ASTS orientations on undirected graphs*

Kai Helge Becker Benjamin Hiller

Abstract

All feasible flows in potential-driven networks induce an orientation on the undirected graph underlying the network. Clearly, these orientations must satisfy two conditions: they are acyclic and there are no "dead ends" in the network, i.e. each source requires outgoing flows, each sink requires incoming flows, and each transhipment node requires both an incoming and an outgoing flow. In this paper we will call orientations that satisfy these conditions acyclic source-transhipment-sink orientations (ASTS-orientation) and study their structure. In particular, we characterize graphs that allow for such an orientation, describe a way to enumerate all possible ASTS-orientations of a given graph, present an algorithm to simplify and decompose a graph before such an enumeration and shed light on the role of zero flows in the context of ASTS-orientations.

1 Introduction

This paper studies a certain type of acyclic orientation of undirected graphs to see how these may be used to improve the optimization of flows on potential-driven networks. In [8] it was shown that MINLP models arising from so-called potential-driven network flow problems can be strengthened by fixing the flow direction and carrying out optimization-based bound tightening for each possible direction. In the present paper we build on this insight and analyse the mathematical structure that results from the flow directions.
All feasible flows in potential-driven networks induce an orientation on the undirected graph underlying the network. Clearly, this orientation must satisfy two conditions: they are acyclic and there are no "dead ends" in the network, i.e. each source requires outgoing flows, each sink requires incoming flows, and each transhipment node requires both an incoming and an outgoing flow. In this paper we will call an orientation that satisfies these two conditions an acyclic source-transhipment-sink orientation (ASTS-orientation).

The remainder of the paper is structured into five sections: In the following section we will formalize the concept of ASTS-orientations and study their fundamental structure. As our aim is to be able to generate all possible ASTS-orientations of a given undirected graph underlying a potential-based network, Section 3 presents methods to simplify and decompose a graph before enummerating ASTS-orientations. In Section 4 we will build on the previous two sections and briefly

*This preprint was first published as [3].

sketch the conditions under which zero flows can occur in the context of ASTS-orientations. Section 5 presents several ways to construct ASTS-orientations and introduces an algorithm to enumerate all ASTS-orientations of a given graph. The conclusion addresses some topics for further research.

2 Definition and fundamental properties of ASTS-orientations

In this section we will consider only nontrivial simple finite (di)graphs, i.e. finite (di)graphs with at least two nodes and without loops, multi-edges and multi-arcs. We begin the defining the main concept of this section.

Definition 1 *Let $D(V, A)$ be a digraph with an underlying graph G, with $V_+ \subsetneq V$ and $V_- \subsetneq V$ being sets of sources and sinks, respectively, with $V_+ \cap V_- = \emptyset$, and with $V_T := V - V_+ - V_-$ being a set of transhipment nodes. A node $v \in V$ is said to satisfy the source-transhipment-sink-condition (STS-condition) if (i) $v \in V_+$ and there exists an arc $a \in A$ with $tail(a) = v$, or (ii) $v \in V_-$ and there exists an arc $a \in A$ with $head(a) = v$, or (iii) $v \in V_T$ and there exist arcs $a_1, a_2 \in A$ with $head(a_1) = tail(a_2) = v$. If all nodes $v \in V$ satisfy the STS-condition, the digraph D is said to be source-transhipment-sink-oriented (STS-oriented) or an STS-orientation of the underlying graph G. If D is also acyclic, it is said to be ASTS-oriented or an ASTS-orientation of the underlying graph G.*

The following observation provides us with a first implication of this concept, which answers the question of what an ASTS-orientation looks like.

Proposition 1 *Let $D(V, A)$ be an ASTS-oriented digraph, $V_+ \subsetneq V$ and $V_- \subsetneq V$ disjoint sets of sources and sinks, respectively, and $V_T := V - V_+ - V_-$ a set of transhipment nodes. Then both V_+ and V_- are non-empty, there exists a "supersource", i.e. a source that is tail of all arcs it is incident with, and a "supersink", i.e. a sink that is head of all arcs it is incident with, and every node $v \in V$ is on a directed path from a supersource to a supersink.*

Proof Assume V_- is empty. As the digraph is ASTS-oriented, each node in $V_+ \cup V_T$ satisfies the STS-condition and, as a consequence, has an outgoing arc. Now take a node in $V_+ \cup V_T$ and follow a directed path from node to node via outgoing arcs. As D is an acyclic digraph, the directed path cannot return to a node we have already visited. Moreover, because D is finite, the directed path must end in a sink, i.e. V_- is non-empty. Analogously, we can show that V_+ is non-empty, and the same line of reasoning implies that every node is on a directed path from a source to a sink and that there must be a super source and a super sink. ■

Remark 1 *We note that the condition of Proposition 1 is only necessary, not sufficient because a digraph with sources and sinks where all nodes are on paths from a source to a sink may have cycles. (Consider, for example, two paths from sources to sinks with nodes $1, 2, 3, 4$ and $5, 6, 7, 8$, where 1 and 5 are sources and 4 and 8 are sinks, with the additional arcs $(3, 6)$ and $(7, 2)$.*

The following result is helpful for enummerating ASTS-orientations because it is related to the fact that we can add a supersink and a supersource to a graph with several sources and sinks, which then become transhipment nodes.

Corollary 1 *Let $D(V, A)$ be an ASTS-oriented digraph and $s \in V_+$ the only supersource. Then the digraph $D'(V - \{s\}, A - \{(s, v) : v \in Succ(s)\})$, i.e. the digraph that results from removing from D the supersource s and the arcs that emanate from it, is ASTS-oriented, too, and one of the successors of s is a supersource on D.*

PROOF Removing the arcs emanating from s does not affect the acyclicity of the graph. For seeing that the resulting graph is still STS-oriented it suffices to notice that the nodes that were successors of the supersource are the only nodes that are affected by removing the supersource. If these nodes were transhipment nodes on D or sources, they are transhipment nodes or sources on D', too. If they wereif they were sources and have an outgoing arc each, i.e. can be sources: Why still STS? Finally, the nodes that were successors of the supersource are the only nodes to be affected by removing the supersource. As there was no other supersource in the graph before removing the supersource and each ASTS-orientation has a supersource according to Proposition 1, the new supersource must be among the successors of s. ■

The following theorem provides us with several ways of characterizing graphs that allow for an ASTS-orientation. It also shows us that on graphs that allow for an ASTS-orientation we can extend all partial ASTS-orientations on subgraphs to an ASTS-orientation on the entire graph. Let us first recall the concepts of a block and of a block tree graph (see e.g. [6], Chapter 3).

Definition 2 *A block of a graph G is a maximal connected subgraph of G that does not have a cut-node.*

As a consequence, a block is either a maximal 2-connected subgraph, a bridge (including its endnodes), or an isolated node. Every edge of a graph lies in a unique block, and two blocks of a graph overlap in at most one node, namely a cut-node of the graph G.

Definition 3 *The block graph of a graph G is the graph whose node set consists of the cut-nodes of G and the blocks of G and where two nodes are adjacent iff one node is a cut-node and the other node is a block that the cut-node is a node of.*

We note that the block graph of a graph is a bipartite tree. We are now prepared for our characterization theorem.

Theorem 1 *Let $G(V, E)$ be a graph with node set V, $V_+ \subsetneq V$ and $V_- \subsetneq V$ disjoint sets of sources and sinks, respectively, and $V_T := V - V_+ - V_-$ the set of transshipment nodes. Then the following four statements are equivalent:*

(i) G has an ASTS-orientation.
(ii) Every node $v \in V$ is on a path from a source to a sink.
(iii) All components of G have both a source and a sink, and the leaves of the block tree graph of G correspond to blocks that have a source or sink other than the cut-node of the block.
(iv) There exists an ASTS-orientation of some subgraph of G, and for every ASTS-orientation $D_0(V_0, A_0)$ of every subgraph $G_0(V_0, E_0)$ of G there exists an orientation A_1 of the edge set $E - E_0$ such that $D(V, A_0 \cup A_1)$ is an ASTS-orientation of G.

