Math. Prog. Comp. (2010) 2:125–165
DOI 10.1007/s12532-010-0015-3

FULL LENGTH PAPER

The MCF-separator: detecting and exploiting multi-commodity flow structures in MIPs

Tobias Achterberg · Christian Raack

Received: 30 November 2009 / Accepted: 27 May 2010 / Published online: 3 July 2010

Abstract Given a general mixed integer program, 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 but provide a special purpose aggregation heuristic that exploits the network structure. Our separation scheme speeds-up the computation for a large set of mixed integer programs coming from network design problems by a factor two on average. We show that almost 10% of the instances in general testsets contain consistent embedded networks. For these instances the computation time is decreased by 18% on average.

Keywords Mixed integer programming · Network detection · Cut-based inequalities · SCIP · CPLEX

Mathematics Subject Classification (2000) 90C11 · 90C35 · 90-08

This research has been supported by the DFG research Center MATHEON.

T. Achterberg
IBM Schoenaicher, Str. 220, 71003 Boeblingen, Germany
e-mail: achterberg@de.ibm.com

C. Raack (✉)
Zuse Institute Berlin (ZIB), Takustr. 7, 14195 Berlin, Germany
e-mail: raack@zib.de

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 and Wolsey [39–41]. 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 [16]:

> 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.*

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 indices of $\mathcal{A}$. The set of integer variables is given by $I \subseteq N$. We consider the mixed integer program (MIP)

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

where the linear constraints of the system (1) are given either as equations or as inequalities. Upper and lower bounds on variables are already included in the constraint system. Associated with (1) we define $X := \{x \in \mathbb{R}^{N\setminus I} \times \mathbb{Z}^I \;:\; \mathcal{A}x \leq b\}$ 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 [41] (also see [39,40]) 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 [1,16,59]. The idea is to generate MIR inequalities from constraints that are valid for X. Marchand and Wolsey [41] employ four different operations to obtain such base inequalities. In the first *aggregation* step a conic combination $u^T \mathcal{A}x \le u^T b$ of the system (1) using weights $u \ge 0$ is considered. This is followed by *bound substitution* trying to substitute continuous variables by variable upper or lower bounds of the form $x_j \le cx_{j'}$ or $cx_{j'} \le x_j$ 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 $x_j \le u_j$ or $l_j \le x_j$ with $j \in C$. Eventually, the resulting mixed integer knapsack inequality is *scaled* using an appropriate multiplier $\gamma > 0$. 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 [43,58]. The c-MIR framework is introduced in [39,40] 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 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 [16].

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 Marchand and Wolsey [41]. 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 2,500 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 $\gamma > 0$ before MIR can be seen as using weight vectors γu^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 Sect. 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 promising cuts in the detected network. Some important extensions and model variants are considered in Sect. 5. In Sect. 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 [17], MIPLIB 2003 [3], the MIP instances of Hans Mittelmann [42], and the CPLEX-internal testset. We conclude with some remarks in Sect. 7.

2 Network design

Combinatorial optimization problems arising for instance from applications in telecommunication and public transportation 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 bandwidth. 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 [6,13,14,24,30,38,45,49] 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 Sect. 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 $a \in A$ 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 [6,13,37].

Let f_a^k be the flow of commodity $k \in K$ on arc $a \in A$. Variables $y_a \in \mathbb{Z}_+$ count the number of capacity modules of size c_a provided on arcs $a \in A$. 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:

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

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

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

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

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 Fig. 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.

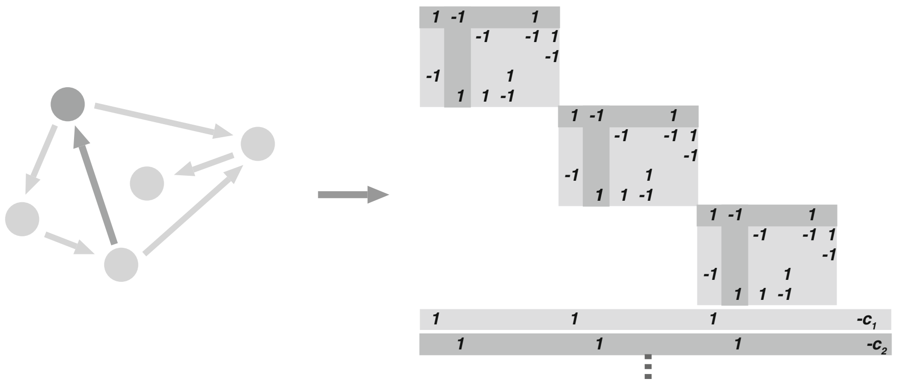

Fig. 1 A digraph and the corresponding matrix representing a coupled multi-commodity flow. A node has a flow-row in every commodity. An arc has a column in every commodity and corresponds to one coupling capacity row

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 [6,50].

