Math. Prog. Comp. (2015) 7:189–217
DOI 10.1007/s12532-015-0079-1

FULL LENGTH PAPER

Solving network design problems via iterative aggregation

Andreas Bärmann[1] · Frauke Liers[1] · Alexander Martin[1] · Maximilian Merkert[1] · Christoph Thurner[1] · Dieter Weninger[1]

Received: 20 December 2013 / Accepted: 9 February 2015 / Published online: 25 March 2015

Abstract In this work, we present an exact approach for solving network design problems that is based on an iterative graph aggregation procedure. The scheme allows existing preinstalled capacities. Starting with an initial aggregation, we solve a sequence of network design master problems over increasingly fine-grained representations of the original network. In each step, a subproblem is solved that either proves optimality of the solution or gives a directive where to refine the representation of the network in the subsequent iteration. The algorithm terminates with a globally optimal solution to the original problem. Our implementation uses a standard integer programming solver for solving the master problems as well as the subproblems. The computational results on random and realistic instances confirm the profitable use of the iterative aggregation technique. The computing time often reduces drastically when our method is compared to solving the original problem from scratch.

✉ Andreas Bärmann
Andreas.Baermann@math.uni-erlangen.de

Frauke Liers
Frauke.Liers@math.uni-erlangen.de

Alexander Martin
Alexander.Martin@math.uni-erlangen.de

Maximilian Merkert
Maximilian.Merkert@math.uni-erlangen.de

Christoph Thurner
Christoph.Thurner@math.uni-erlangen.de

Dieter Weninger
Dieter.Weninger@math.uni-erlangen.de

1 Department Mathematik, Lehrstuhl für Wirtschaftsmathematik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstraße 11, 91058 Erlangen, Germany

Keywords Aggregation · Network design · Combinatorial optimization · Mixed-integer programming · Branch-and-cut

Mathematics Subject Classification 90C35 · 90C27 · 90C11 · 90C57

1 Introduction

In modern applications, mathematical optimization problems usually have to be solved for very large instances. As the problems are often NP-hard, this poses challenges to state-of-the-art solution approaches. In fact, some practical instances cannot be solved with the techniques that are currently available. A question that arises naturally is whether the problem sizes can be reduced in practice. Furthermore, it is desirable to maintain global optimality in the sense that an optimal solution to the smaller problem can be translated to a globally optimal solution of the original problem. To this end, decomposition methods as well as aggregation and disaggregation procedures can be used. In this work, we focus on (dis-)aggregation methods.

Aggregation is a coarsening process that omits details but ensures a global view on the complete problem. Disaggregation can be seen as an inverse procedure that reintroduces more detailed information. Aggregation and disaggregation techniques (see [30]) typically combine (aggregate) parts of the original problem and solve the aggregated instance. Then certain decisions are taken. The process is iterated in case the stopping criteria are not satisfied.

Aggregation techniques have frequently been investigated. Balas [2] suggested a solution method for large-scale transportation problems that does not consider all data simultaneously. Zipkin [33] derived a posteriori and a priori bounds for the linear programming case. A comprehensive survey on aggregation techniques has been given in Dudkin et al. [9]. Litvinchev and Tsurkov [25] wrote a book about aggregation in the context of large-scale optimization. In addition, aggregation techniques are applied to a wide field of applications, for example network flow problems [11,20] and the optimization of production planning systems [21,22]. There are only few results about the usability of aggregation techniques in discrete problem settings. Rosenberg [31] described aggregation of equations, Chvátal et al. [4] analyzed aggregation of inequalities and Hallefjord et al. [16] presented column aggregation in integer programming. Aggregation has proven useful for handling highly symmetric problems. In [24] it is one of several tools to grind a very hard problem instance from coding theory; [32] uses aggregation to form a master problem of a decomposition method for multi-activity shift scheduling. Especially, shortest path algorithms based on graph contractions [12] are very successful in practice. Recent examples for the use of aggregation are [26] for a vehicle routing application with time windows and [28] for scheduling the excavation at an underground mine.

The capacitated network design problem (NDP) is well known to be NP-hard [18]. A standard solution method is Lagrangian relaxation, which was first proposed by Geoffrion [14]. Lemarechal [23] evaluated advanced theoretical results, numerical aspects, and related it to other techniques such as column generation. Karwan and Rardin [19] investigated relationships to surrogate duality in integer programming.

Broad studies on the applicability and possible formulations of the Lagrangian dual for multi-commodity network design problems are given in [6,13].

Another approach that has been used intensively for capacitated network design problems is Benders decomposition. This method was first proposed in [3]. More theoretical background is given in [17,27]. Costa [5] gives a broad survey on the application of Benders decomposition to fixed-charge network design problems. A favorable cut selection criterion for Benders cuts is proposed and analyzed computationally in [10].

In practical applications, it is very common that a relatively developed network has to be upgraded in order to allow for the routing of additional demand requirements. In such a network expansion problem, typically only a small percentage of the arcs has to be upgraded. These arcs are frequently referred to as *bottlenecks*. Bottleneck arcs constitute the limiting factor for additional demands to be routed on top of those that can already be accommodated. As an example application, in its initial state in 2010, the German railway network was able to accommodate about 80 % of the forecasted demand for the year 2030. An appropriate upgrade requires capacity-increasing measures on less than 20 % of the tracks. This fact is a motivation to devise an algorithm that continuously updates a set of potential bottleneck arcs, whereas non-bottleneck arcs are aggregated.

This paper is organized as follows. Section 2 introduces the optimization problem. In Sect. 3, we present a description of the iterative aggregation scheme for network expansion that uses master problems and subproblems. We also relate our method to Benders decomposition. In particular, we prove that our method strictly implies the corresponding Benders feasibility cuts. Then, in Sect. 4, we report some implementation details and describe three different iterative aggregation approaches. In Sect. 5, we show computational results on random as well as realistic instances. Although we focus on single-commodity instances, we also report computational results for the multi-commodity case in Sect. 5.3. We end with conclusions in Sect. 6.

2 The model

We consider a bidirected graph $G = (V, A)$, where a set of nodes V and a set of arcs A underlies the problem. Each arc $a \in A$ possesses initial arc capacities $c_a \geq 0$. In addition, each arc can be upgraded by installing a module with an upgrade capacity of C_a at a price of k_a per unit, available in integral multiples y_a. Let δ_v^- be the set of arcs entering node v and δ_v^+ be the set of arcs leaving node v. The flow on arc a is represented by the variable x_a. The aim is to determine a feasible routing of a specified demand vector $d \in \mathbb{R}^{|V|}$ that incurs a minimal cost upgrade of the network while respecting the capacities of the arcs. Although the method is not restricted to single-commodity network design problems, we focus on this case here for ease of exposition. In fact, the procedure can easily be extended to the multi-commodity case. A mixed-integer programming (MIP) formulation of the single-commodity flow network expansion problem is

$$
\begin{array}{lll}
\min & \sum_{a\in A} k_a y_a & \\
\text{s.t.} & \sum_{a\in\delta_v^+} x_a - \sum_{a\in\delta_v^-} x_a = d_v & (\forall v \in V), \\
& x_a \le c_a + C_a y_a & (\forall a \in A), \\
& x \in \mathbb{R}_+^{|A|}, & \\
& y \in \mathbb{Z}_+^{|A|}. &
\end{array}
\tag{P}
$$

The first set of constraints in (P) is referred to as the *flow conservation constraints*, while the second set are the *capacity constraints*.

3 Iterative aggregation scheme

From a bird's eye view, the aggregation scheme for solving (P) proceeds as follows:

1. Partition the node set of the graph into components, i.e., choose an initial aggregation.
2. Master problem: Solve the network expansion problem over the aggregated graph.
3. Subproblem: Check the feasibility of the network upgrade w.r.t. the original graph.
4. (a) In case of feasibility: Terminate and return a network expansion.
   (b) In case of infeasibility: Refine the partition and go to Step 2.

We will prove in Sect. 3.3 that at termination, the returned solution is a global optimum for the original network. The algorithm is detailed in the following sections.

3.1 Graph aggregation and the definition of the master problem

In our context, aggregating a directed graph $G = (V, A)$ means clustering its nodes into subsets. We define the *aggregated graph* $\mathcal{G}_\varphi = (\mathcal{V}_\varphi, \mathcal{A}_\varphi)$ with respect to a surjective *clustering function* $\varphi : V \to \{1, \ldots, k\}$, $k \in \mathbb{N}$, as follows. Its node set $\mathcal{V}_\varphi = \{V_1, \ldots, V_k\}$ is a partition of V into k *components* with $v \in V_i \in \mathcal{V}_\varphi \iff \varphi(v) = i \in \{1, \ldots, k\}$. Its arc set $\mathcal{A}_\varphi$ contains a directed arc from $V_i \in \mathcal{V}_\varphi$ to $V_j \in \mathcal{V}_\varphi$ for each arc $(u, v) \in A$ with $i = \varphi(u) \neq \varphi(v) = j$, i.e., u and v belong to different components. Note that G as well as $\mathcal{G}_\varphi$ are allowed to contain multiple arcs between the same two nodes. Figure 1 illustrates the above definitions. Nodes in the same component are depicted encircled. Only those arcs connecting different components in $\mathcal{V}_\varphi$ are part of $\mathcal{A}_\varphi$.

