].

Algorithm 1: Flow Detection

Input : flow-row candidates M_F, scoring $s_F : M_F \to \mathbb{R}_+$
Output : set of commodities K, mappings $rowcom : M \to K \cup \{0\}$, $colcom : N \to K \cup \{0\}$
1 Sort M_F in non-increasing order of s_F
2 Initialize $rowcom(i) := 0$ for all $i \in M$
3 Initialize $k := 0$
4 **for** $i \in M_F$ *with* $rowcom(i) = 0$ **do** `// Scan flow-row candidates`
5     $k := k + 1$ `// Create new commodity`
6     $i' := i$
7     $rowcom(i') := k$ `// Add row` i' `to commodity` k
8     $colcom(j) := k$ for all $j \in N[i']$ `// Add non-zero columns of` i' `to commodity` k
9   **for** j *with* $colcom(j) = k$ **do** `// Search for adjacent rows`
10   **for** $i' \in M[j]$ *with* $rowcom(i') = 0$ **do**
11   **if** *row* i' *fits to system* $M_F(k)$ **then goto** 7; `//` i' `is best row of` j
12   **end**
13   **end**
14   **if** *flow-system* $M_F(k)$ *is too small* **then** `// Delete commodity` k
15         $colcom(j) := 0$ for all $j \in N_F(k)$
16         $rowcom(i') := 0$ for all $i' \in M_F(k)$
17         $k := k - 1$
18   **end**
19 **end**
20 $M_F := M_F \setminus \{i \in M_F \; : \; rowcom(i) = 0\}$ `// Remove nonassigned candidates`
21 $N_F := N[M_F]$

To fit a row into a flow-system one can reflect it, i.e., multiply it by -1. Since our separation approach (see Section 4) relies on aggregating flow-rows, this operation can be applied as long as the current row i is an equation (or similarly, every row of the current system $M_F(k)$ is an equation). In case there is a $\leq$-inequality among $M_F(k)$ and a $\leq$-constraint has to be reflected in Step 11 to make it fit to $M_F(k)$ we decrease its score such that it is considered later in the loop 10-12. This way we avoid to introduce slacks when aggregating subsets of the rows of (1) in the c-MIR procedure by summing up $\leq$ and $\geq$-constraints.

After calculating a maximal embedded network within M_F all rows that do not participate in the flow-system are removed from M_F (Step 21 of Algorithm 1).

3.5 Arc Detection

The goal of the arc detection procedure given by Algorithm 2 is to identify the coupling of the commodities K and to assign arc-ids to the (coupling) capacity constraints as well as to all involved flow-variables. The set M_C of capacity-row candidates initially contains all rows in M that are not flow-rows and that contain at least one flow-variable. Hence:

$$M_C := \{i \in M \setminus M_F \; : \; N[i] \cap N_F \neq \emptyset\}$$

These candidates are sorted in non-increasing order of a score $s_C : M_C \to \mathbb{R}_+$ similar to the flow-row candidates in Algorithm 1. The most important property of a capacity-row candidate in this context is to contain a flow-variable for every commodity, i.e., to couple the flow-systems $M_F(k)$, $k \in K$. Basically, the score $s_C(i)$ is largest if the constraint $i \in M_C$ is of the form (2b). Properties that influence the score of capacity-row candidates are given in the following in decreasing order of their importance. Note that capacity-row candidates given as equations can always be reflected. For simplicity we assume that they are given as $\leq$-inequalities in the following. We increase $s_C(i)$

- for every covered commodity, i.e., for every $k \in K$ such that $N[i] \cap N_F(k) \neq \emptyset$,
- if i contains (close to) one flow-variable per commodity, i.e., if the number of flowvariables $|N[i] \cap N_F|$ divided by the number of covered commodities $|\{k \in K \,:\, N[i] \cap N_F(k) \neq \emptyset\}|$ is close to 1,
- if i is a (capacity) constraint bounding flow from above, i.e., it holds that $\alpha_{ij} > 0$ for all $j \in N[i] \cap M_F$ and $\alpha_{ij} < 0$ for all $j \in N[i] \setminus M_F$,
- if $\alpha_{ij} = 1$ for all $j \in N[i] \cap M_F$ without scaling, or
- if $\alpha_{ij} = 1$ for all $j \in N[i] \cap M_F$ by scaling.

We use the absolute dual values of the capacity-row candidates for tie-breaking. As for flow-rows we keep track of the necessary scaling factors. These will be used as weights in the c-MIR aggregation, see Section 4.

Algorithm 2: Arc Detection

Input : capacity-row candidates M_C, scoring $s_C : M_C \to \mathbb{R}_+$, mappings $rowcom : M \to K \cup \{0\}$ and $colcom : N \to K \cup \{0\}$

Output : mappings $rowarc : M \to A \cup \{0\}$ and $colarc : N \to A \cup \{0\}$

1 Sort M_C in non-increasing order of s_C

2 Initialize $colarc(j) := 0$ for all $j \in N$, $rowarc(i) := 0$ for all $i \in M$

3 Initialize $a := 0$

4 **for** $i \in M_C$ **do** `// Scan capacity-row candidates`

5     $flowvars := |N[i] \cap N_F|$

6     $unassigned := |\{j \in N[i] \cap N_F \,:\, colarc(j) = 0\}|$ `// Count unassigned flow-variables`

7   **if** $unassigned > flowvars/3$ **then** `// 1/3 of the flow-variables unassigned`

8         $a := a + 1$ `// Create new arc`

9         $rowarc(i) := a$

10         $colarc(j) := a$ for all $j \in N[i] \cap N_F$ with $colarc(j) = 0$

11   **end**

12 **end**

13 $M_C := M_C \setminus \{i \in M_C \,:\, rowarc(i) = 0\}$ `// Remove nonassigned candidates`

Algorithm 2 simply assigns an arc-id to every capacity-row candidate and all unassigned flow-variables in the support of the capacity-row candidate if one third of the flow-variables is still unassigned. Note that in a perfect network all capacity constraints are disjoint w.r.t. the flow-variables hence the flow-variables are all unassigned in Step 7. Here we allow for some overlap between capacity constraints, for example to cope with presolving reduction. The loop 4 is carried out in non-increasing order of s_C. Eventually, all capacity-rows without an arc-id are removed from M_C. Algorithm 2 terminates with a bijection $rowarc : M_C \leftrightarrow A$ of capacity-rows to arc-ids and vice versa. Flow-variables $j \in N_F$ with $colarc(j) = 0$ are considered to be uncapacitated since they do not have a supporting capacity constraint. For these variables we will create (uncapacitated) arcs in a final step after the construction of the network, see below.

3.6 Node Detection

Algorithm 3 uses the incidence information given by the arc-id mapping $colarc$ to identify (almost) isomorphic nodes in the different commodities. Two flow-rows in different commodities are considered to belong to the same node if they have a similar incidence pattern w.r.t. to their arc-ids.

Algorithm 3 scans all flow-rows in non-increasing order of s_F. In the single-commodity case every flow-row simply gets a different node-id. Given a flow-row $i \in M_F$ belonging to commodity k in the case $|K| > 1$, we try to identify flow-rows in all commodities $k' \neq k$ with a similar arc-pattern. To calculate the arc-pattern of a flow-row i we count for every arc a, how often it appears as an outgoing and incoming arc in the support of the constraint, i.e., how many flow-variables with positive and negative coefficients in the support of i are assigned to arc a.

Algorithm 3: Node Detection

Input : flow-row candidates M_F, mappings $rowcom : M \to K \cup \{0\}$ and $colarc : N \to A \cup \{0\}$
Output : mapping $rownode : M \to V \cup \{0\}$

1 Initialize $rownode(i) := 0$ for all $i \in M$
2 Initialize $v := 0$
3 **for** $i \in M_F$ *with* $rownode(i) = 0$ **do** `// Scan flow-rows`
4     $v := v + 1$ `// Create new node`
5     $rownode(i) := v$ `// Assign node` v `to flow-row` i
6   **if** $|K| = 1$ **then continue**
7     $k = rowcom(i)$
8     $pattern :=$ `PatternOf(`i`)`
9   **for** $i' \in M_F$ *with* $rowcom(i') \neq k$ **do** `// Scan flow-rows of commodities` $k' \neq k$
10         $k' := rowcom(i')$
11         $score :=$ `ComparePattern(`$pattern,$ `PatternOf(`i'`))`
12   Remember $bestrow(k')$ with largest $score$ for k'
13   **end**
14   **for** $k' \in K \setminus \{k\}$ **do** `// Assign node` v `to rows with closest arc-pattern to` i
15         $rownode(bestrow(k')) := v$
16   **end**
17 **end**

If the problem formulation is of the ideal form (2) and if we managed to detect the flow-system and arcs correctly in Algorithm 1 and 2, then `PatternOf`(i) returns an incidence vector in $\{0, +1, -1\}^A$ giving all outgoing and incoming arcs of the flow-row i. Due to inconsistencies in the system or matrix preprocessing the entries might differ from 0, +1, or −1.

Function `PatternOf(`i`)`

Input : flow-row i
Output : $pattern \in \mathbb{Z}^A$

$pattern(a) = 0$ for all $a \in A$ `// Initialize arc-pattern of row` i
for $j \in N[i]$ **do**
    $a = colarc(j)$
    if $a = 0$ **then continue** `// Uncapacitated flow-variable`
    if $\alpha_{ij} > 0$ **then** $pattern(a) = pattern(a) + 1$ `// Outgoing arc`
    if $\alpha_{ij} < 0$ **then** $pattern(a) = pattern(a) - 1$ `// Incoming arc`
end
return $pattern$

In Step 11 of Algorithm 3 we compare all arc-patterns of flow-rows i' of commodities $k' \neq k$ with the pattern of row i of commodity k using the function `ComparePattern`. Notice that it suffices to scan only those flow-rows i' in the loop 9-13 that are coupled with i by a capacity constraint. Hence we consider only rows $i' \in M_F$ that have at least one arc in common with i. The function `ComparePattern` returns the (weighted) overlap of the two arc-patterns. As a tie-breaker we use the number of non-overlapping entries of the two pattern vectors divided by the number of columns of the matrix $\mathcal{A}$. As already mentioned, there might be uncapacitated flow-variables that have no arc-id. In our implementation we also count the number of these uncapacitated flow-variables (both with positive and negative sign) in the two flow-rows and use this information as an additional tie-breaker when comparing two patterns. Hence flow-rows should have a similar number of uncapacitated flow-variables in addition to a similar arc-pattern to receive a large score.

Function `ComparePattern`($pattern1$, $pattern2$)

Input : arc-patterns $pattern1, pattern2$
Output : $score \in \mathbb{R}_+$

Initialize $score := 0$
for $a \in A$ **do**
  if $pattern1(a) \cdot pattern2(a) > 0$ **then** `// Patterns overlap and signs match`
    $score := score + \min(\frac{pattern1(a)}{pattern2(a)}, \frac{pattern2(a)}{pattern1(a)})$ `// Increase weighted overlap`
  end
  Calculate tiebreaker $0 \leq \epsilon \ll 1$
  $score := score - \epsilon$
end
return $score$

It should be mentioned that every individual commodity flow-system $M_F(k)$ can be reflected once, which means to reflect every flow-row in the system. This has to be considered when comparing the arc-patterns.

Subsequent to the the node-detection procedure we perform a cleanup of the network information obtained so far. We remove commodities that have no arcs (no flow-variable with an arc-id) or too few nodes (too few flow-rows assigned to different nodes). A commodity has too few nodes if its total number is smaller than 3 or it has less then 50% of the nodes of the largest commodity. In general one wishes to have commodity systems of almost the same size. Removing a commodity means to release the corresponding data structures and assignments to nodes and arcs.

3.7 Network Construction

For constructing a digraph G based on the nodeset V and arcset A it remains to construct the source and target incidence functions $s : A \to V$ and $t : A \to V$. The corresponding information is hidden in our data structures. Given a flow-variable $j \in N_F$ assigned to some arc $a \in A$, there are at most two flow-rows in M_F having j in their support, one with positive and one with negative coefficient. These flow-rows are assigned to nodes. It follows that every flow-variable, if assigned to an arc, has a source and a target node.

Algorithm 4 iterates all arcs in A and asks all the corresponding flow-variables for their source and target node. Due to inconsistencies in the formulation or in the network detection the flow-variables of the same arc might answer differently. Based on the majority of the votes the incidence function is constructed. For every arc $a \in A$ we evaluate its inconsistency $\Psi(a) \in [0, 1)$ in Step 24, where $\Psi(a)$ corresponds to the number of minority votes divided by the number of involved commodities. Inconsistent arcs, that is, arcs a with $\Psi(a) > \Psi_a^{\max}$, are deleted. The mean of the arc inconsistencies defines the network inconsistency ratio $\Psi(G) \in [0, 1)$. This ratio is used to decide about the quality of the detected network structure. If the inconsistency ratio is too large, that is $\Psi(G) > \Psi^{\max}$, all data structures are released and we do not try to generate inequalities based on the detected network, see Algorithm 5. The influence of the parameters $\Psi^{\max}$ and $\Psi_a^{\max}$ is tested in Section 6.1.

It remains to answer the question what happens with uncapacitated flow-variables that could not be assigned to arcs in the arc detection Algorithm 2. For these variables we try to create uncapacitated arcs in a procedure following the network construction. We create a new uncapacitated arc (s, t) for $s, t \in V$ if there are enough uncapacitated flow-variables in different commodities having s as source and t as target node. More precisely, if for the number $uncap_{(s,t)}$ of uncapacitated flow-variables corresponding to (s, t) it holds that $uncap_{(s,t)} \geq \lceil 0.8|K| \rceil$ we create a new arc $a = (s, t)$. Notice that for (s, t) there can only be one matching flow-variable for every commodity. Also notice that for the single-commodity case this means that we create a new arc for each uncapacitated flow-variable in the flow-system.

The constructed graph $G = (V, A)$ might be disconnected for two reasons. First, the arc-capacity constraints do not necessarily couple all commodity flowsystems but only subsets of them. Secondly, the network might get disconnected by deleting inconsistent arcs. Our separation procedure is applied to every individual component of G. Each of these components might correspond to a multi-commodity system. For simplicity, in the rest of this paper we assume that there is only one such component in the sequel, i.e., G is connected.

Algorithm 4: Network Construction

Input : nodes V, arcs A, mappings $rowarc : M \to A \cup \{0\}$ and $colcom : N \to K \cup \{0\}$
Output : digraph $G = (V, A)$ with incidence functions $s : A \to V$ and $t : A \to V$, network inconsistency $\Psi(G) \in [0, 1)$

Initialize $inconsistencies := 0$
for $a \in A$ **do**
    $i := rowarc^{-1}(a)$ `// Capacity row of arc` a
    for $k \in K$ **do**
        $nvars(k) := |\{j \in N[i] : colcom(j) = k\}|$ `// # flow-variables per commodity`
    end
    $ncom := |\{k \in K : nvars(k) > 0\}|$ `// # commodities in row` i
    `// Ask all flow-variables for source and target node`
    Initialize $scount(v) := 0$, $tcount(v) := 0$ for all $v \in V$
    for $j \in N[i]$ *with* $colcom(j) > 0$ **do**
        $k := colcom(j)$
        for $i' \in M[j] \cap M_F$ **do** `//` j `has at most two incident flow-rows`
            $v := rownode(i')$
            if $\alpha_{i'j} > 0$ **then** `// Increase source count for` v
                $scount(v) := scount(v) + 1/nvars(k)$
            else `// Increase target count for` v
                $tcount(v) := tcount(v) + 1/nvars(k)$
        end
    end
    `// Majority vote wins`
    $s(a) := \text{argmax}\{scount(v) : v \in V, scount(v) \geq tcount(v)\}$ `// Assign best source to` a
    $t(a) := \text{argmax}\{tcount(v) : v \in V, tcount(v) \geq scount(v)\}$ `// Assign best target to` a
    `// Minority votes give arc inconsistency`
    $totalcount := \sum_{v \in V}(scount(v) + tcount(v))$
    $\Psi(a) := (totalcount - scount(s(a)) - tcount(t(a)))/ncom$ `// Arc inconsistency`
    $\Psi(G) := \Psi(G) + \Psi(a)/|A|$ `// Network inconsistency`
end
for $a \in A$ **do**
    if $\Psi(a) > \Psi_a^{\max}$ **then** $A := A \setminus a$; `// Delete inconsistent arcs`
end

3.8 Detection – Results

In the following we discuss the success of our detection strategy. For our tests we selected publicly available network design instances (with formulations similar to type (2)) as well as general MIP instances. Table 1 introduces these testsets. It states the name of the testset, its source, and the number of instances contained. For the network design instances we also give details about the used formulations within the testset. For model variations also see Section 5. There are single-commodity (SCF) and multi-commodity (MCF) instances. The flow can be splittable (S) or unsplittable (US). The capacity formulation can be directed (DI) or undirected (UN) with a single arc facility (SF), multiple arc facilities (MF), or a big-M capacity (M) in case of uncapacitated problems. The capacity variables are either binary (BIN) with an additional generalized upper bound constraint (BIN+GUB) or they are integer (INT). Some instances are randomly generated (RG).

The *miplib* testset contains all instances from the Miplib 3 [15] and Miplib 2003 [2] libraries.

Testset	Size	Source	Paper	Problem description
arc.set	35	A. Atamtürk, [8]	[9]	MCF, S, US, BIN, DI, SF
avub	60	A. Atamtürk, [8]	[10]	SCF, BIN, DI, MF, RG
cut.set	15	A. Atamtürk, [8]	[5]	MCF, INT, DI
fc	20	A. Atamtürk, [8]	[4]	SCF, BIN, DI, SF, RG
fctp	32	J. Gottlieb, [27]		SCF, BIN, DI, SF, bipartite graphs
sndlib	52	ZIB, [12]	[44]	MCF, INT, BIN+GUB, DI, UN, MF
ufcn	83	L.A. Wolsey, [58]	[45]	SCF, BIN, DI, M
miplib	92	ZIB, [2, 15]	[2, 15]	general MIP instances
mittelmann	59	H. Mittelmann, [42]	–	general MIP instances

Table 1: Publicly available network design instances with different formulations and general MIP testsets

The *mittelmann* testset subsumes instances available on the webpage of Hans Mittelmann [42, January 2009] used to benchmark MIP-solvers. From the latter we removed instances that are already contained in *miplib* and *fctp* such that all testsets in Table 1 are disjoint. Note that Hans Mittelmann changes the definition of his testset frequently. Table 8 in Appendix A provides information about all instances regarding the number of rows (*rows*) and columns (*vars*), the value of the LP relaxation (*lp*), and the best dual (*bestdual*) and primal (*bestprimal*) bounds obtained during all our computational tests.

For all the network design instances except for the *cut.set* testset we could determine the original network the formulations are based on, that is, we know the correct number of nodes, arcs, and commodities. Thus we can compare the detected with the original networks.

All the presented results correspond to our implementation in Scip 1.1.0.8 using Cplex 11.2.1 as linear programming solver. This development version can be made available on request by the authors. There is no difference in the Mcf separator of Scip 1.1.0.8 and the publicly available Scip 1.2 [60], but note that changes in other Scip plugins and the framework itself could affect the computational results.

All calculations were done on a 64bit 3.00GHz Quad-Core machine with 6144 KB of cache and 8 GB of RAM using a single CPU. The detailed computational results for the network detection are presented in Appendix B with Table 9 and Table 10 giving the results for instances with known and unknown original network, respectively. Table 2 summarizes these results. We performed two tests. First, we switched off the preprocessing of Scip such that our network detection procedures worked on the original formulation (*detection – no presolve*). But note that the original formulation might already contain model reductions by user presolving. In the second test, Scip was run in its default settings with preprocessing switched on (*detection – presolve*). For both tests and every testset, Table 2 reports on the number of instances for which we detect a network (*nets*), the number of instances with a detected network and inconsistency ratio of at most $\Psi^{\max} = 0.02$ (*nice*), and the maximum inconsistency ratio among all instances in the testset ($max(\Psi)$). Recall that the value $\Psi^{\max}$ is used in our framework as a default parameter to decide whether or not to separate. In case the original network is available, we compare it with the detected network by taking the arithmetic mean of the percentage deviation from the original number of nodes (V), arcs (A), and commodities (K). A single node deviation, for instance, is given by the ratio $100 \cdot ||V| - |V^\star||/|V^\star|$, where $|V^\star|$ and $|V|$ correspond to the number of nodes in the original and detected network, respectively.

Let us first discuss the results for the network design instances. With solver presolving switched off we find a consistent network in almost all of the instances. The inconsistency ratio is close to zero on average and the deviations from the original network are insignificant. There are only a few exceptions. Two *fctp*-networks are not detected (bk4x3 and gr4x6), and one detected *fctp*-network significantly differs from the original one (ran4x64). (All other *fctp*-networks are correctly identified, see Table 9 in Appendix B). It turns out that some of the *fctp* flow-rows are rejected because they have a density exceeding 10% of the total number of variables, which is done by the flow-detection procedure for efficiency reasons, see above. Note that the *fctp* instances are based on complete

testset	#	detection – no presolve						detection – presolve					
					mean diff %						mean diff %		
		nets	nice	max($\Psi(G)$)	V	A	K	nets	nice	max($\Psi(G)$)	V	A	K
arc.set	35	35	35	0.009	0.0	0.0	0.7	35	35	0.008	20.1	13.4	0.9
cut.set	15	15	0	0.403	-	-	-	15	0	0.366	-	-	-
fc	20	20	20	0.000	0.0	0.0	0.0	20	20	0.002	23.3	11.5	0.0
fctp	32	30	30	0.000	3.1	3.3	6.7	30	30	0.000	3.1	3.3	6.7
avub	60	60	60	0.000	0.3	0.4	0.0	60	60	0.002	26.9	21.8	0.0
sndlib	52	52	52	0.000	0.3	0.0	0.0	52	51	0.023	0.4	0.1	0.0
ufcn	83	83	83	0.018	3.8	4.2	0.0	83	83	0.009	9.3	8.3	0.0
miplib	92	46	20	0.669	-	-	-	57	23	0.656	-	-	-
mittelmann	59	41	2	0.712	-	-	-	41	6	0.621	-	-	-

Table 2: Network detection results summary

bipartite graphs which can result in dense flow-rows. In addition, the proposed algorithm is not able to identify consistent networks in the *cut.set* instances. We observed that the algorithm already fails in the flow-detection procedure. For individual *cut.set* instances we do not know the original network but according to Atamtürk [5] the set consists of problems with 19 to 29 nodes and 23 to 93 commodities. In contrast, our flow-detection procedure detects 1 to 6 commodities with up to 168 nodes (see Table 10 in Appendix B). The matrix is not correctly decomposed into commodity blocks caused by additional coupling constraints that are misleadingly used as flow-rows.

If presolving is switched on, the detected networks obviously differ in size from the original ones. The mean deviation in the number of nodes and arcs exceeds 20% for *arc.set*, *fc*, and *avub* while the number of commodities is stable for all network design instances. For most of the instances the network size is decreased because of deleted flow-rows or flow-variables. This does however not mean that these networks are less consistent. Only for the *sndlib* testset the inconsistency ratios are noticeably increasing.

