Incremental Network Design with Maximum Flows

Thomas Kalinowski[†] Dmytro Matsypura[⋆]
Martin W.P. Savelsbergh[†]
[†] *University of Newcastle, Australia* [⋆] *University of Sydney, Australia*

November 13, 2018

Abstract

We study an incremental network design problem, where in each time period of the planning horizon an arc can be added to the network and a maximum flow problem is solved, and where the objective is to maximize the cumulative flow over the entire planning horizon. After presenting two mixed integer programming (MIP) formulations for this NP-complete problem, we describe several heuristics and prove performance bounds for some special cases. In a series of computational experiments, we compare the performance of the MIP formulations as well as the heuristics.

1 Introduction

In the planning process for many network infrastructures, when the network is constructed over a significant period of time, the properties of the intermediate partial networks have to be taken into account. *Incremental network design*, introduced in [1], represents a class of optimization problems capturing that feature and combining two types of decisions: which arcs should be added to a given network in order to achieve a certain goal, and when should these arcs be added?

Variants of this problem have been studied in diverse contexts, for instance, the design of transportation networks [4, 11], network infrastructure restoration after disruptions due to natural disasters [2, 5, 6], and the transformation of an electrical power grid into a smart grid [7, 8].

A general class of mathematical optimization problems that captures essential features of the described decision problems, and includes the problem discussed in this paper as a special case, was introduced in [9, 10], where an *integrated network design and scheduling* problem is specified by (1) a scheduling environment that describes the available resources for adding arcs to the network, (2) a performance measure, which prescribes how a given network is evaluated (for instance by the shortest s-t path or by the maximum s-t flow in the network), and (3) by the optimization goal, which is either to reach a certain level of performance as quickly as possible or to optimize the cumulative performance over the entire planning horizon.

We focus on the special case corresponding to incremental network design as introduced in [1]. Our scheduling environment is such that at most one arc can be added to the network in each time period, and we optimize the cumulative performance, i.e., the sum of the performance measures of the networks in all time periods. Even in this simple setting, the problem has been shown to be NP-complete for classical network optimization problems: for the shortest s-t path problem in [1], and for the maximum s-t flow problem in [10]. Interestingly, the incremental variant of the minimum spanning tree problem can be solved efficiently by a greedy algorithm [3], while it becomes NP-complete in the more general setup of [10]. The performance measure considered in this paper is the value of a maximum s-t flow.

In Section 2, we introduce notation, state the problem precisely, and present two MIP formulations. In Section 3, we describe three heuristics, the first one seeks to increment the flow as quickly as possible, the second seeks to reach a maximum flow as quickly as possible, and the third one is a hybrid of the first two. In Section 4, we prove performance guarantees for the first two heuristics in special cases: for unit capacity networks they provide a 2-approximation algorithm and a 3/2-approximation algorithm, respectively, and these bounds can be strengthened when the network has a special structure. Section 5 discusses the results of

a set of computational experiments using randomly generated instances. After describing the instance generation, we compare the performance of the two MIP formulations, and evaluate and compare the heuristics on hard instances. We end, in Section 6, with some final remarks.

2 Problem formulation

We are given a network $D = (N, A)$ with node set N and arc set $A = A_e \cup A_p$, where A_e contains existing arcs and A_p contains potential arcs, as well as a source $s \in N$ and sink $t \in N$. For each arc, we are given an integer capacity $u_a > 0$, and for every node $v \in N$, we denote with $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ the set of arcs entering v and the set of arcs leaving v, respectively. Let $T > |A_p|$ be the length of the planning horizon. In every period, we have the option to expand the useable network, which initially consists of only the existing arcs, by "building" a single potential arc $a \in A_p$, which will be available for use in the following time period. In every period, the value of a maximum s-t flow is recorded (using only useable arcs, i.e., existing arcs and potential arcs that have been built in previous periods). The objective is to maximize the total flow over the planning horizon. Note that the length of the planning horizon ensures that every potential arc can be built. We refer to a maximum s-t flow using only existing arcs as an *initial maximum flow*, and to a maximum flow for the complete network as an *ultimate maximum flow*. This problem is strongly NP-hard [10] even when restricted to instances where every existing arc has capacity 1 and every potential arc has capacity 3. (A simple proof of this result can be found in the appendix.)

The problem can be formulated as a mixed integer program. For every $a \in A$ and $k \in \{1, \dots, T\}$, we have a flow variable $x_{ak} \geqslant 0$, and for every $a \in A_p$ and $k \in \{1, \dots, T\}$, we have a binary variable y_{ak} which indicates if arc a is built before period k ($y_{ak} = 1$) or not ($y_{ak} = 0$). The incremental maximum flow problem is then

$$\max \sum_{k=1}^{T} \left(\sum_{a \in \delta^{\text{out}}(s)} x_{ak} - \sum_{a \in \delta^{\text{in}}(s)} x_{ak} \right)$$

subject to

$$\begin{aligned}
\sum_{a \in \delta^{\text{out}}(v)} x_{ak} - \sum_{a \in \delta^{\text{in}}(v)} x_{ak} &= 0 && \text{for } v \in N \setminus \{s,t\},\ k \in \{1, \dots, T\}, \\
x_{ak} &\leqslant u_a && \text{for } a \in A_e,\ k \in \{1, \dots, T\}, \\
x_{ak} &\leqslant u_a y_{ak} && \text{for } a \in A_p,\ k \in \{1, \dots, T\}, \\
y_{ak} &\geqslant y_{a,k-1} && \text{for } a \in A_p,\ k \in \{2, \dots, T\}, \\
y_{a1} &= 0 && \text{for } a \in A_p, \\
\sum_{a \in A_p} (y_{ak} - y_{a,k-1}) &\leqslant 1 && \text{for } k \in \{2, \dots, T-1\}, \\
x_{ak} &\geqslant 0 && \text{for } a \in A,\ k \in \{1, \dots, T\}, \\
y_{ak} &\in \{0,1\} && \text{for } a \in A_p,\ k \in \{1, \dots, T\}.
\end{aligned}$$

We denote this formulation by IMFP^1.

A potential weakness of IMFP^1 is that it may suffer from symmetry. If multiple arcs need to be build to increase the maximum s-t flow, then the order in which these arcs are build does not matter, which introduces alternative, symmetrical solutions. Next, we present an alternative MIP formulation which avoids this difficulty. Let f and F denote the initial and the ultimate maximum flow value, respectively, and let $r = F - f$. We introduce binary variables y_{ak} for $a \in A_p$ and $k = 1, 2, \dots, r$ with the interpretation

$$y_{ak} = \begin{cases} 1 & \text{if arc } a \text{ is build while the max flow value is less than } f + k, \\ 0 & \text{otherwise.} \end{cases}$$

The number of time periods with maximum flow value f is $\sum_{a \in A_p} y_{a1}$, and for $k = 1, \dots, r-1$, the number

of time periods with maximum flow value $f+k$ is $\sum_{a\in A_p}(y_{a,k+1}-y_{ak})$. Consequently, the total flow is

$$f\sum_{a\in A_p} y_{a1}+\sum_{k=1}^{r-1}(f+k)\sum_{a\in A_p}(y_{a,k+1}-y_{ak})+F\left(T-\sum_{a\in A_p} y_{ar}\right)$$
$$=TF+\sum_{a\in A_p}\sum_{k=1}^{r} y_{ak}\left[(f+k-1)-(f+k)\right]=TF-\sum_{a\in A_p}\sum_{k=1}^{r} y_{ak}.$$

Hence the incremental maximum flow problem can also be formulated as follows

$$\min \sum_{a\in A_p}\sum_{k=1}^{r} y_{ak}$$

subject to

$$\sum_{a\in\delta^{\mathrm{out}}(v)} x_{ak}-\sum_{a\in\delta^{\mathrm{in}}(v)} x_{ak}=\begin{cases}0 & \text{for } v\notin\{s,t\}\\ f+k & \text{for } v=s\\ -f-k & \text{for } v=t\end{cases} \qquad \text{for } v\in N,\ k\in\{1,\dots,r\},$$
$$x_{ak}\leqslant u_a \qquad \text{for } a\in A_e,\ k\in\{1,\dots,r\},$$
$$x_{ak}\leqslant u_a y_{ak} \qquad \text{for } a\in A_p,\ k\in\{1,\dots,r\},$$
$$y_{ak}\leqslant y_{a,k+1} \qquad \text{for } a\in A_p,\ k\in\{1,\dots,r-1\},$$
$$x_{ak}\geqslant 0 \qquad \text{for } a\in A,\ k\in\{1,\dots,r\},$$
$$y_{ak}\in\{0,1\} \qquad \text{for } a\in A_p,\ k\in\{1,\dots,r\}.$$

We denote this formulation by IMFP^2.

Observe that the size of IMFP^1 strongly depends on the length of the planning horizon, whereas the size of IMFP^2 strongly depends on the difference between the initial and ultimate maximum flow values.

3 Heuristics

In this section, we introduce two natural strategies for trying to obtain high quality solutions: (1) getting a flow increment as quickly as possible, and (2) reaching a maximum possible flow as quickly as possible, as well as a hybrid strategy.

3.1 Quickest flow increment

A natural greedy strategy is to build the arcs such that a flow increment is always reached as quickly as possible. Suppose we have already built the arcs in $B\subseteq A_p$ to reach a maximum flow value $f+k$. A smallest set of potential arcs to be built, in addition to B, to reach a flow of value at least $f+k+1$ can be determined by solving a fixed charge network flow problem: find a flow of value $f+k+1$ where arcs in $A_e\cup B$ have zero cost, and arcs in $A_p\setminus B$ incur a cost of 1 if they carry a nonzero flow. More formally, in order to determine the smallest number of potential arcs that have to be built to increase the flow from