The aggregated demand vector d_φ is defined via $d_\varphi(V_i) = \sum_{v\in V_i} d_v$ for all $i = 1, \ldots, k$. The capacity $c_\varphi(a)$ and the installable module of an arc $a \in \mathcal{A}_\varphi$ are those of the corresponding original arc. In order to simplify notation, we identify a component $V_i \in \mathcal{V}_\varphi$ with its index i and identify each arc $a \in \mathcal{A}_\varphi$ with the corresponding original one in A. The master problem (P_φ) with respect to $\mathcal{G}_\varphi$ can then be stated as

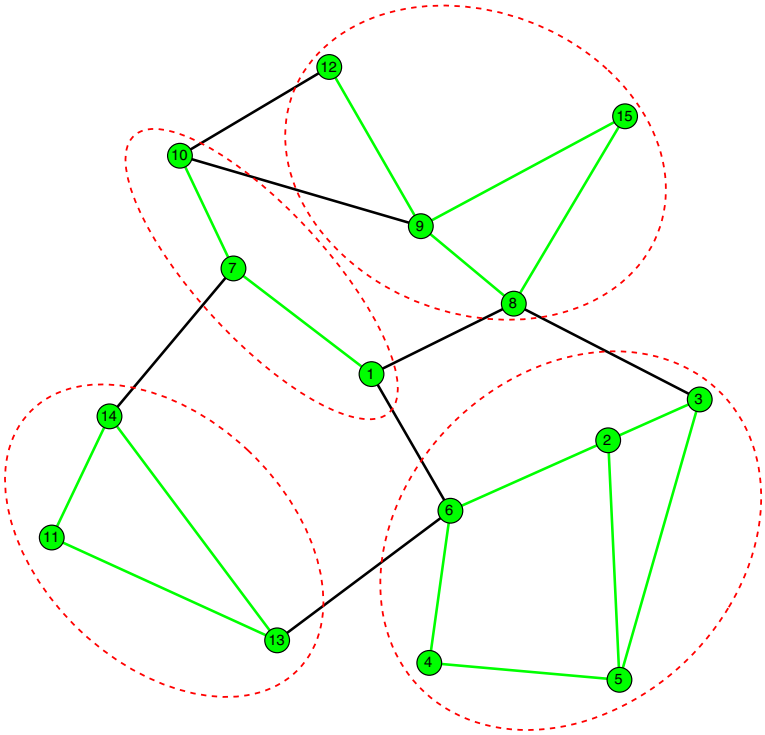

Fig. 1 Aggregation of a graph $G = (V, A)$ with respect to a clustering function φ with $\varphi^{-1}(1) = \{1, 7, 10\}$, $\varphi^{-1}(2) = \{2, 3, 4, 5, 6\}$, $\varphi^{-1}(3) = \{8, 9, 12, 15\}$, and $\varphi^{-1}(4) = \{11, 13, 14\}$

$$
\begin{array}{lll}
\min & \sum_{a \in \mathcal{A}_\varphi} k_a y_a & \\
\text{s.t.} & \sum_{a \in \delta_i^+} x_a - \sum_{a \in \delta_i^-} x_a = d_i & (\forall i \in \mathcal{V}_\varphi), \\
 & x_a \le c_a + C_a y_a & (\forall a \in \mathcal{A}_\varphi), \\
 & x \in \mathbb{R}_+^{|\mathcal{A}_\varphi|}, & \\
 & y \in \mathbb{Z}_+^{|\mathcal{A}_\varphi|}. &
\end{array}
\qquad (\mathrm{P}_\varphi)
$$

We note that by the definition of the aggregated demand vector, the flow conservation constraint at component $i \in \mathcal{V}_\varphi$ is exactly the sum of the original flow conservation constraints that belong to the nodes $v \in V_i$. A capacity constraint for arc $a \in A$ of the original problem (P) is part of $(\mathrm{P}_\varphi) \iff a \in \mathcal{A}_\varphi$ holds. As a result, (P_φ) is a relaxation of the original problem (P). Consequently, the optimal value of (P_φ) with respect to an arbitrary clustering function φ is a lower bound on the optimal value of (P).

By construction of (P_φ), a feasible solution naturally translates to a (not necessarily feasible) solution for (P) by performing two steps: first, all expansion decisions (y-variables) corresponding to arcs within any of the components are set to zero. Second, the induced flows (x-variables) within the components are computed by solving a

maximum-flow problem. This *extendibility test* is the purpose of the subproblems that are derived in the next section.

3.2 Definition of the subproblems and graph disaggregation

The solution of (P_φ) induces new demands within the aggregated components. The purpose of the subproblems is to validate whether these demands can be routed without additional capacity upgrades inside the components. An example of this situation is depicted in Fig. 2.

Let $H_i = (V_i, A_i)$ be the subgraph of $G = (V, A)$ induced by component V_i of the partition of V according to φ. Checking feasibility of a network expansion involves the solution of a maximum-flow problem within each component as follows. The nodes V_i of H_i have an original demand of d_v, $v \in V_i$. The optimal flows of the master problem induce new demands within H_i as each flow x_a on an arc $a = (u, v) \in \mathcal{A}_\varphi$, where $u \in V_i$ and $v \in V_j$, changes the demand of u to $d_u^* := d_u + x_a$ and that of v to $d_v^* := d_v - x_a$. By introducing artificial nodes s as super source and t as super sink, the check whether a feasible flow exists can be formulated as a single-source maximum-flow problem.

A feasible component V_i requires no further examination in the current iteration. An infeasible subproblem can occur for two reasons. It either suggests that the initial capacities within the associated component are not sufficient to route the demands induced by the master problem solution. In other words, the master problem has mistakenly neglected the capacity limitations within the component. The other reason is that the algorithm might not be able to prove that all extension decisions are already optimal. We explain in Sect. 4.4, paragraph *Global testing*, how to detect this case. When an infeasible subproblem is encountered, the partition is refined in order to

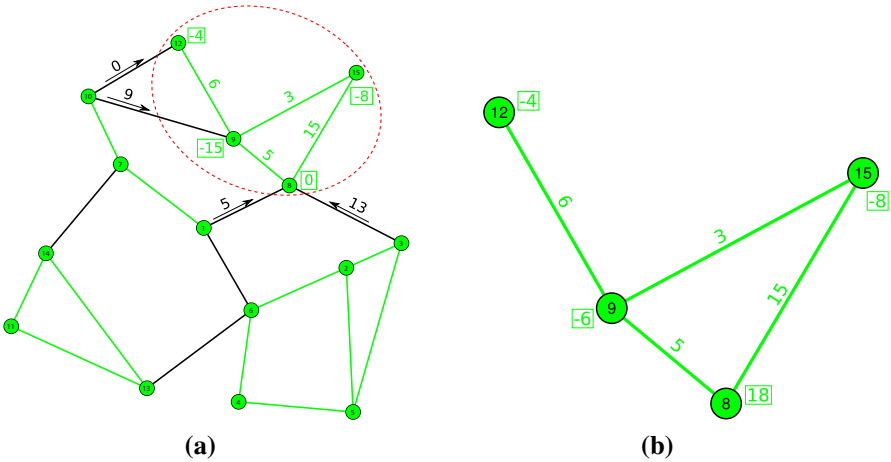

Fig. 2 Illustration of a subproblem of the aggregation scheme: **a** shows part of the solution of the master problem which sends flow into component {8, 9, 12, 15} via several arcs. The corresponding subproblem for this component is depicted in **b**

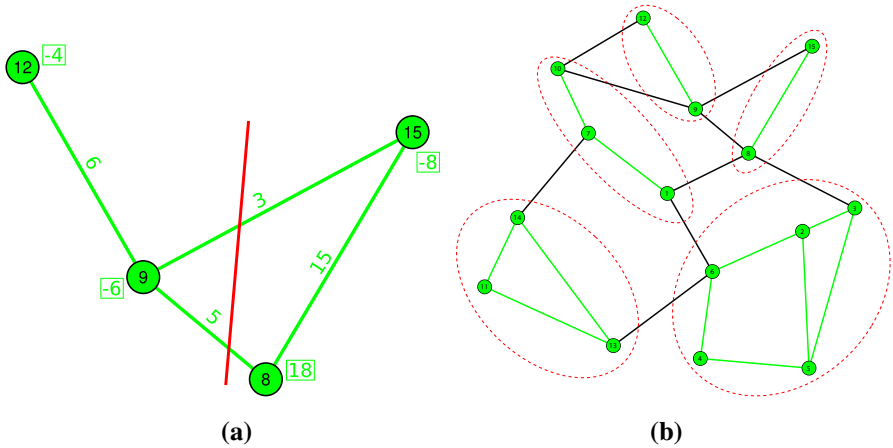

Fig. 3 Disaggregation of an infeasible component. **a** Disaggregation along a minimal cut. **b** Resulting graph after disaggregation

consider additional arcs in the master problem. This arc set is chosen as a minimum s-t-cut that limits the flow, see Fig. 3a. The algorithm terminates as soon as all subproblems are feasible.

Updating the master problem (P_φ) is done by disaggregating the infeasible component along this minimum cut. Let w.l.o.g. V_k be an infeasible component and let $V_k^1, \ldots, V_k^l$ be the components into which V_k disaggregates. We define a new clustering function $\overline{\varphi} : V \to \{1, \ldots, k+l-1\}$ with $\overline{\varphi}(v) = \varphi(v)$ if $v \notin V_k$ and $\overline{\varphi}(v) = k+i$ if $v \in V_k^i \subset V_k$, see Fig. 3b.

3.3 Correctness of the algorithm

We show next that for non-negative expansion costs, the above method always terminates with an optimal solution to the original network expansion instance.

Theorem 3.1 (Correctness of the Algorithm) *For non-negative expansion costs k in* (P)*, the proposed algorithmic scheme always terminates after a finite number of iterations with an optimal solution to the network expansion problem for the original graph.*

Proof Termination follows from the fact that only finitely many disaggregation steps are possible until the original graph is reached. Clearly, the returned solution is feasible for the original network by the termination criterion.

In order to prove optimality, let (x^*, y^*) be the optimum solution of the final master problem on $\mathcal{G}_\varphi$ that is extended to a solution of the original problem as described above and let $(\overline{x}, \overline{y})$ be an arbitrary solution to the network expansion problem for the original graph G. We split the expansion costs of the two solutions into the cost k_M arising at the arcs $\mathcal{A}_\varphi$ of $\mathcal{G}_\varphi$ and the cost k_S arising at $A \backslash \mathcal{A}_\varphi$. The corresponding expansion variables are y_M and y_S. We derive

$$k^T y^* = k_M^T y_M^* + k_S^T y_S^* = k_M^T y_M^*$$

as no expansion is performed within the components. As y_M^* belongs to an optimal upgrade of $\mathcal{G}_\varphi$, we conclude

$$k_M^T y_M^* \leq k_M^T \overline{y}_M.$$

As the expansion costs k are non-negative, it is

$$k_M^T \overline{y}_M \leq k_M^T \overline{y}_M + k_S^T \overline{y}_S = k^T \overline{y}.$$

Altogether, we have shown $k^T y^* \leq k^T \overline{y}$, and therefore (x^*, y^*) is an optimal solution to the original problem. □

3.4 Relation to Benders decomposition

The aggregation procedure developed in this paper possesses some obvious similarities to Benders decomposition. Both algorithms solve a succession of increasingly stronger relaxations of the original problem, which is achieved by introducing cutting planes. In the case of the aggregation framework, these are part of the primal (aggregated) flow conservation and capacity constraints. For Benders decomposition, these are the Benders feasibility and optimality cuts. Both algorithms stop as soon as the optimality of the relaxed solution is proven. Furthermore, the subproblem used in the aggregative approach coincides with the subproblem in Benders decomposition if the x-variables for the arcs in $\mathcal{A}_\varphi$ are chosen to belong to the Benders master problem.