For roughly half of the general MIP instances (*miplib* and *mittelmann*) we detect a network but only a few of them are consistent. The inconsistency ratio can be close to one in general, which is not surprising. It is remarkable that with presolving switched on the number of detected networks and the number of consistent networks increases while the maximum inconsistency ratio decreases for both the *miplib* and *mittelmann* testset. It seems that some networks can be identified easier if redundant rows and columns are removed from the system. In the default settings, we find 6 *mittelmann* and 23 *miplib* instances with a network that can be considered being consistent.

4 Separation

In case the described network detection scheme identified a network $G = (V, A)$ with $\Psi(G) \leq \Psi^{\max}$, we apply the following separation scheme. Our separation heuristic relies on calculating a weight vector $u \in \mathbb{Q}_+^M$ that is used to aggregate original constraints of the system (1). For every weight vector we additionally provide a set Δ of multipliers $\delta > 0$ where $1/\delta > 0$ is chosen among the (absolute values of the) coefficients of integer variables in the capacity coupling constraints $i \in M_C$ with $u_i \neq 0$. The final base mixed integer rows are given by $\delta u^T \mathcal{A} x \leq \delta u^T b$ for all $\delta \in \Delta$.

The vector u and a multiplier $\delta \in \Delta$ are passed to the the c-MIR framework of SCIP which carries out the aggregation and scaling. It additionally applies bound substitution, complementing, and scaling as proposed by Marchand and Wolsey [40] before the MIR inequality is generated, see [1, 59] for details. We chose the vector u such that the resulting inequality is of the form (6) corresponding to a cut in the detected network. The vector u already incorporates the necessary scaling and reflecting of flow and capacity rows from the network detection procedures. In the following description we ignore this fact and assume that all flow and capacity rows are correctly scaled, that is, $u_i \in \{0, 1\}$ for all $i \in M$. To select constraints for aggregation based on the network structure we make use of the calculated mappings *rowarc* : $M_C \to A$, *rowcom* : $M_F \to K$, and *rownode* : $M_F \to V$. From *rowarc* we construct a function *arcrow* : $A \to M_C \cup \{0\}$ that returns the capacity constraint for every arc $a \in A$ or 0 if arc a is uncapacitated. From *rownode* and *rowcom* we construct a function $\mathit{nodecomrow} : V \times K \to M_F \cup \{0\}$ that returns the flow-row corresponding to node $v \in V$ and commodity $k \in K$ or 0 if node v (and hence the corresponding flow-row) is not existing for commodity k. Notice that our detection algorithm ensures that there can be at most one capacity row for every arc and at most one flow-row for every node and commodity.

The high-level separation scheme of our implementation is given by Algorithm 5. Our cut selection strategy is very close to procedures proposed in [13, 14, 30, 45, 49] which have been successfully used in branch-and-cut frameworks to solve different types of network design problems. We favor the generation of cut-based inequalities in the space of the capacity variables over the generation of mixed inequalities containing both flow and capacity variables. This is based on experimental observations that the latter are not as efficient in improving the dual bounds and performance, see for instance [13, 45, 49].

Algorithm 5: Separation scheme

Input : mappings $\mathit{arcrow} : A \to M_C \cup \{0\}$ and $\mathit{nodecomrow} : V \times K \to M_F \cup \{0\}$, primal and dual solution (x^*, π^*) of the linear programming relaxation

1 **if** $\Psi(G) > \Psi^{\max}$ **then return** `// Stop if network is inconsistent`
2 Initialize weights $u_i := 0$ for all $i \in M$
3 Calculate a collection $\mathcal{C}$ of nodesets $S \subset V$ using (x^*, π^*)
4 **for** $S \in \mathcal{C}$ **do**
5   **for** $k \in K$ **do**
6   Determine cut demand $d_S^k := \sum_{v \in S} d_v^k$, where $d_v^k := b_i$ with $i := \mathit{nodecomrow}(v, k)$
7   **end**
8   Determine demand commodities $K_S^- := \{k \in K : d_S^k < 0\}$
9   **for** $v \in S, k \in K_S^-$ **do** $i := \mathit{nodecomrow}(v, k)$ and $u_i := 1$ `// Set flow-row weights`
10   Initialize set of multipliers $\Delta := \emptyset$.
11   **for** $a \in \delta^-(S)$ **do**
12         $i := \mathit{arcrow}(a)$ and $u_i := 1$ `// Set capacity-row weights`
13   **for** $j \in I \cap N[i]$ **do** add $1/|\alpha_{ij}|$ to Δ `// Use coeffs of int variables for scaling`
14   **end**
15   **for** $\delta \in \Delta$ **do**
16         $\mathit{violation} =$ `cMIR(`u`, `δ`)` `// Generate cutset inequality (8)`
17   **if** $\mathit{violation} > 0$ **then** `add c-MIR-cut to the cut-pool`
18   **end**
19   **if** *no violated c-MIR-cut was found* **then**
20   Chose $\delta \in \Delta$ and determine a subset $A^- \subseteq \delta^-(S)$
21   **for** $a \in \delta^-(S) \backslash A^-$ **do**
22             $i := \mathit{arcrow}(a)$ and $u_i := 0$ `// Remove capacity-row for` a `from aggregation`
23   **end**
24         $\mathit{violation} =$ `cMIR(`u`, `δ`)` `// Generate flow-cutset inequality (7)`
25   **if** $\mathit{violation} > 0$ **then** `add c-MIR-cut to the cut-pool`
26   **end**
27 **end**

Algorithm 5 starts by calculating a set of cuts in the detected network (see below for details). If nodeset S is in the list $\mathcal{C}$, then also the reverse direction is considered, i.e., $V \backslash S \in \mathcal{C}$ (for the undirected case see Section 5). For every nodeset S we determine the set of demand commodities K_S^-, i.e., the set of commodities that has to be routed from $V \backslash S$ to S. We set the weights u such that in the ideal case a cutset inequality of the form (8) is generated by the c-MIR framework (Step 16 of Algorithm 5). This inequality contains only capacity variables since flow-variables for arcs in $\delta^-(S)$ are canceled out by the corresponding capacity constraints and flow-variables for arcs in $\delta^+(S)$ get a zero-coefficient by MIR. Several such inequalities might be generated because we try different scaling factors $\delta > 0$. In our implementation we use a maximum of 20 from the largest multipliers in Δ. Two multipliers $\delta_1 \geq \delta_2$ are considered to be identical if $\delta_1/\delta_2 < 1.001$.

To get tight base inequalities (having no slack) we only accept tight flow-rows for aggregation in Step 9 of Algorithm 5, and we are not accepting capacity-rows with a slack greater than 0.1 (the largest coefficient being normalized to 1) in Step 12. Recall that in the ideal case flow-rows are equations. The nodesets $S \in \mathcal{C}$ are selected to prefer tight capacity rows on the cut, see below.

In a second step, if no violated cutset inequality was found, we try to generate a flow-cutset inequality (mixed dicut inequality, flow-cover inequality) of the form (7) containing both flow and capacity-variables. For these mixed inequalities we only try the multiplier in Δ (Step 20) that gave the tightest cutset inequality. Among all possible subsets A^- of the cut-arcs $\delta^-(S)$ we determine the one that gives the most violated mixed inequality in Steps 20-23, see also [5, 45, 49]. This can be done in linear time as follows. We heuristically assume that the right-hand side of the capacity constraints is zero as in (5b), i.e., there is no pre-installed capacity on $\delta^-(S)$. In this case the right-hand side of the base inequality (6) does not depend on the chosen subset A^-. It follows that the MIR coefficients do not depend on A^-. Hence the change of the violation of the MIR inequality can be pre-calculated for every arc a that is removed from A^-. Remove, as an example, arc a from A^- in the base inequality (6). This changes the activity of the MIR inequality (7) by $f_a^* - ry_a^*$. We can start with $A^- = \delta^-(S)$ and remove all arcs a from A^- with $f_a^* < ry_a^*$ which gives the most violated inequality (7) for the given scaling factor $\delta \in \Delta$

Network cut selection – shrinking It remains to explain our cut selection strategy in Step 3. We always add all (singleton) nodesets S with $|S| = 1$ or $|V \backslash S| = 1$ to the cut-collection $\mathcal{C}$. In addition we apply a shrinking heuristic, which has been first proposed by Bienstock et al. [14] and Günlük [30], see also [45, 49]. To every nodepair $\{s,t\} \in V \times V$, for which an arc $a = (s,t)$ or $a = (t,s)$ exists, we assign a weight $w_{st} \in \mathbb{R}$ and iteratively contract the two nodes with the largest weight until $\Omega \geq 2$ node clusters are left. In the remaining graph we enumerate all cuts and add the corresponding sets S and $V \setminus S$ in the original graph to $\mathcal{C}$. The weight of a nodepair $\{s,t\}$ is initialized with the minimum of all corresponding arc-weights w_a defined by

$$w_a := s_a^* - |\pi_a^*|,$$

where s_a^* denotes the slack value of the capacity constraint $arcrow(a)$ with respect to the solution x^*. Similarly, π_a^* denotes the dual value of the row $arcrow(a)$. Note that by complementary slackness s_a^* and $|\pi_a^*|$ cannot be positive simultaneously. We set w_a to infinity if the arc a is uncapacitated. With shrinking weights defined this way, cuts are preferred that have many arcs with small slack and large absolute dual. If (all of) the capacity constraints in the cut are tight then also the base inequality will be tight. For tight base constraints, it is more likely to derive a violated MIR inequality. To substract the dual values for tight arcs is based on the heuristic argument that the inequalities we generate increase the capacity on the cut. Hence, they introduce slacks in the capacity constraints on the cut. It follows that using large absolute duals should maximally improve the dual bound. With weights that can be positive and negative, this shrinking scheme is a fast max-cut heuristic.

Obviously, the number of considered cuts increases exponentially with the size of Ω. In our implementation we use a value of $\Omega = 5$. The effect of the parameter Ω is evaluated in Section 6.2.

Many authors studying different network design problems showed that a crucial condition for a cut-based inequality to be strong (to define a facet) is that the two subgraphs $G[S]$ and $G[V \backslash S]$, i.e., the graphs defined by the nodes in S ($V \backslash S$) and the arcs with both endnodes in S ($V \backslash S$), are (strongly) connected, see [13, 23, 35, 37, 50]. In our implementation we remove nodesets S from the list $\mathcal{C}$ if either $G[S]$ or $G[V \backslash S]$ is disconnected. The connectivity check is carried out using a breadth first search algorithm on these graphs. Note, however, that every individual shore in the network partition is connected since we start with a connected network and only contract arcs.

5 Extensions

In the following we present some extensions to the algorithms introduced above. Our implementation also incorporates different model alternatives of (2). We show how these variations influence our detection and cutting plane procedure.

Multi-facility problems The capacity on a given arc is not necessarily the single product of a capacity and an (integer) capacity variable. It can be a general scalar product. In this case we speak of *multi-facility* problems [5, 10, 50, 54]. Given a set of admissible facilities T_a for every arc $a \in A$ and capacity values $c_a^t \in \mathbb{Q}_+$, $t \in T_a$, the capacity constraints change to

$$\sum_{k \in K} f_a^k - \sum_{t \in T_a} c_a^t y_a^t \leq 0 \quad \forall a \in A \tag{2b'}$$

Capacity constraints of the form (2b') do not influence our algorithms. In fact, our detection and separation framework is independent from the structure of the capacity variables and their coefficients in the capacity constraints. It only relies on the fact that (almost) all arc-flow-variables appear in the coupling inequality. For models with unbounded capacity variables it is known that the aggregation described in Section 2 and Algorithm 5 results in strong valid multi-facility cutset and flow-cutset inequalities. One simply uses the capacity constraints (2b') for bound substitution (or similarly adds them to (5a)) and considers all the facility capacities for scaling before MIR, see [5, 49, 50]. Exactly the same is done by our procedure.

Unsplittable flow models Many applications require that the flow is *unsplittable*, that is, the flow of a commodity has to use a single path from the source to the destination [9, 18, 31, 32]. To model unsplittable flow one typically introduces binary flow-variables f_a^k that state whether or not the flow of commodity k uses arc a. Additionally, the flow conservation constraints are formulated with a vector d^k defined by

$$d_v^k = \begin{cases} 1 & v = s \\ -1 & v = t \\ 0 & \text{else} \end{cases}, \qquad \forall v \in V,\, k = (s,t) \in K,$$

where $(s,t) \in V \times V$ denotes the source-target node-pair of commodity k. The actual demand values $d^{(s,t)} \in \mathbb{Q}_+$ that have to be routed on a single path from s to t are included in the capacity constraints:

$$\sum_{k=(s,t) \in K} d^{(s,t)} f_a^k - c_a y_a \leq 0 \quad \forall a \in A. \tag{2b''}$$

This results in capacity constraints with coefficients for flow-variables that are commodity dependent. Note that the same formulation alternative can be used in the context of splittable flow and single-source, single-target commodities. In this case the flow-variable f_a^k denotes the fraction of flow routed on arc a for commodity k. This formulation does not affect any of the detection procedures since none of them evaluates the coefficients in the capacity constraints. Because of a potentially smaller score, capacity-rows of type (2b'') will be considered later in the arc detection Algorithm 2. But this is only of interest if there are also capacity-rows of type (2b) among M_C that cover all commodities.

While this model variant has no influence on the network detection, we have to adapt the weights in the separation scheme. The aim to add the capacity constraints for $\delta^-(S)$ to the aggregated flow-system (5a) is to cancel out the corresponding flow variables in order to obtain the base constraint (6). Since the coefficients of the capacity constrains depend on the commodities we have to scale the flow-rows for every commodity accordingly. Before applying Algorithm 5 we heuristically normalize all capacity constraints in such a way that the coefficients for flow-variables of the same commodity are identical. Let us assume that the coefficient of the single-source, single-target commodity $k = (s,t)$ is $d^{(s,t)}$ in all capacity constraints after normalization. In this case, every flow-row for commodity k has to be scaled by $d^{(s,t)}$ which is carried out in Step 9 of Algorithm 5. Notice that normalization and scaling has no effect on the standard model with capacity constraints of type (2b).

We refer to Brockmüller et al. [18] for MIR cutset inequalities and the case that the flow is unsplittable.

Undirected capacity models Undirected capacity models appear frequently in telecommunication applications since capacities in practice are typically installed bidirectional and (s,t) demands are routed using the same paths as (t, s) demands [31, 32, 35, 37, 38, 49, 50, 54]. Assume that the digraph $G = (V, A)$ has anti-parallel arcs, i.e., for every arc $a = (v, w)$ there exists the inverted arc $a' = (w, v)$. In undirected (single-facility) formulations there is only one capacity-variable y_{vw} for every of these anti-parallel arc-pairs, and the anti-parallel flows have to share the common capacity $c_{vw}y_{vw}$:

$$\sum_{k\in K}(f^k_{(v,w)} + f^k_{(w,v)}) - c_{vw}y_{vw} \leq 0 \quad \forall a = (v, w) \in A, v < w \tag{2b'''}$$

For every commodity there are two flow-variables in every capacity constraint. While the flow-system is still directed, the direction of the flow is arbitrary and the capacitated network can be considered being undirected.

Our implementation is able to distinguish directed and undirected capacity models. Basically one can think of two different implementations. The user is able to decide which algorithm to use by setting a *modeltype* parameter. In the default setting we try to detect the modeltype automatically.

The flow detection Algorithm 1 is identical for both model types, directed or undirected. If the user is not explicitly claiming either of two detection variants, we decide about the modeltype when assigning the score to the potential capacity-rows just before the arc detection Algorithm 2. If, on average, the number of flow-variables per commodity in the capacity-row is greater than or equal to two we switch to the undirected detection algorithm. The scoring is modified accordingly. In the arc detection procedure we then construct edges instead of arcs. Again, every edge corresponds to either exactly one capacity constraint or it is uncapacitated. The node detection in Algorithm 3 is adapted in the way that the incidence pattern of a node does not depend on the direction of the incident arcs. We only compare the arc-ids of two flow-rows but not their $\{+1, -1\}$ pattern. When constructing the incidence functions in Algorithm 4 we do not distinguish between source and target node count but consider the sum $count(v) := scount(v) + tcount(v)$ and simply assign the two nodes with largest $count(V)$ to be source and target of edge a.

Given a nodeset S, the separation scheme Algorithm 5 considers the set of cut-edges $\delta(S)$ for undirected models. Since the generated inequalities are identical for S and $V\backslash S$ we only add one of the two nodesets to the list $\mathcal{C}$. Since the direction of traffic is arbitrary we calculate the set $K^+_S \cup K^-_S$. Flow-rows corresponding to K^+_S are reflected, i.e., the weight u_i is set to -1 for $i = nodecomrow(v, k)$ with $v \in S$ and $k \in K^+_S$. Hence, the right-hand side value in the base constraints (4) and (6) (the cut demand) gives $d_S = -\sum_{k\in K^+_S} d^k_S + \sum_{k\in K^-_S} d^k_S < 0$. For flow-cutset inequalities we consider a subset A^* of the cut-edges $\delta(S)$ instead of the set A^-. For more details on general flow-cutset inequalities and undirected models the interested reader is referred to [49, 50].

6 Computational Results

In this section we evaluate the performance of the Mcf-separator implemented in Scip and Cplex. We start by comparing the solvers in their default settings (*mcf*) with the Mcf-separator being switched off (*nomcf*) which is summarized in Tables 3 and 4 for Scip as well as Table 5 for Cplex. The corresponding detailed results for Scip can be found in Tables 11 and 12 in Appendix C.

For the maximum performance of our separation strategy we fixed a series of parameters based on extensive computational tests. We intended to accelerate the solvers by an order of magnitude for the network design instances without cutting too aggressively and without decreasing the performance for general MIPs. For the main test *mcf* versus *nomcf* we fixed the inconsistency parameters to $\Psi^{\max} = 0.02$ and $\Psi^{\max}_a = 0.5$ and set $\Omega = 5$. The effect of changing $\Psi^{\max}_a$ and $\Psi^{\max}$ is studied in Section 6.1. In Section 6.2 we report on the impact of the parameter Ω which relates to the number of network cuts used for separation.

For Scip we used the testsets introduced in Table 1. Table 8 in Appendix A gives an overview of the instances together with the best primal and dual bounds found in all Scip runs. For Cplex, in addition, we report on the results using the Cplex-internal testset. The Scip tests have been carried out using the same machine and the same Scip version as in Section 3.8. For the Cplex 12.1 tests we used a single CPU of a 64bit 3.33GHz Quad-Core machine with 6144 KB of cache and 16 GB of RAM. For all tests we fixed the time limit to one hour and the memory limit to 6 GB. We will distinguish *easy* and *hard* instances in our exposition. Hard instances cannot be solved

testset		#	nomcf – means rootgap closed %	time	nodes	mcf – means rootgap closed %	time	nodes	mcf/nomcf wins	t-outs	time	nodes
arc.set	all	25	58.4	31.7	3326	71.4	16.6	1103	20/4	0/0	0.52	0.33
	sep	25	58.4	31.7	3326	71.4	16.6	1103			0.52	0.33
cut.set	all	11	88.7	16.6	1232	88.7	16.5	1232	0/0	0/0	1.00	1.00
	nosep	11	88.7	16.6	1232	88.7	16.5	1232			1.00	1.00
fc	all	20	93.6	3.3	415	94.2	3.5	305	2/10	0/0	1.08	0.74
	sep	19	93.8	3.2	384	94.5	3.4	276			1.08	0.72
	nosep	1	89.1	5.9	1570	89.1	5.9	1570			1.00	1.00
fctp	all	16	76.9	4.5	1679	77.3	4.6	1603	3/5	0/0	1.02	0.95
	sep	13	73.8	6.7	3049	74.3	6.8	2885			1.02	0.95
	nosep	3	89.9	0.3	50	89.9	0.3	50			1.06	1.00
avub	all	45	86.8	55.2	4267	94.2	17.8	1396	25/8	0/14	0.32	0.33
	sep	44	86.5	60.0	4658	94.1	18.9	1491			0.32	0.32
	nosep	1	100.0	0.5	1	100.0	0.6	1			1.20	1.00
sndlib	all	22	47.7	84.7	24197	64.1	45.1	10710	18/2	0/3	0.53	0.44
	sep	21	48.7	95.3	25943	65.8	49.2	11049			0.52	0.43
	nosep	1	26.3	6.4	5555	26.3	6.5	5555			1.02	1.00
ufcn	all	58	85.7	22.1	3984	89.7	11.6	1804	32/11	0/9	0.52	0.45
	sep	58	85.7	22.1	3984	89.7	11.6	1804			0.52	0.45
miplib	all	67	62.7	7.0	816	62.5	6.9	784	4/2	0/1	0.99	0.96
	sep	13	86.5	9.0	1479	85.5	8.4	1212			0.94	0.82
	nosep	54	56.9	6.5	704	56.9	6.5	704			1.00	1.00
mittelmann	all	56	61.3	82.2	3579	61.1	85.2	3676	1/2	0/0	1.04	1.03
	sep	3	68.0	31.6	22815	64.1	57.8	37098			1.83	1.63
	nosep	53	61.0	86.8	3217	61.0	87.1	3217			1.00	1.00

Table 3: Summary for easy instances – *mcf* versus *nomcf* – SCIP

to optimality by the considered solver within the time limit regardless of whether the separator is switched on or off. All other instances are considered to be easy. Notice that this definition depends on the solver.

Tables 3-5 contain 2-3 rows for every individual testset, where row *all* refers to all instances, row *sep* corresponds to those instances for which the MCF-separator was switched on and found at least one violated inequality, and row *nosep* summarizes the results for the rest of the instances (no network found, network inconsistent, or no inequality found). The respective number of instances is given in the third column (#). If there are no instances in *sep* or *nosep* the corresponding rows are omitted.

For the instances that are easy (Table 3 SCIP and Table 5 CPLEX) we report on the geometric means of the CPU time in seconds (*time*) and the explored branch-and-bound nodes (*nodes*) used to solve the problems. For the SCIP runs in Table 3 we additionally provide the arithmetic means of the closed root gap (*closed rootgap*) which is defined as

$$100 \cdot (root - lp)/(bestprimal - lp),$$

where lp denotes the value of the initial LP relaxation, $bestprimal$ the best known primal solution value (see Table 8 in Appendix A), and $root$ the value of the LP at the root node after cutting before branching. All mean values are given for both the *mcf* and *nomcf* runs. The last four columns in Table 3 and Table 5 (*mcf/nomcf*) compare the *mcf* and *nomcf* runs with respect to the number of wins (*wins*) and the number of time or memory limit hits (*t-outs*), and they provide the time (*time*) and node (*nodes*) ratios of the respective geometric means. If by switching on the MCF-separator the time to solve the problem is decreased by at least 10% we say that the *mcf*-run "wins". If it

testset			nomcf – means				mcf – means				mcf/nomcf	
			closed gaps %			endgap	closed gaps %			endgap		
		#	root	dual	primal	%	root	dual	primal	%	wins	endgap
arc.set	all	10	33.9	59.3	86.9	1.4	35.6	61.8	86.5	1.3	2/1	0.93
	sep	10	33.9	59.3	86.9	1.4	35.6	61.8	86.5	1.3		0.93
cut.set	all	4	58.0	68.0	100.0	12.6	58.0	68.0	100.0	12.6	0/0	1.00
	nosep	4	58.0	68.0	100.0	12.6	58.0	68.0	100.0	12.6		1.00
fctp	all	16	21.2	24.0	97.1	24.9	21.3	24.1	97.5	24.8	0/0	0.99
	sep	16	21.2	24.0	97.1	24.9	21.3	24.1	97.5	24.8		0.99
avub	all	15	31.1	37.7	29.1	83.9	72.5	75.6	91.7	10.2	14/0	0.12
	sep	15	31.1	37.7	29.1	83.9	72.5	75.6	91.7	10.2		0.12
sndlib	all	30	31.9	56.5	90.5	7.6	42.0	63.8	94.2	6.2	17/2	0.82
	sep	29	32.7	57.8	90.5	8.5	43.2	65.3	94.3	6.9		0.81
	nosep	1	9.6	20.4	91.7	0.2	9.6	20.4	91.7	0.2		1.00
ufcn	all	25	74.5	80.9	84.5	10.7	81.8	87.7	91.5	7.2	19/2	0.67
	sep	25	74.5	80.9	84.5	10.7	81.8	87.7	91.5	7.2		0.67
miplib	all	25	19.0	36.3	38.2	15.9	19.0	36.5	38.2	15.9	0/0	1.00
	sep	3	11.4	28.8	99.7	14.8	11.5	30.2	99.7	14.2		0.97
	nosep	22	20.0	37.3	32.0	16.1	20.0	37.4	32.0	16.1		1.00
mittelmann	all	3	19.6	52.4	100.0	2.7	19.6	52.4	100.0	2.7	0/0	1.00
	nosep	3	19.6	52.4	100.0	2.7	19.6	52.4	100.0	2.7		1.00

Table 4: Summary for hard instances – *mcf* versus *nomcf* – Scip

increases by at least 10% the *nomcf*-run "wins".