$f + k$ to at least $f + k + 1$, we solve the problem MinArcs$(B, k + 1)$:

$$\min\ z = \sum_{a \in A_p \setminus B} y_a$$

$$\text{subject to} \quad \sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{a \in \delta^{\text{in}}(v)} x_a = \begin{cases} 0 & \text{for } v \notin \{s, t\} \\ f + k + 1 & \text{for } v = s \\ -f - k - 1 & \text{for } v = t \end{cases} \qquad \text{for } v \in N,$$

$$x_a \leqslant u_a \qquad \text{for } a \in A_e \cup B,$$

$$x_a \leqslant u_a y_a \qquad \text{for } a \in A_p \setminus B,$$

$$x_a \geqslant 0 \qquad \text{for } a \in A,$$

$$y_a \in \{0, 1\} \qquad \text{for } a \in A_p \setminus B.$$

Next, among all the possibilities that increase the flow by building the smallest possible number of potential arcs, we want to choose one that maximizes the flow increase. Given the optimal value z^* for the problem MinArcs$(B, k + 1)$, this can be achieved by solving another MIP, denoted by MaxVal(B, z^*):

$$\max\ \xi$$

$$\text{subject to} \quad \sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{a \in \delta^{\text{in}}(v)} x_a = \begin{cases} 0 & \text{for } v \notin \{s, t\} \\ \xi & \text{for } v = s \\ -\xi & \text{for } v = t \end{cases} \qquad \text{for } v \in N,$$

$$x_a \leqslant u_a \qquad \text{for } a \in A_e \cup B, ,$$

$$x_a \leqslant u_a y_a \qquad \text{for } a \in A_p \setminus B,$$

$$\sum_{a \in A_p \setminus B} y_a = z^*$$

$$x_a \geqslant 0 \qquad \text{for } a \in A,$$

$$y_a \in \{0, 1\} \qquad \text{for } a \in A_p \setminus B.$$

The greedy heuristic is described formally in Algorithm 1 (assuming the entire set A_p is provided as input). In general, this heuristic still requires the solution of MIPs MinArcs$(B, k + 1)$ and MaxVal(B, z^*), hence

Algorithm 1: Quickest-increment

```
input : Ā_p ⊂ A_p
D ← (V, A) with A = A_e ∪ Ā_p ;
B ← ∅ { initialize the set of built arcs } ;
k ← 0 { index of initial flow value } ;
while k < F − f do
    z* ← optimal value of MinArcs(B, k + 1) ;
    (x, y) ← optimal solution for MaxVal(B, z*) ;
    { add potential arcs to be built to reach a flow of at least f + k + 1 } ;
    B ← B ∪ {a ∈ Ā_p : y_a = 1} ;
    k ← (maximum flow value using only arcs in A_e ∪ B) − f ;
```

we do not get a polynomial bound on the runtime. But these MIPs are much smaller than IMFP[1] and IMFP[2], so we might expect that they can be solved more efficiently.

However, if the capacities of all potential arcs are equal to 1, then the problems MinArcs$(B, k + 1)$ reduce to minimum cost flow problems, and their solutions are already optimal for MaxVal(B, z^*), because the optimal flow increment is 1. Thus for these instances, the quickest increment heuristic runs in polynomial time.

Furthermore, if the maximization of the flow increment is omitted, then the heuristic can be implemented to run in polynomial time even for general capacities. For this purpose, suppose we have already built the arcs in $B \subset A_p$ and we have a maximum flow $\boldsymbol{x}$ of value $f + k$ for the network $(V, A_e \cup B)$. Using the residual network with respect to $\boldsymbol{x}$, we can apply a labeling procedure which computes for each node v a triple $(d(v), \delta(v), p(v))$, where

- $d(v)$ is the minimum number of arcs in $A_p \setminus B$ on any augmenting s-v-path,
- $\delta(v)$ is the maximum augmentation for any s-v-path using at most $d(v)$ arcs from $A_p \setminus B$, and
- $p(v)$ is the predecessor of v on an s-v-path which contains $d(v)$ arcs from $A_p \setminus B$ and can be augmented by $\delta(v)$ units of flow.

This can be done in time $O(|A|)$, for instance using BFS starting from s, and the distance label $d(t)$ equals the minimum size of a set $B' \subseteq A_p \setminus B$ with the property that the maximum s-t-flow value in the network $(V, A_e \cup B \cup B')$ is strictly larger than $f + k$. Furthermore, $B' = P \cap (A_p \setminus B)$ is such a set of size $d(t)$, where P is the augmenting s-t-path corresponding to the result of the labeling procedure. Suppose there is a set $B'' \subseteq A_p \setminus B$ with $|B''| < d(t)$ such that the maximum flow in $(V, A_e \cup B \cup B'')$ is strictly larger than $f + k$. Then the residual network for $(V, A_e \cup B \cup B'')$ with respect to the flow $\boldsymbol{x}$ of value F_B must contain an augmenting path P'. Now $P' \cap (A_P \setminus B) \subseteq B''$, thus P' is an augmenting s-t-path which contains less than $d(t)$ arcs from $A_p \setminus B$, which contradicts the construction of $d(t)$. Using this labeling procedure, a quickest-increment heuristic can be described as follows.

- Initialize the set of built arcs $B = \emptyset$ and a zero flow $\boldsymbol{x} = \mathbf{0}$
- while $\boldsymbol{x}$ is not a maximum flow for (V, A) do
  - Find an augmenting path P using the smallest number of new potential arcs.
  - Add the potential arcs on P to B.
  - Update the flow $\boldsymbol{x}$.

This algorithm runs in time $O(|V||A|^3)$. The δ-component in our node labels ensures that we find the augmenting path that (1) requires the smallest number of new arcs to be built, and (2) allows the maximum augmentation among these paths. Nevertheless, it might be beneficial to build another path which allows more augmentations afterwards without building additional arcs. This is illustrated in Figure 1.

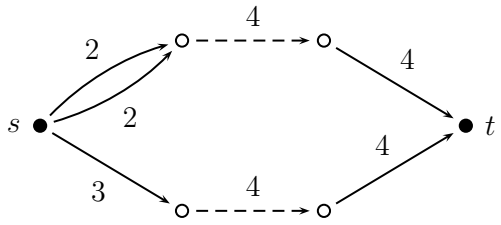

Figure 1: An example where the polynomial variant of *Quickest-increment* builds the "wrong" arc. Every augmenting path requires one potential arc. Our algorithm chooses the lower arc because it allows the augmentation of 3 units of flow. But it would be better to build the upper arc first, and Algorithm 1 does the right thing.

Since Algorithm 1, which solves subproblems MinArcs$(B, k+1)$ and MaxVal(B, z^*), when implemented using state-of-the-art commercial MIP solver, turned out to be sufficiently efficient for our test instances, we decided not to implement the polynomial variant for our computational study, which is presented in Section 5.

3.2 Quickest ultimate flow

A potential drawback of the quickest increment heuristic is that it might build arcs which yield (small) flow increments but that are not used in an ultimate maximum flow. To force the heuristic to build arcs that are used in an ultimate maximum flow, we can first determine a smallest set of potential arcs that admit an ultimate maximum flow by finding an optimal solution (x^*, y^*) to the problem MinArcs$(\emptyset, r)$, and then restricting the arc choices in Algorithm 1 to the arcs used in the optimal solution (x^*, y^*), i.e., by fixing $y_a = 0$ for all $a \in A_p$ with $y^*_a = 0$. This algorithm is described formally in Algorithm 2. Because the

Algorithm 2: Quickest-to-ultimate

input : r
$(x^*, y^*) \leftarrow$ optimal solution for MinArcs$(\emptyset, r)$;
$\overline{A}_p \leftarrow \{a : y^*_a = 1\}$;
$(\tilde{x}, \tilde{y}) \leftarrow$ solution obtained by *Quickest-increment* with input $\overline{A}_p$;

smallest set of potential arcs that admits an ultimate maximum flow is unlikely to be unique, it is possible that the initial choice of arcs may not be the best.

3.3 Quickest target flow

Combining the ideas of quickest flow increment and quickest ultimate flow, we can design a general framework. Let $0 = r_0 < r_1 < r_2 < \cdots < r_k = r$ be a sequence of target values. Put $B_0 = \emptyset$, and for $i \geqslant 1$ suppose that we have already determined a set $B_{i-1} \subseteq A_p$ such that the maximum flow value for $A_e \cup B_{i-1}$ is $f + r_{i-1}$. Following the quickest ultimate flow strategy, we can determine a smallest set of potential arcs B_i with $B_{i-1} \subseteq B_i \subseteq A_p$ such that $A_e \cup B_i$ allows a flow of value $f + r_i$, and then we can use the quickest flow increment strategy to specify the order in which the arcs in $B_i \setminus B_{i-1}$ are built. This is described in detail in Algorithm 3. Note that this framework implicitly also provides a risk mitigation strategy, because we

Algorithm 3: Quickest-to-target

input : $0 = r_0 < r_1 < r_2 < \cdots < r_k = r$
$B_0 \leftarrow \emptyset$;
$\overline{A}_p \leftarrow \emptyset$;
for $i = 1, \ldots, p$ **do**
  $z^* \leftarrow$ optimal objective value for MinArcs(B_{i-1}, r_i) ;
  $(x^*, y^*) \leftarrow$ optimal solution for MaxVal(B_{i-1}, z^*) ;
  $B_i \leftarrow \{a : y^*_a = 1\}$;
  $\overline{A}_p \leftarrow \overline{A}_p \cup B_i$;
  $(\tilde{x}, \tilde{y}) \leftarrow$ solution obtained by *Quickest-increment* with input $\overline{A}_p$;

are not locked in to a set of arcs for the entire planning period, as in *Quickest-to-ultimate*.

4 The unit capacity case

As mentioned in the introduction, the incremental network design problem with maximum flows is strongly NP-hard even when restricted to instances where every existing arc has capacity 1 and every potential arc has capacity 3. That leaves open the possibility that when all arcs have capacity 1, i.e., both existing and potential arcs, the problem is polynomially solvable. Unfortunately, we have been unable to settle the complexity status of the unit-capacity case, but we have been able to derive a number of results regarding the performance of *Quickest-increment* and *Quickest-to-ultimate* in the unit-capacity case, which are presented in this section.

Figures 2 and 3 illustrate that the heuristics can be off by a factor of close to 2 (*Quickest-to-ultimate*) and 3/2 (*Quickest-increment*). The dashed arcs represent paths of potential arcs whose lengths are indicated by

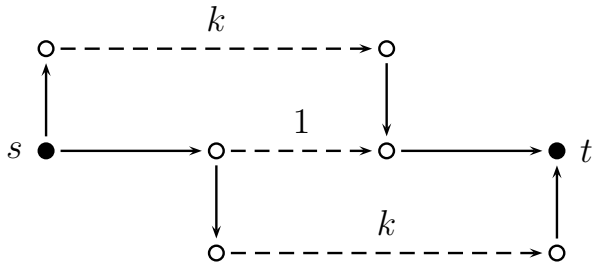

Figure 2: Bad instance for *Quickest-to-ultimate*.