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 $V\backslash S$) and the capacity provided on the dicut $\delta(S)$. We denote by $d_S := \sum_{v\in S} d_v =: -d_{V\backslash S}$ 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:

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

Similarly the supply of S (which is the demand of $V\backslash S$) 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 [4]. 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.

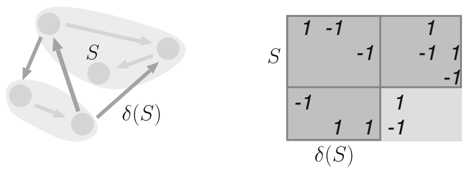

Fig. 2 The cut $\delta(S)$ obtained by aggregating the flow-rows of a nodeset S

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 Fig. 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):

$$\sum_{a\in\delta^+(S)} f_a - \sum_{a\in\delta^-(S)} f_a \leq d_S \tag{5a}$$
$$f_a - c_a y_a \leq 0 \qquad \forall a \in \delta(S) \tag{5b}$$
$$y_a \leq u_a \qquad \forall a \in \delta(S) \tag{5c}$$

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 [5,8,12,28,29,40,46,52,53,56,57] 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 0-1 single node flow sets arise as natural relaxations of general MIPs. Also many 0–1 mixed IP applications have fixed charge network sub-structures. Here we perceive single node flow sets as corresponding to network cuts. Related polyhedral structures with unbounded integer variables have been studied in [6,50] 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 [7,40] 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 of 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* [6,13,21]

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

where r is the remainder of the division of $d_{V\setminus S}$ by c. 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 [6] 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 [6,14,21] (directed models), [35,37,38,49,50] (undirected models), and [35,37,38,49,50] (so-called bi-directed models). If the capacities are not arc-independent (or similarly if there is more than one arc facility), inequality (4) can be divided by one of the given capacities c_a before applying MIR. In particular, the facet defining cutset and flow-cutset inequalities in [6,13,35,38,50] can be obtained this way, 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 whether 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. In this context, certain classes of 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.

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 $\mathcal{A}x \leq b$ of the given MIP. We first identify a subsystem of the form (2) and construct the corresponding network as described in Sect. 3, that is, we resolve the structure given in Fig. 1 backwards. In the detected network we identify interesting cuts, see Sect. 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, this 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 $\gamma = 1/|c|$ 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 $V \setminus S$. The separation scheme is described in more detail in Sect. 4. Clearly, the corresponding aggregation involves a large number of the original constraints (2a) and (2b) already for small sized networks.

Although 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 Sect. 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

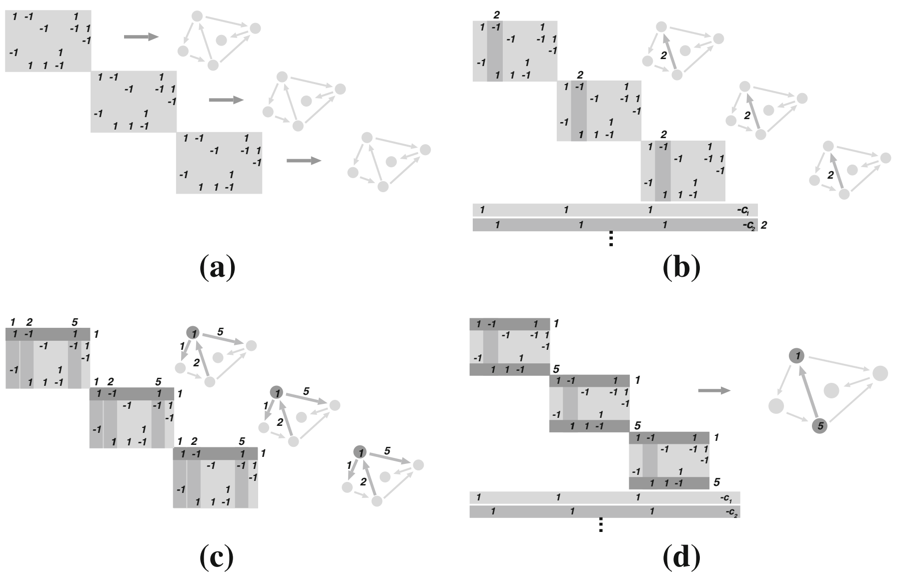

Fig. 3 The network detection algorithm. **a** Flow detection resulting in a disconnected graph with one component per commodity. **b** Arc detection: using capacity constraints to assign arc-ids. **c** Node detection: compare arc-id patterns in the different commodities and assign the same node-id to (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

most one $+1$ and at most one -1 entry as it can be seen in Fig. 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 [15], see also Brown and Wright [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* [15,19].