However, there are also substantial differences. The aggregation scheme introduces both new variables and constraints in each iteration to tighten the master problem formulation. Contrary to this, Benders decomposition is a pure row-generation scheme. Equally important is the fact that the continuous disaggregation of the network graph leads to a shift in the proportions between the master and the subproblem. The master problem grows in size, while the subproblem tends to shrink as bottleneck arcs are transferred from inside the components to the master graph. In comparison, Benders decomposition leaves these proportions fixed.

The following therorem details the relation between the subproblem information used in the two algorithms.

Theorem 3.2 *Let φ be a clustering function according to a given network graph G. For a disaggregation of G along a minimal cut, the primal constraints introduced to the master problem* (P_φ) *in the proposed aggregation scheme strictly imply the Benders feasibility cut obtained from the corresponding subproblem.*

Proof We prove the claim for the special case where the whole graph is aggregated to a single component, i.e., $\varphi \equiv 1$. The corresponding situation in Benders decomposition is that all arc flow variables are projected out of the master problem. The extension of the arguments to the general case is straightforward.

For both algorithms, the subproblem consists in finding a feasible flow in the network and thus a solution to the following feasibility problem:

$$\begin{array}{lll} \min & 0 & \\ \text{s.t.} & \sum_{a\in\delta_v^+} x_a - \sum_{a\in\delta_v^-} x_a = d_v & (\forall v \in V) \\ & x_a \le c_a + C_a y_a^* & (\forall a \in A) \\ & x \in \mathbb{R}_+^{|A|}, & \end{array}$$

where y^* corresponds to the network design solution determined by the master problem. Let α denote the dual variables of the flow conservation constraints and β those of the capacity constraints. In case of infeasibility, Benders decomposition derives its feasibility cut from an unbounded ray of the dual subproblem:

$$\begin{array}{lll} \max & \sum_{v\in V} d_v\alpha_v - \sum_{a\in A}(c_a + C_a y_a^*)\beta_a & \\ \text{s.t.} & \alpha_u - \alpha_v - \beta_a \le 0 & (\forall a = (u,v) \in A) \\ & \beta \in \mathbb{R}_+^{|A|}. & \end{array}$$

Variables β can be eliminated from the problem, as any feasible solution $(\overline{\alpha}, \overline{\beta})$ is dominated by the solution (α^*, β^*) with $\alpha^* = \overline{\alpha}$ and

$$\beta_a^* = \max\{\overline{\alpha}_u - \overline{\alpha}_v, 0\}$$

for $a = (u, v) \in A$. Note that the dual subproblem is exactly that of determining a minimal cut in the network. Thus, an arc $a \in A$ is part of the minimal cut if and only if $\beta_a^* > 0$, which is exactly the case when the flow conservation constraints corresponding to the two end nodes of a obtain different dual values. In case of an infeasible (primal) subproblem, the Benders cut can now be written as

$$\sum_{v\in V} \alpha_v^* d_v \le \sum_{a\in A} \max\{\alpha_u^* - \alpha_v^*, 0\}(c_a + C_a y_a),$$

for α^* belonging to an unbounded dual ray (α^*, β^*). For the same infeasible master solution, the aggregation scheme adds the following system of inequalities to its master problem:

$$\sum_{a\in\delta_{V_i}^+} x_a - \sum_{a\in\delta_{V_i}^-} x_a = d_i \quad (\forall V_i \in \mathcal{V}_{\overline{\varphi}})$$

and

$$x_a \le c_a + c_a y_a \quad (\forall a \in \mathcal{A}_{\overline{\varphi}} \backslash \mathcal{A}_\varphi)$$

together with the new variables x_a for $a \in \mathcal{A}_{\overline{\varphi}} \backslash \mathcal{A}_\varphi$, where $\overline{\varphi}$ is the aggregation induced by the minimal cut. As stated above, the dual subproblem values ${\alpha_u}^*$ and α_v^* coincide if nodes u and v belong to the same component $V_i \in \mathcal{V}_\varphi$. Thus we can define $\alpha_i := {\alpha_u}^*$ for some $u \in V_i$. The claim then follows by taking the sum of the aggregated flow

conservation constraints weighted with $-\alpha_i^*$ and the aggregated capacity constraints weigthed with $-\max\{\alpha_u{}^* - \alpha_v^*, 0\}$. For this weighting, the left hand side becomes

$$\sum\nolimits_{V_i \in \mathcal{V}_\varphi} \alpha_i^* \left(\sum_{a \in \delta_{V_i}^+} x_a - \sum_{a \in \delta_{V_i}^-} x_a \right) - \sum_{a=(u,v) \in \mathcal{A}_\varphi} \max\{\alpha_u^* - \alpha_v^*, 0\} x_a,$$

which can be transformed to

$$\sum_{a=(u,v) \in \mathcal{A}_\varphi} (\alpha_u^* - \alpha_v^* - \max\{\alpha_u^* - \alpha_v^*, 0\}) x_a.$$

This yields

$$\sum_{\substack{a=(u,v) \in \mathcal{A}_\varphi: \\ \alpha_v^* \geq \alpha_u^*}} (\alpha_u^* - \alpha_v^*) x_a,$$

which is non-positive due to the non-negativity of x. Thus, we find

$$\sum_{v \in V} \alpha_v^* d_v = \sum_{V_i \in \mathcal{V}_\varphi} \alpha_i^* d_i \leq \sum_{\substack{a=(u,v) \in \mathcal{A}_\varphi: \\ \alpha_v^* \geq \alpha_u^*}} (\alpha_u^* - \alpha_v^*) x_a + \sum_{a \in A} \max\{\alpha_u^* - \alpha_v^*, 0\}(c_a + C_a y_a),$$

which implies the Benders cut. Finally, this inequality is strict for all solutions to the problem, where flow is sent both along a certain arc as well as along its opposite arc. This completes the proof. □

The theorem above shows that each iteration of the aggregation scheme introduces more information to the master problem than a Benders iteration. Whereas Benders decomposition is often used to solve network design problems, it belongs to the folklore of mathematical programming that the original Benders cuts are weak and numerically instable already for small-scale networks. And this becomes even more problematic in the case of large-scale networks as they are considered here. Therefore, Benders decomposition is most commonly employed for smaller network problems with complicating constraints, such as they arise in the context of demand stochasticity.

4 Implementation

We have implemented three versions of the aggregation scheme. The first one represents what has been described so far. It has the obvious drawback that only very limited information is used when moving from one iteration to the next. To overcome this partly, we integrate the aggregation scheme into a branch-and-bound framework. Finally, we study a hybrid of both.

4.1 Sequential aggregation (SAGG)

The *Sequential Aggregation Algorithm* (SAGG) works in a strictly sequential manner: in each iteration, the network expansion master problem is solved to optimality. In case of feasibility of all subproblems, the algorithm terminates. Otherwise, the graph is disaggregated as described in the previous section, see Fig. 4a for a schematic example.

For speeding up the first iterations, we solve the network expansion linear programming (LP) relaxations only and disaggregate according to the obtained optimal solutions. Our experiments suggest that the savings in runtime compensate for the potentially misleading first disaggregation decisions based on the LP relaxation. Note that the proof of Theorem 3.1 does not require y to be integral.

4.2 Integrated aggregation (IAGG)

In SAGG, most information, including bounds and cutting planes, is lost when proceeding from one iteration to the next. Only the disaggregation is processed to the next iteration. The idea behind the *Integrated Aggregation Algorithm* (IAGG) is to use more

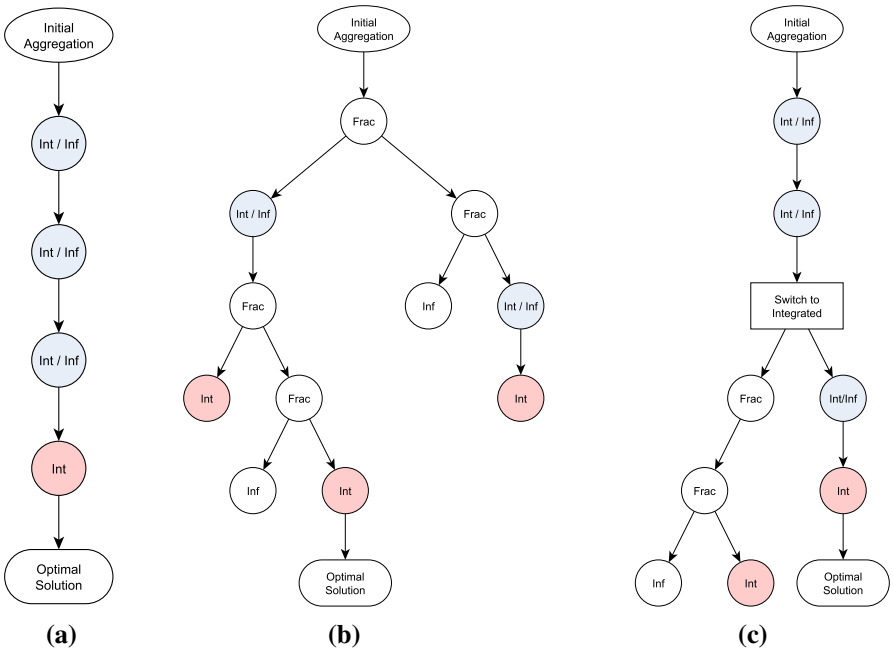

Fig. 4 Schematic outline of the three aggregation schemes. Nodes labeled *Int* correspond to feasible integral solutions of the current master problem, whereas *Int / Inf* indicates that such a solution is infeasible for the original problem, leading to disaggregation. For the branch-and-bound trees, white nodes labeled *Inf* or *Frac* represent infeasible and fractional branch-and-bound nodes (at which branching might occur), respectively. **a** The sequential algorithm (SAGG). **b** The integrated algorithm (IAGG). **c** The hybrid algorithm (HAGG)

of this information by embedding the disaggregation steps into a branch-and-bound tree.

