Operations Research
DOI 10.1287/opre.1080.0524ec
pp. ec1–ec5

e-companion

ONLY AVAILABLE IN ELECTRONIC FORM

Electronic Companion—"The Pseudoflow Algorithm: A New Algorithm for the Maximum-Flow Problem" by Dorit S. Hochbaum, *Operations Research*, DOI 10.1287/opre.1080.0524.

Appendix A: Motivation and Applications of Tree Normalization

When solving a flow problem in practice, it is often the case that numerous similar instances, differing from each other in minor data changes, should be solved. When the input changes slightly it would be beneficial to take advantage of the solution attained for the previous input as a starting point for solving the problem for the new input – a *warm start* process. Such warm start for the pseudoflow algorithm requires normalizing the tree after the capacities have been modified. It is sufficient to consider capacity changes because all changes in the network data can be reduced to changes in arc capacities: Removing a node is accomplished by changing all arc capacities adjacent to the node to 0. Similarly arcs can be removed or included by setting their capacities to 0 or positive values respectively.

The procedure Normalize, presented here, takes as input a tree T and a specification, or a partition $A \setminus T = A^s \cup A^0$, of the out-of-tree arcs where the pseudoflow is to meet their lower bounds A^0 and those where the pseudoflow is to meet the upper bounds A^s. When arc capacities values change, the procedure finds a new pseudoflow f' that meets the new lower or upper bounds on the new capacities of out-of-tree arcs $A \setminus T$. The normalization procedure completes the specification of f' also on the in-tree arcs of T, and delivers a normalized tree T' associated with f'. The in-tree arcs of T' satisfy $T' \subseteq T$ and the out-of-tree arcs $A \setminus T \subseteq A \setminus T'$. In other words, this renormalization procedure can only split each branch of T to a number of branches of T', but branches cannot be merged.

One application where capacities change in a series of instances is the parametric analysis described in Section 9. There the sequence of networks has each subsequent modification restricted to monotone changes in the source adjacent and sink adjacent arcs. In that case all strong nodes remain strong, and the only change in node status is due to some weak nodes becoming strong. In another example the capacity changes apply to arcs interior to G, or the changes are to source and sink adjacent arcs that are not monotone. This happens in a project scheduling problem with resource constraints described by Möhring et al. (2003). A lagrangian relaxation of the shared resource results in a minimum cut problem on a graph. Consecutive iterations in the subgradient method changes the capacities as functions of the Lagrange multipliers, but the changes are small. The motivation of using warm starts is the expectation that if the changes in capacities are small, then only a few iterations will be required to reach an optimal solution to the new instance.

Another application of the normalization procedure is for the directed minimum cut problem. The directed minimum cut problem is to partition a graph $G = (V, A)$ to two subsets $A, \bar{A}$ so that the cut capacity $C(A,\bar{A})$ is minimum. This problem is distinguished from the minimum s,t-cut problem in that the sets are not required to contain the specific nodes s and t respectively. It is possible to solve this problem using the pseudoflow algorithm in complexity $O(mn\log n)$, Hochbaum (2003). The algorithm involves changing the identity of sink nodes between iterations. At the end of an iteration a sink node t_1 is shrunk with the source node and another node t_2 (selected in a particular way) becomes a sink node. This means that the arcs incoming into t_1 are set to their lower bounds, the arcs outgoing from t_1 are set to their upper bounds, and the arcs incoming into t_2 are set to their upper bounds. So rather than changing capacities, we change the pseudoflow values on out-of-tree arcs. Since changing the flow values renders the tree no longer normalized we use the normalization procedure described in the next subsection.

Appendix B: Normalizing a Tree

Given a spanning tree T rooted in r in G^{ext} and a pseudoflow f that has all out-of-tree arcs $A\setminus T$ at their upper or lower bounds, we show how to normalize the tree in linear time, maintaining the values of f on out-of-tree arcs.

The normalization procedure is applied to each tree (or branch) in the forest $T\cap A$ by scanning it from the leaves up assigning each node v a flow going to $p(v)$ which balances the flows from and to leaves and out-of-tree arcs. If the total amount of flow from all nodes other than the parent is positive then we send the flow balance up to $p(v)$. If the total amount is negative then we pull it from $p(v)$ to v. When flow blocking edges are encountered because a residual capacity in the direction required for balancing the flow is not sufficient, the edges $[v,p(v)]$ are split and suspended from r along with their subtree, as strong branches when the flow amount is positive, and as weak ones when the flow amount is negative.