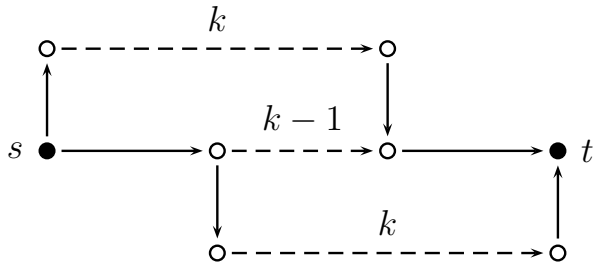

Figure 3: Bad instance for *Quickest-increment*.

the arc labels. For the instance in Figure 2, the time horizon is $T = 2k + 2$ and the optimal solution (which is found by *Quickest-increment*) has value $2k + 2$, while *Quickest-to-ultimate* provides a solution of value $k \cdot 0 + k \cdot 1 + 2 \cdot 2 = k + 4$. For the instance in Figure 3, the time horizon is $T = 3k$ and the optimal solution (which is found by *Quickest-to-ultimate*) has value $3k$, while *Quickest-increment* provides a solution of value $(k - 1) \cdot 0 + 2k \cdot 1 + 2 = 2k + 2$. Combining the two examples we get an instance that fools both heuristics (Figure 4). For this instance, *Quickest-to-ultimate* achieves a total flow of $10k + 4$, *Quickest-increment* gets $11k + 3$, but a combination of the two strategies yields a total flow of $13k$.

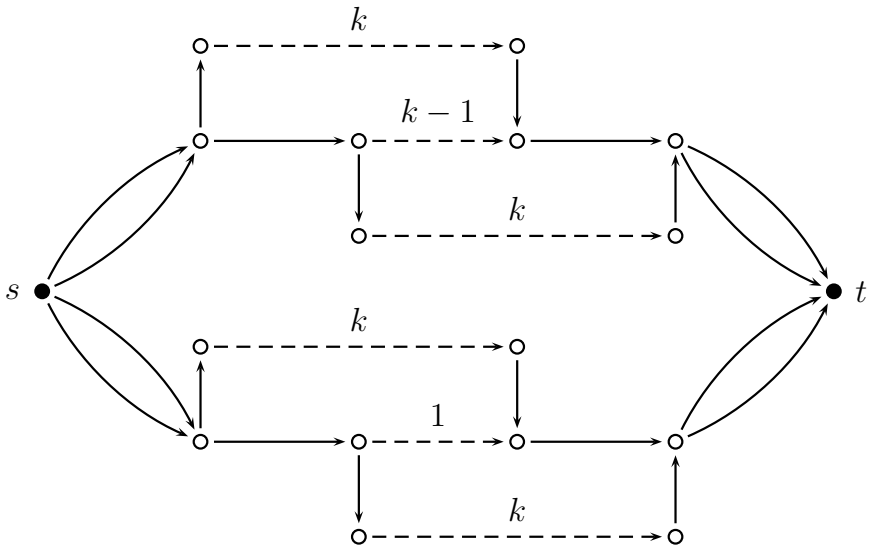

Figure 4: An instance where both *Quickest-to-ultimate* and *Quickest-increment* fail.

4.1 Upper bounds

In this subsection, we show that the instances in Figures 2 and 3 represent the worst situations for the heuristics *Quickest-to-ultimate* and *Quickest-increment*, respectively. Let z^*, z_1 and z_2 denote the optimal objective value, and the values obtained by *Quickest-to-ultimate* and *Quickest-increment*, respectively. For $i = 0, \ldots, r - 1$, let λ_i and μ_i denote the number of time periods with a maximum flow value $f + i$ in the solution from *Quickest-to-ultimate* and *Quickest-increment*, respectively. In other words, λ_i and μ_i are the optimal values of the problems MinArcs$(B, i + 1)$ which are solved for increasing the flow from $f + i$ to $f + i + 1$. Furthermore, let c_j for $0 \leqslant j \leqslant r$ be the minimum number of potential arcs that have to be built

in order to reach a flow of $f + j$:

$$c_j = \min\{ \sum_{a \in A_p} x_a \quad : \quad \sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{a \in \delta^{\text{in}} v)} x_a = \begin{cases} 0 & \text{for } v \in V \setminus \{s,t\}, \\ f+j & \text{for } v = s, \\ -f-j & \text{for } v = t, \end{cases}$$
$$0 \leqslant x_a \leqslant 1 \text{ for all } a \in A\} \tag{1}$$

$$= \max\{(f+j)(\pi_s - \pi_t) + \sum_{a \in A} y_a \quad : \quad \pi_v - \pi_w + y_a \leqslant \begin{cases} 1 & \text{for } a = (v,w) \in A_p, \\ 0 & \text{for } a = (v,w) \in A_e, \end{cases}$$
$$y_a \leqslant 0 \text{ for all } a \in A\}. \tag{2}$$

Observe that if the associated sets of arcs would be nested, then they would give rise to an optimal solution. However, since this is unlikely to be the case in general, the values c_j can only be used to derive an upper bound.

Lemma 1. $z^* \leqslant TF - \sum_{j=1}^{r} c_j \leqslant TF - \frac{1}{2} r(r-1) - c_r$.

Proof. For $i = 0, \ldots, r-1$, let λ_i^* be the number of time periods with flow $f + i$ in an optimal solution. Since any feasible solution must have at least c_{i+1} periods with flow value at most $f + i$, we have

$$\lambda_0^* + \cdots + \lambda_i^* \geqslant c_{i+1}.$$

Summing over i, we obtain

$$\sum_{i=0}^{r-1} \lambda_i^* (r-i) \geqslant \sum_{j=1}^{r} c_j,$$

hence

$$z^* = \sum_{i=0}^{r-1} \lambda_i^* (f+i) + \left(T - \sum_{i=0}^{r-1} \lambda_i^* \right) (f+r) = TF - \sum_{i=0}^{r-1} \lambda_i^* \left[(f+r) - (f+i) \right]$$
$$= TF - \sum_{i=0}^{r-1} \lambda_i^* (r-i) \leqslant TF - \sum_{j=1}^{r} c_j.$$

This proves the first inequality and the second one follows with $c_j \geqslant j$ for $1 \leqslant j \leqslant r$. □

The values z_1 and z_2 can be expressed in terms of the sequences (λ_i) and (μ_i), respectively.

Lemma 2. $z_1 = TF - \sum_{i=0}^{r-1} \lambda_i (r-i)$ *and* $z_2 = TF - \sum_{i=0}^{r-1} \mu_i (r-i)$.

Proof. We have

$$z_1 = \sum_{i=0}^{r-1} \lambda_i (f+i) + \left(T - \sum_{i=0}^{r-1} \lambda_i \right) (f+r) = TF - \sum_{i=0}^{r-1} \lambda_i (r-i),$$

and similarly for z_2. □

The sequences $(\lambda_i)_{i=0,\ldots,r-1}$ and $(\mu_i)_{i=0,\ldots,r-1}$ are non-decreasing.

Lemma 3. *We have* $\lambda_0 \leqslant \lambda_1 \leqslant \cdots \leqslant \lambda_{r-1}$ *and* $\mu_0 \leqslant \mu_1 \leqslant \cdots \leqslant \mu_{r-1}$.

Proof. Let $\overline{A} = A_e \cup \overline{A}_p$ be the set of arcs that are used by *Quickest-to-ultimate*. Set $B_0 = \emptyset$, and for $i = 1, \dots, r-1$ let B_i be the set of arcs which are built by *Quickest-to-ultimate* in the first $\lambda_0 + \cdots + \lambda_{i-1}$ time periods. Then

$$\begin{aligned}\lambda_i = \min\{ \sum_{a \in \overline{A}_p \setminus B_i} x_a \quad : \quad \sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{a \in \delta^{\text{in}}(v)} x_a = \begin{cases} 0 & \text{for } v \in V \setminus \{s,t\}, \\ f+i+1 & \text{for } v = s, \\ -f-i-1 & \text{for } v = t, \end{cases} \\ 0 \leqslant x_a \leqslant 1 \text{ for all } a \in \overline{A}\} \\ = \max\{(f+i+1)(\pi_s - \pi_t) + \sum_{a \in \overline{A}} y_a \ : \ \pi_v - \pi_w + y_a \leqslant \begin{cases} 1 & \text{for } a = (v,w) \in \overline{A}_p \setminus B_i, \\ 0 & \text{for } a = (v,w) \in A_e \cup B_i, \end{cases} \\ y_a \leqslant 0 \text{ for all } a \in \overline{A}\}.\end{aligned}$$

Fix some $i < r-1$, and let (π^*, y^*) be an optimal solution for the dual characterization of λ_i. Since $A_e \cup B_i$ can carry an s-t-flow of value $f+i$, the optimal value for the optimization problem which characterizes λ_i becomes 0 when the flow value $f+i+1$ is replaced by $f+i$. In the dual problem this is corresponds to the coefficient of $(\pi_s - \pi_t)$ in the objective function, and since (π^*, y^*) is also feasible for this modified dual problem, we have

$$(f+i)(\pi_s^* - \pi_t^*) + \sum_{a \in \overline{A}} y_a \leqslant 0,$$

and therefore

$$\lambda_i = (f+i+1)(\pi_s^* - \pi_t^*) + \sum_{a \in \overline{A}} y_a^* \leqslant \pi_s^* - \pi_t^*.$$

On the other hand, we obtain a feasible solution (π', y') for the dual characterization of λ_{i+1} by setting $\pi' = \pi^*$ and

$$y'_a = \begin{cases} y_a^* - 1 & \text{for } a \in B_{i+1} \setminus B_i, \\ y_a^* & \text{for } a \in \overline{A} \setminus (B_{i+1} \setminus B_i), \end{cases}$$

and together with $|B_{i+1} \setminus B_i| = \lambda_i$ this implies

$$\lambda_{i+1} \geqslant (f+i+2)(\pi_s^* - \pi_t^*) + \sum_{a \in \overline{A}} y_a^* - \lambda_i = \pi_s^* - \pi_t^*.$$

The inequality $\mu_i \leqslant \mu_{i+1}$ is proved in the same way (with A instead of $\overline{A}$). □

Theorem 1. *Quickest-to-ultimate is a 2-approximation algorithm for the incremental maximum flow problem with unit capacities, i.e., $z^* \leqslant 2z_1$.*

Proof. Using Lemma 3, we have

$$\sum_{i=0}^{r-1} \lambda_i (r-1-2i) = \sum_{i=0}^{\lfloor (r-1)/2 \rfloor} (\lambda_i - \lambda_{r-1-i})(r-1-2i) \leqslant 0.$$

Adding the inequality $(\lambda_0 + \cdots + \lambda_{r-1})r \leqslant TF$, it follows that

$$TF - \sum_{i=0}^{r-1} \lambda_i (2(r-i)-1) \geqslant 0.$$

By construction, $c_r = \lambda_0 + \cdots + \lambda_{r-1}$, and with Lemma 1 we derive

$$\begin{aligned} z^* \leqslant TF - \frac{1}{2}r(r-1) - \sum_{i=0}^{r-1} \lambda_i \leqslant TF - \frac{1}{2}r(r-1) - \sum_{i=0}^{r-1} \lambda_i + \left(TF - \sum_{i=0}^{r-1} \lambda_i (2(r-i)-1) \right) \\ = 2\left(TF - \sum_{i=0}^{r-1} \lambda_i (r-i) \right) - \frac{1}{2}r(r-1) = 2z_1 - \frac{1}{2}r(r-1). \quad \square \end{aligned}$$

In order to prove that for *Quickest-increment* the case in Figure 3 illustrates an upper bound for the approximation ratio, we need some technical preparations. We start by bounding the numbers μ_i by the numbers c_j.

