Same bang, fewer bucks: efficient discovery of the cost-influence skyline *Supplementary Information*

Matthijs van Leeuwen[*] Antti Ukkonen[†]

1 Proofs of propositions

Proof. **(of Proposition 4.1)** Let $X_i = X_i^{\text{INFMAX}}$ for short. First, observe that $X_1 \in P_1$. This is because the greedy algorithm maximizes marginal gain, and no point on P_1 can dominate this, no matter what the costs are. Next, suppose we have $X_{i-1} \in P_{i-1}$. Since X_i is obtained from X_{i-1} by adding that u that maximizes marginal gain in $I(\cdot)$, no point on P_i can dominate X_i. This is because for some $X' \in P_i$ to dominate X_i, there would have to have been some set in P_{i-1}, such that when some item u' is added to this set, the marginal gain is larger than the one found by greedy when forming X_i. Because for a submodular $I(\cdot)$ the marginal gains are decreasing, such a u' can not exist. If it existed, the greedy algorithm would have added u' at some earlier point. □

Proof. **(of Proposition 4.2)** We provide an example that shows that there can exist some $X \in P_k$ that dominates X_k^{BFTB}. Suppose we have only three vertices, A, B, and C, with the costs and influences as shown in Table 1, and let $k = 2$. Observe first that the influence function is indeed submodular, and that the costs are linear. The greedy BFTB algorithm would first choose B, and then the set BC in the second iteration, as these maximize the bang-for-the-buck ratio. However, the levelwise skyline algorithm would first find that $P_1 = \{A, B, C\}$, and then extend this to $P_2 = \{AB, AC\}$. However, the set AC clearly dominates the set BC as it has both lower cost and higher influence.

Table 1: Example costs and influences for Proof of proposition 4.2

set	Cost	Influence
A	1	2
B	2	5
C	3	6.5
AB	3	5.5
AC	4	8.33
BC	5	8

2 Additional speedups to FAST-SKYLINE

Below we discuss a number of implementation details that are relatively simple but nonetheless useful in practice.

2.1 Implementing the marginal gain cache In our implementation the marginal gain cache is a collection of lists indexed by vertex id. For every $u \in V$ we have a list R_u of triplets of marginal gains of u for different subsets Y. We scan over R_u until we find a case where the resulting upper bound $\Gamma(Y, u) + I(X)$ is less than $I_{\max}$, as this is enough to prune $X_j \cup u$ from consideration. If no such case is found, we return the smallest valid $\Gamma(Y, u)$ found in R_u.

2.2 Managing the priority queue The QUEUE-PUSH procedure implements an additional optimization that avoids inserting candidates into Q that we can prune already at that point. That is, we keep incrementing pos_j until we either reach the end of U, or we find a case that has a upper bound above the current $I_{\max}$. Notice that when the candidate is popped from Q, the value of $I_{\max}$ might have increased, and we have to check the upper bound again. This leads to a substantial speedup in practice as we can prune many of the points in $\mathcal{E}(P_{i-1})$ from entering the queue in the first place.

2.3 More speedups We mention some further implementation details that help making the algorithm run faster in practice.

1. The algorithm will generate duplicate sets. To avoid these, we conduct an additional check after line 10 of the main algorithm to find out if the resulting set $X_j \cup u$ has already been processed. In practice we do this by checking if the freshly popped set is equal to the set that was processed immediately before.

[*]Machine Learning, Department of Computer Science, KU Leuven,Leuven, Belgium matthijs.vanleeuwen@cs.kuleuven.be.

[†]Helsinki Institute for Information Technology HIIT, Aalto University, Finland.

Algorithm 1 Details of subroutines used in FAST-SKYLINE

```
1: procedure MGC-GET($u, X$):
2: $R \leftarrow M_u$
3: for $i = 1$ to $|R|$ do
4:     $(u, Y, \Gamma(Y, u)) \leftarrow R_i$
5:     if $Y \subset X$ and $\Gamma(Y, u) + I(X) < I_{\max}$ then
6:         return $\Gamma(Y, u)$
7: return smallest $\Gamma(Y, u)$ found in $R_u$ with $Y \subset X$
```

```
1: procedure QUEUE-PUSH($j$):
2: repeat
3:     $\mathrm{pos}_j \leftarrow \mathrm{pos}_j + 1$
4: until $\mathrm{pos}_j = |U|$ or MGC-GET$(U_{\mathrm{pos}_j}, X_j) + I(X_j) > I_{\max}$
5: if $\mathrm{pos}_j < |U|$ then
6:     insert $(j, \mathrm{pos}_j)$ into $Q$ with priority $C(X_j \cup U_{\mathrm{pos}_j})$
```

2. In MGC-GET, often an item can be pruned without scanning over R_j, but simply by checking if $\Gamma(\emptyset, u) + I(X) < I_{\max}$. That is, for many vertices even the marginal gain that results from adding u to the empty set is enough to prune u when $I_{\max}$ has grown sufficiently large.