Proof We will prove the theorem in the following order:
$(i) \implies (ii) \implies (iii) \implies (ii) \implies (iv) \implies (i)$.
$(i) \implies (ii)$: The existence of a directed path from a source to a sink for all nodes according to Proposition 1 trivially implies the existence of a path from a source to a sink on the underlying undirected graph for all nodes in $V - V_0$.
$(ii) \implies (iii)$: We show the contrapositive. Clearly, if the graph has components without a source or a sink, not every node can be on a path from a source to a sink. Moreover, if a block of a graph is a leaf of the block tree graph and all nodes that are not the only cut-node of the block are neither a source nor a sink, a path from any of these nodes to a source or to a sink must contain the cut-node. Hence, except possibly the cut-node itself, no node of the block is on a path from a source to a sink.
$(iii) \implies (ii)$: Let $v \in V$ be a node of G. If v is a source (sink) itself, it is clearly on a path from a source to a sink as it is connected with the sink (source) that exists in its component of G. Therefore, in the following let v be neither a source nor a sink. We distinguish between three cases.

Case (1): The node v is in a block of G that has both a source and a sink. By virtue of the 2-connectedness of the block there exist two internally disjoint paths from v to a source and two internally disjoint paths from v to a sink. As a consequence, v must be on a path from a source to a sink.

Case (2): The node v is in a block that has either a source or a sink. We assume w.l.o.g. that the block has a source. If the block is a leaf of the block tree graph of G, the source cannot be the cut-node of the block. If the block is not a leaf of the block tree graph, we will assume for the moment that it is not a cut-node of the block. Now, since each component of G has a source and a sink, the block must have a cut-node that is connected with a sink in a different block. Then, due to the 2-connectedness of blocks, there exist two internally disjoint paths from v to the source and from v via the cut-node to the sink, with the latter being the case only if v is not the cut-node itself. In either case v is on a path from a source to a sink.
Now let us consider the case where our block is not a leaf of the block tree graph and the source is a cut-node of our block. Then the block contains at least two cut-nodes and one of the other cut-nodes of the block must be providing us with a path to a source or a sink in a further block, otherwise the block tree graph of G would have a leaf without source or sink. If the cut-nodes provide us with a path to a sink, there clearly exists, again due to the 2-connectedness of blocks, a path from the block's source to a sink that passes through v. If the cut-nodes only provide us with a path to another source, there must be another sink in a block that is accessible via the cut-node that is a source, otherwise the component that v is a node of would not have a sink. As a consequence, v is

again on a path from a source to a sink.

Case (3): The node v is in a block without source and sink. Then v cannot be in a block that is a leaf of the block tree graph and therefore the block in which v is located contains at least two cut-nodes (one of which may be v itself). All cut-nodes must be on a path from v to a source or from v to a sink, otherwise the block tree graph of G would have a leaf without both source and sink. Moreover, due to the fact that all components of G have both a source and a sink, one of the cut-nodes must be on a path from v to a source and a different cut-node must be on a path from v to a sink. As a consequence, since a block is 2-connected, v must be on a path from a source to a sink.

$(ii) \implies (iv)$: We will procede in two steps. In first step we will extend our digraph D_0 to an ASTS-oriented digraph whose underlying graph is still G and that contains all sources and sinks in V, provided the latter is not the case yet. In the second step we will show that, provided the graph underlying our ASTS-oriented digraph contains all sources and sinks and is a proper subgraph of G, we can always find a path on the remaining unoriented edge set of G that can be oriented to provide a larger ASTS-oriented digraph with an underlying subgraph of G. Statement (ii) then follows by induction.

STEP 1: For each component of G all edges of which are in $E - E_0$ we enlarge the digraph D_0 by connecting, with the sinks in their components, all sources that are not yet in the node set of our digraph, using directed paths whose internal nodes are not in the node set of our digraph either. The prodecure is as follows: for each source, we take an arbitrary path on G from the source to a sink, which is possible because all nodes are on a path from a source to a sink, and orient, into the direction away form the source, either the subpath from the source to the first node of the existing digraph or, if this is not possible because none of the nodes of the path is on the digraph, the entire path from the source to a sink. In a similar fashion we add all remaining sinks to our digraph: by orienting edges that connect these sinks to the closest node of our existing digraph or by directly connecting them with a source. The resulting enlarged digraph has an underlying graph that is a subgraph of G, has a node set that includes all sources and sinks and is ASTS-oriented (note that we did not create any directed cycle because we did not connect any two nodes that were already in the node set of our digraph).

STEP 2: We will now show that given an ASTS-oriented digraph D_0 with an underlying proper subgraph G_0 of G and with a node set that includes all sources and sinks of G, we can always find a path P with edges from $E - E_0$, with distinct endnodes in V_0, and with all other nodes being in $V - V_0$ that can be oriented such that the directed graph that arises from adding the oriented path to D_0 again yields an ASTS-oriented digraph.

We pick an arbitrary edge from $E - E_0$. If both endnodes of this edge are in V_0, this edge is our path P. Otherwise, we extend this edge to a path by adding edges from $E - E_0$ until we have a path P with two distinct endnodes both of which are in V_0. This is always possible because all nodes in V are, on G, on a path from a source to a sink, with the sources and sinks being in V_0 according to the construction in Step 1. We denote the endnodes of P by $i, j \in V_0$, the inner nodes of P by $V' \subseteq V - V_0$, and the edges of P by $E' \subseteq E - E_0$.

We now orient P as a directed path, i.e. from one endnode to the other such that each internal node of the path is both head and tail of an arc, and denote the arcs of the directed path by A'. For constructing our orientation we observe that,

as D_0 is acyclic and i and j are nodes of D_0, there is (a) no directed path on D_0 from i to j and no directed path from j to i, or (b) there is a directed path from i to j, but not from j to i, or (c) there is a directed path from j to i, but not from i to j. In case (a) we choose an arbitrary orientation for our directed path. In cases (b) and (c) we orient the path such that it has the same orientation as the existing path. As a consequence the digraph $D'(V_0 \cup V', A_0 \cup A')$ that results from adding the arcs and nodes of the directed path to D_0 will be acyclic, too. (Note that by construction of P, only the endnodes of P are in V_0, and therefore orienting P cannot create any cycle containing an internal node of P.) As the STS-condition is already satisfied at the endnodes $i, j \in V_1$ and we have oriented the path P to form a directed path, the inner nodes of the path and hence the digraph D' altogether also satisfy the STS-condition. All in all, the resulting digraph $D'(V_0 \cup V', A_0 \cup A')$ is ASTS-oriented.
$(iv) \implies (i)$: As G has a component with both a source and a sink, we can create an ASTS-orientation D_0 of a subgraph G_0 of G by orienting an arbitrary path from a source to a sink into the direction away from the sink. Then we can extend D_0 to an ASTS-orientation $D(V, A_0 \cup A_1)$ of G. ■

In the following we will make some (more or less obvious) complementing remarks on the theorem.

Remark 2 *Note that the subgraph $G_0(V_0, E_0)$ is not required to be the subgraph induced by the node set V_0, i.e. there may be edges in E both endnodes of which are in V_0 that are not edges in E_0. Theorem 1 guarantees that also these edges can be oriented to provide us with an overall ASTS-orientation of G.*

Remark 3 *Note that statement (ii) means that for every node there are two internally disjoint paths, one from the node to a source and one from the node to a sink. This implies that the location of the cut-nodes of the graph is relevant for the possibility of having ASTS-orientations, see statement (iii).*

Remark 4 *Note that in statement (iii) it is crucial that we look at block tree graphs [where the blocks are 2(-node)-connected] and not at bridge block trees [where the "blocks" are 2-edge-connected]. The reason is that in a 2-connected component that contains both a source and a sink it is guaranteed that all nodes are on a path from a source to a sink. For a 2-edge-connected component, this statement does not hold.*
Example: The 2-edge-connected graph
$G(\{1,2,3,4,5\}, \{\{1,2\},\{1,3\},\{2,3\},\{3,4\},\{3,5\},\{4,5\}\})$
with $V_+ = \{4\}$ and $V_- = \{5\}$.
Here nodes 1 and 2 are not on a path from the source to the sink.
So disregarding isolated nodes, we can say that for our purpose the relevant building-blocks of graphs are maximal 2-connected subgraphs and bridges.