Lemma 4. *We have $\mu_i \leqslant c_j/(j-i)$ for all (i,j) with $0 \leqslant i < j \leqslant r$.*

Proof. Fix i and j with $0 \leqslant i < j \leqslant r$, and let X be the feasible region of the maximization problem which characterizes μ_i, i.e.,

$$X = \Big\{(y,\pi) \in \mathbb{R}^{|A|} \times \mathbb{R}^{|V|} \ : \ \pi_v - \pi_w + y_a \leqslant \begin{cases} 0 & \text{for } a = (v,w) \in A_e \cup B_i, \\ 1 & \text{for } a = (v,w) \in A_p \setminus B_i, \end{cases} \\ y_a \leqslant 0 \text{ for all } a \in A\Big\},$$

where B_i is the set of arcs built by *Quickest-increment* in the first $\mu_0 + \cdots + \mu_{i-1}$ time periods. Since the maximum flow value on $A_e \cup B_i$ equals $f + i$, we have that $X \subseteq Y$ where

$$Y = \Big\{(y,\pi) \in \mathbb{R}^{|A|} \times \mathbb{R}^{|V|} \ : \ \pi_v - \pi_w + y_a \leqslant \begin{cases} 0 & \text{for } a = (v,w) \in A_e, \\ 1 & \text{for } a = (v,w) \in A_p, \end{cases} \\ (f+i)(\pi_s - \pi_t) + \sum_{a \in A} y_a \leqslant 0, \ y_a \leqslant 0 \text{ for all } a \in A\Big\},$$

Consequently,

$$\mu_i \leqslant \max\Big\{(f+i+1)(\pi_s - \pi_t) + \sum_{a \in A} y_a \ : \ (y,\pi) \in Y\Big\},$$

and by taking the dual we obtain

$$\mu_i \leqslant \min\Big\{ \sum_{a \in A_p} x_a \ : \ \sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{a \in \delta^{\text{in}} v)} x_a = \begin{cases} 0 & \text{for } v \in V \setminus \{s,t\}, \\ f+i+1-(f+i)z & \text{for } v = s, \\ -f-i-1+(f+i)z & \text{for } v = t, \end{cases} \\ x_a + z \leqslant 1 \text{ for all } a \in A, \quad x_a \geqslant 0 \text{ for all } a \in A, \ z \geqslant 0\Big\}. \quad (3)$$

Let $x \in \mathbb{R}^{|A|}$ be an optimal solution for the problem (1) to determine c_j, i.e., an s-t-flow of value $f + j$. Solving the system

$$(f+j)\gamma = f+i+1-(f+i)z, \qquad \gamma + z = 1$$

for γ and z yields $\gamma = 1/(j-i)$ and $z = (j-i-1)/(j-i)$, and this implies that a feasible solution (x', z) for problem (3) can be obtained by setting $x'_a = x_a/(j-i)$ for all $a \in A$ and $z = (j-i-1)/(j-i)$. Hence the optimal value of problem (3) is at most $c_j/(j-i)$ and this concludes the proof. □

From Lemmas 3 and 4 it follows that $(\mu, c, T) \in \mathbb{R}^r \times \mathbb{R}^{r+1} \times \mathbb{R}$ satisfies the following conditions:

$$\mu_i - \mu_{i+1} \leqslant 0 \qquad \text{for } 0 \leqslant i \leqslant r-2, \quad (4)$$

$$(j-i)\mu_i - c_j \leqslant 0 \qquad \text{for } 0 \leqslant i < j \leqslant r, \quad (5)$$

$$\sum_{i=0}^{r-1} \mu_i - T \leqslant 0, \quad (6)$$

$$c_0 = 0, \quad (7)$$

$$\mu_i,\ c_j \geqslant 0 \qquad \text{for } 0 \leqslant i \leqslant r-1,\ 1 \leqslant j \leqslant r. \quad (8)$$

Let $X(r) \subseteq \mathbb{R}^r \times \mathbb{R}^{r+1} \times \mathbb{R}$ denote the set of all (μ, c, T) satisfying (4) to (8). Using Lemmas 1 and 2, a number $\gamma > 1$ is an upper bound for the approximation ratio of *Quickest-increment*, provided

$$TF - \sum_{j=1}^{r} c_j \leqslant \gamma \left[TF - \sum_{i=0}^{r-1} \mu_i (r-i) \right]$$

is a valid inequality for $X(r)$, or equivalently

$$F(1-\gamma)T - \sum_{j=1}^{r} c_j + \gamma \sum_{i=0}^{r-1} \mu_i (r-i) \leqslant 0$$

for all $(\mu, c, T) \in X(r)$. Since $F \geqslant r$ and $\gamma > 1$, this inequality is strengthened when F is replaced by r on the left hand side. By duality, the stronger inequality is valid for $X(r)$, provided the following system has a solution:

$$\sum_{i=0}^{j-1} y_{ij} \leqslant 1 \qquad \text{for } 1 \leqslant j \leqslant r, \tag{9}$$

$$z + \sum_{j=i+1}^{r} (j-i) y_{ij} + x_i - x_{i-1} \geqslant \gamma(r-i) \qquad \text{for } 0 \leqslant i \leqslant r-1, \tag{10}$$

$$z \leqslant r(\gamma - 1), \tag{11}$$

$$x_{-1} = x_{r-1} = 0, \tag{12}$$

$$z, x_i \geqslant 0 \qquad \text{for } 0 \leqslant i \leqslant r-2, \tag{13}$$

$$y_{ij} \geqslant 0 \qquad \text{for } 0 \leqslant i < j \leqslant r. \tag{14}$$

Let $Y(r, \gamma) \subseteq \mathbb{R}^{r+1} \times \mathbb{R}^{r(r+1)/2} \times \mathbb{R}$ be the set of all (x, y, z) satisfying (9) to (14). The above argument implies the following theorem which provides a sufficient condition for γ to be an approximation factor for *Quickest-increment* on all instances with $F - f = r$.

Theorem 2. *If $Y(r, \gamma) \neq \emptyset$, then Quickest-increment is a γ-approximation algorithm for all instances of the incremental maximum flow problem with unit capacities and $F - f = r$.* □

Consequently, in order to establish that *Quickest-increment* is a 3/2-approximation algorithm, it is sufficient to prove the $Y(r, 3/2) \neq \emptyset$ for all $r \geqslant 2$. To do this, we need another lemma.

Lemma 5. *Let $r \geqslant 2$ and $s = \lfloor (r+2)/3 \rfloor$. Then the following system has a solution with non-negative y_{ij} for $0 \leqslant i \leqslant s$, $i+1 \leqslant j \leqslant r$:*

$$\sum_{j=i+1}^{r} (j-i) y_{ij} \geqslant r - \frac{3}{2} i \qquad 0 \leqslant i \leqslant s,$$

$$\sum_{i=0}^{\min\{j-1, s\}} y_{ij} \leqslant 1 \qquad 1 \leqslant j \leqslant r.$$

Proof. We proceed by induction on r. For $r = 2$, $s = 1$, a solution is given by $y_{01} = 1$, $y_{02} = y_{12} = 1/2$. So we assume $r > 2$, and we distinguish two cases.

Case 1. $r \equiv 0$ or $2 \pmod 3$. By induction, there are nonnegative numbers y_{ij} for $0 \leqslant i \leqslant s$ and $i+1 \leqslant j \leqslant r-1$ satisfying

$$\sum_{j=i+1}^{r-1} (j-i) y_{ij} \geqslant r - 1 - \frac{3}{2} i \qquad 0 \leqslant i \leqslant s,$$

$$\sum_{i=0}^{\min\{j-1, s\}} y_{ij} \leqslant 1 \qquad 1 \leqslant j \leqslant r-1.$$

We extend this by $y_{ir} = 1/(r-i)$ for $0 \leqslant i \leqslant s$. Then $\sum_{j=i+1}^{r}(j-i)y_{ij} \geqslant r - \frac{3}{2}i$ for $0 \leqslant i \leqslant s$ and

$$\sum_{i=0}^{s} y_{ir} = \sum_{i=0}^{s} \frac{1}{r-i} \leqslant \frac{s+1}{r-s} \leqslant 1.$$

Case 2. $r \equiv 1 \pmod 3$ and $s = (r+2)/3$. For $r = 4$, a solution is given by

$$\begin{aligned} &y_{01} = 1, && y_{02} = 1, && y_{03} = 1/3, && y_{04} = 0, \\ &&& y_{12} = 0, && y_{13} = 2/3, && y_{14} = 7/18, \\ &&&&& y_{23} = 0, && y_{24} = 1/2. \end{aligned}$$

For $r > 4$, by induction, there are nonnegative numbers y_{ij} for $0 \leqslant i \leqslant s-1$ and $i+1 \leqslant j \leqslant r-3$ satisfying

$$\begin{aligned} \sum_{j=i+1}^{r-3} (j-i)y_{ij} &\geqslant r - 3 - \frac{3}{2}i && 0 \leqslant i \leqslant s-1, \\ \sum_{i=0}^{\min\{j-1,s-1\}} y_{ij} &\leqslant 1 && 1 \leqslant j \leqslant r-3. \end{aligned}$$

Let $i_0 = \lfloor (r-5)/4 \rfloor$. We extend our solution by

$$\begin{aligned} y_{ij} &= \begin{cases} \frac{3}{(r-2)-i} & \text{for } j = r-2, \\ 0 & \text{for } j \in \{r-1, r\} \end{cases} && \text{for } 0 \leqslant i \leqslant i_0, \\ y_{ij} &= \begin{cases} \frac{3}{(r-1)-i} & \text{for } j = r-1, \\ 0 & \text{for } j \in \{r-2, r\} \end{cases} && \text{for } i_0 + 1 \leqslant i \leqslant s-1, \\ y_{sj} &= \begin{cases} \frac{3(r-2)}{4(r-1)} & \text{for } j = r, \\ 0 & \text{for } j < r. \end{cases} \end{aligned}$$

Then $\sum_{j=i+1}^{r}(j-i)y_{ij} \geqslant r - \frac{3}{2}i$ for $0 \leqslant i \leqslant s$,

$$\sum_{i=0}^{s} y_{i,r-2} = \sum_{i=0}^{i_0} \frac{3}{(r-2)-i} \leqslant \frac{3(i_0+1)}{r-2-i_0} \leqslant \frac{3(r-1)/4}{r-2-(r-5)/4} = 1,$$

$$\begin{aligned} \sum_{i=0}^{s} y_{i,r-1} = \sum_{i=i_0+1}^{s-1} \frac{3}{(r-1)-i} \leqslant \frac{3(s-1-i_0)}{r-1-(s-1)} &\leqslant \frac{3\left(\frac{r+2}{3} - 1 - \frac{r-5}{4}\right)}{r - \frac{r+2}{3}} \\ &= \frac{\frac{1}{4}(r+11)}{\frac{1}{3}(2r-2)} = \frac{3r+33}{8r-8} \leqslant 1, \end{aligned}$$

where we used $r \geqslant 5$ for the last inequality, and $\sum_{i=0}^{s} y_{ir} = y_{sr} \leqslant 1$. □

Using Lemma 5, we can exhibit a point in $Y(r, 3/2)$.