We start with an initial aggregation and form the corresponding master problem. Each integral solution (x, y) found during the branch-and-bound search is immediately tested for extendibility. In case of feasibility, we keep (x, y) as an incumbent solution, otherwise we disaggregate the graph and refuse (x, y), see Fig. 4b for a schematic example. As noted in Sect. 3, aggregation of a network can be performed by removing some constraints and adding up some others from the formulation of the original instance. All constraints from an aggregated graph remain valid for the disaggregated network. Disaggregating a network amounts to inserting the flow conservation constraints for the new components and the arc capacity constraints for the arcs entering the master problem to the problem formulation.

4.3 The hybrid aggregation algorithm (HAGG)

A natural composition of SAGG and IAGG is the *Hybrid Aggregation Algorithm* (HAGG). It starts with a number of sequential iterations such as in SAGG, and then switches to the integrated scheme (Fig. 4c). The idea is to have more information about the graph available at the root node of the branch-and-bound tree once we start to employ the integrated scheme. This is favorable for the cutting planes generated at the root node.

For the first iterations, HAGG and SAGG behave exactly the same. Namely, we solve the LP relaxation of the master problem and proceed with the obtained fractional solution. As a heuristic rule, we switch from the sequential to the integrated scheme when the value of the LP relaxation of the master problem is equal to the value of the LP relaxation of the original problem. Then the optimal fractional expansion is found and the LP relaxation does not give any further disaggregation information.

4.4 Details of the implementation

Initial aggregation The easiest choice is to first aggregate the whole graph to a single vertex so that the disaggregation is completely determined by the minimum-cut strategy. As discussed above, this might not be the most suitable choice for the integrated scheme, which is the motivation for a hybrid scheme. In fact, one can view HAGG as being IAGG with a special heuristic for finding the initial aggregation.

Solving the subproblems The subproblems are maximum-flow problems that are solved by a standard LP solver, which is fast in practice. A potential speedup by using specialized maximum-flow implementations is negligible as the implementation spends almost all of the time in the master problems and not in the subproblems.

Disaggregation We have seen how disaggregation works for a single component in Sect. 3. However, in case of several infeasible components, it is not clear beforehand whether all of them should be disaggregated, or, otherwise, which one(s) should be used for disaggregation. In our implementation, we always disaggregate all infeasible

components, which aims at minimizing the number of iterations. Experiments with other disaggregation policies did not lead to considerable improvement. In addition, we also split components that are not arc-connected into their connected components.

Global testing It should be mentioned that the current network expansion might be optimal although the extendibility test fails for some component. This is because in the subproblems not only the expansions (y-variables) are fixed but also the flow on all edges contained in the master problem. To overcome this problem we can use a simple global test: we fix the expansion and check feasibility of the resulting maximum-flow problem on the complete graph. The case where global testing saves at least one iteration happened regularly in our experiments. As an iteration is relatively expensive, global testing should definitely be included in the default settings. Note that global testing cannot replace the subproblems completely since the cut obtained from a global test might only consist of edges that are already in the master problem. Hence after finding a solution of the master problem, we first apply global testing and in case of infeasibility we use the subproblems to determine how to disaggregate. As global testing just consists of solving one maximum-flow problem, it is performed in each iteration.

5 Computational results

The computational experiments have been performed on a queuing cluster of Intel Xeon E5410 2.33 GHz computers with 12 MB cache and 32 GB RAM, running Linux in 64 bit mode. We have implemented our framework using the C++-API of Gurobi 5.5 [15]. For IAGG and HAGG, it was necessary to adjust Gurobi's parameter settings, which involves a more aggressive cutting plane generation, a focus on improving the bound and downscaling the frequency of the heuristics. Additionally, since those algorithms use *lazy cuts*, dual reductions had to be disabled in order to guarantee correctness. Implementation SAGG uses Gurobi's standard parameter settings. Each job was run on 4 cores and with a time limit of 10 hours.

We compare our aggregation schemes to the solution of the original network expansion integer program (P) by Gurobi 5.5 using standard parameter settings. These reference solution times are denoted by MIP. For MIP, experiments with different parameter settings did not lead to considerably better running times.

5.1 Benchmark instances

The aggregation schemes are tested on three different sets of benchmark instances. These are random scale-free networks according to the preferential attachment model [1], instances created from the *rome99* graph from the 9th DIMACS challenge [7], and instances arising from single-commodity adaptions of the SNDlib instances [29]. In the first two cases, we randomly drew the vector d of demands as well as the vector c of initial arc capacities. For the SNDlib instances, the original initial capacities were retained whenever available and the multi-commodity demands were balanced to obtain single-commodity problems. To demonstrate the extendibil-

ity to the multi-commodity case, we include preliminary results for random scale-free networks with a few commodities.

The initial capacities of each instance were scaled by a constant factor in order to obtain different percentages of initial demand satisfaction l, which was done by solving an auxiliary network flow problem. The parameter l indicates which portion of the demand can be routed given the initial state of the network. For different instance sizes, varying the initial capacities has a significant impact on the solution time and the solvability in general, and is therefore an important parameter for the forthcoming analysis.

Whenever the generation of instances included random elements, we generated 5 instances of the same size and demand satisfaction. The solution times then are (geometric) averages over those five instances. If only a subset of the 5 instances was solvable within the time limit, the average is taken over this subset only. We also state the number of instances that could be solved within the time limit.

5.2 Computational results on scale-free networks with single-commodity demand

The topology of the instances in this benchmark set has been generated according to a preferential attachment model. It produces so-called scale-free graphs [1], which are known to represent the evolutionary behavior of complex real networks well. Starting with a small clique of initial nodes, the model iteratively adds new nodes. Each new node is connected to m of the already existing nodes. This parameter m, the so-called *neighborhood parameter*, influences the average node degree. We set $m = 2$ in order to generate sparse graphs that resemble infrastructure networks. We note that in our experimental computations choosing higher values of m did not influence the results significantly. Furthermore, we chose 80 % of the nodes as terminals, i.e., nodes with non-zero demand, in order to represent a higher but not overly conservative load scenario. The module capacities for these instances were chosen as 0.25 % of the total demand in order to obtain reasonable module sizes with respect to the scale of the demand. Varying these two parameters did not lead to significantly different results either. Finally, the module costs were drawn randomly.

5.2.1 Computational results for small instances

In this subsection, we analyze the aggregation method for small instances with different levels l of initial demand satisfaction. We generated random scale-free networks with 100 nodes. Furthermore, we considered a percentage $l \in \{0, 0.05, 0.1, 0.2, \ldots, 0.8, 0.9, 0.95\}$. First, we determine which implementation of the aggregation scheme performs best. In a second step, we compare the best implementation with MIP.

In Table 1, we report the solution times in seconds, each averaged over five instances with the same value of l. If not all instances could be solved within the time limit, the average is taken over the subset of solved instances. The first number in each cell states the number of solved instances, whereas the second gives the average solution time

Table 1 Number of instances solved/average solution times (s) for the three aggregation algorithms for random scale-free networks with $|V| = 100$ nodes and varying level of initial demand satisfaction

l	IAGG	SAGG	HAGG
0	**3/196.1**	3/502.30	3/230.85
0.05	**5/22.44**	5/186.98	5/63.04
0.1	**5/111.15**	4/676.27	4/106.75
0.2	**5/34.57**	5/110.92	5/55.57
0.3	**5/8.02**	5/25.69	5/10.16
0.4	**5/8.01**	5/35.03	5/8.47
0.5	5/4.18	5/5.65	**5/2.71**
0.6	**5/5.63**	4/4.23	5/6.07
0.7	5/1.17	**5/0.81**	5/0.82
0.8	5/0.57	**5/0.49**	5/0.51
0.9	**5/0.23**	5/0.30	5/0.25
0.95	**5/0.12**	5/0.21	5/0.18

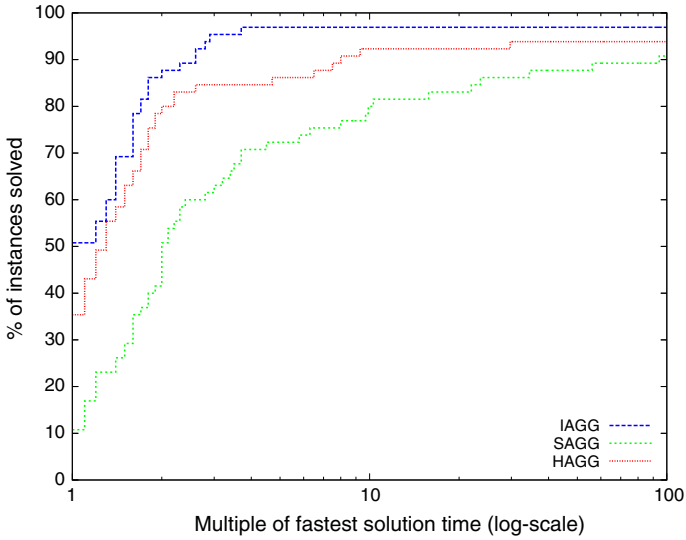

Fig. 5 Performance profile with performance metric solution time (s) for the three aggregation algorithms on random scale-free networks with $|V| = 100$ nodes

in seconds. Note that we apply the geometric mean for the average values in order to account for outliers. If a method could not solve any of the five instances for a given l, we denote this by an average solution time of "∞". The fastest method in each row is emphasized with bold letters. As we assume that those instances that could not be solved within the time limit are very difficult, we rank the methods first by the number of solved instances and second by the average solution time.