Remark 5 *Note that in the proof of statement (iv), in our construction of an ASTS-orientation, we used a technique that is similar to an open ear decomposition (cf. [2]), just that we used a forest instead of a path as the starting point.*

Remark 6 *Obviously in statement (iv) the orientation D_0 is necessarily acyclic. Note that it is also necessary that D_0 is already STS-oriented. Take the digraph*

$D_0(\{1,2,3\},\{(1,2),(2,3)\})$ *with* $V_+ = \{1,3\}$ *and* $V_- = \{2\}$ *and the graph* $G(\{1,2,3,4\},\{\{1,2\},\{1,4\},\{2,3\},\{3,4\}\})$. *Except the fact that node* 3 *does not satisfy the STS-condition, all other conditions of the proposition are satisfied, and yet there exists no orientation of the edges* $\{1,4\}$ *and* $\{3,4\}$ *that would lead to an ASTS-oriented digraph* $D(V, A_0 \cup A_1)$.

Remark 7 *A consequence of Proposition 1 is that graphs where all nodes are transhipment nodes do not have ASTS-orientations, not even when we add zero flows as orientations (except the trivial case where all flows are zero). Example: the complete graph* K_3. *Conversely, Property (iv) in Theorem 1 states that for graphs where every node is on a path from a source to a sink we do not need zero-orientations on individual arcs to achieve an ASTS-orientation because all unoriented edges can be oriented to achieve an overall ASTS-orientation.*

Remark 8 *Regarding the condition in Proposition 1 that every node on an ASTS-oriented digraph is on a directed path from a source to a sink, we noted that this condition is only necessary, but not sufficient. However, we now know from Theorem 1 (ii) that any digraph that satisfies the necessary condition of Theorem 1 allows for an ASTS-orientation. This means that for any cyclic digraph with an STS-orientation that satisfies the condition of Proposition 1, it is possible to construct an acyclic STS-orientation.*

Remark 9 *Note that ASTS-orientations have a rather strong structure. While they are acyclic by definition, adding one node that is connected to all sources and sinks is sufficient to ensure that each node is on a cycle. A different way of looking at the same aspect: The proof that Property (iv) follows from Property (ii) in Theorem 1 is similar to an ear decomposition in the sense that we successively add directed paths to the digraph, with only the endsnodes of the paths being on the existing digraph. But while an ear decompostion requires a 2-connected graph, we just required that every node is on a path from a source to a sink. (This was possible because we did not start our "ear decompostion" from a cycle, but from a forest that connects all sources with sinks.)*

The following direct corollary of the preceding theorem provides us with a way to interpret ASTS-orientations of graphs:

Corollary 2 *For each solution of a potential-based flow problem on an ASTS-orientable graph* $G(V,E)$ *there exists an ASTS-orientation* $D(V,A)$ *of* G *such that all edges with non-zero flows in the solution have the same orientation as on* D.

The subgraph induced by the set of arcs with non-zero flow clearly has an ASTS-orientation where arcs with positive flow have one orientation and arcs with negaive flow the other. According to statement (iv) of the preceding theorem, this orientation can be extended to an ASTS-orientation D of G.

Proof This corollary implies that we can interpret the orientations of the arcs of an ASTS-orientation as non-positive flows (forward arcs) and non-negative flows (backward arcs), i.e. we do not necessarily need to extend our concept of ASTS-orientations to make specific provisions for zero flows. This has the advantage that we will have fewer orientations to account for.

$\lvert V\rvert$	**No of orient.**	**No of ASTS-orient.**
4	64	2
6	3.28×10^4	24
8	2.68×10^8	720
10	3.52×10^{13}	40300
20	1.57×10^{57}	6.40×10^{15}
30	8.87×10^{130}	3.05×10^{29}
40	6.36×10^{234}	5.23×10^{44}
44	5.95×10^{284}	1.41×10^{51}

Table 1: Upper bound on the number of ASTS-orientations with one source and one sink

To finish this section, we will give an upper bound on the number of ASTS-orientations of a graph.

Corollary 3 *Let $G(V, E)$ be a graph that has an ASTS-orientation with $V_+ \subsetneq V$ and $V_- \subsetneq V$ being disjoint sets of sources and sinks, respectively. Then the number of ASTS-orientations of G is less than or equal to $|V_+||V_-|(|V|-2)!$, the number of ASTS-orientations of the complete graph on $|V|$ nodes.*

Proof Due to statement (iv) of the preceding theorem, every ASTS-orientation on G can be interpreted as a subdigraph of an ASTS-orientation on a complete graph K with the same number of nodes, sources and sinks. Moreover, we observe that the set of all ASTS-orientations of K is equal to the set of acyclic tournaments on K where the supersource of the tournament is in V_+ and its supersink in V_-. Disregarding the sets of sources and sinks, there exists a bijection between the number of acyclic tournaments on K and the number of permutations of the node set V. Taking into account that an acyclic tournament has exactly one supersource and one supersink and that we can choose between $|V_+||V_-|$ pairs of supersource and supersink yields a number of $|V_+||V_-|(|V|-2)!$ acyclic tournaments on K such that the supersource and the supersink are from V_+ and V_-, respectively. ■

Table 1 illustrates for complete graphs with one source and one sink that the number of ASTS-orientations can be considerably lower than the number of orientations of the graph. Note that in real-life applications such as gas, water or hydrogen networks, nodes do typically not have a degree greater than 4 or 5. As a consequence, the number of ASTS-orientations of the graphs underlying these networks is significantly lower than the number of ASTS-orientations of complete graphs.

3 Simplication and decomposition of the underlying graph

Having presented some fundamental properties of ASTS-orientation, our aim is now to find a fast way to enummerate all ASTS-orientations of a graph. In this section we will present three basic statements that allow us to simplify a given underlying graph. Moreover, we will give an outline of the general optimization procedure.

3.1 Decomposition of the underlying graph

The first statement, which is a direct consequence of Theorem 1, allows us to decompose any underlying graph into a part that allows for an ASTS-orientation, while the remaining edges must have zero flow, i.e. can be removed form the graph.

Corollary 4 *The edge set E of any graph $G(V, E)$ can be uniquely decomposed into two (potentally empty) sets $E = E_1 + E_2$ such that for any feasible solution of a potential-based flow problem on G*
(i) the flow on all edges in E_1 is zero, and
(ii) the subgraph induced by the edges in E_2 allows for an ASTS-orientation.

Proof We will decompose the underlying graph as follows: One edge set contains the edges from all components without source and sink and the edges in a block of the graph that (a) does not have a source and a sink other than the cut-node of the block or that (b) does not have two cut-nodes, one of which is on the path from nodes of the block to a source and the other one of which is on a path from nodes of the block to a sink. The remaining edges form the second edge set.
Regarding the first edge set: All potential-based flows have an ASTS-orientation. We know from Proposition 1 that all nodes of such an orientation are on a directed path from a source to a sink. This is not possible for the subgraph induced by the first edge set. The subgraph induced by the second edge set allows for an ASTS-orientation by Theorem 1 (iii). ∎

Remark 10 *For a practical algorithmic setting Corollary 1 implies the following procedure. We know that for any given graph we can allocate flow 0 to all edges that are in a component without source and sink. Also we can sucessively allocate flow 0 to all edges in blocks that correpond to a leaf of the block tree graph of G and do not have a source or a sink that is not the cut-node of the block. These blocks can then be removed from the graph and we can recursively re-apply Corollary 1 again, thereby sucessively pruning the block tree graph until no leaf of the block tree graph without source and sink that are not the cut-node of the block is left. We are then guaranteed that there is an ASTS-orientation of the remaining edges of our graph.*

3.2 Elimination of nodes and edges

The first statement allows us to simplify a network by removing serial and parallel edges.

Proposition 2 *Let $G(V, E)$ be a graph. Then all ASTS-orientations of G have the following properties:*
(i) All edges of a path subgraph of G where no internal node is a source or sink and all internal nodes have degree 2 on G have the same orientation.
(ii) All edges of two path subgraphs of G with common endnodes where no internal node is a source or sink and all internal nodes have degree 2 on G have the same orientation.

Proof (i) If the edges were not oriented in the same direction, the subgraph would not have an STS-orientation.
(ii) Due to (i), it remains to observe that the orientation would not be acyclic if two paths with the same endnodes were given different orientations. ■

Remark 11 *It is interesting that the general insight that in networks parallel and serial edges can be replaced by equivalent single edges with a so-called equivalent diameter (see e.g. [9]) is already relevant on the level of ASTS-orientations. More precisely, parallel edges can be removed due to the acyclicity condition and serial edges can be contracted due to the STS condition.*

Remark 12 *Obviously we can also remove leaves that are sourcs and sinks. But it is not clear whether doing so would solve algoithmic time.*

3.3 Fixing flows through cut-nodes

In the following we will fix the orientation of some edges by looking at the demand distribution in the network.