Lemma 6. *$Y(r, 3/2) \neq \emptyset$ for all $r \geqslant 2$.*

Proof. Let $s = \lfloor (r+2)/3 \rfloor$ and let y_{ij} for $0 \leqslant i \leqslant s$ and $i+1 \leqslant j \leqslant r$ be a point as described in Lemma 5. For $i > s$ and $i+1 \leqslant j \leqslant r$, let $y_{ij} = 0$. In addition, let $x_i = 0$ for $0 \leqslant i \leqslant s$, and $z = r/2$. Finally, let

$$x_i = (i-s)\left(r - \frac{3}{2}s - \frac{3}{4}(i-s+1)\right) \qquad \text{for } s+1 \leqslant i \leqslant r-2.$$

We claim that $(x, y, z) \in Y(r, 3/2)$. Constraints (9) follow immediately Lemma 5. The same is true for constraints (10) for $0 \leqslant i \leqslant s$. For $s < i \leqslant r-2$, constraints (10) are satisfied because

$$x_i - x_{i-1} = (i-s)\left(r - \frac{3}{2}s - \frac{3}{4}(i-s+1)\right) - (i-1-s)\left(r - \frac{3}{2}s - \frac{3}{4}(i-s)\right) = r - \frac{3}{2}i.$$

Finally, constraint (10) for $i = r-1$ follows from

$$x_{r-2} = (r-2-s)\left(r - \frac{3}{2}s - \frac{3}{4}(r-1-s)\right) = \begin{cases} r/2 - 3/2 & \text{if } r \equiv 0 \pmod 3, \\ r/6 - 2/3 & \text{if } r \equiv 1 \pmod 3, \\ r/3 - 7/6 & \text{if } r \equiv 2 \pmod 3, \end{cases}$$

which also implies $x_i \geqslant 0$ for all i. □

The bound of 3/2 for the approximation ratio of *Quickest-increment* is a consequence of Theorem 2 and Lemma 6.

Theorem 3. *Quickest-increment is a* 3/2*-approximation algorithm for the incremental maximum flow problem with unit capacities.*

Let us examine the example in Figure 3 more closely. It has initial flow value $f = 0$ and ultimate flow value $F = 2$ and the essence of the example is that the arcs that are built to reach a flow value 1 cannot be used for the (ultimate) flow value 2. Figure 5 shows an attempt to generalize the example to one with initial flow value $f = 0$ and ultimate flow value $F = 3$. The instance has a time horizon $T = 7k + 3$. *Quickest-*

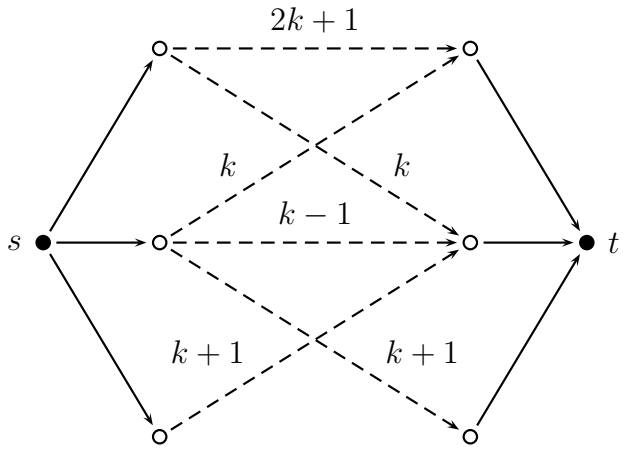

Figure 5: An instance with $r = 3$.

increment starts with building the middle path with $k-1$ potential arcs. Then the cheapest way to achieve a flow of value 2 is to build the two paths containing k potential arcs each. Finally, all the remaining arcs have to be built in order to achieve a flow of value 3. This gives a total flow of

$$(k-1) \cdot 0 + 2k \cdot 1 + (4k+3) \cdot 2 + 3 = 10k + 9.$$

On the other hand, the optimum for large k is to build the ultimate flow immediately which yields

$$(k+1) \cdot 0 + (k+1) \cdot 1 + (2k+1) \cdot 2 + 3k \cdot 3 = 14k + 3.$$

What works in favor of the heuristic, in terms of performance, is that in order for the heuristic not to build the upper path to achieve a flow of value 2, it has to be long enough, i.e. longer than the sum of the lengths of the diagonal paths that the heuristic chooses.

If we define

$$\varphi(r) = \inf\{\gamma \ : \ \textit{Quickest-increment} \text{ is a } \gamma\text{-approximation for instances with } F - f = r\},$$

then Theorem 3 shows that $\varphi(r) \leqslant 3/2$. It is possible to show that $\varphi(r) \leqslant 4/3$ for $r \geqslant 70$, and we conjecture that it is possible to show that $\lim_{r\to\infty} \varphi(r) = 1$.

4.2 The incremental maximum matching problem

In our quest for polynomially solvable cases of incremental network design with maximum flows, we next consider, in addition to unit capacities on all arcs, imposing a restriction on the structure of the network. More specifically, a node set $N = V \cup W \cup \{s, t\}$ and an arc set A consisting of existing arcs (s, v) for all $v \in V$, existing arcs (w, t) for all $w \in W$, and some arcs (v, w) with $v \in V$ and $w \in W$, which can be either existing or potential. Thus, s-t flows correspond to matchings in the bipartite graph $(V \cup W, \{\{v, w\} : (v, w) \in A\})$, and therefore we call this special case the *incremental maximum matching problem*.

The example in Figure 6 shows that *Quickest-to-ultimate* may fail to find an optimal solution. The unique maximum cardinality matching is $\{(1, 2), (3, 4), \ldots, (15, 16)\}$ and building arcs (1,2), (3,4), (5,6), and (7,8), followed by arcs (9,10), (11,12), (13,14), and (15,16), followed by (9,4) results in a total cumulative flow for *Quickest-to-ultimate* of $4 \cdot 6 + 4 \cdot 7 + 2 \cdot 8 = 68$, whereas building arcs (1,2) and (9,4), followed by arcs $(3, 4), (5, 6), \ldots, (15, 16)$ results in a total cumulative flow of $2 \cdot 6 + 7 \cdot 7 + 8 = 69$.

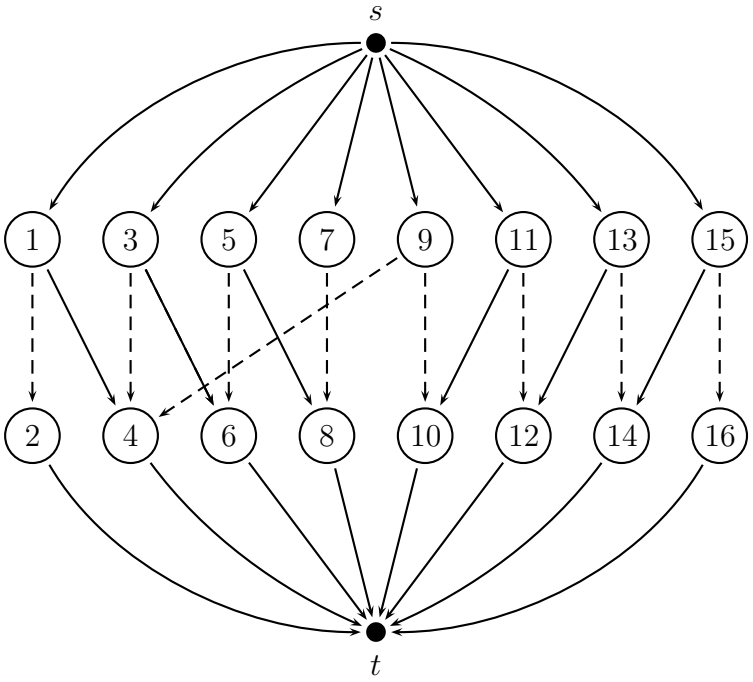

Figure 6: Bad instance for *Quickest-to-ultimate*.

Similarly, the example in Figure 7 shows that *Quickest-increment* may fail to find an optimal solution. The initial maximum matching has size 5, and the unique cheapest way to obtain a matching of size 6 is to build arcs $(5, 8)$ and $(7, 10)$. After that, all the remaining potential arcs need to be build to reach a matching of size 7, and the total cumulative flow for *Quickest-increment* is $2 \cdot 5 + 6 \cdot 6 + 7 = 53$. On the other hand, building arcs $(1, 2), \ldots, (13, 14)$ followed by arcs (5,8) and (7,10) results in a total cumulative flow of $3 \cdot 5 + 3 \cdot 6 + 3 \cdot 7 = 54$.

Thus, even restricting the structure of the network to bipartite graphs does not trivially give a polynomially solvable case of incremental network design with maximum flows. We have been able to improve the performance guarantee of *Quickest-to-ultimate* to 4/3 for instances of the incremental maximum matching problem. The proof is given in the appendix.

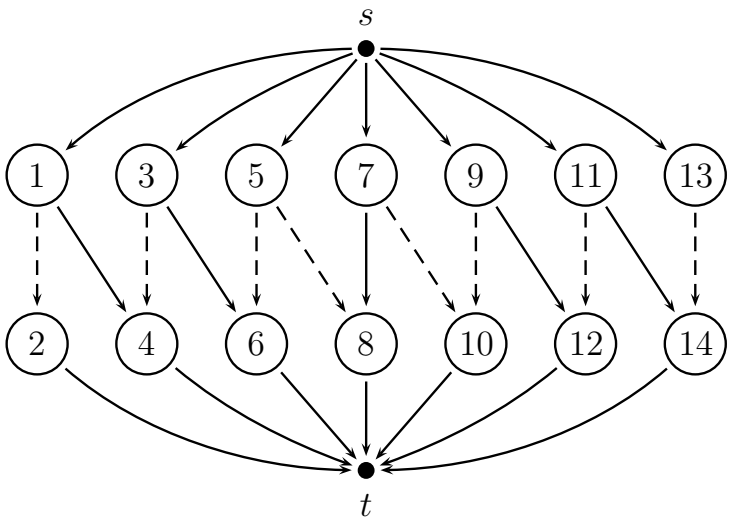

Figure 7: Bad instance for *Quickest-increment*.