The information given in square brackets [ ] is the generation of the values of the pseudoflow f on in-tree arcs that is associated with the newly normalized tree. Recall that $excess(v)$ represents the excess at node v. In this procedure we assume, as we had throughout, that all lower bounds are zero. Adapting it to non-zero lower bounds is straight forward.

```
procedure Normalize $(T, A^s)$
begin $excess(v)=0\ \forall v\in T,\ T'=T$
until $v=r$ Do
    Select a leaf node in the tree of unscanned edges $v\in T$.
    {Compute $excess(v)$}: $excess(v)\leftarrow excess(v)+\sum_{(u,v)\in A^s} c_{u,v}-\sum_{(v,u)\in A^s} c_{v,u}$.
```

```
        If $excess(v) = 0$ then split $\{(v, p(v)), 0\}$;
        Else {Update $excess(p(v))$}:
            For $excess(v) > 0$: If $c_{v,p(v)} > excess(v)$,
            $excess(p(v)) \leftarrow excess(p(v)) + excess(v)$. [$f_{v,p(v)} = excess(v)$.]
            Else, split $\{(v, p(v)), excess(v) - c_{v,p(v)}\}$; $excess(p(v)) \leftarrow excess(p(v)) + c_{v,p(v)}$.
                [ $f_{v,p(v)} = c_{v,p(v)}$.]
            For $excess(v) < 0$: If $c_{p(v),v} > -excess(v)$,
            $excess(p(v)) \leftarrow excess(p(v)) + excess(v)$. [$f_{p(v),v} = c_{p(v),v} + excess(v)$.]
            Else, split $\{(v, p(v)), -excess(v) - c_{p(v),v}\}$; $excess(p(v)) \leftarrow excess(p(v)) - c_{p(v),v}$.
                [$f_{p(v),v} = c_{p(v),v}$.]
        Remove $(v, p(v))$: $T \leftarrow T \setminus (v, p(v))$.
repeat.
end.
```

```
procedure split $\{(a, b), M\}$
$T' \leftarrow T' \setminus (a, b) \cup (a, r)$; $excess(a) = f_{ar} = M$;
$a$ is the root of a weak branch if $M \leq 0$ else it is the root of strong branch;
end
```

The output tree T' is normalized since any node that does not satisfy flow balance is set as a root of a respective strong or weak branch using an excess or deficit arc connecting it to r. Also all downwards residual capacities are positive since if there is 0 excess then the 0-deficit branch is split. Or when a flow is pulled downwards towards v from $p(v)$ there is again a split if the downwards residual capacity is equal to the amount of deficit of v. Thus properties 1, 2, 3 and 4 are satisfied and the tree is normalized.

Lemma B.1 *The complexity of procedure* Normalize *is $O(n)$ when the net inflows into each node are given. Otherwise it is $O(m)$.*

Proof: Given the net inflows into each node resulting from out-of-tree arcs, the algorithm scans each in-tree arc and node at most $O(1)$ times. Thus the complexity if $O(n)$. If these inflows are not given then computing them requires $O(m)$ steps. Q.E.D.

Given a tree and a set of saturated arcs A^s, a corresponding pseudoflow is constructed in linear

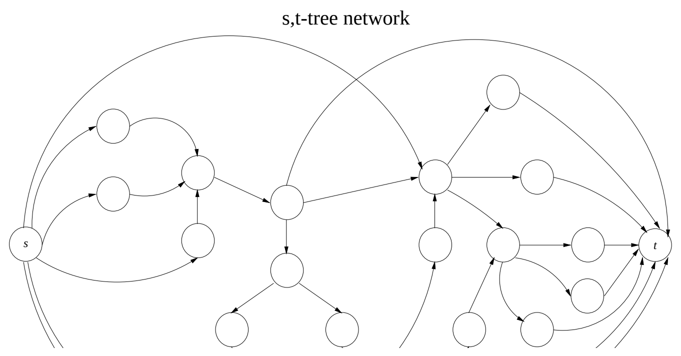

Figure 1 An s,t-tree network

time $O(m)$. We call this pseudoflow generation procedure that includes the [] statements, **procedure** pseudoflow-tree. This pseudoflow can then be used to derive a corresponding feasible flow, as shown in Section 8, in $O(m \log n)$ steps.

Appendix C: Maximum Flow on s,t-tree Network in Linear Time

Let T be a directed acyclic graph which is a tree or a forest (in the undirected sense). An s,t-tree network is a graph $G_{st} = (T \cup \{s,t\}, T \cup A(s) \cup A(t))$ formed by joining T to source and sink nodes that are linked to any subset of nodes in T. All arcs carry capacity values. An example of an s,t-tree network is illustrated in Figure 8. One of the motivation for studying maximum flow on s,t-tree networks is that it serves as a key operation in an efficient minimum cost network flow algorithm, as shown by Vygen (2002).

To solve the maximum flow problem on an s,t-tree network we apply the process of normalization to the tree or forest T where each connected component is arbitrarily rooted and $A^s = A(s) \cup A(t)$ and call for **procedure** pseudoflow-tree. The output is a normalized tree without a residual arc between strong and weak nodes: To see this, observe first that all out-of-tree arcs are generated as split arcs during the normalization process. We show that none of the split arcs is residual from a strong to weak node: If an upwards arc is split, this is because its tail node has positive excess exceeding its capacity. That node becomes then the root of a strong branch and the arc leaving that branch is saturated. Similarly for a downwards arc, if it is split, that is because its head node has deficit that exceeds the capacity of the arc. That head node becomes a weak branch that has the only incoming arc to it, saturated. This normalized tree is thus optimal, and the strong nodes form a maximum blocking cut set and the source set of a minimum cut. It remains to find the corresponding feasible flow by sending the excesses and deficits back to source and sink respectively.

This can be done in linear time $O(n)$ using an algorithm similar to the one for finding feasible flow on closure graphs, Hochbaum (2001). We have thus proved the theorem,

Theorem C.1 *The maximum flow problem on s,t-tree networks is solved in linear time $O(n)$.*

References

Hochbaum D. S. 2001. A new-old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks* **37**(4) 171–193.

Hochbaum D. S. 2003. A pseudoflow algorithm for the directed minimum cut problem. Manuscript, UC Berkeley.

Möhring R. H., A. S. Schultz, F. Stork, M. Uetz. 2003. Solving project scheduling problems by minimum cut computations. *Management Science* **49** 330–350.

Vygen J. On Dual Minimum Cost Flow Algorithms. 2002. *Math. Methods of Oper. Res.* **56** 101–126.