The flow structure could result in a flow network with multiple independent components, see Fig. 3a. 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 proved 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 whether 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 Sect. 5. The Arc Detection procedure assigns arcs to the coupling capacity constraints and all corresponding flow-variables, see Fig. 3b.

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 Fig. 3c. 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 Fig. 3d. 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 Sect. 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

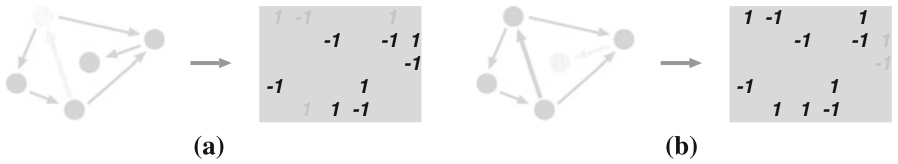

Fig. 4 The impact of preprocessing. **a** A node and all incoming arcs deleted by user presolving. Notice that two of the remaining arcs have no source. **b** Loosely connected nodes and arcs deleted by solver presolving

in every commodity. Moreover, an arc does not necessarily have both a source and a target node, that is, the corresponding flow-rows might be missing:

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 Fig. 4a. 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 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 Fig. 4b. 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

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

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

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

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 [15]. 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 [15] we do

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 cols 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 $N_F := N[M_F]$
```

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 [15] 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, $M_F(k) := \{i \in M_F : rowcom(i) = k\}$. 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 Bixby and Fourer [15].

To fit a row into a flow-system one can reflect it, i.e., multiply it by -1. Since our separation approach (see Sect. 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

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
```

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 Sect. 4.

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

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          $score :=$ CompareP attern($pattern$, PatternOf($i'$))
12          Remember $bestrow(k')$ with largest $score$ for $k'$
13      end
14      for $k' \in K \setminus \{k\}$ do     // Assign $v$ to rows with closest arc-pattern to $i$
15          $rownode(bestrow(k')) := v$
16      end
17  end
```

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. 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.

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.

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. 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 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

```
Function PatternOf(i)
   Input  : flow-row $i$
   Output : $pattern \in \mathbb{Z}^A$
1  $pattern(a) = 0$ for all $a \in A$                  // Initialize arc-pattern of row $i$
2  for $j \in N[i]$ do
3      $a = colarc(j)$
4      if $a = 0$ then continue                         // Uncapacitated flow-variable
5      if $\alpha_{ij} > 0$ then $pattern(a) = pattern(a) + 1$          // Outgoing arc
6      if $\alpha_{ij} < 0$ then $pattern(a) = pattern(a) - 1$          // Incoming arc
7  end
8  return $pattern$
```