5 Computational study

We have conducted a computational study to gain insight into the characteristics that make instances of the incremental maximum flow problem hard to solve, to compare the performance of the two MIP formulations, and to assess the performance of the three heuristics.

The following computational tools were used to develop and analyze the formulations and algorithms: Python 2.7.5, Matplotlib 1.3.1, NetworkX 1.8.1, and Gurobi 5.5. All computational experiments were conducted on a 64-bit Win7 with Intel Xeon CPU (E5-1620) with 8 cores and 32GB of RAM using a single thread.

5.1 Instance generation

For the computational experiments in which we compare the performance of the IP formulations and the heuristics, we use two classes of instances: one using general graphs and one using layered graphs.

General graphs are parameterized by the number of nodes n, the expected density $d = 2|A|/n(n-1) \in \{0.1, 0.3, 0.7\}$, the expected fraction of potential arcs $p = |A_p|/|A| \in \{0.3, 0.7\}$, and the maximum arc capacity $u_{\max} \in \{1, 3, 10\}$. For each combination of parameters, 10 random instances are generated. More specifically, for a given number of nodes n, an arc between two nodes exists with probability d, and if an arc between two nodes exists, the arc is a potential arc with probability p (and thus an existing arc with probability $1 - p$).

Layered graphs consist of a source node, a sink node, and ℓ layers in between ($\ell \geqslant 2$). The source is connected to every node in the first layer and every node in the last layer is connected to the sink. The nodes in layer i may be connected to a node in layer $i+1$. Layered graphs are parameterized by the number of layers ℓ, the number of nodes in each layer n, the expected density $d = |A|/(\ell-1)n^2 \in \{0.1, 0.3, 0.7\}$, the expected fraction of potential arcs $p = |A_p|/|A| \in \{0.3, 0.7\}$, and the maximum arc capacity $u_{\max} \in \{1, 3, 10\}$. For each combination of parameters, 10 random instances are generated. The generation proceeds similar to the generation of general instances. Layered instances are considered because they ensure a minimum distance between the source and the sink.

5.2 Comparison of the MIP models

We start by determining the characteristics that make an instance difficult to solve and, at the same time, comparing the performance of the two MIP formulations presented in Section 2.

Tables 1 and 2 present average performance statistics for various combinations of the instance parameters (with $n = 35$ for general graphs and $\ell = 5$ and $n = 10$ for layered graphs). More specifically, we report the arc

density (d), the fraction of potential arcs (p), the maximum arc capacity ($u_{\max}$), the difference between the value of an ultimate and an initial maximum flow ($F - f$), the number of instances not solved to optimality within the time limit (#), the average initial gap (initial gap), computed as $(z^{\text{LP}} - z^{\text{IP}})/z^{\text{IP}}$, where z^{LP} the value of the LP relaxation and z^{IP} is the value of the best solution found, the average final gap over the instances not solved to optimality (final gap), where in the gap computation z^{LP} is replaced by the best bound at termination, and the average solution time (time) and average number of nodes in the search tree (#nodes) over the instances solved to optimality.

Table 1: Comparison of MIP models for general graphs with $n = 35$; time limit 1 hour

				IMFP^1					IMFP^2				
d	p	$u_{\max}$	$F-f$	#	initial gap	final gap	time	#nodes	#	initial gap	final gap	time	#nodes
0.1	0.3	1	0.9		0.12%		0.13	1.5		0.00%		0.01	0.0
0.1	0.3	3	1.5		0.28%		0.15	0.5		0.43%		0.00	0.0
0.1	0.3	10	7.3		0.36%		0.22	1.5		0.94%		0.05	4.5
0.1	0.7	1	2.1		0.55%		23.88	309.1		0.00%		0.00	0.0
0.1	0.7	3	4.0		1.17%		7.33	325.9		1.11%		0.03	0.0
0.1	0.7	10	11.6		1.46%		73.66	1,694.7		1.83%		1.73	107.1
0.3	0.3	1	1.9		0.02%		1.56	75.5		0.00%		0.01	0.0
0.3	0.3	3	7.7		0.07%		3.01	66.0		0.13%		0.05	0.0
0.3	0.3	10	11.7		0.03%		1.81	7.0		0.13%		0.09	0.0
0.3	0.7	1	6.4		0.13%		288.94	1,461.1		0.00%		0.04	0.0
0.3	0.7	3	12.8	1	0.14%	0.02%	594.46	3,529.1		0.17%		0.46	0.7
0.3	0.7	10	37.0	1	0.14%	0.10%	607.91	925.4	1	0.28%	0.04%	9.91	73.6

Table 2: Comparison of MIP models for layered graphs with $\ell = 5$ and $n = 10$; time limit 1 hour.

				IMFP^1					IMFP^2				
d	p	$u_{\max}$	$F-f$	#	initial gap	final gap	time	#nodes	#	initial gap	final gap	time	#nodes
0.1	0.3	1	1.6		2.13%		0.02	6.3		0.00%		0.00	0.0
0.1	0.3	3	1.3		1.67%		0.01	0.0		0.99%		0.00	0.0
0.1	0.3	10	6.6		4.76%		0.03	8.1		6.46%		0.01	0.0
0.1	0.7	1	3.1		2.84%		1.35	502.9		0.00%		0.00	0.0
0.1	0.7	3	4.0		7.20%		3.24	905.5		6.79%		0.01	0.0
0.1	0.7	10	5.5		5.69%		0.16	119.8		6.77%		0.01	0.0
0.3	0.3	1	1.7		0.09%		0.36	3.3		0.00%		0.00	0.0
0.3	0.3	3	6.5		0.86%		11.23	1603.6		1.00%		0.08	18.7
0.3	0.3	10	15.5		0.80%		3.76	418.4		1.29%		3.36	394.5
0.3	0.7	1	8.9	7	0.72%	0.35%	1473.03	77596.0		0.00%		0.03	0.0
0.3	0.7	3	14.8	8	3.43%	0.52%	1398.13	67872.5	1	3.30%	0.24%	42.99	1574.8
0.3	0.7	10	32.0	9	4.64%	0.66%	1225.42	17924.0	9	5.07%	0.65%	1,705.86	15948.0

First and foremost, we observe that IMFP^2 performs significantly better than IMFP^1; not only are more instances solved to optimality within the time limit of 1 hour, but the instances are solved much faster. (Interestingly, the initial gap for IMFP^2 tends to be larger than the initial gap for IMFP^1 when the maximum arc capacity is greater than 1.)

As expected, the instances get more difficult when the density, and thus the number of arcs, increases, when the fraction of potential arcs, and thus the planning horizon, increases, and when the maximum arc capacity increases.

IMFP^1 is impacted primarily by the length of the planning horizon, i.e., the number of possible build

sequences, whereas IMFP^2 is impacted primarily by the difference between the value of the ultimate flow F and the value of the initial flow f, i.e., the flow increase that has to be accommodated.

The density d relates to the number of arcs in the graph and thus the number of paths from source to sink and thus the maximum flow. Therefore, if the density increases, then the length of the planning horizon and the difference between F and f increase, which impacts the performance of both formulations. The fraction of potential arcs p relates to the planning horizon and thus the number of possible sequences in which arcs can be build, but also impacts f and thus the difference between F and f. Therefore, if this fraction increases, then the length of the planning horizon and the difference between F and f increase, which should impact the performance of both formulations, but most likely the performance of IMFP^1. The maximum arc capacity $u_{\max}$ impacts the difference between F and f. Therefore, if the arc capacity increases, then the difference between F and f increases, which impacts the performance of IMFP^2, but also IMFP^1.

To further investigate the behavior of the two formulations, Tables 3 and 4 present detailed performance statistics for 10 instances in the most difficult class, i.e., $d = 0.3$, $p = 0.7$, and $u_{\max} = 10$, when the time limit is increased to 4 hours. The instances are presented in nondecreasing order of the difference between the value of the ultimate flow and the value of the initial flow, i.e., $F - f$.

Table 3: Comparison of MIP models for general graphs with $n = 35$, $d = 0.3$, $p = 0.7$, $u_{\max} = 10$; time limit 4 hours.

	IMFP^1				IMFP^2			
$F - f$	initial gap	final gap	time	# nodes	initial gap	final gap	time	# nodes
15	0.08%		48.08	34	0.16%		0.25	0
25	0.07%		38.20	152	0.16%		0.50	0
26	0.08%		823.75	980	0.17%		1.72	0
38	0.24%		3,125.46	5,437	0.34%		11.13	254
39	0.20%		83.41	520	0.36%		1.39	0
39	0.12%		69.91	306	0.27%		1.28	0
45	0.26%	0.04%	14,400.00	40,804	0.35%		365.06	865
48	0.26%		5,989.53	23,078	0.39%		40.11	283
55	0.32%		5,134.81	24,585	0.45%		6.83	0
71	0.25%		4,333.61	30,892	0.41%		63.86	135

Table 4: Comparison of MIP models for layered graphs with $\ell = 5$, $n = 10$, $d = 0.3$, $p = 0.7$, $u_{\max} = 10$; time limit 4 hours.

	IMFP^1				IMFP^2			
$F - f$	initial gap	final gap	time	# nodes	initial gap	final gap	time	# nodes
24	2.84%		3,468.53	26,927	3.23%		625.37	21,858
26	3.82%	0.07%	14,400.00	1,041,170	4.27%		1,450.65	102,528
30	5.30%	0.19%	14,400.00	210,589	5.64%		2,602.19	99,424
34	2.85%	0.12%	14,400.00	260,528	3.26%	0.10%	14,400.00	212,306
35	6.09%	0.41%	14,400.00	354,786	6.48%	0.26%	14,400.00	551,579
38	4.21%	0.10%	14,400.00	281,519	4.67%		12,281.30	196,572
39	2.40%		10,759.84	300,278	2.83%	0.22%	14,400.00	202,730
39	5.24%	0.44%	14,400.00	130,664	5.59%	0.24%	14,400.00	312,643
44	3.91%	0.52%	14,400.00	37,954	4.29%	0.42%	14,400.00	72,215
45	3.09%	0.32%	14,400.00	52,449	3.64%	0.39%	14,400.00	97,568

We see that the different between the value of the ultimate flow and the value of the initial flow, i.e., $F - f$, is a good predictor of the difficulty of an instance for both formulations. The larger the value of $F - f$, the more likely it is that the instance cannot be solved and if an instance cannot be solved, the more likely it is that the final gap is large. These "trends" are most clearly seen in the results for the layered

instances. For the layered instances, it is also interesting to note that there is one instance that can be solved by IMFP^1, but not by IMFP^2.