For the hard instances (Table 4 Scip only) we report on the arithmetic mean of the closed root gaps (*root*), the closed dual gaps (*dual*), the closed primal gaps (*primal*), and the endgaps (*endgap*). The closed root gap is defined as above. The closed dual and closed primal gaps are defined as

$$100 \cdot (dual - lp)/(bestprimal - lp)$$

and

$$100 \cdot (bestprimal - bestdual)/(primal - bestdual),$$

respectively. The endgap is given by

$$100 \cdot (primal - dual)/|bestdual|.$$

The values *lp* and *bestprimal* are defined as above and can be found in Table 8 together with the *bestdual* which refers to the best known dual bound. The numbers *primal* and *dual* correspond to the primal and dual bound at the end of the optimization. Note that in case that $primal = bestdual$ for an individual run we set the closed primal gap to 100%. If the LP value is already optimal ($lp = bestprimal = bestdual$) then *rootgap* as well as *dualgap* are considered to be 100%. If primal or dual bounds are not finite or in case that $bestdual = 0$ the corresponding gaps are not defined and hence not considered in the calculation of the mean. Again all mean values are given for both the *mcf* and *nomcf* runs. The last two columns in Table 4 (*mcf/nomcf*) compare the *mcf* and *nomcf* runs with respect to the number of wins (*wins*) and the endgap. If by switching on the Mcf-separator the endgap decreases by at least 10% we say that the *mcf*-run wins. If it increases by at least 10% the *nomcf*-run wins.

In Table 3 it can be seen that our implementation of the Mcf-separator in Scip drastically reduces the computation times and branch-and-bound nodes for almost all of the network design instances. In particular for the testsets *arc.set*, *avub*, *sndlib*, and *ufcn* we save between 56% and 67% of the tree nodes and between 47% and even 68% of the solving time on average. Moreover, 14 *avub*, 2 *sndlib*, and 9 *ufcn* instances can be solved within the time limit of one hour only if the Mcf separator is switched on. Table 4 shows similar effects for the hard instances of these 4 testsets. The average endgap is decreased and the *mcf*-run wins in most of the cases. The results for the hard *avub* instances are remarkable. We decrease the endgap from 83.9% to 10.2% on average which is caused by improving both the dual and the primal bounds. Already at the root node we close the optimality gap by 72.5% compared to 31.1% without the Mcf separator. For the very easy testsets *fc* and also *fctp* (excluding the n37* instances) with an average solving time of less than 5s we decrease the number of nodes but slightly increase the solving time (from 3.3s to 3.5s and from 4.5s to 4.6s on average), see Table 3. For these instances it does not pay off to tighten the relaxation. Our separator has no effect on the hard *fctp* n37* instances (see Table 4 and 12) and for the (hard and easy) *cut.set* instances the Mcf separator is switched off because the networks are not consistent, also compare with Table 2.

testset		#	nomcf – means		mcf – means		mcf/nomcf			
			time	nodes	time	nodes	wins	t-outs	time	nodes
arc.set	all	25	14.9	3244	10.2	1258	15/4	0/0	0.70	0.39
	sep	23	16.9	3344	11.2	1194			0.68	0.36
	nosep	2	3.2	2286	3.2	2286			1.01	1.00
cut.set	all	12	12.7	892	12.7	1232	0/0	0/0	1.00	1.00
	nosep	12	12.7	892	12.7	1232			1.00	1.00
fc	all	20	1.5	270	1.6	260	5/7	0/0	1.04	0.97
	sep	20	1.5	270	1.6	260			1.04	0.97
fctp	all	17	3.9	751	4.1	687	3/5	1/0	1.04	0.92
	sep	15	5.0	1329	5.3	1202			1.05	0.91
	nosep	2	0.1	1	0.1	1			1.00	1.00
avub	all	41	4.6	413	1.8	163	18/2	0/3	0.50	0.33
	sep	40	4.8	454	1.9	175			0.49	0.32
	nosep	1	0.1	1	0.1	1			1.00	1.00
sndlib	all	25	93.3	20353	41.1	7427	18/1	0/4	0.45	0.37
	sep	22	81.1	15624	31.8	4967			0.40	0.32
	nosep	3	258.5	141400	258.4	141400			1.00	1.00
ufcn	all	67	3.3	349	3.5	404	8/11	0/0	1.05	1.15
	sep	59	3.4	379	3.6	448			1.05	1.18
	nosep	8	2.6	186	2.6	186			1.01	1.00
miplib	all	66	3.0	558	2.9	556	3/1	0/0	0.99	1.00
	sep	8	5.7	1060	5.2	1023			0.93	0.97
	nosep	58	2.7	511	2.7	511			1.00	1.00
mittelmann	all	56	23.2	1101	22.4	1073	3/2	0/0	0.97	0.98
	sep	6	73.2	5232	52.2	4135			0.72	0.79
	nosep	50	20.1	912	20.2	912			1.00	1.00
cplex	all	1266	34.9	1201	34.2	1170	45/35	5/5	0.98	0.97
	sep	115	42.8	6665	34.9	5006			0.82	0.75
	nosep	1151	34.1	1011	34.2	1011			1.00	1.00
cplex10s	all	780	132.5	3225	128.8	3118	30/25	5/5	0.97	0.97
	sep	77	142.9	15230	106.8	10856			0.75	0.71
	nosep	703	131.4	2719	131.5	2719			1.00	1.00
cplex100s	all	411	504.4	8989	484.1	8589	20/13	5/5	0.96	0.96
	sep	42	546.3	38192	365.8	24486			0.67	0.64
	nosep	369	499.8	7623	499.8	7623			1.00	1.00

Table 5: Summary for easy instances – *mcf* versus *nomcf* – Cplex

The results for the general MIP sets *miplib* and *mittelmann* are not conclusive since the number of affected instances is very small (only 16 *miplib* and 3 *mittelmann* instances overall). There is some decrease in the computation time and nodes for *miplib* and one instance can only be solved in the *mcf*-run but for 2 out of 3 *mittelmann* instances the performance degrades.

Table 5 shows that the results for CPLEX are comparable to the results for SCIP with respect to the easy network design instances. The effect is not as dramatic since CPLEX is already very fast without the MCF separator (compare the average *nomcf* computation times in Table 3 and Table 5). But the decrease of the computation time is still between 30% and 55% for the *arc.set*, *avub*, and *sndlib* instances with 61%-67% saved branch-and-bound nodes. In contrast to SCIP the time increases for the *ufcn* testset but these instances are very easy for CPLEX with average solving times below 5s in the *nomcf* run similar to the *fc* and *fctp* testsets.

Let us discuss the results in Table 5 for the general MIP instances with CPLEX. In contrast to Table 3 and SCIP we can trust these values since the overall testset is very large with a reasonable number of affected instances. In addition to the *miplib* and *mittelmann* testsets we consider an internal CPLEX-library containing 1266 easy instances (*cplex*). The subsets *cplex10s* and *cplex100s* correspond to those instances within *cplex* that need at least 10s and 100s of CPU time to be solved, respectively, by the slower of the two versions, *mcf* and *nomcf*. Among all 1388 MIP instances (*miplib*, *mittelmann*, and *cplex*) 129 instances or 9.3% are affected by the MCF separator. We save 17.9% of the computation time and 23.6% of the search tree nodes on average for these 129 instances which refers to a 2% time and 2.8% node savings over the whole testset. Moreover, it turns out that the harder the instances are to solve the larger are the benefits of the MCF separator. The saved time amounts to 25% for the 77 affected instances in *cplex10s* and to even 36% for the 42 affected instances in *cplex100s*.

In all cases (SCIP, CPLEX, easy and hard), if the separator is switched off or does not find violated inequalities there is almost no degradation of the computation time (see the *nosep* rows). This means that the detection as well as the separation procedures are very fast. Summarizing it can be said that using the MCF separator many instances can now be solved within 1 hour that could not be solved before. For the instances the separator is designed for a significant reduction in the computation time and the branch-and-bound nodes is observed which is driven by improved dual bounds at the root node. Whenever the solvers struggle in solving a specific problem class switching on the MCF separator gives substantial benefits (see e.g. *avub*, *sndlib* for SCIP and *sndlib*, *cplex100s* for CPLEX).

6.1 The impact of inconsistency

The two parameters $\Psi_a^{\max}$ and $\Psi^{\max}$ control our algorithm with respect to inconsistent or not existing networks in the constraint matrix. If violated inequalities are identified these are always valid since our framework relies on aggregating original constraints and on applying MIR to aggregations. But the larger the inconsistency in the network the lower the chance to produce base inequalities that correspond to network cuts or to have any relation to a network. For very large inconsistency we basically simulate randomized aggregation. Moreover, since the number of considered original constraints can be very large (in contrast to the default c-MIR heuristics) and there is not necessarily a proper cancellation of flow-variables in case of inconsistency we might produce dense and unstable cutting planes.

testset	#	$\Psi^{\max} = 0.02^\star$ $\Psi_a^{\max} = 0.5^\star$			$\Psi^{\max} = 0.02^\star$ $\Psi_a^{\max} = \infty$			$\Psi^{\max} = \infty$ $\Psi_a^{\max} = 0.5^\star$			$\Psi^{\max} = \infty$ $\Psi_a^{\max} = \infty$		
		sep	wins	time	sep	wins	time	sep	wins	time	sep	wins	time
cut.set	11	0	0/0	1.00	0	0/0	1.00	9	4/2	0.90	9	3/3	1.02
sndlib	22	21	18/2	0.53	21	18/2	0.53	22	19/2	0.51	22	19/2	0.51
miplib	67	13	4/2	0.99	13	4/2	0.99	18	5/3	0.98	20	5/6	0.99
mittelmann	56	3	1/2	1.04	3	1/2	1.03	9	3/3	1.03	10	6/3	1.01

Table 6: Impact of inconsistency – SCIP easy. Default values are marked with $\star$.

Increasing $\Psi_a^{\max}$ means to increase the size of the networks (and hence the size of the aggregations) by allowing for more inconsistent arcs (and the corresponding capacity constraints). These are arcs with uncertain source or target assignment. On the other hand, increasing $\Psi^{\max}$ means to consider more instances for separation. In the first test, using SCIP, we released both parameters $\Psi_a^{\max}$ and $\Psi^{\max}$ individually and simultaneously. Table 6 reports on the results for *cut.set*, *sndlib*, *miplib*, and *mittelmann*. All other testsets are not affected since the inconsistency ratios are very small for all contained instances, see also Table 2. The second column in Table 6 gives the total number of easy instances in the testset. For every run there are three columns providing the number of affected instances (*sep*), the ratio of the wins (*wins*), and the time ratio (*time*). The ratios compare the respective *mcf*-run with the *nomcf*-run. The time ratios are based on geometric means over the whole testset (not only the affected instances). The number of clusters Ω is fixed to 5 in all runs. Columns 3-5 in Table 6 summarize the values for the default settings for comparison. These are precisely the values you can already find in Table 3.

It can be seen that releasing the maximum arc inconsistency ratio $\Psi_a^{\max}$ alone (Columns 6-8) does not remarkably change the behavior of the solver. Recall that the network inconsistency ratio $\Psi(G)$ is defined as the mean of the arc inconsistency ratios $\Psi(a)$. Hence in case of a very small value $\Psi(G)$ there cannot be many inconsistent arcs such that releasing $\Psi_a^{\max}$ while keeping $\Psi^{\max}$ small has not a great impact on our algorithm. On the other hand, relaxing the maximum network inconsistency ratio $\Psi^{\max}$ while keeping $\Psi_a^{\max} = 0.5$ (Column 9-11) even seems to improve the performance. There are more instances considered for separation (+9 *cut.set*, +1 *sndlib*, +5 *miplib*, and +6 *mittelmann*) and the wins and time ratios are improving. We even get a 10% decrease in computation time for the *cut.set* instances. It turns out that for networks with large inconsistency ratio it suffices to delete inconsistent arcs (which for many instances might result in empty networks anyway). But, as shown in Columns 12-14 of Table 6, releasing both inconsistency parameters worsens the performance at least for *cut.set* and *miplib* in terms of wins and time ratios.

In a second test we study the impact of different values for $\Psi^{\max}$ while fixing $\Psi_a^{\max} = \infty$. We restrict our attention to the easy instances within the *miplib* testset. Figure 6 shows the number of instances that are considered to contain a consistent network (*nice*) and the number of instances for which at least one violated inequality was found (*sep*). The value *wins(mcf)-wins(nomcf)* refers to the difference of the number of instances for which the computation time was decreased (*wins(mcf)*) and increased (*wins(nomcf)*) by at least 10% using the MCF separator. There are 14 easy instances in *miplib* with an embedded network and $\Psi(G) = 0$. For 11 of these instances we found at least one violated inequality. It is no surprise that the number of considered instances increases with $\Psi^{\max}$. Since the largest inconsistency ratio in *miplib* is 0.656 (compare with Table 2) a value $\Psi^{\max} = 0.8$

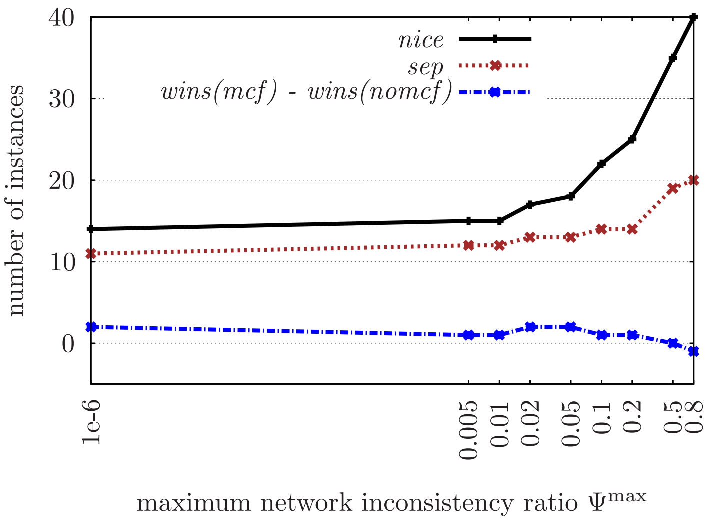


Figure 6: Impact of inconsistency – SCIP *miplib* easy. $\Psi_a^{\max}$ set to ∞ and $\Psi^{\max}$ increasing

means that the multi-commodity flow separator is switched on for all 40 easy *miplib* instances containing an embedded network. But it is remarkable that the number of affected instances only slightly increases to 20. For most of the instances with very large inconsistency ratios the generated inequalities are not violated such that separation based on the network detection has no effect. Moreover, for values of $\Psi^{\max}$ larger than 0.05 there are more and more instances for which using the MCF separator increases the computation time. Notice that *wins(mcf)-wins(nomcf)* decreases. Hence even if violated inequalities are found they do not help.

Summarizing the observed phenomena from Table 6 and Figure 6, one should switch on at least one of the two mechanisms refusing inconsistent networks or network components. Releasing the maximum network inconsistency $\Psi^{\max}$ alone seems to have a positive effect for SCIP. This has not been observed with CPLEX such that we decided to use a more conservative default setting with both $\Psi_a^{\max}$ and $\Psi^{\max}$ being active. For $\Psi^{\max}$ a good trade-off between performance and the number of affected instances seems to be between 0.02 and 0.05. Based on a number of similar test scenarios we fixed $\Psi_a^{\max}$ to 0.5.

6.2 The impact of aggressive cutting

By changing the value of Ω we control the size of the partition used to enumerate network cuts and thus the size of the network cut collection $\mathcal{C}$, see Section 4. For directed networks the number of considered cuts amounts to a maximum of $2^{\Omega} - 2$, but recall that we allow for cutsets only if both cut-shores are connected. The parameter Ω should be large enough to produce enough interesting cutsets and cut-based inequalities. But setting it too large can result in unacceptable computation times for calculating the inequalities itself and also for solving the LP relaxations since too many violated inequalities might be added.

In the test reported in Table 7 we increased the value Ω from 3 to 9 fixing the inconsistency parameters $\Psi^{\max}$ and $\Psi_a^{\max}$ to their default values. Table 7 contains all testsets accept *cut.set* for which no inequalities are separated independent of Ω. We report on the number of easy instances contained in each of the testsets (#) as well as the number of affected instances (*sep*) which is constant over the considered scenarios. For every run we report on the ratio of the wins (*wins*), the geometric mean of the time ratios (*time*), and the arithmetic mean of the number of inequalities added to the LP (*#cuts*). The means are taken over all easy instances of the testset.

First it can be observed that the number of generated cutting planes increases with Ω but the increase is not exponential. The number of added inequalities approximately doubles from $\Omega = 3$ to $\Omega = 9$. Only the *sndlib* instances really benefit from a large Ω value. For this testset the time ratio decreases from 0.64 for $\Omega = 3$ to 0.37 for $\Omega = 9$. The performance is slightly deteriorated for *fc*, *avub*, *ufcn*, *miplib*, and *mittelmann* while it is slightly improved for *arc.set* and *fctp*. We decided to fix Ω to the conservative value 5. For certain classes of network design instances it can be crucial to cut more aggressively.

testset	#	sep	$\Omega = 3$			$\Omega = 5^\star$			$\Omega = 7$			$\Omega = 9$		
			wins	time	#cuts	wins	time	#cuts	wins	time	#cuts	wins	time	#cuts
arc.set	25	25	20/3	0.53	110	20/4	0.52	133	19/3	0.57	169	20/0	0.50	205
fc	20	19	3/13	1.13	423	2/10	1.08	464	6/11	1.10	620	3/14	1.24	819
fctp	16	13	3/3	0.99	424	3/5	1.02	455	4/3	0.98	506	3/6	0.99	699
avub	45	44	27/8	0.32	249	25/8	0.32	256	25/8	0.36	278	22/11	0.34	329
sndlib	22	21	18/2	0.64	102	18/2	0.53	144	15/4	0.38	220	16/2	0.37	296
ufcn	58	58	29/11	0.52	118	32/11	0.52	135	30/13	0.53	202	24/16	0.54	358
miplib	67	13	4/1	0.99	54	4/2	0.99	58	3/3	0.99	66	4/3	1.00	73
mittelmann	56	3	1/2	1.03	6	1/2	1.04	8	0/3	1.04	5	0/2	1.04	5

Table 7: Impact of the partition size Ω – SCIP easy. Default values are marked with $\star$.

7 Concluding remarks

Based on the observation that cut-based MIR inequalities can be used to drastically reduce computation times and gaps when generated within branch & cut procedures to solve network design problems, and based on the fact that these strong inequalities are not detected by state of the art MIP solvers, we proposed a separation framework for general MIP that is now implemented in Scip and Cplex. This algorithm consists of two main steps.

First, we try to identify the block structure of a multi-commodity flow formulation in the constraint matrix of a general MIP. Coupling capacity constraints are used to resolve the isomorphism of the graphs represented by the network matrices of individual commodity blocks. The corresponding underlying network is constructed.

In a second step we derive cutting planes based on the identified network structure. Using mappings from network elements to rows of the original MIP formulation, we replace the default aggregation heuristic of the c-MIR separator implemented in Scip and Cplex. In our framework, rows are aggregated such that the resulting base inequalities correspond to network cuts. These base inequalities are then used to generate MIR cut-set inequalities, flow-cover inequalities, dicut inequalities, and the like, depending on the type of capacity constraints and variables. In contrast to default aggregation of the c-MIR separator the number of aggregated rows depends on the size of the network and can be in the order of hundreds. However, the calculated base inequalities are sparse due to the $\{+1, +1\}$ pattern in the detected network matrices.

One of the key-features in our implementation is to decide about the consistency of the detected networks. On the one hand, we delete inconsistent network elements in order to work on the consistent network core. In addition, only if the calculated overall network inconsistency ratio is very small we trust the detected structures on which we try not generate cutting planes. With this machinery we are able to recognize network design type models for which the methods are successful, introducing almost no overhead for other models.

By extensive computational tests we showed that the proposed separation scheme speeds-up the computation for a large set of network design problems by a factor of two on average. Many of these problems can only be solved within one hour of CPU time if the Mcf separator is switched on. In roughly 10% of general MIP instances we found consistent embedded networks. For these instances the computation time is decreased by 18% on average. There is almost no degradation for the remaining instances.

Given these results and the fact that state-of-the-art MIP solvers have almost no knowledge about the underlying problem, one might consider a new paradigm of exploiting structure in MIP solving. Many known and very successful approaches (cutting planes, heuristics, branching rules) for special purpose problems can be used within the MIP solver if the constraint matrices are scanned for known structures more consequently.

References

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007. `http://opus.kobv.de/tuberlin/volltexte/2007/1611/`.

[2] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4): 361–372, 2006. URL `http://miplib.zib.de/`.

[3] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[4] A. Atamtürk. Flow pack facets of the single node fixed-charge flow polytope. *Operations Research Letters*, 29:107–114, 2001.

[5] A. Atamtürk. On capacitated network design cut-set polyhedra. *Mathematical Programming*, 92:425–437, 2002.

[6] A. Atamtürk. On the facets of the mixed-integer knapsack polyhedron. *Mathematical Programming*, 98:145–175, 2003.

[7] A. Atamtürk. Cover and pack inequalities for mixed integer programming. *Annals of Operations Research*, 139(1):21–38, 2005.

[8] A. Atamtürk. MIP instances. University of California, Berkeley `http://www.ieor.berkeley.edu/~atamturk/data/`, 2009.

[9] A. Atamtürk and D. Rajan. On splittable and unsplittable capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

[10] A. Atamtürk, , G. L. Nemhauser, and M. W. P. Savelsbergh. Valid inequalities for problems with additive variable upper bounds. *Mathematical Programming*, 91:145–162, 2001.

[11] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.

[12] Zuse Institute Berlin. SNDlib – Survivable Network Design Library. `http://sndlib.zib.de/`, 2009.

[13] D. Bienstock and O. Günlük. Capacitated network design – polyhedral structure and computation. *INFORMS Journal on Computing*, 8:243–259, 1996.