3 Effect of the filtering heuristic

We make a note about the effect the filtering heuristic has on the running time, as well as on the resulting skyline. Prior to calling FAST-SKYLINE given P_{i-1}, we prune P_{i-1} by discarding points as described in the main manuscript. We set the parameter $\theta = 1$, meaning the resulting skylines will have a resolution of 1 unit of cost. As the reason for applying the filter is to reduce redundancy as the size of the skyline grows, we only apply the filter when the size of P_{i-1} is above some threshold. In the following experiment we used the thresholds 100 and 200 (denoted by f100 and f200 below); in all other experiments we used 200 as default threshold.

Figure 1 shows the unfiltered skyline, as well as the filtered ones for *soc-Epinions1* and *ca-CondMat*. The plots on the left show the entire skyline, while on the right we zoom into a small region indicated by the dashed rectangle in the left figures. We can observe that the curves for f100 and f200 overlap almost perfectly with the solid black line (no filtering). The small differences that are present appear in the low-cost parts of the skylines, as indicated by the detail plots on the right. The quality of the resulting skylines is hardly affected by applying filtering, and unlikely to make a noticeable difference in practice.

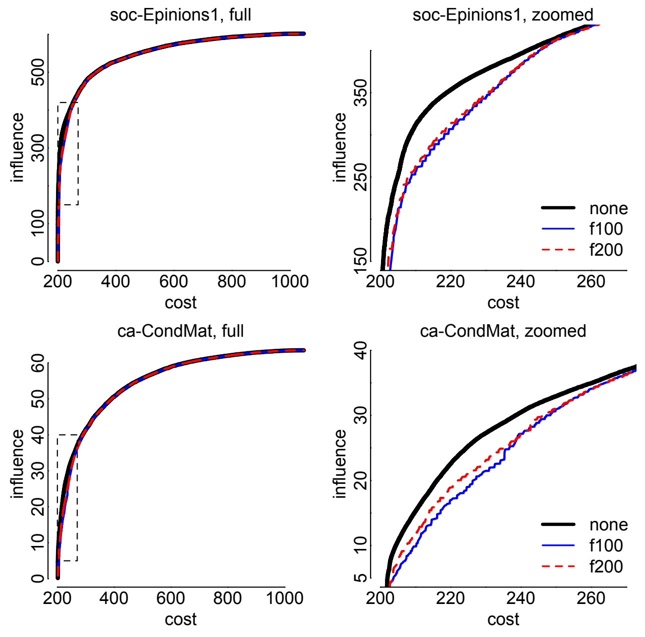

Figure 1: Effect of filtering the skyline P_{i-1} prior to calling FAST-SKYLINE. Black curve shows the unfiltered skyline, while the other lines show filtered ones. Figures on the left show the entire skyline, while figures on the right zoom in on the region indicated by the dashed rectangle.

Table 2: Effect of skyline filtering (with $k = 40$)

Network	filter	time (sec)	P_k
ca-HepTh	none	105	2,612
	f100	29	1,191
	f200	29	1,218
ca-CondMat	none	612	4,795
	f100	69	1,517
	f200	71	1,573
soc-Epinions1	none	7,684	11,128
	f100	1,640	1,648
	f200	1,655	1,654

Table 2 shows both running time (with $k = 40$) and the size of P_{40} for some of our smaller datasets, either without filtering or with one of the two thresholds are applied.[1] We can observe that filtering leads to a substantial reduction in running time, no matter what the number of vertices is. The resulting skylines are obviously substantially smaller.

Note that with filtering, it cannot be guaranteed that the discovered skyline contains exactly the solution found by the greedy InfMax algorithm. In practice, however, the skyline contains at least solutions that are virtually identical in terms of both cost and influence even with (modest) filtering.

[1]The values are computed with a different random allocation of vertex costs, hence the small differences to Table 1 in the article itself.