```
Function ComparePattern(pattern1, pattern2)
   Input  : arc-patterns $pattern1$, $pattern2$
   Output : $score \in \mathbb{R}_+$
1  Initialize $score := 0$
2  for $a \in A$ do
3      if $pattern1(a) \cdot pattern2(a) > 0$ then    // Patterns overlap and signs match
4          $score := score + \min(\frac{pattern1(a)}{pattern2(a)}, \frac{pattern2(a)}{pattern1(a)})$ // Increase weighted overlap
5      end
6      Calculate tiebreaker $0 \leq \varepsilon \ll 1$
7      $score := score - \varepsilon$
8  end
9  return $score$
```

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 Sect. 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 $\Psi(G) \in [0, 1)$
1  Initialize $inconsistencies := 0$
2  for $a \in A$ do
3      $i := rowarc^{-1}(a)$                                   // Capacity row of arc $a$
4      for $k \in K$ do
5          $nvars(k) := |\{j \in N[i] : colcom(j) = k\}|$         // # flow-variables per
                                                                    commodity
6      end
7      $ncom := |\{k \in K : nvars(k) > 0\}|$                   // # commodities in row $i$
       // Ask all flow-variables for source and target node
8      Initialize $scount(v) := 0$, $tcount(v) := 0$ for all $v \in V$
9      for $j \in N[i]$ with $colcom(j) > 0$ do
10         $k := colcom(j)$
11         for $i' \in M[j] \cap M_F$ do       // $j$ has at most two incident flow-rows
12             $v := rownode(i')$
13             if $\alpha_{i'j} > 0$ then                       // Increase source count for $v$
14                 $scount(v) := scount(v) + 1/nvars(k)$
15             else                                        // Increase target count for $v$
16                 $tcount(v) := tcount(v) + 1/nvars(k)$
18         end
19     end
       // Majority vote wins
20     $s(a) := \text{argmax}\{scount(v) : v \in V, scount(v) \geq tcount(v)\}$     // Assign best
                                                                         source to $a$
21     $t(a) := \text{argmax}\{tcount(v) : v \in V, tcount(v) \geq scount(v)\}$     // Assign best
                                                                         target to $a$
       // Minority votes give arc inconsistency
22     $totalcount := \sum_{v \in V}(scount(v) + tcount(v))$
23     $\Psi(a) := (totalcount - scount(s(a)) - tcount(t(a)))/2 \cdot ncom$  // Arc inconsistency
24     $\Psi(G) := \Psi(G) + \Psi(a)/|A|$                        // Network inconsistency
25 end
26 for $a \in A$ do
27     if $\Psi(a) > \Psi_a^{\max}$ then $A := A \backslash a$;              // Delete inconsistent arcs
28 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. Achterberg and Raack [2] provide a complete list of the considered instances with more detailed information. Table 1 introduces the 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 possible model variations also see Sect. 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

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

Testset	Size	Source	Paper	Problem description
arc.set	35	Atamtürk [9]	[10]	MCF, S, US, BIN, DI, SF
avub	60	Atamtürk [9]	[11]	SCF, BIN, DI, MF, RG
cut.set	15	Atamtürk [9]	[6]	MCF, INT, DI
fc	20	Atamtürk [9]	[5]	SCF, BIN, DI, SF, RG
fctp	32	Gottlieb [27]	–	SCF, BIN, DI, SF, bipartite graphs
sndlib	52	ZIB [61]	[44]	MCF, INT, BIN+GUB, DI, UN, MF
ufcn	83	Wolsey [55]	[45]	SCF, BIN, DI, M
miplib	92	ZIB [3,17]	[3,17]	General MIP instances
mittelmann	59	Mittelmann [42]	–	General MIP instances

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 [17] and MIPLIB 2003 [3] libraries. The *mittelmann* testset subsumes instances available on the website 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.

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 3.00 GHz 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 [2] (Tables 9 and 10 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*(Ψ)). 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

Table 2 Network detection results summary

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	–	–	–

is available, we compare it with the detected network by taking the arithmetic mean (*mean diff %*) 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 [2]). 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 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 [6] the set consists of problems with 19–29 nodes and 23–93 commodities. In contrast, our flow-detection procedure detects 1–6 commodities with up to 168 nodes (see Table 10 in [2]). 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 $\mathcal{G}$ of multipliers $\gamma > 0$ where $1/\gamma$ 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 $\gamma u^T \mathcal{A}x \leq \gamma u^T b$ for all $\gamma \in \mathcal{G}$.

The vector u and a multiplier $\gamma \in \mathcal{G}$ 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 [41] 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 $nodecomrow : V \times K \to M_F \cup \{0\}$ 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

Algorithm 5: Separation scheme