[14] D. Bienstock, S. Chopra, Oktay Günlük, and C. Y. Tsai. Minimum cost capacity installation for multicommodity network flows. *Mathematical Programming*, 81:177–199, 1998.

[15] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, June 1998. URL `http://www.caam.rice.edu/~bixby/miplib/miplib.html`.

[16] R.E. Bixby and R. Fourer. Finding embedded network rows in linear programs I. Extraction heuristics. *Management Science*, 34(3):342–376, 1988.

[17] R.E. Bixby and E. Rothberg. Progress in computational mixed integer programming – a look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, 2007.

[18] B. Brockmüller, O. Günlük, and L. A. Wolsey. Designing private line networks: polyhedral analysis and computation. *Transactions on Operational Research*, 16:7–24, 2004.

[19] G.G. Brown and W.G. Wright. Automatic identification of embedded network rows in large-scale optimization models. *Mathematical Programming*, 29:41–56, 1984.

[20] M. R. Bussieck, P. Kreuzer, and U. T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming*, 79(1–3):415–444, 1997.

[21] S. Chopra, I. Gilboa, and S. T. Sastry. Source sink flows with capacity installation in batches. *Discrete Applied Mathematics*, 86:165–192, 1998.

[22] COmputational INfrastructure for Operations Research (COIN-OR). Cut Generation Library (CGL). `https://projects.coin-or.org/Cgl`, 2009.

[23] G. Dahl and M. Stoer. A polyhedral approach to multicommodity survivable network design. *Numerische Mathematik*, 68:149–167, 1994.

[24] G. Dahl and M. Stoer. A cutting plane algorithm for multicommodity survivable network design problems. *INFORMS Journal on Computing*, 10:1–11, 1998.

[25] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[26] J. P. M. Gonçalves and Laszlo Ladanyi. An implementation of a separation procedure for mixed integer rounding inequalities. IBM Research Report RC23686 (W0508-022), IBM, 2005.

[27] J. Gottlieb and H. Mittelmann. FCTP instances. Arizona State University, `http://plato.la.asu.edu/ftp/fctp/`, 2009.

[28] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:436–467, 1999.

[29] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Sequence independent lifting in mixed integer programming. *INFORMS Journal on Computing*, pages 109–129, 2000.

[30] O. Günlük. A branch and cut algorithm for capacitated network design problems. *Mathematical Programming*, 86:17–39, 1999.

[31] S. P. M. Hoesel, A. M. C. A. Koster, R. L. M. J. van de Leensel, and M. W. P. Savelsbergh. Polyhedral results for the edge capacity polytope. *Mathematical Programming*, 92(2):335–358, 2002.

[32] S. P. M. Hoesel, A. M. C. A. Koster, R. L. M. J. van de Leensel, and M. W. P. Savelsbergh. Bidirected and unidirected capacity installation in telecommunication networks. *Discrete Applied Mathematics*, 133:103–121, 2004.

[33] IBM-ILOG. Cplex. `http://www.ilog.com/products/cplex/`, 2009.

[34] Q. Louveaux and L. A. Wolsey. Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. *4OR*, 1(3):173–207, 2003.

[35] T. L. Magnanti and P. Mirchandani. Shortest paths, single origin-destination network design and associated polyhedra. *Networks*, 33:103–121, 1993.

[36] T. L. Magnanti and R. T. Wong. Network Design and Transportation Planning: Models and Algorithms. *TRANSPORTATION SCIENCE*, 18(1):1–55, 1984.

[37] T. L. Magnanti, P. Mirchandani, and R. Vachani. The convex hull of two core capacitated network design problems. *Mathematical Programming*, 60:233–250, 1993.

[38] T. L. Magnanti, P. Mirchandani, and R. Vachani. Modelling and solving the two-facility capacitated network loading problem. *Operations Research*, 43:142–157, 1995.

[39] H. Marchand. *A polyhedral Study of the Mixed Knapsack Set and its use to Solve Mixed Integer Programs.* PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, July 1997.

[40] H. Marchand and L. A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001.

[41] H. Marchand and L. A. Wolsey. The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85:15–33, 1999.

[42] H. Mittelmann. Benchmarks for optimization software. `http://plato.asu.edu/bench.html`, January 2009.

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

[44] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. *Networks*, 2009. To appear.

[45] F. Ortega and L. A. Wolsey. A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, 41:143–158, 2003.

[46] M. W. Padberg, T. J. Van Roy, and L. A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

[47] M. Pióro and D. Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann Publishers, 2004.

[48] Y. Pochet and L. A. Wolsey. Integer knapsack and flow covers with divisible coefficients. *Discrete Applied Mathematics*, 59:57–74, 1995.

[49] C. Raack, A. M. C. A. Koster, S. Orlowski, and R. Wessaely. Capacitated network design using general flow-cutset inequalities. In *Proceedings of the Third International Network Optimization Conference (INOC 2007), Spa, Belgium*, 2007.

[50] C. Raack, A. M. C. A. Koster, S. Orlowski, and R. Wessäly. On the strength of cut-based inequalities for capacitated network design polyhedra. ZIB-Report 07-08, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2007.

[51] M.G.C. Resende and P.M. Pardalos, editors. *Handbook of Optimization in Telecommunications*. Springer, 2006.

[52] T. J. Van Roy and L. A. Wolsey. Valid inequalities for mixed 0-1 programs. *Discrete Applied Mathematics*, 14:199–213, 1986.

[53] R. Weismantel. On the 0/1 knapsack polytope. *Mathematical Programming*, 77:49–68, 1997.

[54] R. Wessäly. *Dimensioning Survivable Capacitated NETworks*. PhD thesis, Technische Universität Berlin, April 2000.

[55] L. A. Wolsey. Faces for a linear inequality in 0-1 variables. *Mathematical Programming*, 8: 165–178, 1975.

[56] L. A. Wolsey. Valid inequalities and superadditivity for 0-1 integer programs. *Mathematics of Operations Research*, 2:66–77, 1977.

[57] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

[58] L.A. Wolsey. UFCN instances. CORE, Université catholique de Louvain `http://www.core.ucl.ac.be/wolsey/ufcn.htm`, 2009.

[59] K. Wolter. Implementation of Cutting Plane Separators for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006.

[60] Zuse Institut Berlin. SCIP - Solving Constraint Integer Programs. `http://scip.zib.de/`, 2009.

A Used instances

Table 8 contains all instances used in the computational tests presented in Section 6 except for the CPLEX-internal testset. We report on the number of rows (*rows*), the number of columns (*vars*), the value of the linear programming relaxation (*lp*), and the values of the best dual (*bestdual*) and primal bounds (*bestprimal*) we found in all of our SCIP runs. These values are used to calculate the measures *rootgap*, *primalgap*, and *dualgap* presented in Table 11 and 12 and summarized in Table 3 and 4.

problem	rows	vars	lp	bestdual	bestprimal
arc.set					
ns25-pr12	2313	5868	52957.5	53905	53905
ns25-pr3	2210	8601	29920	30575	30575
ns25-pr4	1138	3393	16175	16705	16705
ns25-pr6	2639	6919	25362.5	26175	26175
ns25-pr9	2220	7350	29175	29430	29430

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
ns4-pr12	2313	5868	63911	64072	64072
ns4-pr3	2210	8601	36073	36124.5	36141
ns4-pr4	1138	3393	19257.16667	19323	19323
ns4-pr6	2639	6919	29170.75	29314	29314
ns4-pr9	2220	7350	35212.5	35214.5	35231
ns60-pr12	2313	5868	43501.25	45795	45795
ns60-pr3	2210	8601	23572.5	26310	26310
ns60-pr4	1138	3393	12623.75	13825	13825
ns60-pr6	2639	6919	21205	23165	23165
ns60-pr9	2220	7350	22962.5	24475	24475
nu120-pr12	2313	5868	38370	42215	42215
nu120-pr3	2210	8601	21306.44231	26305.33451	28785
nu120-pr4	1138	3393	12067.91667	15755	15755
nu120-pr6	2639	6919	21615	24515	24515
nu120-pr9	2220	7350	19512.5	23642.34321	24945
nu25-pr12	2313	5868	52957.5	53905	53905
nu25-pr3	2210	8601	29920	31341.97133	31720
nu25-pr4	1138	3393	16175	16780	16780
nu25-pr6	2639	6919	25362.5	26175	26175
nu25-pr9	2220	7350	29175	29676.12093	30015
nu4-pr12	2313	5868	63911	64150	64150
nu4-pr3	2210	8601	36073	36613.21154	36915
nu4-pr4	1138	3393	19257.16667	19640	19640
nu4-pr6	2639	6919	29170.75	29400	29400
nu4-pr9	2220	7350	35212.5	35420.77222	35520
nu60-pr12	2313	5868	43501.25	45840	45840
nu60-pr3	2210	8601	23572.5	26178.20136	26830
nu60-pr4	1138	3393	12623.75	14195	14195
nu60-pr6	2639	6919	21205	23165	23165
nu60-pr9	2220	7350	22962.5	24420.96148	24940
cut.set					
n1-3	600	1248	3319.117647	7235	7235
n10-3	912	1710	2764.117647	8985	8985
n11-3	432	864	1360.294118	4356	4356
n12-3	2430	6120	13298.23529	20407.86241	20560
n13-3	1723	3472	4287.647059	13385	13385
n14-3	1078	2300	3438.823529	9566	9566
n15-3	29494	153140	23703.94118	30621.81273	50765.41948
n2-3	1800	3456	6181.176471	12640	12640
n3-3	2425	9028	7465.294118	14022.23351	16001
n4-3	1236	3596	4080.882353	8993	8993
n5-3	1062	2550	2883.823529	8105	8105
n6-3	2760	7178	6311.764706	15175	15175
n7-3	2336	5626	9380.588235	15426	15426
n8-3	1362	2790	4200.588235	12535	12535
n9-3	2364	7644	7889.705882	12950.45195	14409
fc					
fc.30.50.1	247	434	120	307	307
fc.30.50.10	247	434	93	204	204
fc.30.50.2	247	434	152	325	325
fc.30.50.3	247	434	98.3125	294	294
fc.30.50.4	247	434	100	763	763
fc.30.50.5	247	434	160	301	301
fc.30.50.6	247	434	129	272	272
fc.30.50.7	247	434	98	231	231
fc.30.50.8	247	434	186	347	347

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
fc.30.50.9	247	434	85	741	741
fc.60.20.1	414	708	171	487	487
fc.60.20.10	414	708	180	913	913
fc.60.20.2	414	708	241.8522167	584	584
fc.60.20.3	414	708	168.8730159	493	493
fc.60.20.4	414	708	123	442	442
fc.60.20.5	414	708	182	414	414
fc.60.20.6	414	708	203	480	480
fc.60.20.7	414	708	177	492	492
fc.60.20.8	414	708	202	500	500
fc.60.20.9	414	708	171	397	397
fctp					
bal8x12	116	192	451.1880952	471.55	471.55
bk4x3	19	24	321.6666667	350	350
gr4x6	34	48	185.55	202.35	202.35
n3700	5150	10000	972305.748	1044288.906	1333058
n3701	5150	10000	961764.0223	1032761.019	1310970
n3702	5150	10000	961184.2125	1030732.544	1315025
n3703	5150	10000	934212.2189	1004210.525	1316442
n3704	5150	10000	969800.2071	1041739.908	1327179
n3705	5150	10000	973361.017	1041675.373	1329843
n3706	5150	10000	960882.1498	1038148.26	1310913
n3707	5150	10000	935136.8543	1011762.868	1301218
n3708	5150	10000	967522.8052	1034600.74	1329479
n3709	5150	10000	959314.2185	1032234.141	1324904
n370a	5150	10000	979219.5448	1058369.149	1353042
n370b	5150	10000	988308.1237	1054184.669	1346406
n370c	5150	10000	965430.9494	1041864.223	1329640
n370d	5150	10000	965430.9494	1041864.223	1329640
n370e	5150	10000	961651.3119	1038332.371	1296715
ran10x10a	120	200	1252.742491	1499	1499
ran10x10b	120	200	2613.469453	3073	3073
ran10x10c	120	200	11203.09246	13007	13007
ran10x12	142	240	2426.224725	2714	2714
ran10x26	296	520	3857.022783	4270	4270
ran12x12	168	288	1826.549744	2291	2291
ran12x21	285	504	3157.377442	3664	3664
ran13x13	195	338	2691.439469	3252	3252
ran14x18	284	504	3016.944354	3618.383345	3712
ran16x16	288	512	3116.429512	3823	3823
ran17x17	323	578	1215.2457	1373	1373
ran4x64	324	512	9637.933333	9711	9711
ran6x43	307	516	6244.707023	6330	6330
ran8x32	296	512	4937.584531	5247	5247
avub					
nexp.100.20.1.1	2080	1980	43.9	48	48
nexp.100.20.1.2	2080	1980	42.16666667	50	50
nexp.100.20.1.3	2080	1980	44.90909091	51	51
nexp.100.20.1.4	2080	1980	42.07142857	55	55
nexp.100.20.1.5	2080	1980	41.66666667	48	48
nexp.100.20.2.1	2080	2970	79.81818182	91	91
nexp.100.20.2.2	2080	2970	67.73333333	75.58832584	77
nexp.100.20.2.3	2080	2970	79.63636364	88	88
nexp.100.20.2.4	2080	2970	52.4	63	63
nexp.100.20.2.5	2080	2970	66.93333333	76	76
nexp.100.20.4.1	2080	4950	84	121	121

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
nexp.100.20.4.2	2080	4950	79.81818182	124	124
nexp.100.20.4.3	2080	4950	81.27272727	127	127
nexp.100.20.4.4	2080	4950	59.52941176	113	113
nexp.100.20.4.5	2080	4950	80.18181818	131	131
nexp.100.20.8.1	2080	8910	12.70103093	152.8761993	157
nexp.100.20.8.2	2080	8910	12.06872852	159.1561744	164
nexp.100.20.8.3	2080	8910	15.45017182	173.2858026	222
nexp.100.20.8.4	2080	8910	13.91065292	163.9339052	169
nexp.100.20.8.5	2080	8910	12.12371134	145.0365993	219
nexp.150.20.1.1	4620	4470	59	69	69
nexp.150.20.1.2	4620	4470	58.54545454	66	66
nexp.150.20.1.3	4620	4470	52.16666667	68	68
nexp.150.20.1.4	4620	4470	57.63636364	71	71
nexp.150.20.1.5	4620	4470	56.33333333	66	66
nexp.150.20.2.1	4620	6705	96.93333333	107	107
nexp.150.20.2.2	4620	6705	90.4	96.74020384	103
nexp.150.20.2.3	4620	6705	117.0909091	123	123
nexp.150.20.2.4	4620	6705	111.2307692	123.4045085	126
nexp.150.20.2.5	4620	6705	96.30769231	106	106
nexp.150.20.4.1	4620	11175	137.6363636	191.8333333	193
nexp.150.20.4.2	4620	11175	83.29411765	159	159
nexp.150.20.4.3	4620	11175	113.8181818	173	173
nexp.150.20.4.4	4620	11175	73.64705882	147	147
nexp.150.20.4.5	4620	11175	115.2727273	176	176
nexp.150.20.8.1	4620	20115	20.81099656	239.20596	318
nexp.150.20.8.2	4620	20115	17.70446735	216.7981222	275
nexp.150.20.8.3	4620	20115	17.20962199	210.573206	367
nexp.150.20.8.4	4620	20115	13.66323024	182.4681098	260
nexp.150.20.8.5	4620	20115	18.22680412	217.6294399	325
nexp.50.20.1.1	540	490	12.03030303	29	29
nexp.50.20.1.2	540	490	24	27	27
nexp.50.20.1.3	540	490	24.72727273	28	28
nexp.50.20.1.4	540	490	33.75	37	37
nexp.50.20.1.5	540	490	18.6	30	30
nexp.50.20.2.1	540	735	22.05555556	37	37
nexp.50.20.2.2	540	735	37.6	44	44
nexp.50.20.2.3	540	735	32.18181818	41	41
nexp.50.20.2.4	540	735	33.2	42	42
nexp.50.20.2.5	540	735	50	54	54
nexp.50.20.4.1	540	1225	37.45454546	61	61
nexp.50.20.4.2	540	1225	49.09090909	71	71
nexp.50.20.4.3	540	1225	42.36363636	70	70
nexp.50.20.4.4	540	1225	41.53030303	64	64
nexp.50.20.4.5	540	1225	44.63636364	67	67
nexp.50.20.8.1	540	2205	7.257731959	90	90
nexp.50.20.8.2	540	2205	8.384879725	104	104
nexp.50.20.8.3	540	2205	6.130584192	83	83
nexp.50.20.8.4	540	2205	6.955326461	91	91
nexp.50.20.8.5	540	2205	7.972508591	94	94
sndlib					
atlanta-DBM	269	660	39014475.95	46244642.4	46244642.4
atlanta-UUM	232	618	80132352.9	86492550.3	86492550.3
cost266-DBE	1540	4275	16058601.95	19247538.53	19787597.22
cost266-DBM	1483	4275	16058601.95	18346585.66	18928995.52
cost266-UUE	1446	4161	20161515.66	24446908.15	25148940.56
cost266-UUM	1389	4161	20086679.37	22718580	23574240

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
dfn-bwin-DBE	235	3285	17890.9462	53983.92272	82322.82
dfn-bwin-UUE	180	3196	28912.17693	70174.56047	92143.71
dfn-gwin-DBE	261	1031	15018.81029	23572	23572
dfn-gwin-DBM	216	1031	15018.81029	22660	22660
dfn-gwin-UUE	203	938	27467.25724	38904	38904
dfn-gwin-UUM	158	938	27467.25724	38752	38752
di-yuan-DBE	214	855	274606.25	553700	553700
di-yuan-UUE	161	774	316143.75	656600	656600
france-DBM	715	2205	10964	12400	12400
france-UUM	645	2115	18878	20200	20200
germany50-DBM	2526	8189	438028	461662.2689	474700
germany50-UUM	2088	6971	597932.5	617513.2	628490
giul39-DDE	1865	7052	1706.732812	2050.952458	2761
janos-us-DDM	760	2184	1488134.75	1490871.1	1493112
janos-us-ca-DDM	1643	4758	1488681.897	1490273.263	1492416
newyork-DBE	403	1568	41753.8	824721.6604	999702
newyork-DBM	354	1568	41753.8	846739.7	999702
newyork-UUE	338	1470	74943.8	864347.1473	999702
newyork-UUM	289	1470	74943.8	852034.6	999702
nobel-eu-DBE	879	3771	570687.5	601505.1037	610910
nobel-eu-UUE	838	3771	817152.5	840162.4697	863900
nobel-ger-DBE	333	1771	115406	126070.7711	129940
nobel-ger-UUE	307	1770	147488	156094.3439	162420
norway-DBE	882	2754	84836.5925	331034.6295	383890
norway-DBM	831	2754	84836.5925	333307.2	383890
norway-UUE	804	2652	162715.395	412130.1817	452680
norway-UUM	753	2652	162715.395	415215.0786	452680
pdh-DBE	212	703	4489313.45	9689062	9689062
pdh-DBM	178	703	4489313.45	9615107	9615107
pdh-UUE	145	527	4593661.173	11114202	11114202
pdh-UUM	111	527	4593661.173	10903843	10903843
pioro40-DBM	1738	6942	254029.2926	254632.7885	255454
pioro40-UUM	1649	6942	412076.045	412774.4549	414045
polska-DBM	168	396	14948.55627	15717	15717
polska-UUM	150	396	22633.7508	23619	23619
sun-DDM	264	695	684.6945	859.3029916	897.72
ta1-DBE	621	2606	2344400.755	5896728	5896728
ta1-DBM	566	2606	2344400.755	4686829.552	5896728
ta1-UUE	494	2288	3693674.557	6855895	7533070
ta1-UUM	439	2288	3693674.557	5331466	7621520
ta2-DBE	3055	10101	36065210.45	36471255.99	36471255.99
ta2-DBM	2947	10101	36065210.45	36471255.99	36471255.99
ta2-UUE	2687	9241	36964014.23	37871728.59	37871728.59
ta2-UUM	2579	9241	36964014.18	37534290	37871728.59
zib54-DBE	2430	6748	2224025.844	9773826.66	9773826.66
zib54-UUE	1809	5150	4081019.858	10334015.82	10334015.82
ufcn					
beasleyC1	1750	2500	52	85	85
beasleyC2	1750	2500	47.33333333	144	144
beasleyC3	1750	2500	237.9878049	653.7467057	779
beavma	372	390	293174.2758	383284.9997	383284.9997
berlin	2704	5304	130.0666667	1008.759977	1044
brasil	3364	6612	2208.708333	12976.96639	13680
fixnet6	600	1000	3192.042	3983	3983
g150x1100	1250	2200	34562.12	67196.46414	71827
g150x1650	1800	3300	31884.515	63313.75911	69130

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
g180x666	846	1332	496101.2425	624632	624632
g200x740	940	1480	34077.54267	43807.57406	44316
g200x740b	940	1480	145993.88	178286.1284	179279
g200x740c	940	1480	655765.895	680124	680124
g200x740d	940	1480	548918.9429	586038	586038
g200x740e	940	1480	559469.2194	600396	600396
g200x740f	940	1480	572132.2166	617872	617872
g200x740g	940	1480	7911.513	36249.13905	45308
g200x740h	940	1480	86698.8429	125673.348	131639
g200x740i	940	1480	2292.465	25884.70859	31175
g40x132	172	264	13997.266	26629	26629
g50x170	220	340	10711.592	25576	25576
g55x188	243	376	9178.605	24487	24487
g55x188c	243	376	22549.394	35464	35464
h50x2450	2549	4900	11147.73151	32906.88083	32906.88083
h50x2450b	2549	4900	2822.584647	3030.198086	3030.198086
h50x2450c	2549	4900	3400.372829	4896.196619	4896.196619
h50x2450d	2549	4900	3251.877445	4639.264786	4639.264786
h50x2450e	2549	4900	2999.379052	4077.684199	4077.684199
h80x6320b	6558	12640	5086.547046	6003.179413	6003.179413
h80x6320c	6558	12640	5461.33783	6273.627361	6273.627361
h80x6320d	6558	12640	5325.160104	6382.09905	6382.09905
k10x90	100	180	399.73	568	568
k14x182	196	364	3995.514	8491	8491
k14x182b	196	364	4442.844	11042	11042
k15x210	225	420	1982.74	16128	16128
k15x420	435	840	350.08	819	819
k15x630	645	1260	449.89	936	936
k16x240	256	480	2769.838	10072.41885	10674
k16x240b	256	480	3320.771	11146.48643	11393
k20x380	400	760	952.566	1941	1941
k20x380b	400	760	1764.92	11343	11343
k20x380c	400	760	1858.48	17159	17159
k20x380d	400	760	1937.75	20979	20979
k20x380e	400	760	2013.35	6904	6904
l121x232	353	464	169104.5134	192446	192446
l451x885	1336	1770	399355.0014	431050	431050
l451x885b	1336	1770	530266.6196	560847	560847
l61x114	175	228	56275.99471	60085	60085
mc11	1920	3040	608.8443396	9711.1359	12008
mc7	1920	3040	367.7573964	3167.928932	3696
mc8	1920	3040	91.76470588	1284.278026	1615
mtest4ma	1174	1950	33968.06567	52148	52148
p100x588	688	1176	3957.217778	8711.17038	8999
p100x588b	688	1176	5554.011111	43614.81892	48793
p100x588c	688	1176	97579.11063	172770	172770
p100x588d	688	1176	1.899352402	5	5
p200x1188	1388	2376	5575.126667	11109.11095	11396
p200x1188b	1388	2376	7860.811111	50629.67929	56403
p200x1188c	1388	2376	5678.607089	15078	15078
p500x2988	3488	5976	59130.6729	71280.72127	71917
p500x2988b	3488	5976	61188.729	160303.2449	179975
p500x2988c	3488	5976	14122.96285	15215	15215
p500x2988d	3488	5976	2.161832061	6	6
p50x288	338	576	4178.478	6134	6134
p50x288b	338	576	5818.04	21753	21753
p50x576	626	1152	12203.142	19407	19407