The results for the instances from Table 1 are also presented as a performance profile in Fig. 5. For each aggregation method, the percentage of all solved scale-free instances with $|V| = 100$ is shown as a function of the solution time that is given in multiples of the time the fastest method needed in order to solve it. The information

Table 2 Number of instances solved/average solution times (s) of MIP and IAGG for random scale-free networks with $|V| = 100$ nodes and varying values of l

l	MIP	IAGG
0	3/1373.11	**3/196.1**
0.05	5/266.07	**5/22.44**
0.1	4/904.29	**5/111.15**
0.2	5/136.43	**5/34.57**
0.3	5/29.30	**5/8.02**
0.4	5/28.16	**5/8.01**
0.5	**5/2.4**	5/4.18
0.6	5/19.81	**5/5.63**
0.7	**5/0.21**	5/1.17
0.8	**5/0.08**	5/0.57
0.9	**5/0.05**	5/0.23
0.95	**5/0.04**	5/0.12

deduced from this kind of plot is twofold. First, the intercept of each curve with the left vertical axis shows the percentage of instances for which the corresponding method achieves the shortest solution time. Thus, the method attaining the highest intercept on the vertical axis is the one which solves the highest number of instances in the shortest time. Second, for each value m on the horizontal axis, the plot shows the percentage of instances that a method was able to solve within m times the shortest solution time achieved by any of the methods. In particular, the vertical intercept corresponds to $m = 1$. The interpretation of this information is how good a method is in catching up on instances for which it is not the fastest. A more detailed introduction to performance profiles can be found in [8].

We see that IAGG performs best for the scale-free networks with 100 nodes when compared to SAGG and HAGG. It solves the majority of instances within the shortest solution time and solves 97 % of the instances, which is the largest value among the three methods. Furthermore, there is no instance for which it requires more than 4 times the shortest solution time.

In Table 2, we thus compare IAGG with MIP. We find that our aggregation approach is beneficial whenever the instance cannot trivially be solved within a few seconds. For instances with small initial demand satisfaction, we observe significantly faster solution times for IAGG, and we see that it is able to solve more instances within the time limit. Even without any preinstalled capacities ($l = 0$), the aggregation scheme attains an average solution time which is 6 times smaller than that of MIP. From $l = 0.7$ upwards, the running times of both algorithms are negligible, and the tiny advantage for MIP can be attributed to the overhead caused by performing the aggregation. The superior performance on instances with small initial demand satisfaction seems surprising. It contrasts the fact that the number of components in the final state of network aggregation in IAGG converges to the number of nodes in the original instance. This is presented in Fig. 6, where we show the average number of components in the final iteration as a function of the percentage of initial demand satisfaction.

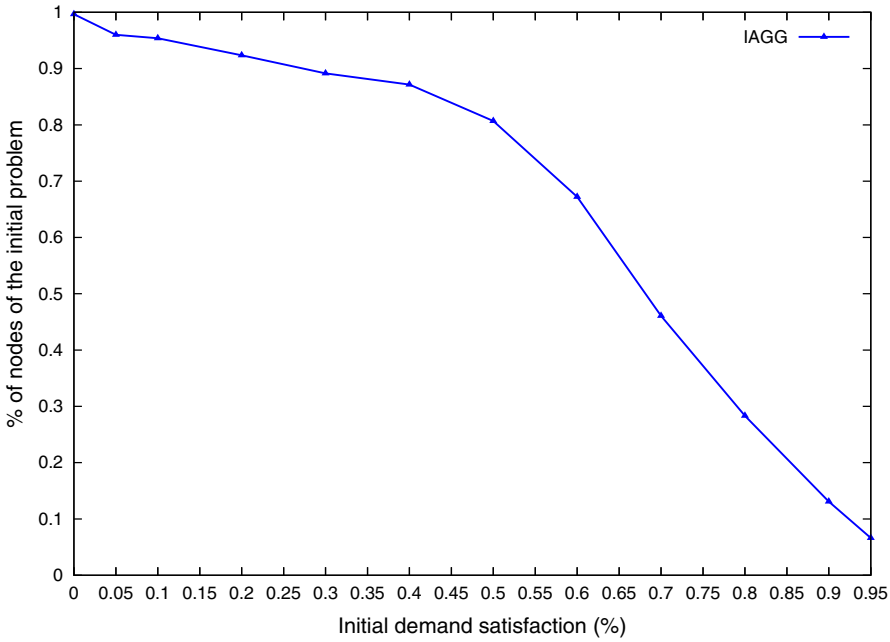

Fig. 6 Average number of components in the last iteration of IAGG in relation to the original $|V| = 100$ nodes for random scale-free networks for varying level of initial demand satisfaction

Obviously, the aggregation framework performs better than the standard approach MIP even in case of complete disaggregation. In order to determine what causes this behavior, we tested whether the aggregation approach could determine more effective branching decisions within the branch-and-bound procedure. To this end, in the standard solver MIP, we increased the branching priority on variables that enter the master problems of the aggregation scheme in early iterations. We found that these branching priorities did not lead to better running times for MIP. This suggests that the cutting planes generated within the aggregation procedure are more powerful than the ones generated within MIP.

5.2.2 Behavior for medium-sized and large instances

We now consider larger instances within a range of 1000 to 25,000 nodes as well as an initial demand satisfaction from 60 to 98 %. We applied a selection rule to sort out instances which are "too easy" or "too hard" to solve. We required that for at least three out of five instances per class, any of the four methods has a solution time in the time interval reaching from 10 s to the time limit of 10 h. Table 3 lists the instances which comply with this selection rule. Note that the relevant instances can be located mainly on the diagonal as an increasing instance size requires an increasing level of initial capacities in order to remain solvable within the time limit.

Figure 7 shows the performance profile for the medium-sized instances with up to 5000 nodes. We observe that for these instances, the HAGG implementation performs

Table 3 Instance sizes $|V|$ and extension degrees l that comply with the selection rule for medium-sized and large instances are marked by an × in the table

$\|V\|\backslash l$	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.92	0.94	0.96	0.98
1000	×	×	×	×	×	×					
2000				×	×	×	×	×			
3000					×	×	×	×	×		
4000						×	×	×	×	×	
5000						×	×	×	×	×	×
10,000							×	×	×	×	×
15,000										×	×
20,000											×
25,000											×

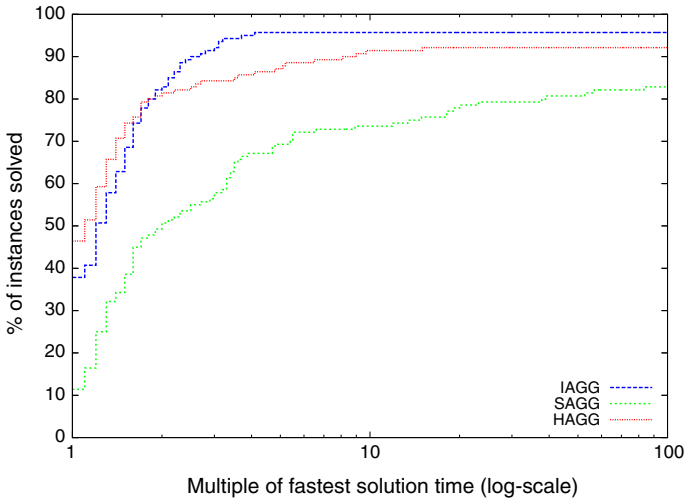

Fig. 7 Performance profile for the medium-sized random scale-free networks from Table 3 with up to $|V| = 5000$ nodes, comparing the three aggregation schemes

best and IAGG is almost as good. Accordingly, these results suggest the choice of one of those two methods. However, the picture changes for the large instances with at least 10,000 nodes, which have a high level of pre-installed capacities, see Fig. 8. Here, HAGG clearly performs very poorly. Instead, SAGG solves a majority of the instances fastest (42 %), while IAGG performs only slightly worse.

As a result of Figs. 7 and 8, we come to the conclusion that the overall best choice is IAGG, as it is not much worse than HAGG on the medium-sized instances and much better on the large networks. Furthermore, it outperforms both other implementations when considering small multiples of the shortest solution times.

The comparison between IAGG and MIP on the instances from Table 3 is shown in Table 4. The instances are grouped by initial demand satisfaction. We see that IAGG is better comparing the average solution times for almost all instances under

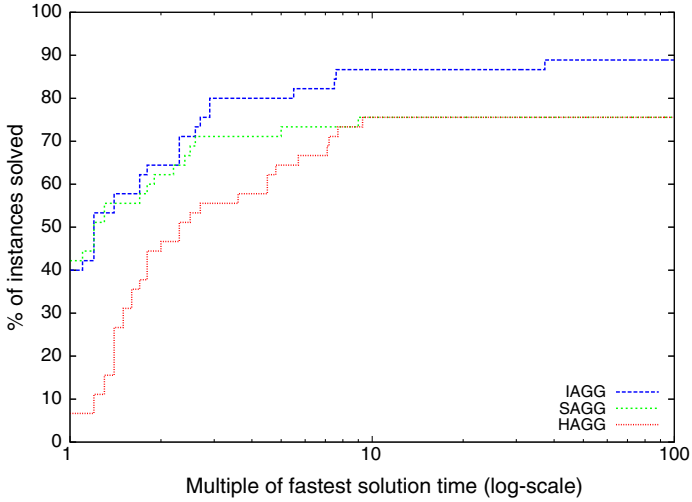

Fig. 8 Performance profile for the large random scale-free networks from Table 3 from $|V| = 10{,}000$ nodes on, comparing the three aggregation schemes

Table 4 Number of instances solved/average solution times (s) of MIP and IAGG for random scale-free instances with $|V|$ nodes and initial demand satisfaction l