```
Input : mappings $arcrow : A \to M_C \cup \{0\}$ and $nodecomrow : V \times K \to M_F \cup \{0\}$, primal and
        dual solution $(x^*, \pi^*)$ 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 $(x^*, \pi^*)$
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 $i := nodecomrow(v, k)$
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 $i := nodecomrow(v, k)$ and $u_i := 1$ // Set flow-row weights
10     Initialize set of multipliers $\mathcal{G} := \emptyset$.
11     for $a \in \delta^-(S)$ do
12         $i := arcrow(a)$ and $u_i := 1$                  // Set capacity-row weights
13         for $j \in I \cap N[i]$ do add $1/|\alpha_{ij}|$ to $\mathcal{G}$    // Use coeffs of int variables for
                                                              scaling
14     end
15     for $\gamma \in \mathcal{G}$ do
16         $violation$ = cMIR($u$, $\gamma$)          // Generate cutset inequality (8)
17         if $violation > 0$ then add c-MIR-cut to the cut-pool
18     end
19     if no violated c-MIR-cut was found then
20         Chose $\gamma \in \mathcal{G}$ and determine a subset $A^- \subseteq \delta^-(S)$
21         for $a \in \delta^-(S) \backslash A^-$ do
22             $i := arcrow(a)$ and $u_i := 0$         // Remove capacity-row for a from
                                                          aggregation
23         end
24         $violation$ = cMIR($u$, $\gamma$)      // Generate flow-cutset inequality (7)
25         if $violation > 0$ then add c-MIR-cut to the cut-pool
26     end
27 end
```

that the latter are not as efficient in improving the dual bounds and performance, see for instance [13,45,49].

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 Sect. 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 $\gamma > 0$. In our implementation we use a maximum of 20 from the largest multipliers in $\mathcal{G}$. Two multipliers $\gamma_1 \geq \gamma_2$ are considered to be identical if $\gamma_1/\gamma_2 < 1.001$.

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 $\mathcal{G}$ (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 [6,45,49]. 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 $\gamma \in \mathcal{G}$.

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 $V \backslash S$ 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 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 subtract 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 Sect. 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 $G[V \setminus S]$, i.e., the graphs defined by the nodes in S (resp. $V \setminus S$) and the arcs with both endnodes in S (resp. $V \setminus S$), 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 $G[V \setminus S]$ 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 [6,11,50,54]. 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

$$\sum_{k \in K} f_a^k - \sum_{t \in T_a} c_a^t y_a^t \leq 0 \quad \forall a \in A \tag{9}$$

Capacity constraints of the form (9) 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 Sect. 2 and Algorithm 5 results in strong valid multi-facility cutset and flow-cutset inequalities. One simply uses the capacity constraints (9) for bound substitution (or similarly adds them to (5a)) and considers all the facility capacities for scaling before MIR, see [6,49,50]. 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 [10,18,31,32]. 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

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

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:

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

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 [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 $c_{vw} y_{vw}$:

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

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 $d_S = -\sum_{k \in K_S^+} d_S^k + \sum_{k \in K_S^-} d_S^k < 0$. 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].

Additional cut generation procedures. Both SCIP and CPLEX contain procedures to generate (mixed) knapsack-cover inequalities and flow-cover inequalities (using exact and heuristic lifting methods) for a base inequality such as (6), provided that this base inequality is already in the required form or can be relaxed accordingly.

Our SCIP implementation of the MCF-separator tries to generate exactly lifted knapsack-cover inequalities (based on knapsack-covers) in addition to applying default c-MIR to (6). We have not tried flow-cover inequalities with SCIP. Notice that the corresponding implementation also uses the c-MIR framework but with a different complementing heuristic based on flow-covers, see [59].