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
p50x864	914	1728	11284.682	19007	19007
p80x400	480	800	4824.65	8458.861661	8548
p80x400b	480	800	6418.8	36136.51828	39915
r20x100	120	200	6747.217	15603	15603
r20x200	220	400	5088.257	14783	14783
r30x160	190	320	10608.074	21827	21827
r50x360	410	720	575.02	1653	1653
r80x800	880	1600	3651.48	5150.009169	5338
sp100x200	300	400	18517.54178	34507	34507
sp150x300	450	600	13333.96896	30918	30918
sp150x300b	450	600	38.09703259	56	56
sp150x300c	450	600	406383.7402	560735.9965	560735.9965
sp150x300d	450	600	34.1089901	68.99999927	68.99999927
sp50x100	150	200	49287.62252	50968	50968
sp80x160	240	320	14573.39492	19549	19549
sp90x180	270	360	58915.83556	68862	68862
sp90x250	340	500	18733.09473	23571	23571
miplib					
10teams	230	2025	917	924	924
a1c1s1	3312	3648	2364.279909	11503	11503
aflow30a	479	842	983.1674253	1158	1158
aflow40b	1442	2728	1005.664817	1168	1168
air03	124	10757	338864.25	340160	340160
air04	823	8904	55535.43639	56137	56137
air05	426	7195	25877.60927	26374	26374
arki001	1048	1388	7579621.831	7580813.046	7580813.046
atlanta-ip	21732	48738	81.24319887	90.00987861	90.00987861
bell3a	123	133	869515.1309	878430.316	878430.316
bell5	91	104	8912505.574	8966406.491	8966406.491
blend2	274	353	6.915675114	7.598985	7.598985
cap6000	2176	6000	-2451537.325	-2451377	-2451377
dano3mip	3202	13873	576.2316203	577.2959336	716.6428571
danoint	664	521	62.63728042	65.66666667	65.66666667
dcmulti	290	548	184569.1643	188182	188182
disctom	399	10000	-5000	-5000	-5000
ds	656	67732	57.23456605	59.3658192	116.59
dsbmip	1182	1886	-305.198175	-305.198175	-305.198175
egout	98	141	511.4151251	568.1007	568.1007
enigma	21	100	0	0	0
fast0507	507	63009	172.1455667	174	174
fiber	363	1298	198107.3575	405935.18	405935.18
fixnet6	478	878	3192.042	3983	3983
flugpl	18	18	1167875.166	1201500	1201500
gen	780	870	112273.8902	112313.3627	112313.3627
gesa2	1392	1224	25492512.14	25779856.37	25779856.37
gesa2-o	1248	1224	25489759.78	25779856.37	25779856.37
gesa3	1368	1152	27846437.46	27991042.65	27991042.65
gesa3_o	1224	1152	27844600.02	27991042.65	27991042.65
glass4	396	322	800002400	1200012600	1200012600
gt2	29	188	20146.7613	21166	21166
harp2	112	2993	-74325169.34	-73899813	-73899813
khb05250	101	1350	95919464	106940226	106940226
l152lav	97	1989	4656.363636	4722	4722
liu	2178	1156	560	560	1138
lseu	28	89	947.9572368	1120	1120
manna81	6480	3321	-13297	-13164	-13164

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
markshare1	6	62	0	1	1
markshare2	7	74	0	1	1
mas74	13	151	10482.79528	11801.18573	11801.18573
mas76	12	151	38893.90364	40005.05414	40005.05414
misc03	96	160	1910	3360	3360
misc06	820	1808	12841.68939	12850.86074	12850.86074
misc07	212	260	1415	2810	2810
mitre	2054	10724	114782.4674	115155	115155
mkc	3411	5325	-605.25	-566.33	-557.206
mod008	6	319	290.9310727	307	307
mod010	146	2655	6532.083333	6548	6548
mod011	4480	10958	-62081950.29	-54558535.01	-54558535.01
modglob	291	422	20430947.62	20740508.09	20740508.09
momentum1	42680	5174	79192.66721	109143.4935	109143.4935
momentum2	24237	3732	10696.11156	12314.21959	12314.21959
momentum3	56822	13532	94175.457	95161.32147	236426.335
msc98-ip	15850	21143	19530897.71	19839497.01	19839497.01
mzzv11	9499	10240	-22944.98755	-21718	-21718
mzzv42z	10460	11717	-21622.99848	-20540	-20540
net12	14021	14115	68.39787582	214	214
noswot	182	128	-43	-41	-41
nsrand-ipx	735	6621	49667.89226	51200	51200
nw04	36	87482	16310.66667	16862	16862
opt1217	64	769	-20	-16	-16
p0033	16	33	2838.546739	3089	3089
p0201	133	201	7125	7615	7615
p0282	241	282	246740.1476	258411	258411
p0548	176	548	7740.671835	8691	8691
p2756	755	2756	2704.482763	3124	3124
pk1	45	86	0	11	11
pp08a	136	240	2748.345238	7350	7350
pp08aCUTS	246	240	5480.606156	7350	7350
protfold	2112	1835	-41.95744681	-31	-31
qiu	1192	840	-931.6388569	-132.873137	-132.873137
qnet1	503	1541	14274.10267	16029.69268	16029.69268
qnet1_o	456	1541	12557.24792	16029.69268	16029.69268
rd-rplusc-21	125899	622	100	165395.2753	165395.2753
rentacar	6803	9557	28928379.62	30356760.98	30356760.98
rgn	24	180	48.79999856	82.19999765	82.19999765
roll3000	2295	1166	11099.05045	12890	12890
rout	291	556	981.8642857	1077.56	1077.56
set1ch	492	712	35118.10985	54537.75	54537.75
seymour	4944	1372	403.8464741	414.4202538	423
sp97ar	1761	14101	652560391.1	659537423	661670441.4
stein27	118	27	13	18	18
stein45	331	45	22	30	30
stp3d	159488	204880	481.8777862	482.4321875	500.736
swath	884	6805	334.4968581	467.407491	467.407491
t1717	551	73885	134531.0214	135582.8176	170195.1
timtab1	171	397	157896.0366	764772	764772
timtab2	294	675	210652.4709	669812.8095	1145245
tr12-30	750	1080	21260.24478	130596	130596
vpm1	234	378	16.76666667	20	20
vpm2	234	378	11.17074055	13.75	13.75
mittelmann					
30:70:4_5:0_5:100	12050	10772	8.1	9	9

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
30:70:4_5:0_95:100	12526	10976	3	3	3
30:70:4_5:0_95:98	12471	10990	11.5	12	12
acc-1	2286	1620	0	0	0
acc-2	2520	1620	0	0	0
acc-3	3249	1620	0	0	0
acc-4	3285	1620	0	0	0
acc-5	3052	1339	0	0	0
acc-6	3047	1335	0	0	0
bc1	1913	1751	2.146801502	3.338362548	3.338362548
bienst1	576	505	11.72413793	46.75	46.75
bienst2	576	505	11.72413793	54.6	54.6
binkar10_1	1026	2298	6637.188027	6742.200024	6742.200024
dano3_3	3202	13873	576.2316203	576.344633	576.344633
dano3_4	3202	13873	576.2316203	576.4352247	576.4352247
dano3_5	3202	13873	576.2316203	576.924916	576.924916
lrn	8491	7253	44246903.22	44482699.34	44482699.34
markshare_4_0	4	34	0	1	1
markshare_5_0	5	45	0	0	2
mik.250-20-75.1	195	270	-59156.75737	-49716	-49716
mik.250-20-75.2	195	270	-59987.19637	-50768	-50768
mik.250-20-75.3	195	270	-60670.36664	-52242	-52242
mik.250-20-75.4	195	270	-61651.2271	-52301	-52301
mik.250-20-75.5	195	270	-60527.43721	-51532	-51532
neos1	5020	2112	15.5	19	19
neos10	46793	23489	-1196.333333	-1135	-1135
neos11	2706	1220	6	9	9
neos12	8317	3983	9.411612426	13	13
neos13	20852	1827	-126.1783778	-95.47480656	-95.47480656
neos14	552	792	32734.11478	74333.34334	74333.34334
neos17	486	535	0.0006814985015	0.1500025774	0.1500025774
neos2	1103	2101	-4407.097239	454.864697	454.864697
neos20	2446	1165	-475	-434	-434
neos21	1085	614	2.216483516	7	7
neos22	5208	3240	777191.4286	779715	779715
neos23	1568	477	56	137	137
neos3	1442	2747	-6158.209105	368.842751	368.842751
neos4	38577	22884	-4.860344075e+10	-4.860344075e+10	-4.860344075e+10
neos5	63	63	13	15	15
neos6	1036	8786	83	83	83
neos648910	1491	814	16	32	32
neos7	1994	1556	562977.4297	721934	721934
neos8	46324	23228	-3725	-3719	-3719
neos808444	18329	19846	0	0	0
neos818918	2450	2750	1680	1700	1700
neos823206	709	1830	19.81417814	83.86019578	83.86019578
neos897005	11612	44630	14	14	14
neos9	31600	81408	780	798	798
ns1648184	806	705	-1260.954861	-1234.666667	-1231.31746
ns1671066	316	2840	7.634607843	7.634607843	7.634607843
ns1688347	4191	2685	15.11176471	27	27
ns1692855	4562	3047	15.11176471	26	30
nug08	912	1632	203.5	214	214
prod1	208	250	-84.41587189	-56	-56
prod2	211	301	-86.98076893	-62	-62
qap10	1820	4150	332.5662276	340	340
seymour1	4944	1372	403.8464741	410.7637014	410.7637014
swath2	884	6805	334.4968581	385.1996929	385.1996929

Table 8: continued on next page

problem	rows	vars	lp	bestdual	bestprimal
swath3	884	6805	334.4968581	397.7613437	397.7613437

Table 8: General information for all instances – Scip runs

B Results – Network Detection

Table 9 presents the results of the network detection for those instances for which the original network is known. It reports on the number of nodes ($|V|$), the number of arcs ($|A|$), and the number of commodities ($|K|$) in the original network (*original*). For the detected networks with the presolving of Scip switched off respectively on (*detection – no presolve* respectively *detection – presolve*) the network inconsistency ($\Psi(G)$) and the difference in the network size (nodes $|V|$, arcs $|A|$, commodities $|K|$) compared with the original network is provided. If no network has been found the corresponding entries are marked with '-'. For every testset we present the arithmetic mean of the (absolute) difference of the original and the detected network elements. The detection results are summarized in Table 2.

problem	original			detection – no presolve				detection – presolve			
	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
arc.set											
ns25-pr12	27	72	81	0.000	0	0	0	0.000	-7	-14	0
ns25-pr3	29	122	70	0.006	0	0	+1	0.007	-3	-6	+1
ns25-pr4	18	58	58	0.009	0	0	+1	0.008	-4	-8	+1
ns25-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
ns25-pr9	24	84	87	0.007	0	0	+1	0.006	-3	-6	+1
ns4-pr12	27	72	81	0.000	0	0	0	0.006	-7	-14	+1
ns4-pr3	29	122	70	0.007	0	0	+1	0.007	-3	-6	+1
ns4-pr4	18	58	58	0.000	0	0	0	0.008	-4	-8	+1
ns4-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
ns4-pr9	24	84	87	0.006	0	0	+1	0.007	-3	-6	+1
ns60-pr12	27	72	81	0.000	0	0	0	0.000	-7	-14	0
ns60-pr3	29	122	70	0.006	0	0	+1	0.007	-3	-6	+1
ns60-pr4	18	58	58	0.000	0	0	0	0.008	-4	-8	+1
ns60-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
ns60-pr9	24	84	87	0.007	0	0	+1	0.007	-3	-6	+1
nu120-pr12	27	72	81	0.000	0	0	0	0.006	-7	-14	+1
nu120-pr3	29	122	70	0.007	0	0	+1	0.006	-3	-6	+1
nu120-pr4	18	58	58	0.009	0	0	+1	0.008	-4	-8	+1
nu120-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
nu120-pr9	24	84	87	0.007	0	0	+1	0.006	-3	-6	+1
nu25-pr12	27	72	81	0.000	0	0	0	0.000	-7	-14	0
nu25-pr3	29	122	70	0.007	0	0	+1	0.006	-3	-6	+1
nu25-pr4	18	58	58	0.008	0	0	+1	0.008	-4	-8	+1
nu25-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
nu25-pr9	24	84	87	0.007	0	0	+1	0.006	-3	-6	+1
nu4-pr12	27	72	81	0.000	0	0	0	0.005	-7	-14	+1
nu4-pr3	29	122	70	0.007	0	0	+1	0.007	-3	-6	+1
nu4-pr4	18	58	58	0.000	0	0	0	0.000	-4	-8	0
nu4-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
nu4-pr9	24	84	87	0.006	0	0	+1	0.007	-3	-6	+1
nu60-pr12	27	72	81	0.000	0	0	0	0.000	-7	-14	0
nu60-pr3	29	122	70	0.007	0	0	+1	0.006	-3	-6	+1
nu60-pr4	18	58	58	0.000	0	0	0	0.000	-4	-8	0
nu60-pr6	27	74	93	0.000	0	0	0	0.000	-8	-16	0
nu60-pr9	24	84	87	0.006	0	0	+1	0.006	-3	-6	+1

Table 9: continued on next page

problem	original			detection – no presolve				detection – presolve			
	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
mean diff in %					**0.0**	**0.0**	**0.7**		**20.1**	**13.4**	**0.9**
fc											
fc.30.50.1	30	217	1	0.000	0	0	0	0.000	-6	-16	0
fc.30.50.10	30	217	1	0.000	0	0	0	0.000	-8	-27	0
fc.30.50.2	30	217	1	0.000	0	0	0	0.000	-4	-19	0
fc.30.50.3	30	217	1	0.000	0	0	0	0.000	-6	-15	0
fc.30.50.4	30	217	1	0.000	0	0	0	0.000	-9	-23	0
fc.30.50.5	30	217	1	0.000	0	0	0	0.002	-9	-44	0
fc.30.50.6	30	217	1	0.000	0	0	0	0.000	-3	-8	0
fc.30.50.7	30	217	1	0.000	0	0	0	0.000	-9	-28	0
fc.30.50.8	30	217	1	0.000	0	0	0	0.000	-6	-16	0
fc.30.50.9	30	217	1	0.000	0	0	0	0.000	-8	-24	0
fc.60.20.1	60	354	1	0.000	0	0	0	0.000	-14	-39	0
fc.60.20.10	60	354	1	0.000	0	0	0	0.000	-13	-32	0
fc.60.20.2	60	354	1	0.000	0	0	0	0.000	-13	-38	0
fc.60.20.3	60	354	1	0.000	0	0	0	0.000	-18	-43	0
fc.60.20.4	60	354	1	0.000	0	0	0	0.000	-16	-49	0
fc.60.20.5	60	354	1	0.000	0	0	0	0.001	-17	-56	0
fc.60.20.6	60	354	1	0.000	0	0	0	0.000	-18	-78	0
fc.60.20.7	60	354	1	0.000	0	0	0	0.000	-7	-24	0
fc.60.20.8	60	354	1	0.000	0	0	0	0.000	-8	-30	0
fc.60.20.9	60	354	1	0.000	0	0	0	0.000	-20	-67	0
mean diff in %					**0.0**	**0.0**	**0.0**		**23.3**	**11.5**	**0.0**
fctp											
bal8x12	20	96	1	0.000	0	0	0	0.000	0	0	0
bk4x3	7	12	1	-	-	-	-	-	-	-	-
gr4x6	10	24	1	-	-	-	-	-	-	-	-
n3700	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3701	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3702	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3703	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3704	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3705	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3706	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3707	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3708	150	5000	1	0.000	0	0	0	0.000	0	0	0
n3709	150	5000	1	0.000	0	0	0	0.000	0	0	0
n370a	150	5000	1	0.000	0	0	0	0.000	0	0	0
n370b	150	5000	1	0.000	0	0	0	0.000	0	0	0
n370c	150	5000	1	0.000	0	0	0	0.000	0	0	0
n370d	150	5000	1	0.000	0	0	0	0.000	0	0	0
n370e	150	5000	1	0.000	0	0	0	0.000	0	0	0
ran10x10a	20	100	1	0.000	0	0	0	0.000	0	0	0
ran10x10b	20	100	1	0.000	0	0	0	0.000	0	0	0
ran10x10c	20	100	1	0.000	0	0	0	0.000	0	0	0
ran10x12	22	120	1	0.000	0	0	0	0.000	0	0	0
ran10x26	36	260	1	0.000	0	0	0	0.000	0	0	0
ran12x12	24	144	1	0.000	0	0	0	0.000	0	0	0
ran12x21	33	252	1	0.000	0	0	0	0.000	0	0	0
ran13x13	26	169	1	0.000	0	0	0	0.000	0	0	0
ran14x18	32	252	1	0.000	0	0	0	0.000	0	0	0
ran16x16	32	256	1	0.000	0	0	0	0.000	0	0	0
ran17x17	34	289	1	0.000	0	0	0	0.000	0	0	0
ran4x64	68	256	1	0.000	-63	-252	+2	0.000	-63	-252	+2
ran6x43	49	258	1	0.000	0	0	0	0.000	0	0	0

Table 9: continued on next page

problem	original			detection – no presolve				detection – presolve			
	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$	$\Psi(G)$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$	$\Psi(G)$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$
ran8x32	40	256	1	0.000	0	0	0	0.000	0	0	0
mean diff in %					**3.1**	**3.3**	**6.7**		**3.1**	**3.3**	**6.7**
avub											
nexp.100.20.1.1	100	990	1	0.000	0	0	0	0.000	-61	-388	0
nexp.100.20.1.2	100	990	1	0.000	0	0	0	0.000	-59	-356	0
nexp.100.20.1.3	100	990	1	0.000	0	0	0	0.000	-66	-457	0
nexp.100.20.1.4	100	990	1	0.000	0	0	0	0.000	-64	-447	0
nexp.100.20.1.5	100	990	1	0.000	0	0	0	0.000	-65	-431	0
nexp.100.20.2.1	100	990	1	0.000	0	0	0	0.000	-61	-388	0
nexp.100.20.2.2	100	990	1	0.000	0	0	0	0.000	-54	-325	0
nexp.100.20.2.3	100	990	1	0.000	0	0	0	0.000	-63	-414	0
nexp.100.20.2.4	100	990	1	0.000	0	0	0	0.000	-51	-313	0
nexp.100.20.2.5	100	990	1	0.000	0	0	0	0.000	-58	-351	0
nexp.100.20.4.1	100	990	1	0.000	0	0	0	0.000	0	-102	0
nexp.100.20.4.2	100	990	1	0.000	0	0	0	0.000	0	-107	0
nexp.100.20.4.3	100	990	1	0.000	0	0	0	0.000	0	-92	0
nexp.100.20.4.4	100	990	1	0.000	0	0	0	0.000	0	-85	0
nexp.100.20.4.5	100	990	1	0.000	0	0	0	0.000	0	-133	0
nexp.100.20.8.1	100	990	1	0.000	0	0	0	0.000	0	-102	0
nexp.100.20.8.2	100	990	1	0.000	0	0	0	0.000	0	-107	0
nexp.100.20.8.3	100	990	1	0.000	0	0	0	0.000	0	-119	0
nexp.100.20.8.4	100	990	1	0.000	0	0	0	0.000	0	-85	0
nexp.100.20.8.5	100	990	1	0.000	0	0	0	0.000	0	-133	0
nexp.150.20.1.1	150	2235	1	0.000	0	0	0	0.000	-91	-852	0
nexp.150.20.1.2	150	2235	1	0.000	0	0	0	0.000	-87	-806	0
nexp.150.20.1.3	150	2235	1	0.000	0	0	0	0.000	-88	-776	0
nexp.150.20.1.4	150	2235	1	0.000	0	0	0	0.000	-95	-907	0
nexp.150.20.1.5	150	2235	1	0.000	0	0	0	0.000	-88	-786	0
nexp.150.20.2.1	150	2235	1	0.000	0	0	0	0.000	-91	-842	0
nexp.150.20.2.2	150	2235	1	0.000	0	0	0	0.000	-81	-673	0
nexp.150.20.2.3	150	2235	1	0.000	0	0	0	0.000	-87	-806	0
nexp.150.20.2.4	150	2235	1	0.000	0	0	0	0.000	-95	-906	0
nexp.150.20.2.5	150	2235	1	0.000	0	0	0	0.000	-88	-776	0
nexp.150.20.4.1	150	2235	1	0.000	0	0	0	0.000	0	-220	0
nexp.150.20.4.2	150	2235	1	0.000	0	0	0	0.000	0	-190	0
nexp.150.20.4.3	150	2235	1	0.000	0	0	0	0.000	0	-212	0
nexp.150.20.4.4	150	2235	1	0.000	0	0	0	0.000	0	-176	0
nexp.150.20.4.5	150	2235	1	0.000	0	0	0	0.000	0	-193	0
nexp.150.20.8.1	150	2235	1	0.000	0	0	0	0.000	0	-220	0
nexp.150.20.8.2	150	2235	1	0.000	0	0	0	0.000	0	-177	0
nexp.150.20.8.3	150	2235	1	0.000	0	0	0	0.000	0	-176	0
nexp.150.20.8.4	150	2235	1	0.000	0	0	0	0.000	0	-154	0
nexp.150.20.8.5	150	2235	1	0.000	0	0	0	0.000	0	-190	0
nexp.50.20.1.1	50	245	1	0.000	0	0	0	0.000	-8	-23	0
nexp.50.20.1.2	50	245	1	0.000	-2	-16	0	0.000	-26	-82	0
nexp.50.20.1.3	50	245	1	0.000	-2	-16	0	0.000	-26	-89	0
nexp.50.20.1.4	50	245	1	0.000	-1	-5	0	0.000	-24	-78	0
nexp.50.20.1.5	50	245	1	0.000	0	0	0	0.000	-15	-39	0
nexp.50.20.2.1	50	245	1	0.000	0	0	0	0.000	-11	-25	0
nexp.50.20.2.2	50	245	1	0.000	0	0	0	0.000	-19	-80	0
nexp.50.20.2.3	50	245	1	0.000	0	0	0	0.000	-16	-39	0
nexp.50.20.2.4	50	245	1	0.000	0	0	0	0.000	-25	-78	0
nexp.50.20.2.5	50	245	1	0.000	0	0	0	0.002	-28	-95	0
nexp.50.20.4.1	50	245	1	0.000	0	0	0	0.000	0	-19	0
nexp.50.20.4.2	50	245	1	0.000	-1	-5	0	0.000	-1	-23	0