$\|V\|$	l	MIP	IAGG	$\|V\|$	l	MIP	IAGG
1000	0.75	4/142.31	**5/92.4**	4000	0.92	5/979.59	**5/51.89**
2000	0.75	3/6715.29	**4/1583.72**	5000	0.92	5/1499.94	**5/61.39**
1000	0.8	4/52.75	**5/34.25**	10,000	0.92	$0/\infty$	**3/6139.25**
2000	0.8	5/944.84	**5/143.04**	3000	0.94	5/21.08	**5/13.17**
3000	0.8	2/6605.03	**3/1806.96**	4000	0.94	5/121.86	**5/21.3**
1000	0.85	5/19.68	**5/9.31**	5000	0.94	5/450.84	**5/36.23**
2000	0.85	5/140.75	**5/43.15**	10,000	0.94	3/9004.99	**5/338.51**
3000	0.85	5/2783.47	**5/397.87**	3000	0.96	**5/5.62**	5/11.76
4000	0.85	3/14164.16	**5/884.52**	4000	0.96	5/30.08	**5/8.76**
5000	0.85	1/31214.39	**3/5484.7**	5000	0.96	5/65.41	**5/20.21**
2000	0.9	5/24.39	**5/12.78**	10,000	0.96	4/4012.20	**5/109.48**
3000	0.9	5/340.59	**5/35.94**	15,000	0.96	2/25343.05	**5/2395.53**
4000	0.9	5/1433.56	**5/115.57**	5000	0.98	**5/6.7**	5/9.53
5000	0.9	5/3113.36	**5/195.83**	10,000	0.98	5/421.14	**5/33.07**
10,000	0.9	$0/\infty$	**3/11951.6**	15,000	0.98	5/3690.52	**5/90.5**
2000	0.92	5/11.32	**5/10.14**	20,000	0.98	5/10,982.81	**5/470.47**
3000	0.92	5/63.97	**5/24.38**	25,000	0.98	2/22,687.73	**4/4372.76**

consideration. A special remark is to be made on the total number of solved instances, which is the first number in each cell. Here, we see that within the time limit, IAGG can always solve at least as many instances as MIP, often more. Furthermore, we note

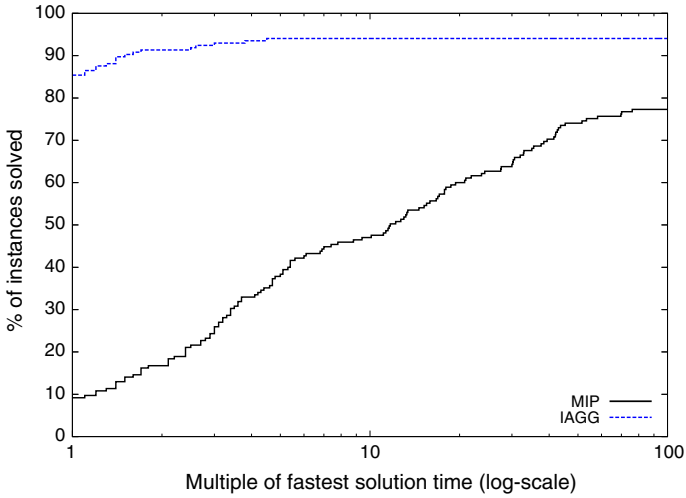

Fig. 9 Performance profile for all instances from Table 3 comparing MIP and IAGG

that even though the number of solved instances is larger for IAGG, the geometric mean of the solution times is still lower compared to the solution times of MIP. Thus, IAGG solves the instances significantly faster than the standard MIP approach.

These statements are underlined by the performance profile over the same instances, which is shown in Fig. 9. The aggregation scheme IAGG clearly outperforms the standard approach MIP. It solves about 86 % of all instances fastest. In addition, IAGG was able to solve a higher percentage of the overall number of instances within the time limit when compared to MIP.

To investigate why the aggregation scheme solves the instances so much faster, we examine the average number of network components in the final iteration for four selected instance sizes, $|V| \in \{1000, 2000, 3000, 4000\}$, as this number strongly influences the size of the aggregated network design problem, see Fig. 10.

The results are comparable to those for random scale-free networks with $|V| = 100$ nodes as shown in Fig. 6. Due to the larger size, these instances could only be solved for higher levels of initial demand satisfaction, for example at least 60 % for graphs with 1000 nodes. The plot shows that the aggregation algorithms can indeed reduce the number of nodes significantly when compared to the number of nodes in the original graph. Hence, the NDP master problems are much smaller than the original NDP. It is also interesting to note that for the tested instances, the factor by which the number of components is smaller compared to the original graph increases with the number of nodes. That means that for the large evaluated instances, the number of components is significantly smaller than the original number of nodes. Consequently, IAGG obtains a big advantage in computation times and a higher number of solved instances as can be seen in Table 4.

As an example, Table 5 presents the results for the instances with $V = 3000$ nodes. For different values of l, average solution times of the aggregation schemes are compared with those of MIP.

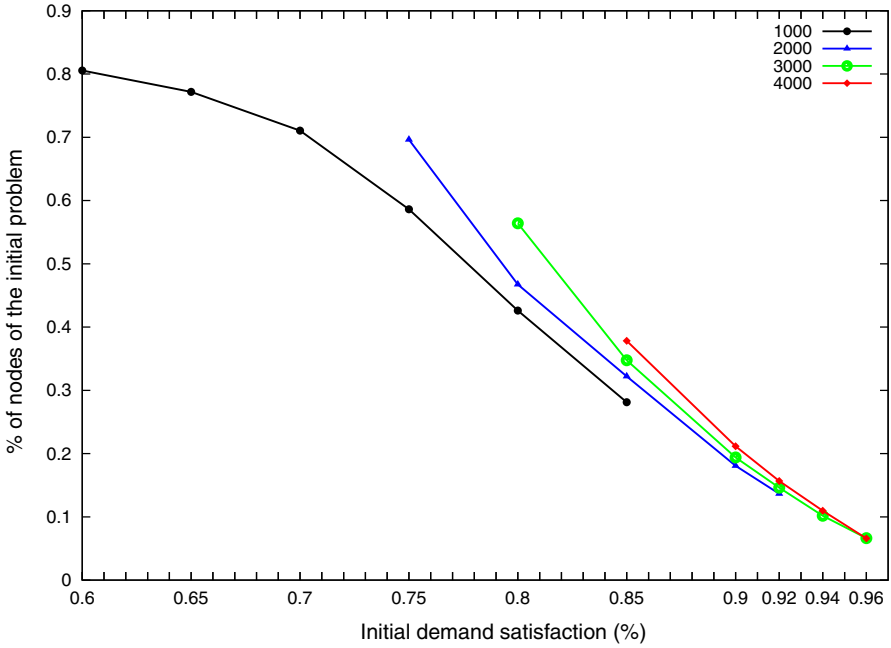

Fig. 10 Average number of components in the last iteration of IAGG and the number of original nodes for $|V| \in \{1000, 2000, 3000, 4000\}$ for random scale-free networks with varying level of initial demand satisfaction

Table 5 Number of instances solved/average solution times (s) for random scale-free networks with $|V| = 3000$ nodes and initial demand satisfaction l

l	MIP	IAGG	SAGG	HAGG
0.8	2/6605.03	**3/1806.96**	2/5914.47	2/1299.07
0.85	5/2783.47	**5/397.87**	5/3761.89	5/623.59
0.9	5/340.59	5/35.94	5/58.96	**5/28.68**
0.92	5/63.97	5/24.38	**5/16.23**	5/18.00
0.94	5/21.08	5/13.17	**5/9.84**	5/10.44
0.96	**5/5.62**	5/11.76	5/7.99	5/7.81

In total, these results for medium and large instances confirm our findings for instances with $|V| = 100$ nodes. Namely, MIP is vastly outperformed by the aggregation schemes. IAGG generally performs best with respect to the number of solved instances and with respect to the solution times.

5.3 Computational results on scale-free networks with multi-commodity demand

To show the wider applicability of the proposed aggregation framework, we state some preliminary results on instances with a small number of commodities. These instances

Table 6 Number of instances solved/average solution time (s) for small multi-commodity instances with $|V| = 100$ nodes, an initial demand satisfaction of $l = 0.8$, and an increasing number of commodities b

b	MIP	IAGG	SAGG	HAGG
5	5/7.13	5/9.72	**5/6.78**	5/6.93
10	5/258.43	**5/52.47**	5/126.54	5/86.12
15	4/166.98	5/147.35	5/334.15	**5/140.81**
20	3/966.64	**4/337.68**	4/1779.86	4/363.40
25	3/2358.80	**5/876.46**	3/2678.55	4/2258.37

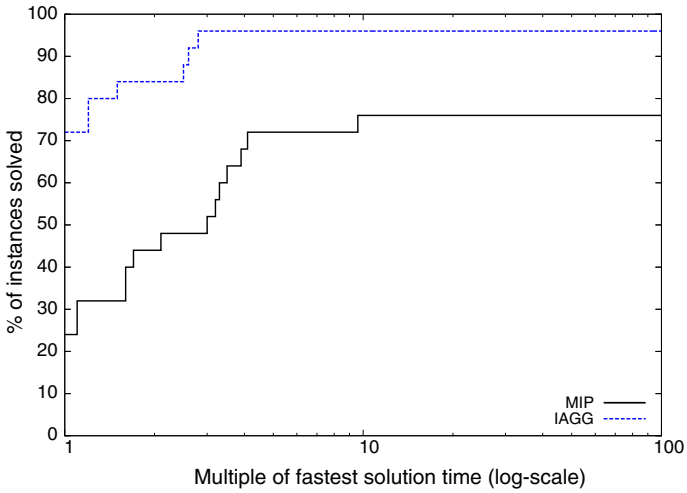

Fig. 11 Performance profile comparing MIP and IAGG on random multi-commodity scale-free networks with $|V| = 100$ nodes

are random scale-free networks with $|V| = 100$ and $l = 0.8$ as well as medium-sized networks with $|V| = 3000$ and $l = 0.96$. The short solution times for these settings obtained in the single-commodity case allow for a multi-commodity study with a varying number of commodities b, for which the demands were again randomly drawn.

Table 6 compares the number of instances solved and the average solution times obtained by MIP and the three implementations of our aggregation scheme for the small instances with up to 25 commodities. We see that an increasing number of commodities increases the difficulty of the problem significantly. On the one hand, this is due to the obvious fact that the subproblems are now multi-commodity network flow problems. On the other hand, we also observe that the graphs tend to be disaggregated much further when optimality can be proven. Nevertheless, implementations IAGG and HAGG both outperform MIP, and for a higher number of commodities, IAGG is preferable. The performance profile for the latter in comparison to MIP is shown in Fig. 11 and underlines these findings, as 70 % of the instances are solved fasted by IAGG.