Observation 1 *Let $G(V, E)$ be a graph that allows for an ASTS-orientation with given supplies and demands for sources and sinks, respectively. Then we can use the block tree graph of G to restrict the orientation of edges that the cut-nodes of G are incident with by classifying the cut nodes of a block as sources, sinks or transshipment nodes relative to the block considered and can consider the orientations of each block independent of the orientations of the other blocks.*

The following proposition expresses this insight.

Proposition 3 *Let $G(V, E)$ be a graph that allows for an ASTS-orientation and let V be the node set, $V_+ \subsetneq V$ and $V_- \subsetneq V$ disjoint sets of sources and sinks, respectively, and $V_T := V - V_+ - V_-$ the set of transhipment nodes. Let $F(R, f, g)$ be flow structure of G. Then the block tree of G also has an ASTS-orientation D, every ASTS-orientation D corresponds to a set of ASTS-orientations of G, and any flow on G that satisfies the mass balance corresponds to a flow on D, where the arcs on D represent the flows in and out of blocks.*

PROOF As G allows for an ASTS-orientation, each leaf of the block tree graph of G has a source or a sink that is not the cut-node of the block that the leaf represents by Theorem 1. As a consequence, also by Theorem 1, the block tree has an ASTS-orientation D if we identify each node that represents a block by its non-cut-node source or sink (depending on which has the higher demand). Such an ASTS-orientation D corresponds to all ASTS-orientation on G where the arcs on D represent the incoming and outgoing flows of blocks on G. By construction of R and f, the flows on R satify mass balances corresponding to the mass balances on G.

Remark 13 *In a practical setting we can use this insight as follows: We decompose the underlying graph into its blocks and declare each cut-node of each block source or sink depending on whether the block has in the flow structure graph, at the point of the cut-node, an incoming or an out-going arc, respectively. Then the number of ASTS-orientations of a graph is the product of the number of the orientations of the blocks (where blocks with flow zero throughout are considered to have one orientation) and .*

3.4 General procedure for enumerating ASTS-orientations

Based on the insights of this and the previous section we can now present the general framework for enumerating ASTS-orientations.

We start our algorithm in Step 1 with removing all connected components without source and sink (because these will trivially have zero flows) and will look at all components seperately in the following.

In Step 2, to further simplify the computational effort, we will remove all nodes with degree 1. In the above framework a node of degree 1 (including the edge incident to it and its neighbour) would be treated as a block with zero flow, provided that the node is not a source or a sink, or would be treated as a block with a source or sink other than its cut node. A computationally less expensive approach is to remove a node with degree 1 *ab initio* and add its demand to its neighbour because the flow direction on the one arc incident to the node is determined by the demand at the node.

After removing all nodes with degree 1, we apply Corollary 2. Figure 1 demonstrates how the sucessive application of this corollary leads to the elimination of nodes and nodes. Note that doing so may lead to new nodes with degree 1, the removal of which may allow for the further application of Corollary 2.

In Step 3 we analyse the block structure of our graph, to be prepared for Corollary 4 and Corollary 1.
Now we apply Corollary 4 in Step 4 and iteratively remove the leaves of the block tree graph if the coresponding blocks have no source or sink other than the cut node of the block.
Removing blocks in Step 4 may lead to a graph in which we can further apply Corollary 2 and remove nodes of degree 1. Therefore, in Step 5, we will repeat

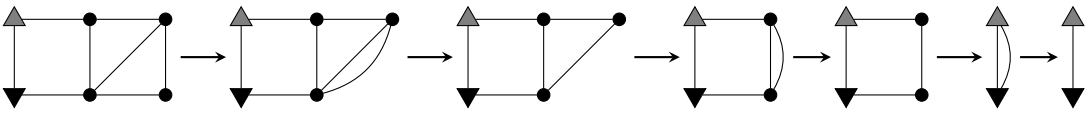

Figure 1: Successive elimination of serial and parallel edges and nodes with degree 1

Step 2 on the current graph.
In Step 6 we use Observation 1 and classify the cut nodes of all blocks as sources, sinks or transshipment nodes *relative to the block*. This provides the basis for enumerating, in Step 7, the ASTS-orientations of each block separately.
Altogether our algorithm for enumerating ASTS-orientations looks as follows:

1. Remove connected components without source and sink and look at all remaining components separately.
2. Iteratively remove all nodes with degree 1, all nodes with degree 2 that are transhipment nodes and all parallel edges.
3. Construct block tree graph.
4. Iteratively remove blocks that are leaves of the block tree graph and do not have a source or sink other than the cut-node of the block.
5. Repeat Step 2.
6. Calculate flows through cut-nodes and classify the cut-nodes as sources, sinks or transhipment nodes relative to their blocks.
7. Enumerate the ASTS-orientations of each block separately.

Remark 14 *It may also be useful to decompose the underlying graph not only into blocks, but also, more generally, into subgraphs that are connected by rather small edge-cuts. In such a case, the orientations of the two subgraphs can be treated in separate columns within a column generation framework, with some additional constraints that ensure compatibility between the orientations of the subgraphs. Altogether this will lead to fewer columns than having a column for the orientation of the entire undecomposed graph.*

We will demonstrate the approach developed in the present and the previous section with an example. We would like to find all ASTS-orientations of the network given in Figure 2.

As the entire graph is connected, Step 1 does not change our graph. After carrying out Step 2, i.e. removing all nodes with degree 1 and all serial and parallel arcs, we arrive at the reduced network in Figure 3. Note in particular that the sucessive removal of nodes with degree 1 led to the removal of a sink with demand -100, which has turned the sink's neighbouring node into the new sink with demand -100.

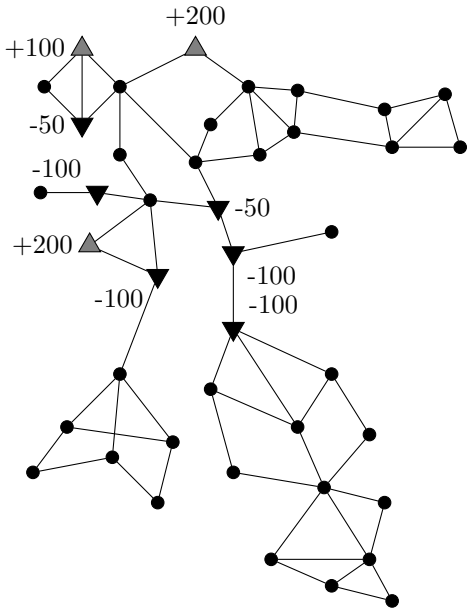

Figure 2: Network example with 3 sources (grey upward triangles), 6 sinks (black downward triangles) and 33 transhipment nodes (dots). Numbers indicate the demands at sources (demand > 0) and sinks (demand < 0)

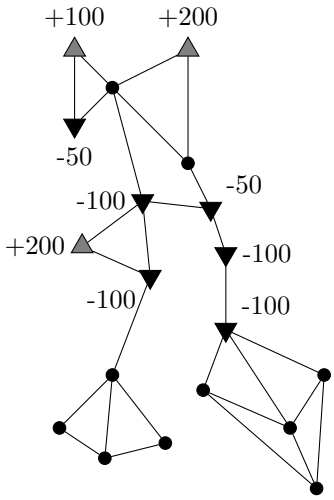

Figure 3: Reduced network after carrying out Step 2

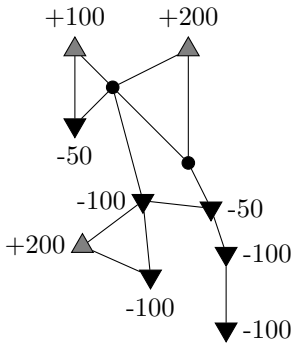

Figure 4: Network after Step 4

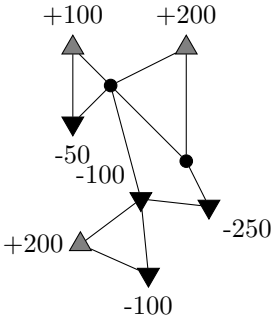

Figure 5: Network after Step 5

We can now carry out Step 4 and remove three blocks (one of which is a bridge) that have no sources and sinks or no sources or sinks other than their cut node. Figure 4 shows the resulting network.