Table 9: continued on next page

problem	original			detection – no presolve				detection – presolve			
	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$	$\Psi(G)$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$	$\Psi(G)$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$
nexp.50.20.4.3	50	245	1	0.000	0	0	0	0.000	0	-32	0
nexp.50.20.4.4	50	245	1	0.000	-1	-2	0	0.000	-4	-26	0
nexp.50.20.4.5	50	245	1	0.000	0	0	0	0.000	-1	-25	0
nexp.50.20.8.1	50	245	1	0.000	-1	-4	0	0.000	-2	-27	0
nexp.50.20.8.2	50	245	1	0.000	0	0	0	0.000	0	-22	0
nexp.50.20.8.3	50	245	1	0.000	-1	-2	0	0.000	-2	-25	0
nexp.50.20.8.4	50	245	1	0.000	0	0	0	0.000	-1	-29	0
nexp.50.20.8.5	50	245	1	0.000	-1	-2	0	0.000	-1	-23	0
mean diff in %					**0.3**	**0.4**	**0.0**		**26.9**	**21.8**	**0.0**
sndlib											
atlanta-DBM	15	44	15	0.000	0	0	0	0.010	0	0	0
atlanta-UUM	15	22	14	0.000	0	0	0	0.023	0	0	0
cost266-DBE	37	114	37	0.000	0	0	0	0.003	0	0	0
cost266-DBM	37	114	37	0.000	0	0	0	0.003	0	0	0
cost266-UUE	37	57	36	0.000	0	0	0	0.004	0	0	0
cost266-UUM	37	57	36	0.000	0	0	0	0.008	0	0	0
dfn-bwin-DBE	10	90	10	0.000	0	0	0	0.000	0	0	0
dfn-bwin-UUE	10	45	9	0.000	0	0	0	0.000	0	0	0
dfn-gwin-DBE	11	94	11	0.000	0	0	0	0.002	0	0	0
dfn-gwin-DBM	11	94	11	0.000	0	0	0	0.002	0	0	0
dfn-gwin-UUE	11	47	10	0.000	0	0	0	0.002	0	0	0
dfn-gwin-UUM	11	47	10	0.000	0	0	0	0.002	0	0	0
di-yuan-DBE	11	84	8	0.000	0	0	0	0.000	0	0	0
di-yuan-UUE	11	42	7	0.000	0	0	0	0.000	0	0	0
france-DBM	25	90	25	0.000	0	0	0	0.008	0	0	0
france-UUM	25	45	24	0.000	0	0	0	0.012	0	0	0
germany50-DBM	50	176	47	0.000	0	0	0	0.002	0	0	0
germany50-UUM	50	88	40	0.000	0	0	0	0.003	0	0	0
giul39-DDE	39	172	39	0.000	0	0	0	0.000	0	0	0
janos-us-DDM	30	84	26	0.000	-4	0	0	0.004	-4	0	0
janos-us-ca-DDM	39	122	39	0.000	0	0	0	0.004	0	0	0
newyork-DBE	16	98	16	0.000	0	0	0	0.001	0	0	0
newyork-DBM	16	98	16	0.000	0	0	0	0.001	0	0	0
newyork-UUE	16	49	15	0.000	0	0	0	0.001	0	0	0
newyork-UUM	16	49	15	0.000	0	0	0	0.001	0	0	0
nobel-eu-DBE	28	82	27	0.000	0	0	0	0.006	0	0	0
nobel-eu-UUE	28	41	27	0.000	0	0	0	0.009	0	0	0
nobel-ger-DBE	17	52	15	0.000	0	0	0	0.012	0	0	0
nobel-ger-UUE	17	26	15	0.000	0	0	0	0.015	0	0	0
norway-DBE	27	102	27	0.000	0	0	0	0.001	0	0	0
norway-DBM	27	102	27	0.000	0	0	0	0.001	0	0	0
norway-UUE	27	51	26	0.000	0	0	0	0.002	0	0	0
norway-UUM	27	51	26	0.000	0	0	0	0.003	0	0	0
pdh-DBE	11	68	10	0.000	0	0	0	0.000	0	0	0
pdh-DBM	11	68	10	0.000	0	0	0	0.000	0	0	0
pdh-UUE	11	34	7	0.000	0	0	0	0.000	0	0	0
pdh-UUM	11	34	7	0.000	0	0	0	0.000	0	0	0
pioro40-DBM	40	178	39	0.000	0	0	0	0.000	0	0	0
pioro40-UUM	40	89	39	0.000	0	0	0	0.000	0	0	0
polska-DBM	12	36	11	0.000	0	0	0	0.008	0	0	0
polska-UUM	12	18	11	0.000	0	0	0	0.010	0	0	0
sun-DDM	27	102	6	0.000	0	0	0	0.003	0	0	0
ta1-DBE	24	110	19	0.000	0	0	0	0.002	0	0	0
ta1-DBM	24	110	19	0.000	0	0	0	0.002	0	0	0
ta1-UUE	24	55	16	0.000	0	0	0	0.003	0	0	0

Table 9: continued on next page

problem	original			detection – no presolve				detection – presolve			
	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
ta1-UUM	24	55	16	0.000	0	0	0	0.003	0	0	0
ta2-DBE	65	216	42	0.000	0	0	0	0.002	-1	-2	0
ta2-DBM	65	216	42	0.000	0	0	0	0.003	-1	-2	0
ta2-UUE	65	108	38	0.000	0	0	0	0.003	-1	-1	0
ta2-UUM	65	108	38	0.000	0	0	0	0.005	-1	-1	0
zib54-DBE	54	162	42	0.000	0	0	0	0.005	-1	-2	0
zib54-UUE	54	81	32	0.000	0	0	0	0.006	-1	-1	0
mean diff in %					**0.3**	**0.0**	**0.0**		**0.4**	**0.1**	**0.0**
ufcn											
beasleyC1	500	1250	1	0.000	0	0	0	0.000	-193	-386	0
beasleyC2	500	1250	1	0.000	0	0	0	0.000	-190	-380	0
beasleyC3	500	1250	1	0.000	-1	-2	0	0.000	-199	-398	0
beavma	89	195	1	0.000	0	0	0	0.000	-11	-15	0
berlin	52	2652	1	0.000	0	0	0	0.000	0	0	0
brasil	58	3306	1	0.000	0	0	0	0.000	0	0	0
fixnet6	100	500	1	0.000	0	0	0	0.000	-1	-51	0
g150x1100	150	1100	1	0.000	0	0	0	0.000	0	0	0
g150x1650	150	1650	1	0.000	0	0	0	0.000	0	0	0
g180x666	180	666	1	0.000	0	0	0	0.000	0	0	0
g200x740	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740b	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740c	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740d	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740e	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740f	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740g	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740h	200	740	1	0.000	0	0	0	0.000	0	0	0
g200x740i	200	740	1	0.000	0	0	0	0.000	0	0	0
g40x132	40	132	1	0.000	0	0	0	0.000	0	0	0
g50x170	50	170	1	0.000	0	0	0	0.000	0	0	0
g55x188	55	188	1	0.000	0	0	0	0.000	0	0	0
g55x188c	55	188	1	0.000	0	0	0	0.000	0	0	0
h50x2450	50	2450	1	0.000	0	-49	0	0.000	0	-49	0
h50x2450b	50	2450	1	0.000	0	-49	0	0.000	0	-49	0
h50x2450c	50	2450	1	0.000	0	-49	0	0.000	0	-49	0
h50x2450d	50	2450	1	0.000	0	-49	0	0.000	0	-49	0
h50x2450e	50	2450	1	0.000	0	-49	0	0.000	0	-49	0
h80x6320b	80	6320	1	0.000	0	-79	0	0.000	0	-79	0
h80x6320c	80	6320	1	0.000	0	-79	0	0.000	0	-79	0
h80x6320d	80	6320	1	0.000	0	-79	0	0.000	0	-79	0
k10x90	10	90	1	0.000	0	0	0	0.000	0	0	0
k14x182	14	182	1	0.000	0	0	0	0.000	0	0	0
k14x182b	14	182	1	0.000	0	0	0	0.000	0	0	0
k15x210	15	210	1	0.000	0	0	0	0.000	0	0	0
k15x420	15	420	1	0.000	0	0	0	0.000	0	0	0
k15x630	15	630	1	0.000	0	0	0	0.000	0	0	0
k16x240	16	240	1	0.000	0	0	0	0.000	0	0	0
k16x240b	16	240	1	0.000	0	0	0	0.000	0	0	0
k20x380	20	380	1	0.000	0	0	0	0.000	0	0	0
k20x380b	20	380	1	0.000	0	0	0	0.000	0	0	0
k20x380c	20	380	1	0.000	0	0	0	0.000	0	0	0
k20x380d	20	380	1	0.000	0	0	0	0.000	0	0	0
k20x380e	20	380	1	0.000	0	0	0	0.000	0	0	0
l121x232	121	232	1	0.000	0	0	0	0.006	-6	-36	0
l451x885	451	885	1	0.000	0	0	0	0.004	-9	-129	0

Table 9: continued on next page

problem	original			detection – no presolve				detection – presolve			
	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
l451x885b	451	885	1	0.000	0	-1	0	0.004	-9	-134	0
l61x114	61	114	1	0.000	0	-1	0	0.009	-5	-20	0
mc11	400	1520	1	0.000	0	0	0	0.000	0	0	0
mc7	400	1520	1	0.000	0	0	0	0.000	0	0	0
mc8	400	1520	1	0.000	0	0	0	0.000	0	0	0
mtest4ma	100	975	1	0.000	0	0	0	0.000	0	0	0
p100x588	100	588	1	0.000	0	0	0	0.000	0	0	0
p100x588b	100	588	1	0.000	0	0	0	0.000	0	0	0
p100x588c	100	588	1	0.000	0	0	0	0.000	0	0	0
p100x588d	100	588	1	0.000	0	0	0	0.000	0	0	0
p200x1188	200	1188	1	0.000	0	0	0	0.000	0	0	0
p200x1188b	200	1188	1	0.000	0	0	0	0.000	0	0	0
p200x1188c	200	1188	1	0.000	0	0	0	0.000	0	0	0
p500x2988	500	2988	1	0.000	0	0	0	0.000	0	0	0
p500x2988b	500	2988	1	0.000	0	0	0	0.000	0	0	0
p500x2988c	500	2988	1	0.000	0	0	0	0.000	0	0	0
p500x2988d	500	2988	1	0.000	0	0	0	0.000	0	0	0
p50x288	50	288	1	0.000	0	0	0	0.000	0	0	0
p50x288b	50	288	1	0.000	0	0	0	0.000	0	0	0
p50x576	50	576	1	0.000	0	0	0	0.000	0	0	0
p50x864	50	864	1	0.000	0	0	0	0.000	0	0	0
p80x400	80	400	1	0.000	0	0	0	0.000	-2	-4	0
p80x400b	80	400	1	0.000	0	0	0	0.000	-2	-4	0
r20x100	20	100	1	0.000	0	0	0	0.000	-1	-2	0
r20x200	20	200	1	0.000	0	0	0	0.000	0	0	0
r30x160	30	160	1	0.000	0	0	0	0.000	-4	-8	0
r50x360	50	360	1	0.000	0	0	0	0.000	-1	-2	0
r80x800	80	800	1	0.000	0	0	0	0.000	0	0	0
sp100x200	100	200	1	0.000	-60	-124	0	0.000	-76	-136	0
sp150x300	150	300	1	0.000	-52	-105	0	0.000	-89	-143	0
sp150x300b	150	300	1	0.000	-23	-53	0	0.000	-79	-110	0
sp150x300c	150	300	1	0.000	-13	-30	0	0.000	-68	-85	0
sp150x300d	150	300	1	0.000	0	-4	0	0.000	-86	-86	0
sp50x100	50	100	1	0.018	-39	-82	0	0.000	-47	-94	0
sp80x160	80	160	1	0.000	-19	-39	0	0.000	-47	-68	0
sp90x180	90	180	1	0.000	-17	-41	0	0.000	-65	-110	0
sp90x250	90	250	1	0.000	-71	-205	0	0.000	-79	-210	0
mean diff in %					**3.8**	**4.2**	**0.0**		**9.3**	**8.3**	**0.0**

Table 9: Network detection results with Scip– Instances with known original network

Table 10 presents the results of the network detection for those instances for which no original network is known. It reports on the number of nodes ($|V|$), the number of arcs ($|A|$), and the number of commodities ($|K|$) in the detected networks in case the presolving of Scip is switched off respectively on (*detection – no presolve* respectively *detection – presolve*). In addition, the network inconsistency ($\Psi(G)$) of the detected networks is provided. If no network has been found the corresponding entries are marked with '-'. The detection results are summarized in Table 2.

problem	detection – no presolve				detection – presolve			
	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
cut.set								
n1-3	0.239	51	228	2	0.350	9	16	1
n10-3	0.248	61	311	4	0.327	5	8	1

Table 10: continued on next page

problem	detection – no presolve				detection – presolve			
	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
n11-3	0.258	48	197	5	0.302	7	12	1
n12-3	0.258	96	108	1	0.305	14	26	1
n13-3	0.352	55	54	1	0.291	7	12	1
n14-3	0.259	74	353	4	0.268	51	54	1
n15-3	0.346	168	326	1	0.346	168	326	1
n2-3	0.244	74	79	1	0.246	5	8	1
n3-3	0.403	17	30	1	0.318	73	82	1
n4-3	0.267	73	86	1	0.239	83	668	3
n5-3	0.268	52	57	1	0.299	11	20	1
n6-3	0.243	141	2007	6	0.307	61	109	1
n7-3	0.257	98	103	1	0.329	74	75	1
n8-3	0.280	46	49	1	0.326	9	15	1
n9-3	0.279	86	89	1	0.366	17	32	1
miplib								
10teams	-	-	-	-	-	-	-	-
a1c1s1	0.019	340	648	6	0.045	126	191	4
aflow30a	0.000	29	421	1	0.000	29	421	1
aflow40b	0.000	39	1364	1	0.000	39	1364	1
air03	-	-	-	-	-	-	-	-
air04	0.625	4	9	1	0.646	7	12	1
air05	0.668	6	14	1	0.635	3	5	1
arki001	0.000	11	36	1	0.000	11	36	1
atlanta-ip	0.518	16	46	1	0.512	149	200	1
bell3a	0.000	19	38	1	0.018	8	16	2
bell5	0.016	12	26	2	0.000	3	9	1
blend2	-	-	-	-	-	-	-	-
cap6000	-	-	-	-	-	-	-	-
dano3mip	0.214	550	1274	1	0.081	78	612	1
danoint	0.000	72	464	1	0.000	55	404	1
dcmulti	-	-	-	-	-	-	-	-
disctom	-	-	-	-	-	-	-	-
ds	-	-	-	-	-	-	-	-
dsbmip	-	-	-	-	-	-	-	-
egout	0.016	11	15	1	0.000	8	21	1
enigma	-	-	-	-	-	-	-	-
fast0507	-	-	-	-	-	-	-	-
fiber	0.015	28	44	11	0.005	23	38	11
fixnet6	0.000	100	448	1	0.000	99	449	1
flugpl	-	-	-	-	-	-	-	-
gen	0.000	52	55	5	0.486	16	18	2
gesa2	0.000	24	24	13	0.000	24	24	13
gesa2-o	0.000	24	24	1	0.000	24	24	1
gesa3	0.093	36	70	12	0.093	35	70	11
gesa3_o	0.020	26	28	1	0.022	25	50	1
glass4	-	-	-	-	-	-	-	-
gt2	-	-	-	-	-	-	-	-
harp2	-	-	-	-	-	-	-	-
khb05250	0.000	76	100	1	0.000	76	100	1
l152lav	-	-	-	-	-	-	-	-
liu	-	-	-	-	-	-	-	-
lseu	-	-	-	-	-	-	-	-
manna81	0.190	46	51	7	0.190	46	51	7
markshare1	-	-	-	-	-	-	-	-
markshare2	-	-	-	-	-	-	-	-
mas74	-	-	-	-	-	-	-	-

Table 10: continued on next page

problem	detection – no presolve				detection – presolve			
	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
mas76	-	-	-	-	-	-	-	-
misc03	-	-	-	-	0.417	5	4	1
misc06	0.041	8	14	1	0.099	6	6	1
misc07	-	-	-	-	0.438	6	5	1
mitre	-	-	-	-	-	-	-	-
mkc	0.095	319	319	1	0.271	7	6	24
mod008	-	-	-	-	-	-	-	-
mod010	-	-	-	-	-	-	-	-
mod011	-	-	-	-	0.445	19	24	3
modglob	-	-	-	-	-	-	-	-
momentum1	0.021	74	184	1	0.026	311	658	1
momentum2	0.003	287	1375	1	0.002	285	1193	1
momentum3	0.000	502	4519	1	0.005	898	4035	1
msc98-ip	0.529	186	345	1	0.535	143	303	1
mzzv11	0.669	6	7	1	0.645	10	23	1
mzzv42z	0.661	10	17	2	0.656	7	11	2
net12	0.209	12	17	2	0.395	20	32	1
noswot	0.250	6	5	1	0.250	6	5	1
nsrand-ipx	-	-	-	-	0.322	11	17	5
nw04	-	-	-	-	-	-	-	-
opt1217	-	-	-	-	-	-	-	-
p0033	-	-	-	-	-	-	-	-
p0201	-	-	-	-	0.561	4	3	2
p0282	-	-	-	-	0.089	8	9	5
p0548	-	-	-	-	0.137	4	6	3
p2756	-	-	-	-	0.195	11	14	1
pk1	-	-	-	-	-	-	-	-
pp08a	0.000	72	162	1	0.000	69	154	1
pp08aCUTS	0.095	72	132	1	0.091	69	121	1
protfold	0.479	142	368	1	0.499	140	263	1
qiu	-	-	-	-	-	-	-	-
qnet1	0.478	13	49	5	0.313	8	28	6
qnet1_o	0.213	36	103	24	0.000	10	37	24
rd-rplusc-21	-	-	-	-	0.000	18	19	1
rentacar	0.240	206	414	1	0.405	15	44	1
rgn	-	-	-	-	-	-	-	-
roll3000	0.474	33	82	1	0.457	63	129	1
rout	0.344	30	240	1	0.344	30	240	1
set1ch	0.000	251	451	1	0.000	191	385	1
seymour	0.445	58	91	1	0.450	111	201	1
sp97ar	0.219	16	43	3	0.233	17	50	4
stein27	-	-	-	-	-	-	-	-
stein45	0.481	19	26	1	0.481	19	26	1
stp3d	0.408	26645	99201	1	0.004	2352	10915	2
swath	-	-	-	-	-	-	-	-
t1717	-	-	-	-	-	-	-	-
timtab1	-	-	-	-	0.000	3	5	2
timtab2	-	-	-	-	0.000	4	7	2
tr12-30	0.000	24	47	12	0.017	20	39	12
vpm1	0.000	8	7	4	0.000	5	8	4
vpm2	-	-	-	-	-	-	-	-
mittelmann								
30:70:4_5:0_5:100	0.404	91	104	1	0.416	176	201	1
30:70:4_5:0_95:100	0.424	108	128	1	0.421	108	129	1
30:70:4_5:0_95:98	0.418	50	62	1	0.411	58	69	1

Table 10: continued on next page

problem	detection – no presolve				detection – presolve			
	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$	$\Psi(G)$	$\|V\|$	$\|A\|$	$\|K\|$
acc-1	0.514	293	764	1	0.515	155	673	1
acc-2	0.517	116	422	1	0.514	155	673	1
acc-3	0.373	149	168	1	0.407	175	220	1
acc-4	0.374	169	204	1	0.387	180	225	1
acc-5	0.488	250	334	1	0.469	286	440	1
acc-6	0.466	238	355	1	0.473	287	470	1
bc1	-	-	-	-	-	-	-	-
bienst1	0.022	64	347	1	0.000	51	386	1
bienst2	0.022	64	347	1	0.000	51	386	1
binkar10_1	0.249	21	20	1	0.500	14	13	1
dano3_3	0.214	550	1278	1	0.081	78	612	1
dano3_4	0.214	550	1268	1	0.081	78	612	1
dano3_5	0.214	550	1261	1	0.081	78	612	1
lrn	0.298	41	93	1	0.280	30	81	1
markshare_4_0	-	-	-	-	-	-	-	-
markshare_5_0	-	-	-	-	-	-	-	-
mik.250-20-75.1	-	-	-	-	-	-	-	-
mik.250-20-75.2	-	-	-	-	-	-	-	-
mik.250-20-75.3	-	-	-	-	-	-	-	-
mik.250-20-75.4	-	-	-	-	-	-	-	-
mik.250-20-75.5	-	-	-	-	-	-	-	-
neos1	0.243	82	216	23	0.267	42	312	8
neos10	0.375	17	54	1	0.221	8	53	2
neos11	0.481	165	372	1	0.410	105	528	1
neos12	0.440	596	1531	1	0.415	254	2081	1
neos13	-	-	-	-	-	-	-	-
neos14	0.138	147	233	1	0.189	176	214	1
neos17	-	-	-	-	-	-	-	-
neos2	0.476	5	82	2	0.479	3	31	1
neos20	0.307	28	37	1	0.394	31	44	1
neos21	0.165	61	87	1	0.031	27	44	9
neos22	0.193	20	34	1	0.210	30	33	18
neos23	0.298	7	16	2	0.124	23	35	3
neos3	0.476	5	82	2	0.478	4	83	2
neos4	-	-	-	-	0.054	267	971	1
neos5	0.404	91	104	1	-	-	-	-
neos6	-	-	-	-	-	-	-	-
neos648910	0.058	3	3	1	0.000	7	7	1
neos7	0.007	38	141	1	0.007	38	141	1
neos8	0.488	23	56	1	0.451	16	44	1
neos808444	0.557	220	337	5	0.465	289	517	5
neos818918	0.712	6	5	1	0.304	70	110	5
neos823206	0.232	30	63	55	0.000	28	29	1
neos897005	0.391	14	13	1	0.460	485	535	1
neos9	0.596	55	1302	1	0.621	46	1217	1
ns1648184	0.055	30	244	2	0.055	32	244	2
ns1671066	0.000	19	20	1	0.000	18	19	1
ns1688347	0.366	156	320	1	0.491	304	968	1
ns1692855	0.366	156	313	1	0.449	164	327	1
nug08	-	-	-	-	-	-	-	-
prod1	-	-	-	-	-	-	-	-
prod2	-	-	-	-	-	-	-	-
qap10	-	-	-	-	-	-	-	-
seymour1	0.451	36	59	1	0.459	60	102	1
swath2	-	-	-	-	-	-	-	-
swath3	-	-	-	-	-	-	-	-

Table 10: continued on next page

problem	detection – no presolve				detection – presolve			
	$\Psi(G)$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$	$\Psi(G)$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$

Table 10: Network detection results with Scip– Instances with unknown original network

C Results – Separation

Table 11 presents the results of the separation for all easy instances. We compare the performance of Scip with (*mcf*) and without (*nomcf*) the Mcf separator. Every easy instance can be solved by Scip in one of the two settings within one hour of CPU time. We provide the closed gap at the root node before branching (*rootgap*) which is defined as

$$100 \cdot (root - lp)/(bestprimal - lp),$$

where *lp* denotes the value of the LP relaxation, *bestprimal* the best known primal solution value (see Table 8 in Appendix A), and *root* the value of the LP at the root node after cutting before branching. In addition, the time in seconds and the number of branch-and-bound nodes used to solve the instance are provided. Note that if an easy problem hits the time limit we use the number of nodes explored so far. If it hits the memory limit before the time limit has been reached we set $time = 3600s$ and scale the nodes accordingly. The number of cuts (*# cuts*) found by the Mcf separator is given in the last column. Values in bold face indicate that they are at least 10% below the opposite value (*time* and *nodes*) or at least 1 percentage point above the opposite value (*rootgap*). If the Mcf separator is switched off (no network has been found or the network inconsistency ratio $\Psi(G)$ is above $\Psi^{\max} = 0.02$) then column *#cuts* contains a "-". Runs marked with "*" have hit the memory limit. For every testset two additional rows provide the arithmetic mean of the *rootgap* and the geometric means of CPU time and nodes (*means*) as wells as the number of values in bold face for every measure (*wins*).