The corresponding results for the medium-sized instances in Table 7 and Fig. 12 show a similar picture. The dimension of the graph allows for fewer commodities to

Table 7 Number of instances solved/average solution time (s) for medium-sized multi-commodity instances with $|V| = 3000$ nodes, an initial demand satisfaction of $l = 0.96$, and an increasing number of commodities b

b	MIP	IAGG	SAGG	HAGG
2	5/120.01	5/54.13	5/59.98	**5/50.08**
3	5/337.98	5/152.21	5/156.87	**5/132.78**
5	5/4436.23	5/620.31	4/863.54	**5/533.49**
7	3/15302.56	4/2249.79	1/15708.12	**5/5169.41**
10	$0/\infty$	**2/5465.29**	$0/\infty$	$0/\infty$

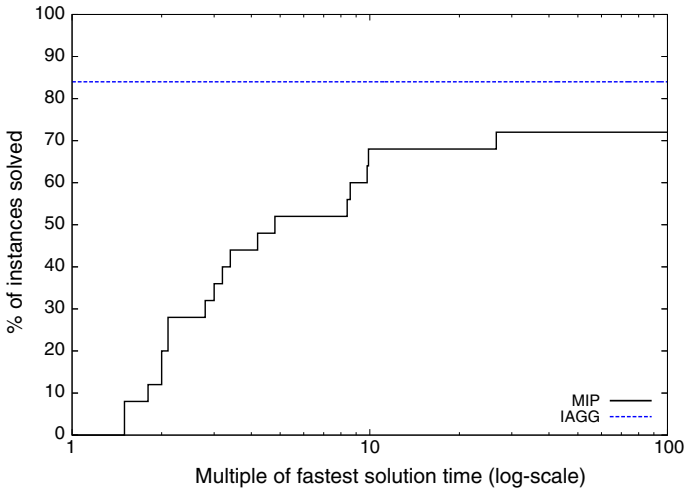

Fig. 12 Performance profile comparing MIP and IAGG on random multi-commodity scale-free networks with $|V| = 3000$ nodes

be considered, but the instances are still best solved by the aggregation schemes. In this case, IAGG does not only solve more instances to optimality than MIP; there even is no single instance which is solved faster by MIP than by IAGG.

As a conclusion, we note that our aggregation approach can also be used successfully in the multi-commodity case. We restricted ourselves to the case of a relatively small number of commodities. For a large number of commodities, further algorithmic enhancements should be included, such as methods that aggregate commodities, in addition to aggregating the network topology.

5.4 Performance on a real-world street network

The graph *rome99* from the 9th DIMACS Implementation Challenge on shortest path problems [7] describes a large portion of the road network of the city of Rome from 1999 (3353 vertices, 8870 directed edges). Its size is comparable to that of the scale-free networks with 3000 nodes. The corresponding results can be found in Table 5. It not only provides a realistic network topology but also comes with a distance measure

Table 8 Number of instances solved/average solution time (s) for a real street network with initial demand satisfaction l

l	MIP	IAGG	SAGG	HAGG
0.9	$0/\infty$	$0/\infty$	$0/\infty$	$0/\infty$
0.91	1/11041.70	**1/1201.86**	1/6297.84	1/1894.52
0.92	1/94.92	1/130.39	1/157.05	**1/56.22**
0.93	2/1109.27	2/478.40	2/747.56	**2/169.59**
0.94	4/826.98	5/311.36	4/421.08	**5/269.42**
0.95	5/133.37	5/76.08	5/122.41	**5/58.84**
0.96	**5/11.9**	5/27.63	5/36.57	5/20.13
0.97	**5/2.57**	5/9.78	5/8.89	5/8.56
0.98	**5/1.53**	5/4.65	5/4.95	5/4.61

on the edges. We use the latter values as module expansion costs, as a distance-proportional cost seems a plausible choice. The module sizes are of the size of 0.25 % of the total demand as for the previous test instances. The demands and initial capacities were again generated randomly.

Table 8 shows the solution times for each aggregation method as well as for plainly solving the network expansion problem via MIP for initial capacities ranging from 90 % up to 98 % in steps of 1 %.

We see a very similar behavior as for the scale-free instances: MIP is fastest only for very easy instances and our aggregation algorithms start to take the lead from a certain level of difficulty onwards.

5.5 Performance on instances from SNDlib

The last benchmark set comprises instances which originate from the popular library of network design problems SNDlib [29]. We adapted the multi-commodity instances to single-commodity problems by balancing the flows in a first step, i.e., by taking the net demand of a node as its single-commodity demand. In a second step, the demands are scaled up such that the total flow in the single-commodity instance is equal to that in the multi-commodity instance. For those SNDlib instances that originally do not possess initial capacities, the initial capacities were generated randomly. Then, results are averaged over five instances. Furthermore, the instances in this section usually have more than one type of module per arc to choose from, as the original SNDlib instances. This is a slight generalization of problem (P), which can be handled straightforwardly in the aggregation scheme. As in the previous sections, we examine the behavior for different levels of initial demand satisfaction.

IAGG performs best on the scale-free network instances with $|V| = 100$ nodes. This suggests choosing it for the comparison over the SNDlib instances too, as they have up to $|V| = 65$ nodes. This choice is confirmed by a limited number of experiments. Table 9 shows the results on those instances that are neither "too hard" nor "too easy", using the same definition as for the scale-free networks. As an example, for the *nobel-us* topology with initial demand satisfaction of $l = 0.1$, only two of the five

Table 9 Number of instances solved/average MIP and IAGG solution times (s) for the SNDlib instances with initial demand satisfaction of l percent

Instance	l	MIP	IAGG	Instance	l	MIP	IAGG
dfn-bwin	0.05	0/∞	**5/372.29**	nobel-germany	0.15	4/703.50	**5/3.41**
dfn-bwin	0.1	0/∞	**5/514.83**	nobel-germany	0.2	3/527.90	**5/2.83**
dfn-bwin	0.15	3/215.86	**4/468.4**	nobel-germany	0.25	5/245.17	**5/2.62**
dfn-bwin	0.2	**5/134.03**	5/203.62	nobel-germany	0.3	4/155.74	**5/3.16**
dfn-bwin	0.25	**5/33.96**	5/51.10	nobel-germany	0.35	5/55.27	**5/1.97**
dfn-bwin	0.3	**5/6.84**	5/47.49	nobel-germany	0.45	5/4.93	**5/0.96**
germany50	0	0/∞	0/∞	nobel-uS	0	0/∞	**5/936.26**
germany50	0.05	4/3989.60	**5/1390.99**	nobel-uS	0.15	0/∞	**1/10.65**
germany50	0.1	5/531.34	**5/302.43**	nobel-uS	0.2	1/966.94	**4/34.06**
germany50	0.15	5/1471.83	**5/280.75**	nobel-uS	0.35	0/∞	**4/21.81**
germany50	0.2	5/1580.57	**5/116.88**	nobel-uS	0.4	0/∞	**3/71.5**
germany50	0.25	5/301.82	**5/39.92**	nobel-uS	0.45	1/2.55	**3/52.0**
germany50	0.3	5/39.51	**5/10.1**	nobel-uS	0.5	0/∞	**5/88.19**
germany50	0.35	5/34.71	**5/13.01**	nobel-uS	0.55	0/∞	**3/4365.75**
germany50	0.45	5/62.14	**5/7.77**	nobel-uS	0.6	1/3258.98	**4/19.86**
giul39	0.05	1/11979.07	**3/12818.92**	nobel-uS	0.65	2/19.24	**5/2.26**
giul39	0.1	3/1074.27	**5/1037.79**	nobel-uS	0.7	1/0.35	**5/9.75**
giul39	0.15	5/1748.12	**5/255.6**	nobel-uS	0.75	4/23.38	**5/0.67**
giul39	0.2	5/598.93	**5/154.78**	norway	0	**5/577.05**	5/972.13
giul39	0.25	5/269.30	**5/79.69**	norway	0.05	**5/18.2**	5/35.30
giul39	0.3	5/80.15	**5/23.33**	norway	0.1	**5/14.4**	5/21.89
giul39	0.35	5/48.11	**5/16.14**	norway	0.15	**5/6.61**	5/10.65
giul39	0.4	5/52.92	**5/9.95**	norway	0.2	**5/4.11**	5/9.92
giul39	0.45	5/18.62	**5/7.88**	pioro40	0.05	4/1225.40	**5/130.93**
giul39	0.5	5/5.49	**5/5.27**	pioro40	0.1	3/6196.60	**5/270.36**
india35	0.1	**2/21952.26**	0/∞	pioro40	0.15	3/3859.20	**5/206.34**
india35	0.15	**5/1856.75**	5/8391.71	pioro40	0.2	4/2594.55	**5/208.51**
india35	0.2	**5/140.59**	5/243.80	pioro40	0.25	5/2619.35	**5/147.14**
india35	0.25	**5/120.72**	5/396.54	pioro40	0.3	5/453.17	**5/25.8**
india35	0.3	**5/36.0**	5/79.56	pioro40	0.35	4/362.43	**5/56.76**
india35	0.35	**5/26.52**	5/33.18	pioro40	0.4	4/175.76	**5/41.31**
india35	0.4	**5/10.68**	5/21.35	pioro40	0.45	5/42.19	**5/9.02**
newyork	0.05	**5/346.2**	5/1286.77	pioro40	0.5	5/813.45	**5/42.27**
newyork	0.1	**5/53.33**	5/462.61	pioro40	0.55	5/38.11	**5/5.93**
newyork	0.15	**5/16.7**	5/89.49	pioro40	0.6	5/36.30	**5/7.08**
newyork	0.2	**5/6.46**	5/31.99	pioro40	0.65	5/12.22	**5/3.83**
nobel-eu	0.05	0/∞	**4/4520.18**	ta1	0.05	0/∞	**4/595.3**
nobel-eu	0.1	0/∞	**3/9269.41**	ta1	0.2	0/∞	**3/4779.79**
nobel-eu	0.15	0/∞	**3/2591.37**	ta1	0.25	0/∞	**3/10246.12**