Additional analysis and experimentation revealed that in addition to the difference $F - f$, the number of flow increases from f to F also seems to impact solution time, with more flow increases usually resulting in longer solution times. The number of flow increases from f to F depends on the arc capacities in an instance, and on how these arc capacities interact on the different paths from source to sink.

5.3 Heuristics

For the difficult instances, we compare the performance of *Quickest-increment*, *Quickest-to-ultimate*, and *Quickest-to-target* with $p = 2$, $r_0 = f, r_1 = \lfloor (F - f)/2 \rfloor$, and $r_2 = F$. The results are shown in Tables 5 and 6, where we report the difference between the ultimate and initial flow values ($F - f$), for each of the heuristics the cumulative flow (flow), the relative difference to the best known cumulative flow ($\Delta = (z^{\text{best}} - z^{\text{heur}})/z^{\text{best}}$), and the solution time (time), and for IMFP^2 the value of the first feasible solution (first), the relative difference to the best solution ($\Delta = (z^{\text{best}} - z^{\text{first}})/z^{\text{best}}$), the time to reach the first feasible solution, the value of the best solution (best), and the time to reach the best feasible solution (time). A time limit of 3,600 seconds was imposed for the solution of IMFP^2.

As expected, the three heuristics are able to produce high-quality solutions quickly (with *Quickest-increment* possibly being slightly slower than the other two). For these instances, the hybrid heuristic *Quickest-to-target* seems to performs best (on average within 0.03% from the best known solution for instances on general graphs and within 0.47% from the best known solution for instances on layered graphs). However, IMFP^2 also produces high-quality solutions when given an hour of computing time, often noticeably better than the heuristic solutions.

As the instances used for the above experiments are relatively small, we performed a final experiment in which we use general and layered graphs with 300 nodes to investigate whether the performance of the heuristics (both in terms of quality and time) scales well. Instances of this size are beyond what can be solved to optimality in a reasonable amount of time with IMFP^2. Therefore, instead, we solved the instances twice with Gurobi parameter `SolutionLimit` set to one the first time and to two the second time. When `SolutionLimit` is set to one, Gurobi uses its embedded heuristics to generate a feasible solution and does not even solve the linear programming relaxation. When `SolutionLimit` is set to two, Gurobi solves at least one linear programming relaxation. The results can be found in Table 7 and Table 8.

A few interesting observations can be made. For general instances of this size, *Quickest-increment* performs best (it produces the best solution for all but one instance), but it is noticeably slower than *Quickest-to-ultimate* and *Quickest-to-target*. The first solution found by IMFP^2 is the worst for all instances, but the second solution found by IMFP^2 matches the best in all but one instance. However, it takes substantially longer to find that solution. For layered instances of this size, the situation is not as clear cut, both *Quickest-increment* and *Quickest-to-ultimate* perform well and outperform IMFP^2 in all but one instance. Overall, *Quickest-to-target* produces high-quality solution very efficiently for all instances and should be the method of choice when time is important.

6 Final remarks

We have studied the incremental network design with maximum flows. We have investigated the performance of mixed integer programming formulations and we have analyzed the performance of natural heuristics, both theoretically and empirically.

On the theoretic side, the complexity status of the incremental maximum flow problem for instances with unit arc capacities remains open, even when the network is restricted to bipartite graphs. On the algorithmic side, we have identified classes of instances where integer programming solvers struggle and where the natural heuristics, although fast, do not necessarily provide high-quality solutions. Thus, there is an opportunity to explore more sophisticated heuristics, e.g., metaheuristics.

Finally, there are various other incremental network design problems that are worth studying, e.g., the incremental multicommodity flow problem.

Table 5: Comparison of heuristic performance for general graphs with $n = 35$, $d = 0.3$, $p = 0.7$, and $u_{\max} = 10$.

		Quickest-increment			*Quickest-to-ultimate*			*Quickest-to-target*			IMFP^2				
instance	$F - f$	flow	Δ	time	flow	Δ	time	flow	Δ	time	first	Δ	time	best	time
1	37	12,887	0.0002	1.20	12,884	0.0005	0.61	12,884	0.0005	0.70	12,868	0.0017	0.13	12,890	6.13
2	14	8,690	0.0000	0.37	8,690	0.0000	0.24	8,690	0.0000	0.34	8,690	0.0000	0.03	8,690	0.03
3	61	16,331	0.0021	2.23	16,347	0.0012	0.86	16,341	0.0015	0.91	16,304	0.0038	0.27	16,366	147.27
4	30	11,673	0.0005	1.07	11,679	0.0000	0.52	11,679	0.0000	0.61	11,676	0.0003	0.09	11,679	5.09
5	26	9,456	0.0002	0.74	9,456	0.0002	0.47	9,458	0.0000	0.60	9,453	0.0005	0.06	9,458	1.06
6	33	9,687	0.0003	1.06	9,689	0.0001	0.43	9,689	0.0001	0.55	9,658	0.0033	0.08	9,690	2.08
7	58	18,902	0.0002	1.48	18,904	0.0001	0.65	18,905	0.0000	0.77	18,808	0.0051	1.31	18,905	55.31
8	44	15,706	0.0000	1.25	15,705	0.0001	0.63	15,706	0.0000	0.70	15,701	0.0003	0.11	15,706	1.11
9	27	10,098	0.0001	1.01	10,097	0.0002	0.52	10,096	0.0003	0.52	10,078	0.0021	0.08	10,099	0.47
10	26	8,418	0.0001	0.85	8,418	0.0001	0.52	8,418	0.0001	0.56	8,418	0.0001	0.03	8,419	2.03
			0.0004	**1.126**		**0.0002**	**0.545**		**0.0003**	**0.626**		**0.0017**	**0.219**		**22.058**

Table 6: Comparison of heuristic performance for layered graphs with $\ell = 5$, $n = 10$, $d = 0.3$, $p = 0.7$, and $u_{\max} = 10$.

		Quickest-increment			*Quickest-to-ultimate*			*Quickest-to-target*			IMFP^2				
instance	$F - f$	flow	Δ	time	flow	Δ	time	flow	Δ	time	first	Δ	time	best	time
1	29	3,254	0.0157	1.22	3,295	0.0033	0.82	3,294	0.0036	0.84	3,249	0.0172	0.08	3,306	19.08
2	30	2,188	0.0271	1.21	2,233	0.0071	0.83	2,226	0.0102	0.80	2,190	0.0262	0.05	2,249	15.05
3	37	2,946	0.0081	0.89	2,956	0.0047	0.60	2,961	0.0030	0.64	2,879	0.0306	0.11	2,970	31.11
4	16	1,739	0.0063	0.49	1,743	0.0040	0.45	1,750	0.0000	0.50	1,731	0.0109	0.01	1,750	0.30
5	41	3,321	0.0160	1.26	3,352	0.0068	0.85	3,347	0.0083	0.93	3,323	0.0154	0.06	3,375	15.06
6	33	2,601	0.0069	1.14	2,612	0.0027	0.90	2,611	0.0031	0.94	2,525	0.0359	0.06	2,619	43.06
7	43	3,543	0.0250	1.41	3,604	0.0083	1.23	3,607	0.0074	1.06	3,505	0.0355	0.19	3,634	298.19
8	22	1,684	0.0024	0.57	1,685	0.0018	0.51	1,685	0.0018	0.56	1,661	0.0160	0.02	1,688	1.02
9	46	3,249	0.0107	1.25	3,206	0.0238	1.31	3,260	0.0073	1.10	3,194	0.0274	0.13	3,284	27.13
10	54	3,760	0.0079	1.34	3,736	0.0142	1.51	3,780	0.0026	1.23	3,652	0.0364	0.13	3,790	37.13
			0.0126	**1.078**		**0.0077**	**0.901**		**0.0047**	**0.86**		**0.0252**	**0.084**		**48.713**

Table 7: Comparison of heuristic performance on large instances; general graphs with $n = 300$, $d = 0.3$, $p = 0.7$, and $u_{\max} = 10$.

		Quickest-increment		*Quickest-to-ultimate*		*Quickest-to-target*		IMFP^2			
instance	$F - f$	flow	time	flow	time	flow	time	first	time	second	time
1	349	**8,510,022**	573.89	8,510,019	63.96	8,510,018	69.64	8,501,339	346.50	**8,510,022**	2,618.57
2	276	**7,748,950**	607.13	7,748,926	66.64	7,748,936	80.42	7,741,651	261.88	**7,748,950**	1,551.84
3	306	**7,455,851**	708.26	7,455,825	76.61	7,455,840	82.10	7,446,419	294.29	**7,455,851**	2,043.68
4	302	7,693,036	553.93	7,693,080	62.84	7,693,084	68.21	7,683,753	298.44	**7,693,095**	2,820.65
5	303	**8,561,146**	636.09	8,561,146	74.13	8,561,141	81.39	8,551,909	295.49	**8,561,146**	2,194.76
6	300	**8,173,176**	647.99	8,173,175	68.08	8,173,174	73.66	8,164,741	291.10	8,173,173	1,837.50
7	320	**8,551,244**	677.60	8,551,226	70.46	8,551,237	73.81	8,543,003	318.91	**8,551,244**	2,646.82
8	378	**9,707,335**	760.95	9,707,321	85.66	9,707,334	91.26	9,696,827	390.26	**9,707,335**	2,922.14
9	335	**8,078,470**	626.21	8,078,470	75.93	8,078,468	80.53	8,068,707	331.73	**8,078,470**	2,207.85
10	285	**7,676,476**	546.08	7,676,432	64.63	7,676,418	69.36	7,668,636	272.20	**7,676,476**	2,051.99
			633.81		**70.894**		**77.038**		**310.08**		**2,289.58**

Table 8: Comparison of heuristic performance on large instances; layered graphs with $\ell = 10$, $n = 30$, $d = 0.3$, $p = 0.7$, and $u_{\max} = 10$.

		Quickest-increment		*Quickest-to-ultimate*		*Quickest-to-target*		IMFP^2			
instance	$F - f$	flow	time	flow	time	flow	time	first	time	second	time
1	52	289,131	54.44	289,171	5.98	**289,185**	22.98	288,993	52.97	289,164	165.88
2	25	237,934	8.18	237,935	2.63	**237,937**	3.40	237,919	7.03	237,920	23.67
3	47	304,763	20.03	304,779	32.59	**304,781**	6.77	304,586	21.80	304,775	89.02
4	40	**233,395**	10.85	233,394	5.75	**233,395**	3.91	233,354	13.92	233,362	49.63
5	21	251,797	10.33	251,800	4.51	**251,801**	4.07	251,762	5.83	251,790	24.74
6	30	258,451	12.97	258,448	4.01	258,448	4.99	258,385	28.83	**258,452**	75.28
7	33	**243,606**	11.22	243,604	4.24	243,604	5.55	243,535	21.50	243,542	73.16
8	23	**195,436**	5.05	195,431	1.93	**195,436**	2.54	195,160	1.19	195,406	4.67
9	29	**255,092**	9.61	255,083	4.26	255,086	6.97	254,978	13.23	255,053	48.85
10	50	280,063	18.89	280,070	5.66	**280,078**	10.04	280,003	21.67	280,077	126.33
			16.157		**7.156**		**7.122**		**18.797**		**68.123**

References

[1] M. Baxter, T. Elgindy, A. Ernst, T. Kalinowski, and M. Savelsbergh. Incremental network design with shortest paths. *European Journal of Operational Research*, 238:675–684, 2014.