Removing has now led to a network that has a node with degree 1 again. Carrying out Step 5, which here just means removing a node with degree 1 twice, yields the network in Figure 5. Note how the removal of two sinks led to a sink withn an overall demand of -250.

To the reduced network we apply Step 6, i.e. we classify the cut nodes of the remaining graph as sources, sinks or transshipment nodes relativ to their blocks. This has been achieved in Figure 6, where the three blocks have been clearly separated. The dashed lines betweek nodes indicate that these two nodes represent the same cut node in the previous graph. Note that in the process of Step 6 a former sink in Block 2 has become a transhipment node because the inflow of $+100$ into this node from Block 3 has been balanced out by the demand of -100 at this node.

In Step 7 we enumerate the orientations of each block separately. In Figure 7 we can see the two ASTS-orientations of each of Block 1 and Block 3 (both blocks have the same structure with one source and two sinks), while Figure 8 shows the

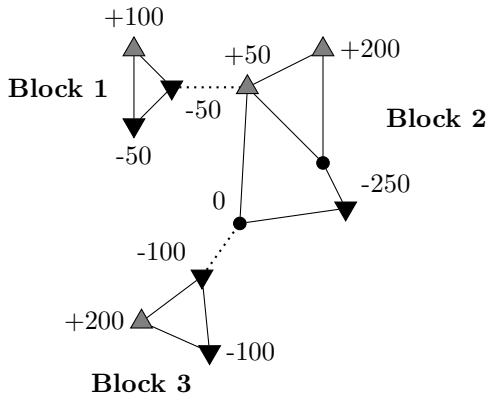

Figure 6: The three remaining blocks of the graph with classified cut nodes

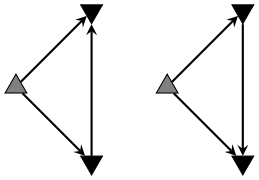

Figure 7: Possible orientations for Blocks 1 and 3 in Figure 6

four possible ASTS-orientations of Block 2. It is easy to see that Blocks 1 and 3 must have 2 ASTS-orientations block: We know from Proposition 1 that each ASTS-orientation must have a supersource and a supersink. The two blocks have only one possible supersource each and two potential supersinks. After fixing the orientations of the edges incident to the supersource, choosing one of the two sinks as a supersink determines the orientation of the blocks. In the case of Block 2, we also have to choose which of the two sinks should become a supersink, but due to the chord of the 5-cycle we have another degree of freedom here, whcih leads to 4 orientations altogether. (Note that in the case of our blocks here, we always had to choose one sink as a supersink. Depending on the network structure also both sinks could become supersinks in an ASTS-orientation. In our cases, however, making one sink a supersink excludes the other sink from being a supersink.)

We can conclude from Figures 6 to 8, but multiplying the number of ASTS-orientations of the blocks, that our graph in Figure 6 and, consequently, also the original graph in Figure 2 have altogether $2 \times 2 \times 4 = 16$ ASTS-orientations each. One such orientation of the original graph in Figure 2 is shown in Figure 9.

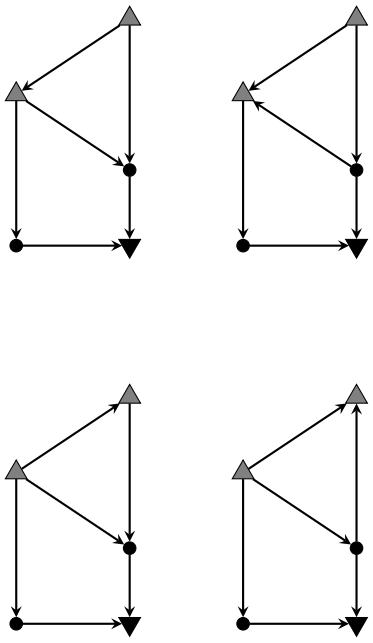

Figure 8: Possible orientations for Block 2 in Figure 6

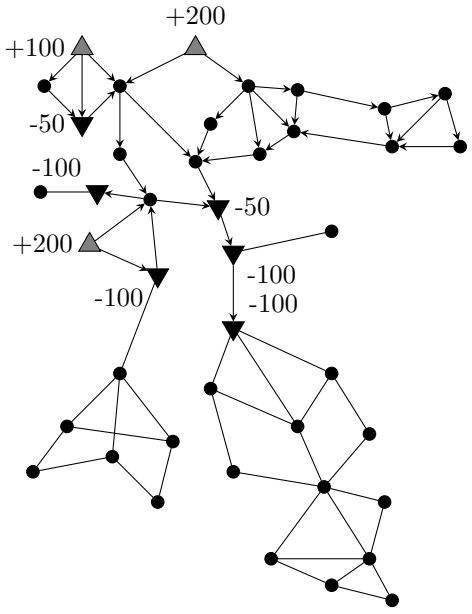

Figure 9: One of the 16 possible ASTS-orientations of the network in Figure 2

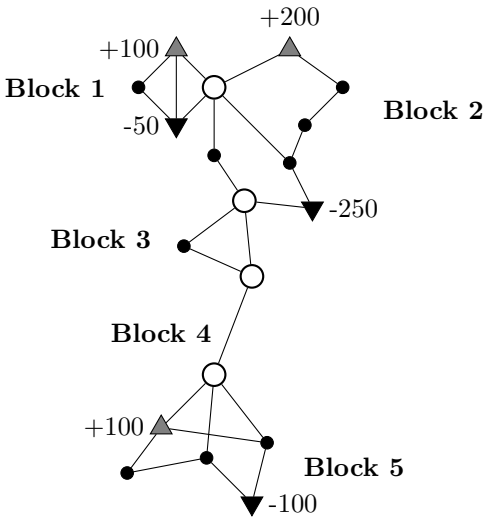

Figure 10: Inner block with zero flow on all edges (Block 3)

4 Remarks on zero flows

Detecting zero flows is of crucial importance to speed up an algorithm: As we have seen above we can interpret a forward arc of an ASTS-orientation as a non-negative flow and a backward arc as a non-positive flow. If we know beforehand whether a flow is zero we can reduce the number of orientations that are possible for a given network flow problem. In the following, based on our insights in the previous sections, we will provide an overview of different types of zero flows that may arise in potential-driven networks.

We can distinguish between four types of zero flows:

1. **Outer blocks** We know from Corollary 4 that we can uniquely decompose any graph into two subgraphs such that the edges on one subgraphs have always zero flows. The subgraph with zero flows consists of the blocks that we obtain by sucessively removing those leaves of the block graph of our underlying graph that represent blocks with no source and no sink other than the cut node of the block. (See the step from Figure 3 to Figure 4 above.) These flows are zero flows due to the structure of the network, independent of the demand at the sources and sinks and the pipeline characteristics, i.e. we can determine these zero flows prior to any optimization algorithm.

2. **Inner blocks** Once the blocks with zero flows from the previous item have been removed from the graph, we know from Theorem 1 that the remaining graph has an ASTS-orientation. However, while all remaining blocks do have an ASTS-orientation, there may be some blocks with flow zero on all edges. In these cases all forward and backward edges of an ASTS-orientation of the block correspond to zero flows. These blocks are characterized by the fact that all cut nodes are transhipment nodes

relative to the block. This means that all cut nodes of the block in question partition the graph into two components in each of which the sum of all demands at sources and sinks is zero. As a consequence, there is no flow through the cut nodes of the block and all edges of the block have a flow of zero. This situation is illustrated in Figure 10, where Block 3 is a block with zero flow. The nodes marked as circles are the cut-nodes of the graph. Zero flows in inner blocks arise due to the block structure of the network in conjunction with the demands at sources and sinks and are independent of any pipeline characteristics. Theorefore, we can determine these zero flows prior to any optimization procedure.