We have tried both the flow-cover and knapsack-cover separators of CPLEX but computational tests indicated that these do not improve the performance. Hence, the MCF-separator in CPLEX is exclusively based on the presented c-MIR procedure.

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. Detailed results for SCIP can be found in Tables 11 and 12 in [2].

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 $\Psi_a^{\max} = 0.5$ and set $\Omega = 5$. The effect of changing $\Psi_a^{\max}$ and $\Psi^{\max}$ is studied in Sect. 6.1. In Sect. 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. 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 Sect. 3.8. For the CPLEX 12.1 tests we used a single CPU of a 64bit 3.33 GHz Quad-Core machine with 6144 KB of cache and 16 GB of RAM. The MCF-separator is called with the default cutting strategy, that is, it is called in every pass of the solver but only in the root node. For all tests we fixed the time limit to 1 h and the memory limit to 6 GB. We will distinguish *easy* and *hard* instances in our exposition. Hard instances cannot be solved 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, 4, and 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 second 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 (*rootgap closed*) 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 [2]), 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

Table 3 Summary for easy instances—*mcf* versus *nomcf*—SCIP

Testset	#	nomcf—means			mcf—means			mcf/nomcf			
		rootgap closed %	time	nodes	rootgap closed %	time	nodes	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

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

Table 4 Summary for hard instances—mcf versus nomcf—SCIP

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

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

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

Table 5 Summary for easy instances—mcf versus nomcf—CPLEX

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	67	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

Table 5 continued

Testset	#	nomcf—means		mcf—means		mcf/nomcf			
		time	nodes	time	nodes	wins	t-outs	time	nodes
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

respectively. The endgap is given by

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

All closed gaps (root, dual, primal) are defined such that larger values correspond to better results with a maximum of 100% whereas the endgap is the better the closer to 0%. The values *lp* and *bestprimal* are defined as above and *bestdual* refers to the best known dual bound. These values can be found in Table 8 in [2]. 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 1 h 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 3.3 to 3.5 s and from 4.5 to 4.6 s 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 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.

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 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 h 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 because there is not necessarily a proper cancellation of flow-variables in case of inconsistency we might produce dense and unstable cutting planes.

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 we released both parameters $\Psi_a^{\max}$ and $\Psi^{\max}$ individually and simultaneously. Table 6 reports on the results for all considered general MIP instances for both SCIP and CPLEX. The third 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 4–6 in Table 6 summarize the values for the default settings for comparison. These are precisely the values you can already find in Table 3 and Table 5 for SCIP and CPLEX, respectively. Notice again that the results for the testsets *miplib* and *mittelmann* should not be overestimated since the number of affected instances is simply too small.

It can be seen that releasing the maximum arc inconsistency ratio $\Psi_a^{\max}$ alone (Columns 7–9) does not remarkably change the behavior of the solver. Recall that

Table 6 Impact of inconsistency—SCIP and CPLEX—easy instances

Solver	Testset	#	$\Psi^{\max} = 0.02^\dagger$ $\Psi_a^{\max} = 0.5^\dagger$			$\Psi^{\max} = 0.02^\dagger$ $\Psi_a^{\max} = \infty$			$\Psi^{\max} = \infty$ $\Psi_a^{\max} = 0.5^\dagger$			$\Psi^{\max} = \infty$ $\Psi_a^{\max} = \infty$		
			sep	wins	time	sep	wins	time	sep	wins	time	sep	wins	time
SCIP	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
CPLEX	miplib	67	8	3/1	0.99	8	3/1	0.99	11	3/2	0.99	12	3/2	0.99
	mittelmann	56	6	3/2	0.97	6	2/2	0.97	9	3/4	0.98	10	3/5	1.00
	cplex	1266	115	45/35	0.98	115	46/33	0.98	214	78/80	0.99	247	84/99	1.00
	cplex10s	780	77	30/25	0.97	77	31/24	0.97	146	58/56	0.98	166	62/71	0.99
	cplex100s	411	42	20/13	0.96	43	20/11	0.96	81	35/28	0.96	90	39/34	1.01

† Default values

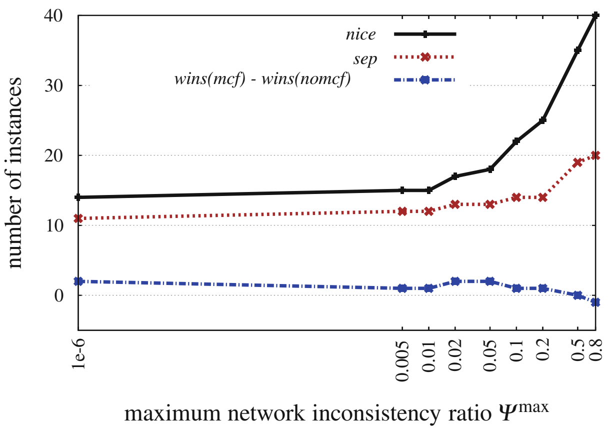