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
arc.set							
ns25-pr12	63.8	13.9	2940	**78.5**	**2.0**	**219**	34
ns25-pr3	45.7	2112.5	159702	**48.8**	**1827.0**	**134719**	27
ns25-pr4	43.9	**11.8**	**2742**	**59.0**	14.5	3078	29
ns25-pr6	74.8	4.3	783	**78.7**	**2.5**	**222**	26
ns25-pr9	10.1	**192.9**	**17314**	10.0	286.0	21740	42
ns4-pr12	67.7	**12.9**	**3151**	67.7	16.4	3925	17
ns4-pr4	9.3	9.2	3512	**68.1**	**4.4**	**1706**	6
ns4-pr6	42.1	121.0	57036	**60.6**	**7.6**	**1531**	10
ns60-pr12	72.7	3.3	409	**86.2**	**2.6**	**46**	87
ns60-pr3	52.4	1975.3	81589	**53.5**	**1347.6**	**49704**	62
ns60-pr4	48.8	4.9	254	**80.8**	5.2	**152**	52
ns60-pr6	61.7	6.7	1190	**84.2**	**1.8**	**27**	65
ns60-pr9	42.7	478.2	27341	43.1	**405.0**	25251	82
nu120-pr12	67.1	11.2	316	**100.0**	**3.9**	**2**	609
nu120-pr4	38.8	104.0	12628	**49.4**	**71.9**	**7959**	262
nu120-pr6	95.5	6.5	39	**100.0**	**3.9**	**1**	412
nu25-pr12	82.5	6.8	72	82.6	**5.6**	**54**	100
nu25-pr4	58.1	64.8	8877	**74.2**	**20.4**	**2484**	81
nu25-pr6	**87.8**	**4.8**	**61**	82.7	5.6	116	160
nu4-pr12	72.1	51.7	13798	**79.4**	**16.4**	**1988**	18
nu4-pr4	65.8	35.8	9526	**78.7**	**9.2**	**1116**	20

Table 11: continued on next page

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
nu4-pr6	82.0	31.6	4257	**87.5**	**9.1**	**572**	14
nu60-pr12	71.9	20.3	1699	**94.8**	**6.3**	**14**	531
nu60-pr4	45.5	38.5	3560	**61.7**	**28.8**	**3193**	224
nu60-pr6	58.1	34.1	4416	**74.1**	**7.8**	**233**	370
means	**58.4**	**31.7**	**3326**	**71.4**	**16.6**	**1103**	
wins	**1**	**4**	**4**	**20**	**20**	**20**	
cut.set							
n1-3	100.0	0.5	3	100.0	0.5	3	-
n10-3	97.9	0.3	16	97.9	0.3	16	-
n11-3	92.9	1.0	10	92.9	1.0	10	-
n13-3	92.2	11.8	2067	92.2	11.8	2067	-
n14-3	89.0	4.1	353	89.0	4.1	353	-
n2-3	96.7	1.1	21	96.7	1.1	21	-
n4-3	78.6	464.4	28958	78.6	464.7	28958	-
n5-3	74.8	97.5	9971	74.8	97.8	9971	-
n6-3	89.6	3568.6	246628	89.6	3569.5	246628	-
n7-3	83.6	52.0	1865	83.6	51.7	1865	-
n8-3	80.7	9.6	965	80.7	9.5	965	-
means	**88.7**	**16.6**	**1232**	**88.7**	**16.5**	**1232**	
wins	**0**	**0**	**0**	**0**	**0**	**0**	
fc							
fc.30.50.1	88.6	**1.3**	**19**	**89.7**	1.5	33	685
fc.30.50.10	100.0	**0.9**	6	100.0	1.0	**4**	649
fc.30.50.2	94.2	**2.0**	15	93.4	2.2	**11**	537
fc.30.50.3	84.6	2.3	815	**88.5**	2.3	**137**	554
fc.30.50.4	97.1	3.2	953	97.5	**2.4**	**495**	549
fc.30.50.5	100.0	**1.2**	5	100.0	1.6	**3**	668
fc.30.50.6	88.8	**1.9**	129	89.7	2.4	125	479
fc.30.50.7	100.0	**1.0**	6	100.0	1.9	**5**	700
fc.30.50.8	89.0	2.5	471	**93.2**	2.5	**23**	289
fc.30.50.9	98.2	3.1	278	98.3	3.0	**90**	773
fc.60.20.1	91.7	9.2	1994	92.2	8.3	2028	337
fc.60.20.10	96.1	6.9	1283	96.9	6.5	**497**	268
fc.60.20.2	88.1	**8.4**	2897	**89.7**	10.4	**1988**	465
fc.60.20.3	92.3	5.8	961	92.8	6.1	961	352
fc.60.20.4	91.8	10.7	8367	91.2	10.3	8260	405
fc.60.20.5	98.6	**1.2**	12	**100.0**	1.6	**3**	292
fc.60.20.6	89.2	9.7	3764	88.6	**8.1**	3578	494
fc.60.20.7	89.1	5.9	1571	89.1	5.9	1571	0
fc.60.20.8	93.9	**3.9**	**288**	93.1	5.1	651	499
fc.60.20.9	100.0	**1.5**	4	100.0	2.1	4	301
means	**93.6**	**3.3**	**415**	**94.2**	**3.5**	**305**	
wins	**0**	**10**	**2**	**5**	**2**	**11**	
fctp							
bal8x12	**100.0**	**0.2**	**1**	97.8	0.4	4	216
bk4x3	100.0	0.0	1	100.0	0.0	1	-
gr4x6	100.0	0.0	1	100.0	0.0	1	-
ran10x10a	94.6	0.5	15	94.2	0.6	**9**	372
ran10x10b	84.4	**0.2**	**15**	**85.5**	0.3	21	527
ran10x10c	70.3	1.9	2550	**72.5**	**1.5**	**1411**	561
ran10x12	100.0	**0.3**	1	100.0	0.4	1	425
ran10x26	55.6	28.4	31469	55.3	26.8	31563	844
ran12x12	68.0	**13.6**	**19096**	68.3	16.9	26827	632

Table 11: continued on next page

problem	nomcf rootgap closed %	nomcf time	nomcf nodes	mcf rootgap closed %	mcf time	mcf nodes	mcf #cuts
ran12x21	**60.1**	**53.2**	**61542**	58.8	71.7	88530	744
ran13x13	60.7	32.1	50831	**65.3**	**26.9**	**41173**	522
ran16x16	61.3	296.4	**394888**	**62.3**	313.3	440247	680
ran17x17	**76.3**	6.7	4697	73.2	**5.4**	**2958**	575
ran4x64	69.8	1.2	231	69.8	1.3	231	0
ran6x43	56.6	1.0	110	**60.4**	0.9	**82**	442
ran8x32	72.0	10.1	10999	72.8	9.4	**9248**	745
means	**76.9**	**4.5**	**1679**	**77.3**	**4.6**	**1603**	
wins	**3**	**5**	**5**	**5**	**3**	**6**	
avub							
nexp.100.20.1.1	100.0	0.8	1	100.0	**0.6**	1	55
nexp.100.20.1.2	100.0	**2.4**	1	100.0	2.9	1	210
nexp.100.20.1.3	100.0	1.6	1	100.0	1.7	1	35
nexp.100.20.1.4	86.6	14.2	933	**100.0**	**3.9**	**1**	170
nexp.100.20.1.5	100.0	2.3	1	100.0	2.1	1	102
nexp.100.20.2.1	89.3	**7.2**	**10**	89.2	24.3	1817	119
nexp.100.20.2.3	100.0	4.2	**1**	100.0	4.4	2	94
nexp.100.20.2.4	75.4	2295.2	634375	**86.3**	**37.4**	**8277**	171
nexp.100.20.2.5	71.0	3600.0	1302612	**80.7**	**87.1**	**14457**	225
nexp.100.20.4.1	63.2	3600.0	325144	**94.6**	**37.2**	**14341**	641
nexp.100.20.4.2	67.3	3600.0	178770	**96.1**	**28.4**	**4158**	620
nexp.100.20.4.3	73.3	3600.0	**361786**	**95.2**	**2546.1**	531289	516
nexp.100.20.4.4	76.7	3600.0	112480	**96.9**	**38.7**	**2434**	698
nexp.100.20.4.5	82.6	3600.0	310722	**99.2**	**15.2**	**762**	518
nexp.150.20.1.1	100.0	**4.9**	1	100.0	6.1	1	67
nexp.150.20.1.2	100.0	**3.1**	1	100.0	3.7	1	172
nexp.150.20.1.3	**100.0**	**7.3**	**1**	96.8	24.5	201	214
nexp.150.20.1.4	**86.3**	**422.8**	**13165**	84.7	1454.5	43179	321
nexp.150.20.1.5	100.0	5.3	1	100.0	5.6	1	267
nexp.150.20.2.1	69.4	3600.0	302168	**80.8**	**589.9**	**42504**	56
nexp.150.20.2.3	67.7	3600.0	354263	**91.5**	**885.4**	**162779**	268
nexp.150.20.2.5	70.7	3600.0	228516	**76.0**	**1305.2**	**78584**	30
nexp.150.20.4.2	71.9	3600.0	**31754**	**95.5**	**2143.0**	85008	854
nexp.150.20.4.3	71.1	3600.0	40544	**97.6**	**1151.2**	39474	597
nexp.150.20.4.4	75.5	3600.0	46633	**97.3**	**440.6**	**15564**	854
nexp.50.20.1.1	100.0	0.2	1	100.0	0.2	1	142
nexp.50.20.1.2	100.0	0.1	1	100.0	0.1	1	76
nexp.50.20.1.3	100.0	0.1	1	100.0	0.1	1	95
nexp.50.20.1.4	100.0	0.0	1	100.0	0.0	1	35
nexp.50.20.1.5	100.0	0.5	1	100.0	0.6	1	0
nexp.50.20.2.1	100.0	1.2	11	100.0	**0.9**	11	214
nexp.50.20.2.2	100.0	**0.4**	**1**	100.0	0.6	4	68
nexp.50.20.2.3	100.0	1.8	13	100.0	**0.9**	**1**	129
nexp.50.20.2.4	75.4	3.1	1454	**80.9**	3.1	**1303**	164
nexp.50.20.2.5	100.0	0.4	1	100.0	0.5	1	62
nexp.50.20.4.1	82.6	16.4	8906	**97.3**	**2.4**	**6**	223
nexp.50.20.4.2	75.2	20.3	**7767**	75.3	19.9	9209	6
nexp.50.20.4.3	86.7	15.6	7242	**90.0**	**11.4**	**3778**	174
nexp.50.20.4.4	85.9	8.7	2710	**87.6**	**6.0**	**902**	64
nexp.50.20.4.5	97.4	2.4	8	**100.0**	**0.7**	**1**	131
nexp.50.20.8.1	77.5	585.8	325852	**83.4**	**273.4**	**156790**	172
nexp.50.20.8.2	78.2	3600.0⋆	2533846⋆	**94.9**	**60.4**	**23889**	582
nexp.50.20.8.3	82.8	829.5	602812	**94.3**	**8.2**	**1799**	496
nexp.50.20.8.4	83.2	3600.0	2656207	**93.7**	**26.5**	**10430**	703

Table 11: continued on next page

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
nexp.50.20.8.5	**85.1**	**192.1**	**99668**	83.4	458.7	218807	122
means	**86.8**	**55.2**	**4267**	**94.2**	**17.8**	**1396**	
wins	**3**	**8**	**9**	**23**	**25**	**21**	
sndlib							
atlanta-DBM	49.4	6.3	3474	**71.6**	**1.9**	**829**	34
atlanta-UUM	26.3	6.4	5556	26.3	6.5	5556	-
dfn-gwin-DBE	60.7	82.4	20260	**67.3**	**61.4**	**10855**	216
dfn-gwin-DBM	66.7	56.6	16285	**69.4**	**44.9**	**8923**	74
dfn-gwin-UUE	64.7	76.1	40495	**66.8**	**61.6**	**24978**	146
dfn-gwin-UUM	64.5	287.5	214582	**66.5**	**107.3**	**60790**	54
di-yuan-DBE	51.5	62.0	19753	**57.6**	**43.2**	**9449**	283
di-yuan-UUE	64.8	168.0	109972	**69.4**	**62.3**	**24589**	215
france-DBM	44.3	407.4	132148	**54.9**	**166.5**	**42159**	64
france-UUM	31.1	130.9	69384	**64.0**	**15.6**	**5123**	28
pdh-DBE	56.1	49.6	22184	**59.3**	51.8	21442	287
pdh-DBM	56.9	60.3	26828	**62.4**	**41.6**	**16895**	157
pdh-UUE	71.7	23.7	12629	**75.6**	**16.1**	**6649**	304
pdh-UUM	74.4	16.0	10149	**76.0**	**11.4**	**5289**	60
polska-DBM	53.0	**0.8**	**659**	**56.4**	1.1	942	16
polska-UUM	35.8	7.4	13897	**49.2**	**3.5**	**5799**	16
ta1-DBE	27.2	3600.0*	1608597*	**87.6**	**812.7**	**488521**	86
ta2-DBE	21.1	14.9	250	**100.0**	**2.6**	**1**	32
ta2-DBM	0.0	47.5	1769	**27.5**	**16.4**	**384**	2
ta2-UUE	0.0	3600.0*	**181005***	**47.4**	**2620.2**	207059	16
zib54-DBE	72.1	**3238.2**	153710	**80.2**	3600.0	**121784**	661
zib54-UUE	57.2	1599.0	105420	**73.7**	**547.6**	**27183**	434
means	**47.7**	**84.7**	**24197**	**64.1**	**45.1**	**10710**	
wins	**0**	**2**	**2**	**21**	**18**	**18**	
ufcn							
beasleyC1	74.5	**1.9**	**17**	**75.5**	3.0	58	9
beasleyC2	**73.4**	**241.5**	**103685**	70.1	429.3	205234	29
beavma	88.4	0.9	31	**100.0**	**0.6**	**7**	270
fixnet6	82.0	1.4	7	82.0	1.4	7	2
g180x666	67.2	3600.0	1407689	**96.3**	**226.1**	**97553**	523
g200x740c	77.1	15.0	7493	**79.8**	**8.8**	**2283**	109
g200x740d	68.2	3600.0	2038110	**85.9**	**242.4**	**116280**	378
g200x740e	62.4	3600.0	1783097	**89.5**	**1479.1**	**752118**	487
g200x740f	62.2	3600.0	1838978	**93.8**	**40.4**	**17293**	609
g40x132	88.7	6.1	9379	**90.0**	**4.8**	**7634**	121
g50x170	88.8	**26.4**	**40826**	88.7	78.6	121031	140
g55x188c	92.8	1.2	31	**100.0**	**0.7**	**2**	109
h50x2450	88.2	3600.0	512965	**94.0**	**646.9**	**62232**	293
h50x2450b	87.8	36.9	1942	**100.0**	**3.6**	**1**	361
h50x2450c	87.0	1888.2	241900	**95.0**	**23.4**	**402**	284
h50x2450d	88.0	3600.0	628139	**96.0**	**22.9**	**300**	367
h50x2450e	93.9	19.2	50	**95.6**	18.3	**7**	389
h80x6320b	93.6	38.0	518	93.6	35.9	**273**	25
h80x6320c	87.8	314.2	45656	87.9	**208.7**	**28161**	23
h80x6320d	92.2	100.1	6345	92.3	**66.0**	**3083**	15
k10x90	90.5	0.5	**42**	**91.9**	0.5	81	83
k14x182	**83.6**	**2.3**	**779**	82.2	4.0	3345	70
k14x182b	**94.7**	**1.0**	**15**	89.5	1.5	111	94
k15x210	93.9	0.8	64	**100.0**	**0.7**	**1**	82
k15x420	92.7	1.6	9	**100.0**	**1.1**	**2**	144

Table 11: continued on next page

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
k15x630	87.2	2.9	49	**99.7**	**1.9**	**3**	185
k20x380	81.8	73.9	120588	**83.5**	**56.5**	**75872**	103
k20x380b	100.0	0.4	1	100.0	**0.2**	1	64
k20x380c	99.9	1.0	4	99.9	**0.8**	**3**	73
k20x380d	100.0	0.8	**2**	100.0	0.8	3	76
k20x380e	100.0	**0.2**	1	100.0	0.4	1	32
l121x232	**93.8**	**0.4**	**8**	89.5	0.5	13	25
l451x885	**58.2**	**785.8**	**644282**	56.6	950.4	744828	35
l451x885b	70.5	11.4	6032	69.8	11.4	6363	34
l61x114	100.0	0.1	1	100.0	0.1	1	36
mtest4ma	85.8	3600.0	1034729	**95.4**	**11.1**	**337**	318
p100x588c	58.5	**6.0**	**1302**	59.4	7.4	2302	11
p100x588d	100.0	0.2	1	100.0	**0.1**	1	19
p200x1188c	48.4	2005.1	918761	**51.3**	**999.6**	**505917**	10
p500x2988c	96.5	3.4	5	**100.0**	3.4	**1**	12
p500x2988d	100.0	0.6	1	100.0	**0.3**	1	16
p50x288	94.2	1.2	169	**95.4**	1.3	**87**	82
p50x288b	88.2	391.3	459786	**89.5**	**150.2**	**149012**	112
p50x576	95.1	1.4	**57**	95.6	**1.2**	99	98
p50x864	96.0	2.7	229	**100.0**	**1.4**	**1**	149
r20x100	87.8	2.8	2567	86.9	2.8	2350	79
r20x200	79.7	**566.9**	**1328162**	80.0	963.7	2253459	90
r30x160	81.8	112.0	287070	**84.1**	**47.3**	**104204**	75
r50x360	83.5	3600.0	2286022	**93.0**	**1001.2**	**593978**	186
sp100x200	86.2	**1.4**	**702**	85.8	1.9	1411	241
sp150x300	93.9	0.8	69	94.3	0.8	**53**	94
sp150x300b	62.7	501.5	1044093	**65.2**	**262.4**	**580764**	49
sp150x300c	77.6	2.5	1336	**84.4**	**1.4**	**677**	75
sp150x300d	77.7	3600.0	10056270	**78.7**	**1616.0**	**3977407**	56
sp50x100	100.0	0.0	1	100.0	0.0	1	6
sp80x160	100.0	0.1	1	100.0	0.0	1	48
sp90x180	100.0	0.2	2	100.0	0.3	2	267
sp90x250	88.7	0.1	7	**96.2**	0.1	**4**	77
means	**85.7**	**22.1**	**3984**	**89.7**	**11.6**	**1804**	
wins	**5**	**11**	**13**	**31**	**32**	**34**	
miplib							
10teams	100.0	6.1	154	100.0	6.1	154	-
aflow30a	**65.1**	10.0	1768	63.7	10.2	**1348**	967
aflow40b	52.3	2732.6	465560	**53.8**	**1432.7**	**250510**	1392
air03	100.0	28.1	4	100.0	28.0	4	-
air04	18.8	50.3	100	18.8	50.2	100	-
air05	19.5	22.7	280	19.5	22.6	280	-
arki001	24.9	1842.0	912993	24.9	1834.5	912993	0
bell3a	48.0	12.5	48539	48.0	12.3	48539	0
bell5	**90.7**	0.8	2003	80.8	**0.4**	**1140**	8
blend2	18.8	0.3	177	18.8	0.3	177	-
cap6000	28.1	2.8	2949	28.1	2.8	2949	-
dcmulti	92.3	0.7	54	92.3	0.7	54	-
disctom	100.0	1.9	1	100.0	1.9	1	-
dsbmip	100.0	0.1	1	100.0	0.1	1	-
egout	100.0	0.0	1	100.0	0.0	1	328
enigma	100.0	0.2	356	100.0	0.2	356	-
fast0507	0.0	663.5	1998	0.0	662.2	1998	-
fiber	91.8	**0.7**	**24**	**94.2**	0.9	37	585

Table 11: continued on next page

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
fixnet6	**83.0**	1.4	11	75.8	1.4	11	2
flugpl	10.6	0.0	80	10.6	0.0	80	-
gen	100.0	0.1	1	100.0	0.1	1	-
gesa2	99.8	1.1	7	99.8	1.1	7	45
gesa2-o	100.0	1.1	10	100.0	1.1	10	0
gesa3	79.1	1.0	16	79.1	1.0	16	-
gesa3_o	85.8	1.2	10	85.8	1.2	10	-
gt2	100.0	0.1	1	100.0	0.1	1	-
khb05250	99.9	0.5	10	99.9	0.6	**8**	3
l152lav	0.2	1.8	29	0.2	1.7	29	-
lseu	56.5	0.2	407	56.5	0.2	407	-
manna81	100.0	0.6	2	100.0	0.6	2	-
mas74	1.5	756.0	3070052	1.5	754.7	3070052	-
mas76	5.9	64.2	329883	5.9	64.1	329883	-
misc03	18.9	0.8	176	18.9	0.8	176	-
misc06	59.1	0.3	7	59.1	0.3	7	-
misc07	0.7	20.9	30984	0.7	20.7	30984	-
mitre	100.0	9.9	1	100.0	9.6	1	-
mod008	47.2	0.2	90	47.2	0.2	90	-
mod010	100.0	1.0	2	100.0	1.0	2	-
mod011	70.6	75.5	2642	70.6	75.1	2642	-
modglob	96.3	0.5	66	96.3	0.5	66	-
mzzv11	82.8	375.7	3166	82.8	373.3	3166	-
mzzv42z	80.3	196.0	251	80.3	195.9	251	-
net12	11.4	897.1	5075	11.4	897.0	5075	-
noswot	0.0	379.6	1072776	0.0	376.0	1072776	-
nw04	0.5	45.9	132	0.5	45.0	132	-
opt1217	100.0	0.3	1	100.0	0.4	1	-
p0033	100.0	0.0	2	100.0	0.0	2	-
p0201	26.9	0.6	113	26.9	0.6	113	-
p0282	100.0	0.4	7	100.0	0.4	7	-
p0548	99.4	0.2	8	99.4	0.2	8	-
p2756	98.8	1.3	21	98.8	1.3	21	-
pk1	0.0	70.8	227351	0.0	70.5	227351	-
pp08a	97.0	1.0	629	97.0	**0.8**	**196**	84
pp08aCUTS	93.3	1.0	91	93.3	1.0	91	-
qiu	0.0	72.7	12812	0.0	72.4	12812	-
qnet1	80.3	2.3	71	80.3	2.2	71	-
qnet1_o	90.2	**1.4**	27	**91.6**	2.0	**24**	130
rentacar	49.1	2.7	15	49.1	2.7	15	-
rgn	98.8	0.2	34	98.8	0.2	34	-
rout	0.3	31.9	29025	0.3	31.4	29025	-
set1ch	99.9	0.4	**9**	99.9	0.5	16	318
stein27	0.0	0.8	4175	0.0	0.8	4175	-
stein45	0.0	19.0	52415	0.0	18.9	52415	-
timtab1	55.8	566.9	925498	55.8	566.6	925498	2
tr12-30	99.6	3600.0	1358717	99.6	**2873.6**	**1090119**	32
vpm1	100.0	0.1	1	100.0	0.1	1	0
vpm2	69.5	1.1	994	69.5	1.1	994	-
means	**62.7**	**7.0**	**816**	**62.5**	**6.9**	**784**	
wins	**3**	**2**	**2**	**3**	**4**	**7**	
mittelmann							
30:70:4_5:0_5:100	90.7	182.4	146	90.7	183.7	146	-
30:70:4_5:0_95:100	100.0	147.7	166	100.0	148.5	166	-