3. **Closed pipelines** Once inner and outer blocks have been removed from the graph, all remaining edges belong to blocks in which some, but not all arcs have non-zero flow.
   Trivially, zero flows can occur in such a block if a pipeline is closed off by a gate valve that blocks any flow into the pipeline. In optimization problems a gate will be closed off when any flow through the pipeline leads to sub-optimal solutions (due to transport costs, for example). This is a decision that depends on the cost structure of the optimization problem, possibly in conjunction with the structure of the block, the pipeline characteristics, the demand at the nodes of the block, and pressure losses in other parts of the network, depending on the variables in the objective function.
   Except in trivial cases, this type of zero flow cannot be detected prior to the optimization problem. We finally note that due to Theorem 1 (iv) is will always be possible to allocate to any such pipeline with zero flow a forward or backward arcs as part of an ASTS-orientation.

4. **Pipelines with pressure symmetry** There is another type of zero flow that can occur in blocks where not all flows are zero. Within a block the flow from the sources to the sinks of the block will, due to the pressure loss caused by the flow, typically distribute across all pipelines that are not closed off. However, a zero flow can occur "accidentally" due to the values of the demands at the sources and sinks of the block in combination with the pipeline characteristic. We will provide an example for the simplest cases in which this phenomenon can happen in the following.
   Consider the block $G(V, E)$ with
   $$V = \{1, 2, 3, 4\} \text{ and } E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}\}$$
   and let us assume that nodes 1 and 2 are sources with a demand of $+100$ each, while nodes 3 and 4 are sinks with a demand of -100 each (see Figure 11). For simplicity's sake let us further assume that all edges are equal with respect to all pipeline characteristics such as lengths and diameters. Disregarding symmetry, this graph has two possible ASTS-orientations: $D_1(V, A_1)$ with $A_1 = \{(1, 2), (2, 3), (1, 4), (4, 3)\}$ and $D_2(V, A_2)$ with $A_2 = \{(1, 2), (2, 3), (1, 4), (3, 4)\}$. (This can be seen easily by taking into account that due to Proposition 1 one of the sources must be a supersource and one of the sinks must be a supersink.)
   Let us consider the flow on the arcs in the case of orientation D_1. We denote by x_ij the flow on arc (i, j). All flows will be non-negative in the direction of the orientation. Any flow x_{12} on arc $(1, 2)$ implies that $x_{23} = 100+x12$. As node 3 has a demand of -100, we must have $x_{23} \leq 100$, hence $x_{12} = 0$. Accordingly, $x_{14} = 100$. Due to the demand at node 4

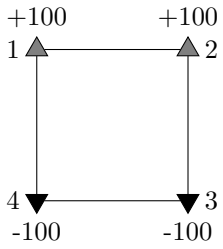

Figure 11: Pipeline with pressure symmetry

being -100, we also have $x_{43} = 3$, i.e. this ASTS-orientation implies zero flows on two arcs, $(1, 2)$ and $(4, 3)$.
Now let us us consider the flow on the arc of orientation D_2. We denote by $\delta\pi_{ij}$ the pressure loss along arc (i, j). If we have $x_{23} > x_{14}$, the pressure loss function in potential-driven graphs (see [8]) implies $\delta\pi_{23} > \delta\pi_{14}$, provided that both pipelines have the same physical characteristics. However, we also have $\delta\pi_{14} = \delta\pi_{12} + \delta\pi_{23} + \delta\pi_{34}$, which yields $\delta\pi_{14} \geq \delta\pi_{23}$. Since our ASTS-orientation in conjunction with the demands at the nodes does not permit the case that $x_{23} < x_{14}$, we must have $x_{23} = x_{14}$, and hence $x_{12} = x_{34} = 0$, i.e. this ASTS-orientation implies zero flows on the same two edges as the previous ASTS-orientation.
As there are no flows on potential-driven networks that do not correspond with an ASTS-orientation (cf. [8]) and we have considered all ASTS-orientations of G (apart from symmetrical cases), we can conclude that all feasible flows on our underlying graph G imply zero flows on the edges $\{1, 2\}$ and $\{3, 4\}$.
We note that in the case of D_1 the zero flow occured due to the demand situation of the network in conjunction with the ASTS-orientation, while in the case of D_2 we also required a symmetric pressure situation in the two components induced by the edge cut $\{(1, 2), (3, 4)\}$. This implies that in most cases it will be difficult to detect such a zero flow prior to the optimization procedure. Clearly, a necessary condition for such a non zero flow to occur in a pipeline is that the pipeline is an element of an edge cut of the graph such that in both node sets of the partition induced by the cut the sum of all demands at sources and sinks is zero. However, this condition is not sufficient as we can see when considering the graph G' that results from subdividing the edges $\{1, 2\}$ and $\{3, 4\}$ of G with new transshipment nodes 5 and 6, respectively, and introducing the edge $\{5, 6\}$. In this case the pressure differentials between nodes 1 and 4 and between nodes 2 and 3 imply that there will be a flow on the edge $\{5, 6\}$, and, as a consequence, the network will not permit any non-zero flows (unless a pipeline is shut down). Further research that would allow to distinguish cases of zero flow that fall into this category as a part of pre-solving an optimization problem may well be useful to save computational time.

Table 2 sums up the cases of zero flows that we have described.

Type	**Cause**	**Independent of pipeline characteristics?**	**Independent of objective function?**	**Does block/edge allow for ASTS-orientation?**	**Part of presolving?**
Outer block	Nodes not on path between source and sink	Yes	Yes	No	Yes
Inner block	Cut nodes are transhipment nodes	Yes	Yes	Yes	Yes
Accidentally unused pipe	Symmetry of pressure loss	No	Possible	Yes	Possibly
Closed pipe	Pipe closed by valve	No	No	Yes	Typically not

Table 2: Types of zero flows

5 Algorithms for generating ASTS-orientations

In this section we present an algorithm for enumerating all ASTS-orientations of a given graph. Throughout the section we will assume that the underlying graph admits an ASTS-orientation according to Theorem 1. In a second section we discuss other methods of generating ASTS-orientations.

5.1 An algorithm for enumerating ASTS orientations

For the problem of finding an acyclic orientation of a graph that has exactly one supersource, no other source and no sink, [4] provide an algorithm the main idea of which can be summarized as follows:

1. Construct a DFS tree that is rooted in the supersource.
2. Start from an empty digraph and traverse the tree backwards from the leaves towards the root, adding one node after the other to the empty digraph.
3. For each new node create a list of nodes that are reachable from the new node and a list of nodes that can reach the new node by drawing on the forward and backward reachabilities of its neighbours that are already nodes of the digraph under construction.
4. For each node, generate all possible orientations of all arcs to nodes that have already been added to the digraph such that all nodes who become saturated by neighbours due to adding the new node have both indegree and outdegree of at least 1. When doing so, update the reachability lists and discard orientations that produce cycles.

This algorithm guarantees an acyclic orientation because it discards orientations that produce cycles. It guarantees an STS-orientation because when traversing the tree, each new transhipment node will have a neighbour further up the tree (towards the root), which ensures that for each node, except the supersource, there will be a neighbour left to prevent an outdegree of 0, which would lead to making a source a supersource other than the given a supersource.
What makes this algorithm very efficient is the fact that the search backwards up to the root prevents the algorithm from reaching a "dead end" that could be caused by violating (a) the STS-condition or (b) the acyclicity condition.
In case (a), the algorithm would have to stop constructing an orientation when adding a new node with arcs to its neighbours would force a neighbour to become a supersource. As mentioned above, this cannot happen due to the search backward up to the root of the tree.
In case (b) the algorithm would have to stop when adding new arcs would inevitably lead to a cycle. It is easy to see that for each new node added to the orientation there exist arcs incident to those of its neighbours that are already part of the orientation that do not cause a cycle: simply orient all arcs as arcs emanating from the new node. Note that this does not conflict with case (a) as we will have come to a node later up the tree that will have an arc that is incoming at the current node. (For details see [4].)
Unfortunately, we cannot use this algorithm directly for ASTS-orientations because ASTS-orientations may have

(i) several sources

(ii) at least one of which must be a supersource, and

(iii) several sinks

(iv) at least one of which must be a supersink.