Fig. 5 Impact of inconsistency—SCIP—*miplib* easy testset. $\Psi_a^{\max}$ set to ∞ and $\Psi^{\max}$ increasing

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 10–12) already deteriorates the performance. There are more instances considered for separation but the wins and time ratios degrade at least for the larger *cplex* testsets. As shown in Columns 13–15 of Table 6, releasing both inconsistency parameters even worsens the performance in terms of wins and time ratios while the number of affected instances is again increasing. It turns out that the arc inconsistency ratio is not as important as the network inconsistency ratio, but releasing both restrictions gives the worst results.

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 SCIP tests and the easy instances within the *miplib* testset. Figure 5 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 $\Psi^{\max} = 0.8$ means that the MCF-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 Fig. 5, both mechanisms for refusing inconsistent networks or network components help to improve the overall performance of the separator and can be used alone. But the best results are obtained with a small maximum network inconsistency ratio $\Psi^{\max}$. 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 Sect. 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 except for *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

Table 7 Impact of the partition size Ω—SCIP easy

Testset	#	sep	$\Omega = 3$			$\Omega = 5^{\dagger}$			$\Omega = 7$			$\Omega = 9$		
			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

† Default values

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.

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 and 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 1 h 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. Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin. http://opus.kobv.de/tuberlin/volltexte/2007/1611/ (2007)
2. Achterberg, T., Raack, C.: The MCF-separator—Detecting and Exploiting Multi-Commodity Flow Structures in MIPs. ZIB-Report 07-38, Konrad-Zuse-Zentrum für Informationstechnik Berlin. http://www.zib.de/ (2009)
3. Achterberg, T., Koch, T., Martin, A.: MIPLIB Oper. Res. Lett. 34(4):361–372, (2003). http://miplib.zib.de/ (2006)
4. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
5. Atamtürk, A.: Flow pack facets of the single node fixed-charge flow polytope. Oper. Res. Lett. **29**, 107–114 (2001)
6. Atamtürk, A.: On capacitated network design cut-set polyhedra. Math. Program. **92**, 425–437 (2002)
7. Atamtürk, A.: On the facets of the mixed-integer knapsack polyhedron. Math. Program. **98**, 145–175 (2003)
8. Atamtürk, A.: Cover and pack inequalities for mixed integer programming. Ann. Oper. Res. **139**(1), 21–38 (2005)
9. Atamtürk, A.: MIP instances. University of California, Berkeley. http://www.ieor.berkeley.edu/~atamturk/data/ (2009)
10. Atamtürk, A., Rajan, D.: On splittable and unsplittable capacitated network design arc-set polyhedra. Math. Program. **92**, 315–333 (2002)
11. Atamtürk, A., Nemhauser, G.L., Savelsbergh, M.W.P.: Valid inequalities for problems with additive variable upper bounds. Math. Program. **91**, 145–162 (2001)
12. Balas, E.: Facets of the knapsack polytope. Math. Program. **8**, 146–164 (1975)
13. Bienstock, D., Günlük, O.: Capacitated network design—polyhedral structure and computation. INFORMS J. Comput. **8**, 243–259 (1996)
14. Bienstock, D., Chopra, S., Günlük, O., Tsai, C.Y.: Minimum cost capacity installation for multicommodity network flows. Math. Program. **81**, 177–199 (1998)
15. Bixby, R., Fourer, R.: Finding embedded network rows in linear programs I. Extraction heuristics. Manage. Sci. **34**(3), 342–376 (1988)
16. Bixby, R., Rothberg, E.: Progress in computational mixed integer programming—a look back from the other side of the tipping point. Ann. Oper. Res. **149**(1), 37–41 (2007)
17. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Optima 58:12–15. http://www.caam.rice.edu/~bixby/miplib/miplib.html (1998)
18. Brockmüller, B., Günlük, O., Wolsey, L.A.: Designing private line networks: polyhedral analysis and computation. Trans. Oper. Res. **16**, 7–24 (2004)
19. Brown, G., Wright, W.: Automatic identification of embedded network rows in large-scale optimization models. Math. Program. **29**, 41–56 (1984)
20. Bussieck, M.R., Kreuzer, P., Zimmermann, U.T.: Discrete optimization in public rail transport. Math. Program. **79**(1–3), 415–444 (1997)
21. Chopra, S., Gilboa, I., Sastry, S.T.: Source sink flows with capacity installation in batches. Discret. Appl. Math. **86**, 165–192 (1998)
22. COmputational INfrastructure for Operations Research (COIN-OR) Cut Generation Library (CGL). https://projects.coin-or.org/Cgl (2009)