Table 9 continued

Instance	l	MIP	IAGG	Instance	l	MIP	IAGG
nobel-eu	0.2	$0/\infty$	**3/1602.6**	ta1	0.3	1/5807.34	**4/1564.45**
nobel-eu	0.25	$0/\infty$	**4/415.85**	ta1	0.35	4/1094.01	**5/446.31**
nobel-eu	0.3	$0/\infty$	**5/641.49**	ta1	0.4	5/88.93	**5/82.31**
nobel-eu	0.35	$0/\infty$	**4/294.9**	ta1	0.45	5/10.73	**5/8.84**
nobel-eu	0.4	1/2300.46	**4/53.98**	ta1	0.5	**5/4.68**	5/6.78
nobel-eu	0.45	4/568.94	**5/8.85**	ta2	0.4	$0/\infty$	**1/396.97**
nobel-eu	0.5	3/362.33	**5/10.95**	ta2	0.45	$0/\infty$	**1/357.95**
nobel-eu	0.55	3/21.96	**5/3.44**	ta2	0.5	**1/21.91**	1/32.46
nobel-eu	0.6	3/16.97	**5/2.38**	ta2	0.55	**1/33.47**	1/52.74
nobel-eu	0.65	4/11.92	**5/2.34**	ta2	0.6	**1/10.48**	1/26.97
nobel-germany	0.05	4/303.37	**5/4.7**	zib54	0	**5/207.68**	5/410.01
nobel-germany	0.1	3/57.31	**5/4.15**	zib54	0.05	**5/8.01**	5/30.68

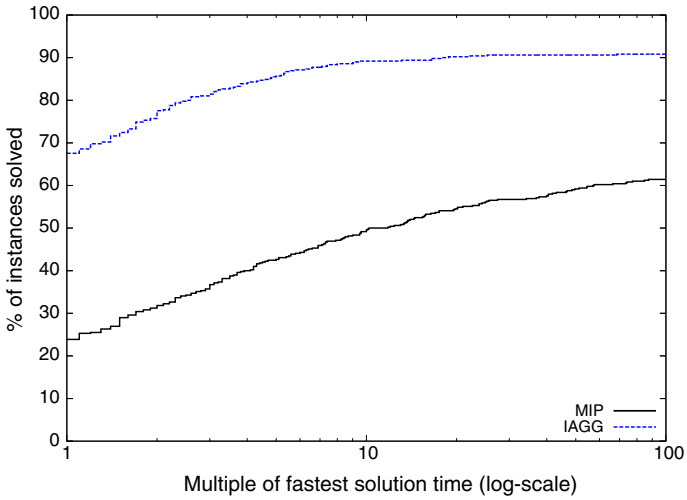

Fig. 13 Performance profile comparing MIP and IAGG on the SNDlib instances

instances can be solved by either MIP and IAGG. The other three could not be solved within the time limit. Hence this class of instances is considered "too hard" and is not listed in Table 9. Those unsolvable instances which are presented here were solved by either SAGG or HAGG. On the other hand, the solution times for three out of the five instances on *nobel-us* topology with $l = 0.8$ are below 10 s for all implemented methods and therefore are not listed either. The same applies to *abilene*, *atlanta*, *di-yuan*, *france*, *geant*, *pdh*, *polska*, and *sun* for any level of initial demand satisfaction. Finally, we had to exclude *dfn-gwin* as its modules did not allow for feasibly scaling the initial demand satisfaction, as well as *cost266*, *janos-us*, and *janos-us-ca* because their nodes all have a net demand of zero.

Clearly, IAGG outperforms MIP on most of the different topologies and most of the initial demand satisfaction levels. Especially interesting are the results on *nobel-germany*, where the solution time of IAGG does not increase significantly with decreasing demand satisfaction. The contrary is the case for MIP, which is unable to solve part of the instances defined on this network. Noteworthy is here as well as for the other *nobel*-instances that they feature an especially high number of modules per arc (40). It is obvious that in such a case aggregating even small portions of the graph can have a tremendous effect because of the number of integral variables which can be saved. The positive results for IAGG are further underscored by the performance profile in Fig. 13, from which is visible that it is the fastest method for almost 70 % of the instances. It is able to solve more than 90 % of the instances while MIP solves slightly more than 60 % within the time limit.

6 Conclusion

In this paper, we presented a new algorithmic framework for the solution of network design problems which is based on iterative aggregation. We described three implementations of our method. They start with an aggregated version of the network graph which is iteratively refined until it represents the whole graph sufficiently well in the sense that an optimal solution to the aggregated problem can easily be extended to an optimal solution to the original problem. This approach mainly aims at reducing the graph size, which directly translates into a smaller size of the mixed-integer program. Our computational experiments clearly show that for the network expansion problem, we could indeed achieve a huge graph compression and consequently we were able to solve most instances much faster than a standard approach. An interesting direction for future research is the combination of graph aggregation with specialized solution techniques for this problem, for example cutting planes, as they would presumably benefit from the much smaller master problem graphs.

Acknowledgments We would like to express our gratitude to Daniel Schmidt for providing us with his preferential attachment graph generator. Furthermore, we thank Andreas Bley for fruitful discussions on the topic. We gratefully acknowledge the computing resources provided by the group of Michael Jünger in Cologne. In particular, we thank Thomas Lange for technical support. We are also indebted to the anonymous reviewers for their constructive comments on this paper. We furthermore acknowledge financial support under BMBF grant 05M10WEC and thank the EnCN for support within research focus Simulation, Projects TP3 and TP6 as well as the DFG for their support within Projects A05, B06, and B07 in CRC TRR 154.

References

1. Albert, R., Barabási, A.-L.: Statistical mechanics of complex networks. Rev. Modern Phys. **74**, 47–97 (2002)
2. Balas, E.: Solution of large-scale transportation problems through aggregation. Oper. Res. **13**(1), 82–93 (1965)
3. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numer. Math. **4**(1), 238–252 (1962)

4. Chvátal, V., Hammer, P.L.: Aggregation of inequalities in integer programming. In: Hammer, P., Johnson, E., Korte, B., Nemhauser, G. (eds.) Studies in Integer Programming, volume 1 of Annals of Discrete Mathematics, vol. 1, pp. 145–162. Elsevier (1977)
5. Costa, A.M.: A survey on Benders decomposition applied to fixed-charge network design problems. Comput. Oper. Res. **32**(6), 1429–1450 (2005)
6. Crainic, T.G., Frangioni, A., Gendron, B.: Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. Discrete Appl. Math. **112**, 73–99 (2001)
7. Demetrescu, C., Goldberg, A., Johnson, D.: 9th DIMACS implementation challenge—shortest paths. http://www.dis.uniroma1.it/~challenge9/ (2006)
8. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Math. Progr. A **91**(2), 201–213 (2002)
9. Dudkin, L., Rabinovich, I., Vakhutinsky, I.: Iterative aggregation theory. Number 111 in Pure and applied mathematics. Dekker, New York (1987)
10. Fischetti, M., Salvagnin, D., Zanette, A.: A note on the selection of Benders' cuts. Math. Progr. Series B **124**, 175–182 (2010)
11. Francis, V.E.: Aggregation of network flow problems. Ph.D. thesis, University of California (1985)
12. Geisberger, R.: Advanced route planning in transportation networks. Ph.D. thesis, Karlsruhe Institute of Technology (2011)
13. Gendron, B., Crainic, T.G., Frangioni, A.: Multicommodity capacitated network design. In: Soriano, P., Sansò, B. (eds.) Telecommunications network planning, vol. 98, pp. 1–19. Kluwer Academic Publishers (1998)
14. Geoffrion, A.: Lagrangean relaxation for integer programming. In: Balinski, M. (ed.) Approaches to integer programming, volume 2 of mathematical programming studies, pp. 82–114. Springer, Berlin (1974)
15. Gurobi Optimization, Inc.: Gurobi optimizer reference manual. http://www.gurobi.com (2013)
16. Hallefjord, A., Storoy, S.: Aggregation and disaggregation in integer programming problems. Oper. Res. **38**(4), 619–623 (1990)
17. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Math. Progr. **96**(1), 33–60 (2003)
18. Johnson, D.S., Lenstra, J.K., Kan, A.H.G.R.: The complexity of the network design problem. Networks **8**(4), 279–285 (1978)
19. Karwan, M., Rardin, R.: Some relationships between lagrangian and surrogate duality in integer programming. Math. Progr. **17**(1), 320–334 (1979)
20. Lee, S.: Surrogate programming by aggregation. Ph.D. thesis, University of California (1975)
21. Leisten, R.: Iterative Aggregation und Mehrstufige Entscheidungsmodelle: Einordnung in Den Planerischen Kontext, Analyse Anhand Der Modelle Der Linearen Programmierung und Darstellung Am Anwendungsbeispiel Der Hierarchischen Produktionsplanung. Produktion Und Logistik. Physica (1995)
22. Leisten, R.: An LP-aggregation view on aggregation in multi-level production planning. Ann. Oper. Res. **Bd. 82**, S.413–S.434 (1998)
23. Lemaréchal, C.: Lagrangian relaxation. In: Jünger, M., Naddef, D. (eds.) Computational combinatorial optimization, volume 2241 of lecture notes in computer science, pp. 112–156. Springer, Berlin (2001)
24. Linderoth, J., Margot, F., Thain, G.: Improving bounds on the football pool problem by integer programming and high-throughput computing. INFORMS J. Comput. **21**(3), 445–457 (2009)
25. Litvinchev, I., Tsurkov, V.: Aggregation in large-scale optimization. In: Applied optimization, vol. 83. Springer (2003)
26. Macedo, R., Alves, C., de Carvalho, J.V., Clautiaux, F., Hanafi, S.: Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. Eur. J. Oper. Res. **214**(3), 536–545 (2011)
27. McDaniel, D., Devine, M.: A modified Benders' partitioning algorithm for mixed integer programming. Manag. Sci. **24**(3), 312–319 (1977)
28. Newman, A.M., Kuchta, M.: Using aggregation to optimize long-term production planning at an underground mine. Eur. J. Oper. Res. **176**(2), 1205–1218 (2007)
29. Orlowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0-Survivable Network Design Library. In: Proceedings of the 3rd International Network Optimization Conference (INOC 2007). Spa, Belgium (2007)
30. Rogers, D.F., Plante, R.D., Wong, R.T., Evans, J.R.: Aggregation and disaggregation techniques and methodology in optimization. Oper. Res. **39**(4), 553–582 (1991)

31. Rosenberg, I.: Aggregation of equations in integer programming. Discrete Math. **10**(2), 325–341 (1974)
32. Salvagnin, D., Walsh, T.: A hybrid MIP/CP approach for multi-activity shift scheduling. In: Milano, M. (ed.) Principles and practice of constraint programming, lecture notes in computer science, pp. 633–646. Springer, Berlin (2012)
33. Zipkin, P.H.: Aggregation in linear programming. Ph.D. thesis, Yale University (1977)