In the following we will present a modified version of the enumeration algorithm that takes these aspects of ASTS-orientations into account. Unfortunately, in contrast to the algorithm in [4], we cannot guarantee anymore that our algorithm will not reach any dead end. It is, to the best of our knowledge, an open problem whether an enumeration algorithm for ASTS-orientations exists that is guaranteed to construct an ASTS-orientation by adding one node after the other and orienting the edges incident to those neighbours that are already part of the oriented digraph.
We will deal with aspects (i) and (ii) by introducing a dummy supersource s that is adjacent to all sources of the graph $G(V, E)$ for which we would like to find ASTS-orientations, i.e. we will consider the graph $G'(V \cup \{s\}), E \cup \{\{s, v\} : v \in V_+\}$. Then making sure that s is oriented as the only supersource on G' is equivalent to ensuring that there are no supersources on G other than the nodes in V_+. Additionally we will have to make sure that each source in V_+ will have at least one outgoing arc because the procedure of introducing the dummy supersource may lead to an orientation in which there exists a node in V_+ that is a supersink. We will achieve this by forcing the last unoriented edge incident to a source to be oriented as an emanating arc if the source would become a supersink otherwise. Should this forced orientation lead to a cycle, we will disregard the corresponding branch in our search tree and backtrack.
For taking into account sinks, we will need to sort our node differently. Recall

that in the algorithm in [4] a DFS tree is rooted in the supersource is constructed and then the nodes to be added to build up an orientation are sorted such that the tree is traversded backwards from the leaves towards the root. If the graph has a sink, we require a tree that is rooted in the supersource and has the sink as a leave. We construct such a tree by starting with a path from the supersource to the sink and then proceeding along this path with a BFS based on the neigbourhood relation.
On this basis, we can deal with aspects (iii) and (iv) analogously to aspects (i) and (ii) by introducing a dummy sink that is adjacent to all sinks of the graph, by forcing the orientation of an edge should a node in V_- become a supersource otherwise, and by disregarding the corresponding branch of the search tree should a cycle be the consequence of the forced orientation.
The following we will present our algorithm for enumerating all ASTS-orientations of a given graph $G(V, E)$. In the light of the previous section, this algorithm should be applied after the graph has been simplified, and it should be applied to all blocks of the graph separately.
In a first step, Algorithm 1 sorts the node set V into a list such that the first node of the list is the dummy sink, the last node is the dummy sink, and for all nodes $u \in V$ there exists a node $v \in V$ that is neighbour of u and appears after u in the list. Afterwards it calls the next algorithm that generates the ASTS-orientations.

Algorithm 1: Sorting nodes and initializing **GenerateOrientations**

Input : A graph $G(V, E)$ with sources $V_+ \subseteq V$ and sinks $V_- \subseteq V$
Output : A list L of sorted nodes

$V := V \cup \{s, t\}$;
$E := E \cup \{\{s, v\} : v \in V_+\} \cup \{\{t, v\} : v \in V_-\}$;
$P :=$ list of nodes that constitute a path from s to t;
for $i \in P$ **do**
    Add i to L;
    FollowNeighbour $(G(V, E), P, i, L := \varnothing)$;
Reverse order of nodes in L;
GenerateOrientations $(G(V, E), L, D_0, t, R_0)$;

Function `FollowNeighbour`$((G(V, E), P, i, L))$
    for $j \in N(i)$ **do**
        if $j \notin Lj \notin P$ **then**
            Add j to L;
            FollowNeighbour $(G(V, E), P, i, L)$;

The following algorithm, called from Algorithm 1, starts with an empty digraph and adds one node i after the other from the node list L, each of which will be connected via arcs with those of its neighbours that are already part of the digraph. The collection of all possible combinations of arc orientations is generated by Algorithm 3 and stored in a list of valid assignments A^*. When adding the arcs of a valid assignment, we make sure that no source can be equipped with the orientations of a supersink and that no sink can be equipped with the orientations of a supersource. We denote the oriented digraph that

includes all nodes up to (and including) node i by D_i. A dictionary R_i keeps track of the nodes that can be reached from any node via forward and backward arcs on D_i. One a valid assignment has been added, the function recursively calls itself to add another node from the list L and expand the existing partial orientation D_i.

Algorithm 2: GenerateOrientations

Input : A graph $G(V, E)$, the sorted list L, the current partial digraph D_{i-1}, the current node i to be added, the current reachability dictionary R_{i-1}

Output : An orientation D containing D_{i-1}

if $i = s$ **then**
  Remove dummy sink t from D_{s-1};
  output finished orientation D_{s-1};
else
  $A^* :=$ **ListValidAssignments** $(G(V, E), L, D_{i-1}, R_{i-1}, i, \varnothing)$;
  for *any set of arcs* $A \in A^*$ **do**
    Construct $D_i(V_i, A_i)$ *with* $V_i = V_{i-1} \cup i$ *and* $A_i = A_{i-1} \cup A^*$;
    if $(i \notin V_+ \vee deg_{D_i}(i) \neq deg_G(i) \vee deg_{D_i}(i) \neq indeg_{D_i}(i)) \wedge (i \notin V_- \vee deg_{D_i}(i) \neq deg_G(i) \vee deg_{D_i}(i) \neq outdeg_{D_i}(i))$ **then**
      for $(u, v) \in A$ **do**
        $R_i :=$ **ReachabilityUpdate** (R_{i-1}, u, v) ;
      GenerateOrientations $(G(V, E), L, D_i(V_i, A_i, i + 1, R_i)$;

Called from Algorithm 2, this Algorithm generates for a new node i added to the partial orientation D_{i-1} all valid sets of orientations of all edges that connect i with those of its neighbours that are already nodes in D_{i-1}. The latter neighbours are denoted by the set N'. The incumbent valid set of oriented edges that is being constructed during a particular iteration of the algorithm is denoted by A. All such set are stored in the list A^*. So make sure that we do not create a supersource or a supersink we store those nodes than could become a supersource or a supersink upon adding the node i in the set V_+. The algorithm contains a function *AddArc* that recursively adds one oriented edge after the other to be added to the current set A.

Finally we need an algorithm to update the reachability dictionary R_i. This is achieved by the following Algorithm 4. The dictionary has a forward component Ri^{forw} and a backward component Ri^{back} such that $Ri^{forw}(u)$ denotes the set of nodes to which there exists a dipath from u and $Ri^{back}(u)$ refers to the set of nodes from which there exists a dipath to u.

5.2 Other approaches for constructing ASTS-orientations

In the following we will describe some alternative ways for arriving at ASTS-orientations. While these may not be useful for enumerating all ASTS-orientations, they may turn out to be helpful when ASTS- orientations are used in an optimization context.

Algorithm 3: ListValidAssignments

Input : A graph $G(V, E)$, the sorted list L, the current partial digraph $D_{i-1}(V_{i-1}, A_{i-1}$, the current reachability dictionary R_{i-1}, the current node i to be added, the current list A^* of valid orientations

Output : A list A^* of all valid assignments of orientations to edges incident to i

$A := \varnothing$;
$V_+^* := \varnothing$;
$V_-^* := \varnothing$;
if *i is a sink* **then**
  $A := A \cup \{(i, t)\}$
for $j \in N(i)$ **do**
  if $j \in V_i \wedge deg_{D_i}(j) = outdeg_{D_i}(j) = deg_G(j) - 1$ **then**
    $A := A \cup \{(i, j)\}$;
    ReachabilityUpdate (R_i, i, j);
  if $j \in V_i \wedge deg_{D_i}(j) = indeg_{D_i}(j) = deg_G(j) - 1$ **then**
    $A := A \cup \{(j, i)\}$;
    ReachabilityUpdate (R_i, j, i);
if *j can reach itself via forward or backward arcs* **then**
  Delete update of R_i and added arcs in A;
  return;
$N' := (N(i) \cap V_{i-1}) \setminus V_+^* \setminus V_-^* \setminus \{t\}$;
AddArc $(L, i, A, N', 0, R_i)$;

Function `AddArc`(L, i, A, N', k, R_i)
  if $N' = \varnothing$ **then**
    $A^* := A^* \cup A$;
    return ;
  if $A \neq \varnothing$ **then**
    $(u, v) :=$ last arc added to A ;
    ReachabilityUpdate (R_i, u, v);
  if *i is not forward-reachable from k* **then**
    AddArc $(L, i, A \cup (k, i), N' \setminus \{k\}, k + 1, R_i)$;
  if *i is not backward-reachable from k* **then**
    AddArc $(L, i, A \cup (i, k), N' \setminus \{k\}, k + 1, R_i)$;

Algorithm 4: ReachabilityUpdate

Input : A reachability dictionary R_i, and an arc (u, v) that extends the reachability relation