Table 11: continued on next page

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
30:70:4_5:0_95:98	100.0	120.4	228	100.0	121.1	228	-
acc-1	100.0	36.6	61	100.0	36.7	61	-
acc-2	100.0	57.0	84	100.0	57.1	84	-
acc-3	100.0	229.5	348	100.0	229.9	348	-
acc-4	100.0	1274.0	2102	100.0	1276.0	2102	-
acc-5	100.0	299.5	941	100.0	300.1	941	-
acc-6	100.0	140.0	593	100.0	140.2	593	-
bc1	36.2	183.6	5459	36.2	185.2	5459	-
bienst1	**63.9**	21.7	17068	57.1	**16.9**	**12309**	169
bienst2	51.3	**109.9**	**89778**	50.4	164.8	106597	279
binkar10_1	63.9	274.9	157069	63.9	275.5	157069	-
dano3_3	36.4	54.8	8	36.4	55.0	8	-
dano3_4	29.3	127.2	23	29.3	127.7	23	-
dano3_5	15.2	207.8	226	15.2	204.7	226	-
lrn	84.9	3097.6	115652	84.9	3097.2	115652	-
markshare_4_0	0.0	134.9	2341936	0.0	135.3	2341936	-
mik.250-20-75.1	82.6	2.7	7218	82.6	2.7	7218	-
mik.250-20-75.2	83.7	2.3	4746	83.7	2.3	4746	-
mik.250-20-75.3	81.5	2.6	5684	81.5	2.6	5684	-
mik.250-20-75.4	77.1	23.0	95266	77.1	23.0	95266	-
mik.250-20-75.5	81.7	3.9	12576	81.7	3.9	12576	-
neos1	100.0	4.1	2	100.0	4.1	2	-
neos10	55.5	43.3	7	55.5	43.4	7	-
neos11	0.0	210.7	5602	0.0	210.6	5602	-
neos12	12.4	538.3	1083	12.4	539.9	1083	-
neos13	0.0	226.1	2774	0.0	229.9	2774	-
neos14	87.4	681.4	407182	87.4	679.6	407182	-
neos17	67.0	17.4	14365	67.0	17.3	14365	-
neos2	55.7	27.1	19237	55.7	27.2	19237	-
neos20	2.4	5.8	841	2.4	5.8	841	-
neos21	10.2	17.5	1575	10.2	17.6	1575	-
neos22	100.0	1.2	1	100.0	1.2	1	-
neos23	36.5	15.2	16909	36.5	15.1	16909	-
neos3	35.3	283.9	206128	35.3	287.3	206128	-
neos4	100.0	2.4	1	100.0	2.4	1	-
neos5	19.2	1254.9	5375739	19.2	1253.5	5375739	-
neos6	100.0	190.4	7171	100.0	190.8	7171	-
neos648910	0.0	1.6	80	0.0	1.6	80	0
neos7	**88.8**	**12.7**	**7699**	84.9	67.5	38776	7
neos8	100.0	41.9	1	100.0	42.1	1	-
neos808444	100.0	2822.0	941	100.0	2860.0	941	-
neos818918	0.0	2706.6	1048671	0.0	2705.5	1048671	-
neos823206	95.9	356.4	21033	95.9	360.6	21033	0
neos897005	100.0	204.5	21	100.0	222.5	21	-
neos9	100.0	50.8	1	100.0	51.1	1	-
ns1671066	100.0	169.2	137689	100.0	170.1	137689	0
ns1688347	66.4	2094.7	12380	66.4	2101.6	12380	-
nug08	100.0	41.8	1	100.0	41.8	1	-
prod1	34.3	17.5	23482	34.3	17.6	23482	-
prod2	32.7	81.4	68500	32.7	81.4	68500	-
qap10	18.5	167.9	5	18.5	168.2	5	-
seymour1	12.2	453.0	5772	12.2	452.5	5772	-
swath2	14.7	61.4	5157	14.7	61.5	5157	-
swath3	11.4	866.1	140698	11.4	871.5	140698	-
means	**61.3**	**82.2**	**3579**	**61.1**	**85.2**	**3676**	

Table 11: continued on next page

problem	nomcf			mcf			
	rootgap closed %	time	nodes	rootgap closed %	time	nodes	#cuts
wins	**2**	**2**	**2**	**0**	**1**	**1**	

Table 11: Results of the mcf separator with Scip– *easy* instances

Table 12 presents the results of the separation for all hard instances. We compare the performance of Scip with (*mcf*) and without (*nomcf*) the Mcf separator. Every hard instance cannot be solved by Scip in both settings within one hour of CPU time. We provide the closed gap at the root node before branching (*rootgap*) defined as

$$100 \cdot (root - lp)/(bestprimal - lp),$$

the closed dual gap after optimization (*dualgap*) given by

$$100 \cdot (dual - lp)/(bestprimal - lp),$$

the closed primal gap after optimization (*primalgap*) given by

$$100 \cdot (bestprimal - bestdual)/(primal - bestdual),$$

the *endgap* defined as

$$100 \cdot (primal - dual)/|bestdual|,$$

and the number of cuts (*#cuts*) found by the Mcf separator. The numbers *primal* and *dual* correspond to the primal and dual bound at the end of the optimization. The value *lp* denotes the value of the initial LP relaxation, *bestprimal* and *bestdual* are the best known primal and dual bounds (see Table 8 in Appendix A), and *root* is the value of the LP at the root node after cutting and before branching. In case that *primal* = *bestdual* for an individual run we set the closed primal gap to 100%. If the LP value is already optimal ($lp = bestprimal = bestdual$) then *rootgap* as well as *dualgap* are considered to be 100%. If primal or dual bounds are not finite or in case that $bestdual = 0$ the corresponding gaps are not defined and marked with "-". Values in bold face indicate that they are at least 10% below the opposite value (*endgap*) or at least 1 percentage point above the opposite value (*rootgap*, *primalgap*, *dualgap*). If the Mcf separator is switched off (no network has been found or the network inconsistency ratio $\Psi(G)$ is above $\Psi^{\max} = 0.02$) then column *#cuts* contains a "-". Runs marked with "*" have hit the memory limit. For every testset two additional rows provide the arithmetic means of the gaps (*means*) and the number of values in bold face for every measure (*wins*).

problem	nomcf				mcf				
	closed gaps %			endgap	closed gaps %			endgap	
	root	dual	primal	%	root	dual	primal	%	#cuts
arc.set									
ns4-pr3	15.6	71.8	**100.0**	0.1	**17.3**	71.3	84.6	0.1	5
ns4-pr9	0.0	5.4	100.0	0.0	0.0	**8.3**	100.0	0.0	12
nu120-pr3	37.0	61.1	**100.0**	11.1	**38.6**	**64.1**	93.9	10.8	392
nu120-pr9	43.1	**76.0**	**100.0**	**5.5**	43.2	72.3	79.8	7.8	328
nu25-pr3	62.1	72.2	47.4	2.9	**66.9**	**78.8**	**70.9**	**1.7**	86
nu25-pr9	32.0	56.4	62.9	1.9	**35.0**	**59.6**	**64.1**	1.8	135
nu4-pr3	42.5	**64.1**	89.6	0.9	42.2	61.5	**100.0**	0.9	1
nu4-pr9	20.5	47.1	83.2	0.5	19.6	**51.1**	**100.0**	0.4	21
nu60-pr3	52.2	73.5	**86.7**	3.7	**54.2**	**77.3**	73.1	3.7	299
nu60-pr9	34.4	65.4	99.0	2.8	**39.1**	**73.7**	99.0	**2.1**	351
means	**33.9**	**59.3**	**86.9**	**1.4**	**35.6**	**61.8**	**86.5**	**1.3**	

Table 12: continued on next page

problem	nomcf				mcf				
	closed gaps %			endgap	closed gaps %			endgap	
	root	dual	primal	%	root	dual	primal	%	#cuts
wins	**0**	**2**	**4**	**1**	**6**	**7**	**4**	**2**	
cut.set									
n12-3	71.6	93.7	100.0	2.2	71.6	93.7	100.0	2.2	-
n15-3	24.3	24.5	100.0	66.7	24.3	24.5	100.0	66.7	-
n3-3	70.2	76.2	100.0	14.5	70.2	76.2	100.0	14.5	-
n9-3	65.7	77.4	100.0	11.4	65.7	77.4	100.0	11.4	-
means	**58.0**	**68.0**	**100.0**	**12.6**	**58.0**	**68.0**	**100.0**	**12.6**	
wins	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
fc									
fctp									
n3700	18.7	20.0	96.8	28.6	19.1	19.9	**100.0**	27.7	1307
n3701	18.5	19.8	100.0	27.1	18.8	20.3	100.0	26.9	1363
n3702	18.8	19.6	98.1	28.1	18.9	19.7	98.1	28.1	1377
n3703	16.8	18.3	100.0	31.1	16.6	17.6	100.0	31.4	1108
n3704	18.9	19.8	100.0	27.5	19.2	20.1	100.0	27.4	1242
n3705	18.0	19.2	95.8	28.9	17.9	18.8	95.8	29.0	2059
n3706	21.0	22.1	96.2	27.3	20.6	22.1	**98.3**	26.7	2280
n3707	19.6	20.7	100.0	28.7	19.6	20.9	100.0	28.6	1145
n3708	17.1	18.3	100.0	28.6	17.3	18.5	100.0	28.5	1294
n3709	18.8	19.5	100.0	28.5	19.0	19.9	100.0	28.4	1969
n370a	19.9	21.2	100.0	27.8	20.2	21.1	100.0	27.9	1427
n370b	17.4	18.2	95.5	29.1	17.5	18.4	**100.0**	27.7	1191
n370c	19.6	20.4	95.3	29.2	19.8	21.0	95.4	29.0	1220
n370d	19.6	20.4	95.3	29.2	19.8	21.0	95.4	29.0	1220
n370e	22.2	22.9	**100.0**	24.9	21.9	22.6	97.3	25.7	1345
ran14x18	53.6	83.9	80.3	3.7	**54.7**	83.8	80.3	3.7	1031
means	**21.2**	**24.0**	**97.1**	**24.9**	**21.3**	**24.1**	**97.5**	**24.8**	
wins	**0**	**0**	**1**	**0**	**1**	**0**	**3**	**0**	
avub									
nexp.100.20.2.2	53.1	63.4	16.8	13.7	**74.2**	**81.5**	**58.5**	**3.6**	84
nexp.100.20.8.1	27.3	38.6	2.5	162.0	**93.6**	**96.9**	**100.0**	**2.9**	747
nexp.100.20.8.2	33.0	44.9	2.8	158.2	**92.0**	**95.9**	**82.9**	**4.5**	1309
nexp.100.20.8.3	31.5	39.9	34.6	124.7	**50.3**	**55.8**	**100.0**	**52.7**	350
nexp.100.20.8.4	26.9	36.8	2.3	192.8	**91.9**	**96.4**	**83.5**	**4.0**	1123
nexp.100.20.8.5	23.0	30.1	30.1	218.3	**59.3**	**63.7**	**86.0**	**60.0**	617
nexp.150.20.2.2	46.5	49.3	100.0	6.6	47.4	49.9	100.0	6.5	228
nexp.150.20.2.4	59.2	66.0	100.0	4.1	**73.1**	**78.9**	100.0	**2.5**	62
nexp.150.20.4.1	49.8	57.2	2.1	41.0	**97.9**	**97.9**	**100.0**	**0.6**	1042
nexp.150.20.4.5	53.0	59.0	0.0	59.0	**97.9**	**98.4**	**100.0**	**0.6**	901
nexp.150.20.8.1	11.0	14.8	25.4	202.4	**71.8**	**73.4**	**91.8**	**36.0**	1303
nexp.150.20.8.2	15.7	19.5	17.1	225.6	**75.9**	**77.4**	**100.0**	**26.8**	1597
nexp.150.20.8.3	7.3	10.2	46.6	234.2	**32.5**	**34.3**	**75.8**	**132.9**	540
nexp.150.20.8.4	13.5	16.8	22.5	258.7	**67.2**	**68.5**	**100.0**	**42.5**	1220
nexp.150.20.8.5	15.6	18.7	34.4	208.9	**63.1**	**64.4**	**97.3**	**51.6**	1336
means	**31.1**	**37.7**	**29.1**	**83.9**	**72.5**	**75.6**	**91.7**	**10.2**	
wins	**0**	**0**	**0**	**0**	**14**	**14**	**13**	**14**	
sndlib									
cost266-DBE	40.1	80.0	100.0	3.9	**44.4**	**82.0**	100.0	3.5	312
cost266-DBM	47.6	76.2	95.0	3.9	**54.7**	**78.6**	**100.0**	**3.4**	76
cost266-UUE	21.8	76.6	100.0	4.8	**38.5**	**84.6**	100.0	**3.1**	264
cost266-UUM	19.5	62.1	86.2	$6.4^{\star}$	**42.1**	**73.7**	**90.2**	**$4.4^{\star}$**	81
dfn-bwin-DBE	46.6	54.7	74.1	72.4	**49.4**	55.6	**99.2**	**53.4**	1221
dfn-bwin-UUE	55.3	62.7	72.4	45.5	**57.8**	**65.2**	**100.0**	**31.3**	441

Table 12: continued on next page

problem	nomcf				mcf				
	closed gaps %			endgap	closed gaps %			endgap	
	root	dual	primal	%	root	dual	primal	%	#cuts
germany50-DBM	38.9	58.6	**100.0**	3.3	**40.2**	**64.4**	92.0	3.1	63
germany50-UUM	34.1	**63.1**	54.5	3.3	**36.2**	61.0	**100.0**	**1.9**	23
giul39-DDE	22.4	28.8	96.6	37.8	**26.3**	**30.5**	**98.5**	36.2	264
janos-us-DDM	15.6	32.3	**92.7**	$0.2^{\star}$	**47.0**	**54.0**	91.0	0.2	11
janos-us-ca-DDM	9.6	20.4	91.7	0.2	9.6	20.4	91.7	0.2	0
newyork-DBE	63.2	75.8	**100.0**	28.1	**74.8**	**81.7**	84.5	**25.1**	716
newyork-DBM	67.0	77.0	99.4	26.2	**69.3**	77.5	100.0	$25.4^{\star}$	267
newyork-UUE	57.6	76.2	93.1	26.6	**73.0**	**82.2**	**99.3**	**19.1**	256
newyork-UUM	57.9	76.8	99.3	$25.3^{\star}$	**74.1**	**81.7**	99.9	**19.8***	126
nobel-eu-DBE	28.0	73.8	51.4	3.2	28.3	74.8	**80.4**	**2.1**	720
nobel-eu-UUE	12.0	38.5	**100.0**	3.4	12.2	**46.2**	83.4	3.6	465
nobel-ger-DBE	35.0	66.0	**100.0**	**3.9**	35.6	**67.4**	79.5	4.6	1038
nobel-ger-UUE	**24.3**	53.5	100.0	4.4	21.4	**57.1**	100.0	4.1	876
norway-DBE	48.2	79.9	**100.0**	**18.2**	**60.3**	79.7	80.5	22.2	715
norway-DBM	48.0	74.2	**97.1**	$23.6^{\star}$	**64.8**	**83.1**	93.2	**16.3***	446
norway-UUE	40.4	77.8	100.0	15.7	**53.5**	**84.5**	100.0	**10.9**	261
norway-UUM	36.0	71.3	89.2	$21.2^{\star}$	**51.1**	**79.8**	**100.0**	**14.1***	115
pioro40-DBM	24.9	**42.3**	**100.0**	0.3	**25.9**	41.2	86.5	0.4	6
pioro40-UUM	16.1	28.6	**100.0**	0.3	**23.6**	**35.5**	92.4	0.3	9
sun-DDM	41.8	73.0	100.0	$6.7^{\star}$	**44.5**	**74.5**	100.0	$6.3^{\star}$	28
ta1-DBM	0.0	20.5	76.7	$68.1^{\star}$	**20.7**	**43.5**	**100.0**	**42.8***	11
ta1-UUE	5.7	30.0	65.7	44.4	**26.6**	**55.7**	**90.2**	**25.9***	21
ta1-UUM	0.1	20.5	80.3	$69.1^{\star}$	**15.0**	**39.2**	**94.5**	**47.3***	7
ta2-UUM	0.0	24.7	100.0	$1.8^{\star}$	**40.1**	**58.5**	100.0	**1.0***	4
means	**31.9**	**56.5**	**90.5**	**7.6**	**42.0**	**63.8**	**94.2**	**6.2**	
wins	**1**	**2**	**9**	**2**	**25**	**23**	**12**	**17**	
ufcn									
beasleyC3	**70.2**	**73.9**	79.7	26.5	68.6	72.1	**98.4**	**23.4**	255
berlin	81.5	86.8	89.8	12.3	**84.7**	**89.3**	**100.0**	**9.7**	152
brasil	77.2	84.2	**94.7**	14.3	**87.6**	**92.7**	75.8	**8.2**	114
g150x1100	68.1	78.2	68.4	15.3	**77.2**	**87.6**	**96.8**	**7.1**	196
g150x1650	66.4	75.0	83.7	16.5	**73.6**	**82.3**	**95.2**	**10.9**	193
g200x740	67.4	73.7	99.8	6.1	**85.3**	**93.9**	99.8	**1.4**	370
g200x740b	72.3	82.2	69.4	3.6	**87.0**	**96.5**	**100.0**	**0.6**	297
g200x740g	68.5	69.4	80.7	37.5	**73.4**	**74.3**	**89.5**	**29.4**	584
g200x740h	77.4	81.1	**84.3**	7.6	**82.8**	**86.0**	75.9	**6.5**	561
g200x740i	71.4	72.4	52.5	49.3	**78.7**	**80.6**	**64.9**	**32.7**	487
g55x188	82.2	97.1	100.0	**1.8**	**83.3**	96.7	100.0	2.0	139
k16x240	75.4	**92.1**	100.0	**6.2**	76.2	90.8	100.0	7.2	126
k16x240b	77.9	93.9	100.0	4.4	**79.9**	94.2	100.0	4.2	152
mc11	51.7	53.6	77.9	61.2	**78.4**	**79.3**	**92.1**	**26.3**	532
mc7	39.0	41.4	45.2	81.8	**81.6**	**83.0**	**61.0**	**28.6**	474
mc8	58.3	64.0	94.8	44.0	**75.7**	**77.3**	**98.5**	**27.3**	447
p100x588	88.0	93.2	99.3	4.0	88.7	93.8	99.7	3.6	227
p100x588b	82.5	85.6	78.7	17.5	83.5	**86.7**	**96.0**	**13.7**	340
p200x1188	89.0	94.0	83.4	3.7	89.6	94.5	**100.0**	**2.9**	205
p200x1188b	82.5	85.6	**90.1**	15.1	**84.1**	**88.1**	82.0	13.9	292
p500x2988	87.2	93.7	**92.2**	1.2	**88.9**	**95.0**	82.6	1.1	349
p500x2988b	80.5	82.3	78.5	16.5	81.3	82.8	**96.5**	**13.2**	440
p80x400	86.1	95.5	100.0	2.0	**88.7**	**96.7**	100.0	**1.5**	162
p80x400b	82.4	87.5	87.2	13.2	83.2	**88.7**	**95.8**	**10.9**	205
r80x800	79.4	85.7	83.2	5.4	**83.4**	**88.5**	**86.6**	**4.3**	262
means	**74.5**	**80.9**	**84.5**	**10.7**	**81.8**	**87.7**	**91.5**	**7.2**	
wins	**1**	**2**	**4**	**2**	**18**	**18**	**15**	**19**	

Table 12: continued on next page

problem	nomcf				mcf				
	closed gaps %			endgap	closed gaps %			endgap	
	root	dual	primal	%	root	dual	primal	%	#cuts
miplib									
a1c1s1	67.8	84.6	0.0	13.7	67.8	84.6	0.0	13.7	-
atlanta-ip	0.4	55.5	0.0	6.6	0.4	56.1	0.0	6.5	-
dano3mip	0.6	0.8	99.9	24.2	0.6	0.8	99.9	24.2	-
danoint	1.7	37.9	100.0	2.9	1.9	**42.3**	100.0	2.7	28
ds	0.3	3.6	19.3	498.7	0.3	3.6	19.3	498.7	-
glass4	0.0	25.0	0.0	58.3	0.0	25.0	0.0	58.3	-
harp2	35.1	99.9	100.0	0.0	35.1	99.9	100.0	0.0	-
liu	0.0	0.0	68.8	150.0	0.0	0.0	68.8	150.0	-
markshare1	0.0	0.0	0.0	400.0	0.0	0.0	0.0	400.0	-
markshare2	0.0	0.0	0.0	1500.0	0.0	0.0	0.0	1500.0	-
mkc	42.7	81.0	100.0	1.6	42.7	81.0	100.0	1.6	-
momentum1	56.9	59.1	-	-	56.9	59.1	-	-	-
momentum2	0.5	7.2	0.0	12.2	0.5	7.2	0.0	12.2	0
momentum3	0.7	0.7	-	-	0.7	0.7	-	-	21
msc98-ip	55.7	55.7	0.0	1.0	55.7	55.7	0.0	1.0	-
nsrand-ipx	27.3	63.3	0.0	5.8	27.3	63.3	0.0	5.8	-
protfold	14.5	46.6	0.0	38.2	14.5	46.6	0.0	38.2	-
rd-rplusc-21	0.0	0.0	0.0	99.9	0.0	0.0	0.0	99.9	0
roll3000	76.2	96.1	100.0	0.5	76.2	96.1	100.0	0.5	-
seymour	22.6	55.2	89.6	2.3	22.6	55.2	89.6	2.3	-
sp97ar	8.6	40.1	13.0	3.0	8.6	40.2	13.0	3.0	-
stp3d	0.2	2.0	-	-	0.2	2.9	-	-	0
swath	29.0	41.8	0.0	20.9	29.0	41.8	0.0	20.9	-
t1717	1.5	2.9	49.4	51.7	1.5	2.9	49.4	51.7	-
timtab2	31.8	47.7	99.4	73.5	31.8	47.7	99.4	73.4	2
means	**19.0**	**36.3**	**38.2**	**15.9**	**19.0**	**36.5**	**38.2**	**15.9**	
wins	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	
mittelmann									
markshare_5_0	0.0	0.0	100.0	-	0.0	0.0	100.0	-	-
ns1648184	5.7	84.2	100.0	0.4	5.7	84.2	100.0	0.4	-
ns1692855	53.0	73.1	100.0	15.4	53.0	73.1	100.0	15.4	-
means	**19.6**	**52.4**	**100.0**	**2.7**	**19.6**	**52.4**	**100.0**	**2.7**	
wins	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	

Table 12: Results of the mcf separator with SCIP– *hard* instances