23. Dahl, G., Stoer, M.: A polyhedral approach to multicommodity survivable network design. Numer. Math. **68**, 149–167 (1994)
24. Dahl, G., Stoer, M.: A cutting plane algorithm for multicommodity survivable network design problems. INFORMS J. Comput. **10**, 1–11 (1998)
25. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman and Company, New York (1979)

26. Gonçalves, J.P.M., Ladanyi, L.: An implementation of a separation procedure for mixed integer rounding inequalities. IBM Res. Report RC23686 (W0508-022), IBM (2005)
27. Gottlieb, J., Mittelmann, H.: FCTP instances. Arizona State University. http://plato.la.asu.edu/ftp/fctp/ (2009)
28. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Lifted flow cover inequalities for mixed 0-1 integer programs. Math. Program. **85**, 436–467 (1999)
29. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Sequence independent lifting in mixed integer programming. INFORMS J. Comput. pp. 109–129 (2000)
30. Günlük, O.: A branch and cut algorithm for capacitated network design problems. Math. Program. **86**, 17–39 (1999)
31. Hoesel, S.P.M., Koster, A.M.C.A., van de Leensel, R.L.M.J., Savelsbergh, M.W.P.: Polyhedral results for the edge capacity polytope. Math. Program. **92**(2), 335–358 (2002)
32. Hoesel, S.P.M., Koster, A.M.C.A., van de Leensel, R.L.M.J., Savelsbergh, M.W.P.: Bidirected and unidirected capacity installation in telecommunication networks. Discret. Appl. Math. **133**, 103–121 (2004)
33. IBM-ILOG: CPLEX. http://www.ilog.com/products/cplex/ (2009)
34. Louveaux, Q., Wolsey, L.A.: Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. 4OR **1**(3), 173–207 (2003)
35. Magnanti, T.L., Mirchandani, P.: Shortest paths, single origin-destination network design and associated polyhedra. Networks **33**, 103–121 (1993)
36. Magnanti, T.L., Wong, R.T.: Network design and transportation planning: models and algorithms. Trans. Sci. **18**(1), 1–55 (1984)
37. Magnanti, T.L., Mirchandani, P., Vachani, R.: The convex hull of two core capacitated network design problems. Math. Program. **60**, 233–250 (1993)
38. Magnanti, T.L., Mirchandani, P., Vachani, R.: Modelling and solving the two-facility capacitated network loading problem. Oper. Res. **43**, 142–157 (1995)
39. Marchand, H.: 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 (1997)
40. Marchand, H., Wolsey, L.A.: The 0–1 knapsack problem with a single continuous variable. Math. Program. **85**, 15–33 (1999)
41. Marchand, H., Wolsey, L.A.: Aggregation and mixed integer rounding to solve MIPs. Oper. Res. **49**(3), 363–371 (2001)
42. Mittelmann, H.: Benchmarks for optimization software. http://plato.asu.edu/bench.html (2009)
43. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, London (1988)
44. Orlowski, S„ Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0–Survivable Network Design Library. Networks **55**(3), 276–286 (2010)
45. Ortega, F., Wolsey, L.A.: A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. Networks **41**, 143–158 (2003)
46. Padberg, M.W., Roy, T.J.V., Wolsey, L.A.: Valid linear inequalities for fixed charge problems. Oper. Res. **33**, 842–861 (1985)
47. Pióro, M., Medhi, D.: Routing, Flow, and Capacity Design in Communication and Computer Networks. Morgan Kaufmann Publishers, Menlo Park (2004)
48. Pochet, Y., Wolsey, L.A.: Integer knapsack and flow covers with divisible coefficients. Discret. Appl. Math. **59**, 57–74 (1995)
49. Raack, C., Koster, A.M.C.A., Orlowski, S., Wessäly, R.: Capacitated network design using general flow-cutset inequalities. In: Proceedings of the Third International Network Optimization Conference (INOC 2007), Spa, Belgium (2007)
50. Raack, C., Koster, A.M.C.A., Orlowski, S., Wessäly, R.: On cut-based inequalities for capacitated network design polyhedra. Networks (2010, to appear)
51. Resende, M., Pardalos, P. (eds.): Handbook of Optimization in Telecommunications. Springer, Berlin (2006)
52. Van Roy, T.J., Wolsey, L.A.: Valid inequalities for mixed 0-1 programs. Discret. Appl. Math. **14**, 199–213 (1986)
53. Weismantel, R.: On the 0/1 knapsack polytope. Math. Program. **77**, 49–68 (1997)
54. Wessäly, R.: Dimensioning Survivable Capacitated NETworks. PhD thesis, Technische Universität Berlin (2000)

55. Wolsey, L.: UFCN instances. CORE, Université catholique de Louvain. http://www.core.ucl.ac.be/wolsey/ufcn.htm (2009)
56. Wolsey, L.A.: Faces for a linear inequality in 0–1 variables. Math. Program. **8**, 165–178 (1975)
57. Wolsey, L.A.: Valid inequalities and superadditivity for 0–1 integer programs. Math. Oper. Res. **2**, 66–77 (1977)
58. Wolsey, L.A.: Integer Programming. Wiley, London (1998)
59. Wolter, K.: 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)
61. Zuse Institut Berlin: SNDlib—Survivable Network Design Library. http://sndlib.zib.de/ (2009)