Output : An updated reachability dictionary R_i

$R_i^{forw}(u) := R_i^{forw}(u) \cup \{v\}$;
$R_i^{back}(v) := R_i^{back}(v) \cup \{u\}$;
for $n \in R_i^{forw}(v)$ **do**
    $R_i^{forw}(u) := R_i^{forw}(u) \cup \{n\}$;
    $R_i^{back}(n) := R_i^{back}(n) \cup \{u\}$;
for $n \in R_i^{back}(u)$ **do**
    $R_i^{forw}(n) := R_i^{forw}(n) \cup \{v\}$;
    $R_i^{back}(v) := R_i^{back}(v) \cup \{n\}$;
    for $m \in R_i^{forw}(v)$ **do**
        $R_i^{forw}(n) := R_i^{forw}(n) \cup \{m\}$;
        $R_i^{back}(m) := R_i^{back}(m) \cup \{n\}$;

The first approach we are going to look at proceeds in a way opposite to the the algorithm presented in the previous subsection, which attempts to construct orientations where the nodes satisfy the STS-condition and then checks whether these are acyclic. Obviously, an alternative approach would be to construct acyclic orientations and then choose only those where all nodes satisfy the STS-condition. For the first part we can exploit the fact that all permutations of the nodes of the graph give rise to an acyclic orientation when we go through the list of nodes in the order of the permutation and at each node orient all so far unoriented edges such that they are emanating ftom the node.
Let $G(V, E)$ be the graph for which we would like to generate all ASTS orientations. We proceed as follows:

1. Generate all trees of G that have roots in V_+. Let v_1 denote the rootnode of such a tree.
2. For each tree create all permutations of nodes $(v_1, v_2, ..., v_{|V|})$ that respect the condition that all nodes v_i are in the neighbourhood of some node v_j with $j < i$.
3. Go through the nodes on each tree in the order of each of its permutations and for each node orient all non-oriented edges away from the node. If during this process a node is reached all edges of which have already been oriented, discard the permutation (STS-orientation impossible to satisfy).

Since all acyclic graphs can be constructed via permuations of nodes and orienting edges in the order of the node permutation and since all STS-orientations satisfy the above neighbourhood condition for some tree that has a root in V_+, this procedure even guarantees that we will find all ASTS orientations.
Clearly, the algorithm has two limitations: As some permutations lead to the same acyclic graph, we may count some ASTS-orientations double. As we may have to discard some permutations due to not satisfying the STS property, the algorithm may not be very efficient.

We have seen above that also the algorithm in the previous section can happen to reach "dead ends" where partially constructed orientation has to be discarded. For enumerating acyclic orientations and for enumerating orientations that satisfy the STS-condition it is easy to see that such a "dead end" can be prevented. It must be considered an open question for further research whether it is possible in principle to arrive at a more streamlined algorithm.
A possible candidate procedure for such an algorithm could be the concept of reverse search for enumeration ([1]). In the case of acyclic orientations graphs, a straight-forward reverse search procedure can be built on the insight that each acyclic digraph has an arc the orientation of which can be reversed to yield another acyclic digraph and that in this way all acyclic digraphs on the same underlying graph can be sucessively converted into each other ([7]).
It can easily be seen that the same simple principle does not apply to ASTS-oriented digraphs. However is an open question whether a similar same structural insight may apply to ASTS-oriented graphs, or at least to a subclass of ASTS-oriented graphs.
Another way of generating an ASTS-orientation can be based on the proof of Theorem 1 of the present paper. In the proof we constructed an ASTS-orientation by means of a decomposition of the underlying graph that is similar to an open ear decomposition (cf. [2]), just that we used a forest instead of a path as the starting point. For each oriented path that we added during the process of our "forest-based" ear decomposition, we can freely choose the orientation, as long as no cycle results and as long as we orient the entire path into the same direction. Clearly, by choosing the path to add during this procedure and by choosing its orientation, we can build a large variety of "customized" ASTS-orientations, i.e. orientations that may be useful within an optimization framework.
Finally let us briefly note that further potential for constructing ASTS-orientations may be found in the application of the theory of bipolar graphs, i.e. graphs with exactly one supersource and one supersink, where all other nodes are transhipment nodes. For these graphs there exists a method of counting orientations based on successive minors of the underlying graph ([5]). It would be worth studying the respect in which the theory of bipolar graphs may be extended to complement the theory of ASTS-orientations developed in the present paper.

6 Conclusion

In this paper we have introduced the concept of ASTS-orientations to study the interplay of two graph-theoretical structures: the STS-condition, according to which an orientation must be compatible with the location of sources, sinks and transhipment nodes on a graph, and acyclicity. This double structure is interesting to study because it governs the interplay of the flow conservation constraints and the acyclicity of flows on potential-based networks.
In particular, we have found a theorem with different characterizations of graphs that allow for ASTS-orientations and an upper bound for the number of ASTS-orientations of a given undirected graph, and have achieved an understanding of the role of zero flows in the context of flow orientations in potential-based networks.
The theoretical results from the previous item have been capitalized on to con-

struct an approach to sucessively simplify and decompose a given network for which an ASTS-orientation is to be constructed. It exploits the block structure that can typically be found in real-world infrastructure networks.
Based on these results, an algorithm has been developed that is capable of enumerating all ASTS-orientations of a given network, which may be used to inform a configuration model for optimization-based bound tightening ([8]).
There are several avenues for further research on the structure of ASTS-orientations. In particular, we have mentioned the following three aspects:
For a given ASTS-orientation, changes in the orientation of an individual arc or a subset of arcs may or nor may not lead to a new ASTS-orientation. We will study the conditions (as related to the subset of arcs, the present ASTS-orientation, or the structure of the graph) under which the ASTS-property will be preserved and seek to exploit this insight to find a more efficient algorithm for enumerating all ASTS-orientations, based on an existing approach for enumerating acyclic orientations. In this context it may also be useful to study the relevance of the tree-width of a graph for the complexity of the enumeration of ASTS-orientations.
Many network flow problems come with further restrictions on the flow direction of arcs. It is therefore useful to study the conditions under which restrictions on the orientation of arbitrary arcs can still guarantee the existence of an ASTS-orientation on the entire network and how such an orientation can be constructed.
In the present paper we have identified four types of zero flows that can occur in potential-based networks. Two of these can be identified prior to an optimiztion procedure on the network. It would be useful to find ways to recognize zero flows of the other two types within a pre-solving procedure.
ASTS-orientations are closely related to bipolar orientations of graphs. It would be interesting to study this relationship in detail to see what results from the theory of bipolar orientations translate into new results about the structure of ASTS-orientations. In particular, further research into this direction may lead to new algorithms for enumerating ASTS-orientations, which is not only an interesting mathematical question per se, but is also likely to improve the efficiency of solving flow optimization problems on potential-driven networks.

Acknowledgements This research is carried out in the framework of MATHEON supported by Einstein Foundation Berlin (project MI10). The authors would also like to thank the DFG for their support within project A04 in CRC TRR154 and the BMBF Research Campus Modal (fund number 05M14ZAM) and ICT COST Action TD1207 for additional support.

References

[1] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

[2] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[3] Kai Helge Becker and Benjamin Hiller. ASTS orientations on undirected graphs. ZIB Report 18-31, Zuse Institute Berlin, 2018. `http://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/6963`.

[4] Alessio Conte, Roberto Grossi, Andrea Marino, and Romeo Rizzi. Efficient enumeration of graph orientations with sources. *Discrete Applied Mathematics*, 246:22–37, 2018.

[5] Hubert De Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. Bipolar orientations revisited. *Discrete Applied Mathematics*, 56(2-3):157–179, 1995.

[6] Reinhard Diestel. *Graph theory (Graduate texts in mathematics)*, volume 173. Springer Heidelberg, 2005.

[7] Komei Fukuda, Alain Prodon, and Tadashi Sakuma. Notes on acyclic orientations and the shelling lemma. *Theoretical computer science*, 263(1-2):9–16, 2001.

[8] Benjamin Hiller and Kai Helge Becker. Improving relaxations for potential-driven network flow problems via acyclic flow orientations. *ZIB Report 18–30, Zuse Institute Berlin*, 2018.

[9] Ralf Lenz and Kai Helge Becker. Optimization of capacity expansion in potential-driven networks including multiple looping - a comparison of modelling approaches. *ZIB Report 18–44, Zuse Institute Berlin*, 2018.