[2] B. Cavdaroglu, E. Hammel, J.E. Mitchell, T.C. Sharkey, and W.A. Wallace. Integrating restoration and scheduling decisions for disrupted interdependent infrastructure systems. *Annals of Operations Research*, pages 1–16, 2013.

[3] K. Engel, T. Kalinowski, and M.W.P. Savelsbergh. Incremental network design with minimum spanning trees. arxiv:1306.1926, 2013.

[4] B.J. Kim, W. Kim, and B.H. Song. Sequencing and scheduling highway network expansion using a discrete network design model. *The Annals of Regional Science*, 42(3):621–642, 2008.

[5] E.E. Lee, J.E. Mitchell, and W.A. Wallace. Restoration of services in interdependent infrastructure systems: A network flows approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(6):1303–1317, 2007.

[6] E.E. Lee, J.E. Mitchell, and W.A. Wallace. Network flow approaches for analyzing and managing disruptions to interdependent infrastructure systems. *Wiley handbook of science and technology for Homeland Security*, 2009.

[7] A. Mahmood, M. Aamir, and M.I. Anis. Design and implementation of AMR smart grid system. In *Electric Power Conference, 2008. EPEC 2008. IEEE Canada*, pages 1–6. IEEE, 2008.

[8] J.A. Momoh. Smart grid design for efficient and flexible power networks operation and control. In *Power Systems Conference and Exposition*, pages 1–8. IEEE, 2009.

[9] S.G. Nurre, B. Cavdaroglu, J.E. Mitchell, T.C. Sharkey, and W.A. Wallace. Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem. *European Journal of Operational Research*, 223(3):794–806, 2012.

[10] S.G. Nurre and T.C. Sharkey. Integrated network design and scheduling problems with parallel identical machines: Complexity results and dispatching rules. *Networks*, 63(4):306–326, 2014.

[11] S. V. Ukkusuri and G. Patil. Multi-period transportation network design under demand uncertainty. *Transportation Research Part B: Methodological*, 43(6):625–642, 2009.

Appendix

Complexity of Incremental Maximum Flow Problem

Theorem 4. *The incremental maximum flow problem is NP-hard even when restricted to instances where every existing arc has capacity* 1 *and every potential arc has capacity* 3*.*

Proof. We use reduction from the problem *Exact Cover by 3-sets* (X3C). An instance of X3C is given by a collection $\mathcal{S}$ of 3-element subsets of the set $U = \{1, 2, \ldots, 3n\}$, and the problem is to decide if there are n sets $S_1, \ldots, S_n \in \mathcal{S}$ such that $U = S_1 \cup \cdots \cup S_n$. Such an X3C instance can be reduced to the following IMFP instance. The node set is

$$N = \{s, t\} \cup \{v_S \ : \ S \in \mathcal{S}\} \cup \{w_1, \ldots, w_{3n}\}.$$

For every $S \in \mathcal{S}$, there is a potential arc with capacity 3 from s to v_S, and for every $i \in U$ there is an existing arc of capacity 1 from w_i to t. Moreover, for every $S = \{i, j, k\} \in \mathcal{S}$ there are three existing arcs (v_S, w_i), (v_S, w_j) and (v_S, w_k) with unit capacities. Clearly the flow in time period k is at most $3 \cdot \min\{n, k-1\}$, hence the objective value is bounded by

$$3 \cdot [0 + 1 + \cdots + (n-1)] + 3n(T - n)$$

and this upper bound can be achieved if and only if the underlying X3C instance is a YES-instance. □

Approximation for the incremental maximum matching problem

In order to derive a performance guarantee for *Quickest-to-ultimate* on instances of the incremental maximum matching problem, we strengthen Lemma 2. We need the following auxiliary result.

Lemma 7. *For real numbers* $\alpha_1 \geqslant \alpha_2 \geqslant \cdots \geqslant \alpha_n$ *and* $0 \leqslant \beta_1 \leqslant \beta_2 \leqslant \cdots \leqslant \beta_n$ *with* $\sum_{i=1}^n \beta_i = B$*, we have*

$$\sum_{i=1}^n \alpha_i \beta_i \leqslant \frac{B}{n} \sum_{i=1}^n \alpha_i.$$

Proof. By duality,

$$\max \left\{ \sum_{i=1}^n \alpha_i x_i \ : \ x_i \leqslant x_{i+1} \text{ for } 1 \leqslant i \leqslant n-1, \ \sum_{i=1}^n x_i = B, \ x_i \geqslant 0 \text{ for } 1 \leqslant i \leqslant n \right\}$$
$$= \min \left\{ Bz \ : \ y_i - y_{i-1} + z \geqslant \alpha_i \text{ for } 1 \leqslant i \leqslant n, \ y_0 = y_n = 0, \ y_i \geqslant 0 \text{ for } 1 \leqslant i \leqslant n-1 \right\},$$

and a feasible solution for the minimization problem on the RHS is given by $z = 1/n \sum_{i=1}^n \alpha_i$ and

$$y_i = (n-i)z - \sum_{j=i+1}^n \alpha_j \qquad \text{for } 1 \leqslant i \leqslant n-1.$$ □

Lemma 8. *For the incremental maximum matching problem,*

$$z_1 \geqslant \begin{cases} TF - \frac{1}{2} c_r (r+1) & \text{if } f \geqslant r, \\ TF - \frac{1}{4} \left[c_r F + r(r-f) + 2c_r \right] & \text{if } f < r. \end{cases}$$

Proof. Recall that $\sum_{i=0}^{r-1} \lambda_i = c_r$. Lemmas 2 and 7 yield

$$z_1 \geqslant TF - \sum_{i=0}^r \lambda_i (r-i) \geqslant TF - \frac{c_r}{r-1} \sum_{i=0}^r (r-i) = TF - \frac{c_r}{r} \left(r^2 - \frac{1}{2} r(r-1) \right)$$
$$= TF - \frac{1}{2} c_r (r+1).$$

For the case $r > f$ we define $\rho = \max\{0, \lceil (r-f)/2 \rceil\}$. Note that $f + \rho - 1 < f + (r-f)/2 = F/2$. This implies that in the first ρ time periods, there is always a potential arc which is used in the ultimate maximum matching, and is not adjacent to any arc in the current maximum matching. Hence $\lambda_i = 0$ for $0 \leqslant i \leqslant \rho - 1$. From Lemma 2 we obtain

$$z_1 \geqslant TF - \sum_{i=0}^{\rho-1}(r-i) - \sum_{i=\rho+1}^{r-1} \lambda_i (r-i).$$

Using Lemma 7 and $\sum_{i=\rho}^{r-1} \lambda_i = c_r - \rho$, this implies

$$\begin{aligned} z_1 &\geqslant TF - r\rho + \frac{\rho(\rho-1)}{2} - \frac{c_r - \rho}{r - \rho} \sum_{i=\rho+1}^{r-1} (r-i) \\ &= TF - r\rho + \frac{\rho(\rho-1)}{2} - \frac{c_r - \rho}{r-\rho} \left[r(r-\rho) - \frac{1}{2}(r-t)(r+t-1) \right] \\ &= TF - r\rho + \frac{\rho(\rho-1)}{2} - \frac{(c_r - \rho)(r-\rho+1)}{2}. \end{aligned}$$

If $\rho = (r-f)/2$ then

$$\begin{aligned} z_1 &\geqslant TF - \frac{1}{2} r(r-f) + \frac{(r-f)(r-f-2)}{8} - \frac{(c_r - (r-f)/2)(r - (r-f)/2 + 1)}{2} \\ &= TF - \frac{1}{4} \left[c_r F + r(r-f) + 2c_r \right], \end{aligned}$$

and if $\rho = (r-f)/2$ then

$$\begin{aligned} z_1 &\geqslant TF - \frac{1}{2} r(r-f+1) + \frac{(r-f+1)(r-f-1)}{8} - \frac{(c_r - (r-f+1)/2)(r - (r-f+1)/2 + 1)}{2} \\ &= TF - \frac{1}{4} \left[c_r F + r(r-f) + c_r + r \right]. \end{aligned}$$

Now the claim follows from $c_r \geqslant r$. □

Proposition 1. *Quickest-to-ultimate is a 4/3-approximation algorithm for the incremental maximum matching problem.*

Proof. We need to show that $z^* - \frac{4}{3} z_1 \leqslant 0$, and we distinguish two cases.

Case 1. $r \leqslant f$, i.e. $\rho = 0$. Using Lemmas 1 and 8, we obtain

$$z^* - \frac{4}{3} z_1 \leqslant TF - \sum_{j=1}^{r} c_j \leqslant TF - \frac{r(r-1)}{2} - c_r - \frac{4}{3} \left[TF - \frac{c_r(r+1)}{2} \right] \leqslant \frac{2c_r r}{3} - \frac{TF}{3}.$$

The required inequality follows from $T \geqslant c_r$ and $F = f + r \geqslant 2r$.

Case 2. $r > f$. Lemma 8 implies

$$\begin{aligned} z^* - \frac{4}{3} z_1 &\leqslant TF - \sum_{j=1}^{r} c_j \\ &\leqslant TF - \frac{r(r-1)}{2} - c_r - \frac{4}{3} \left[TF - \frac{1}{4} \left[c_r F + r(r-f) + 2c_r \right] \right] \\ &= -\frac{TF}{3} - \frac{r(r-1)}{2} - c_r + \frac{c_r F}{3} + \frac{r(r-f)}{3} + \frac{2c_r}{3}. \end{aligned}$$

Using $T \geqslant c_r + 1$ and then substituting $F = f + r$, we obtain

$$z^* - \frac{4}{3}z_1 \leqslant -\frac{c_r F}{3} - \frac{F}{3} - \frac{r^2}{2} + \frac{r}{2} - c_r + \frac{c_r F}{3} + \frac{r^2}{3} - \frac{rf}{3} + \frac{2c_r}{3}$$
$$\leqslant -\frac{f}{3} - \frac{r}{3} + \frac{r}{2} - c_r - \frac{rf}{3} - \frac{c_r}{3} \leqslant \frac{r}{6} - \frac{c_r}{3}.$$

Now the claim follows from $c_r \geqslant r$. □