DP_Kruskal: a concurrent algorithm to maintain dynamically minimum spanning trees

D. Benedí*, A. Duch, E. Pasarella, C. Zoltan

Universitat Politècnica de Catalunya

*Contact: daniel.benedi@upc.edu

EURO-PAR CONFERENCE 2024

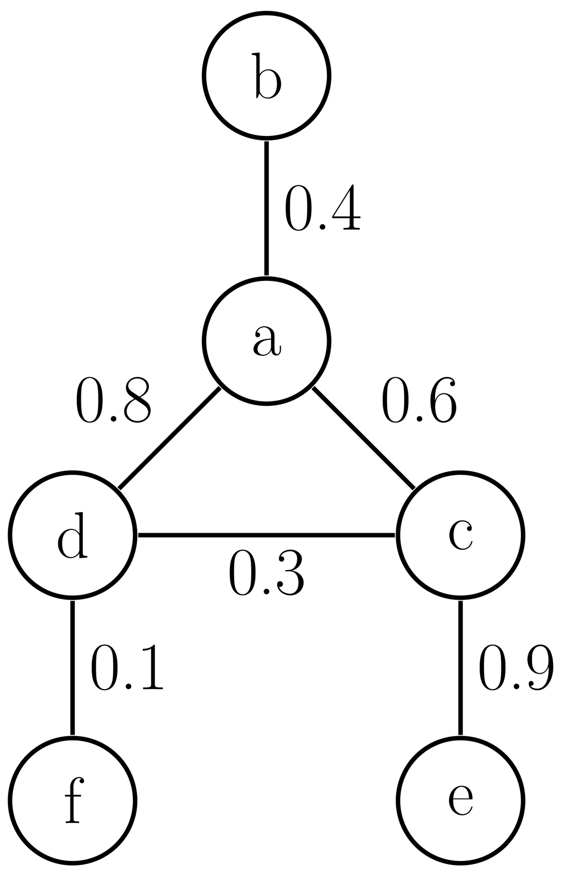
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

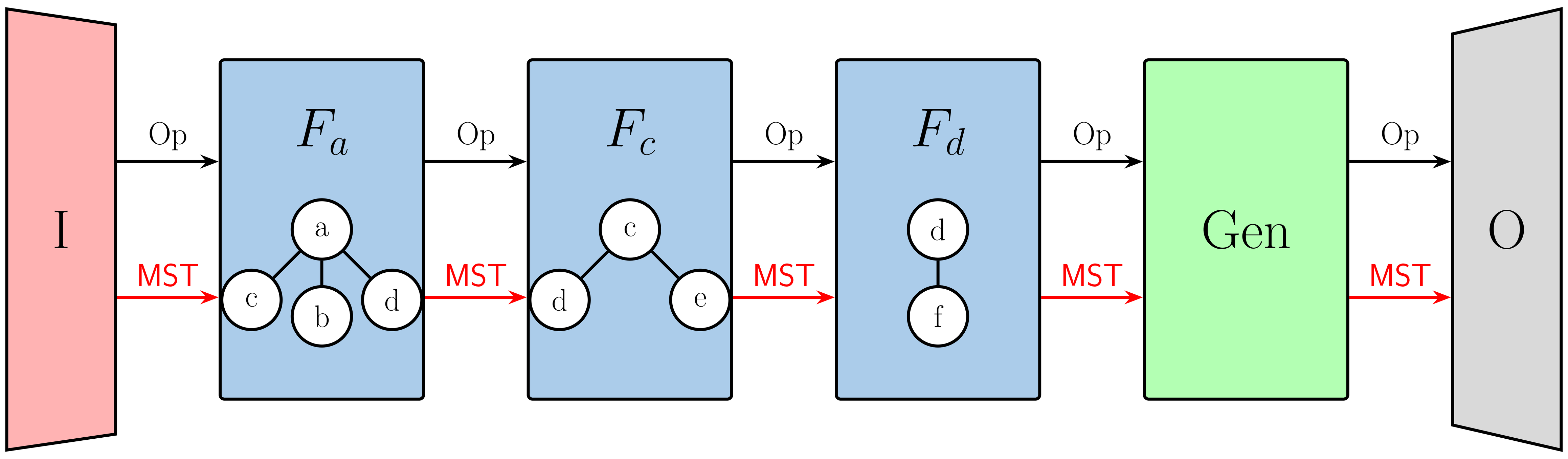
Introduction

We **introduce** `DP_Kruskal`: a parallel and concurrent Kruskal-based algorithm, **defined** on the Dynamic Pipeline Model (DPM) [5], to **solve** the fundamental Minimum Spanning Tree (MST) problem. `DP_Kruskal` **outperforms** other algorithms and is particularly **effective** for **dense** graphs.

DP_Kruskal

A *dynamic pipeline* (DP) is a sequence of **connected** (via communication channels) **concurrent, stateful stages**. `DP_Kruskal` distributes the input graph $G = (V, E)$ along a DP with event and graph channels carrying events and MSTs, respectively.





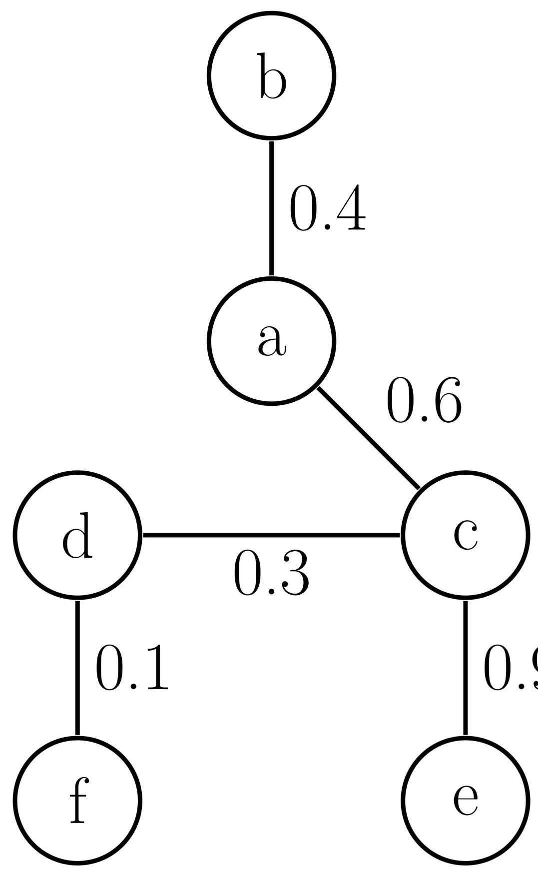


Figure 1: An instance of `DP_Kruskal`

Instance of DP_Kruskal

Figure 1 illustrates an instance of `DP_Kruskal`. It consists of:

- *Input (I):* Receives events of the form (`op`, `e`) and passes them through the event channel; if `op` = `mst`, sends an empty set via the graph channel.
- *Filter (F_v):* v is a vertex, called the *root* of the filter. When an event (`op`, `e`) arrives: (i) If `op` $\in$ {`insert`, `update`, `delete`} and e is incident to v, processes the event; otherwise, passes it to the next stage. (ii) If `op` = `mst`, computes a partial MST from the union of the arriving MST and the tree in $F(v)$, then passes it to the next stage.
- *Generator (Gen):* If `op` $\in$ {`insert`, `update`} arrives, spawns a new Filter stage $F(v)$, adding it to the pipeline between the last filter and *Gen*.
- *Output (O):* If `op` = `mst` arrives, outputs the MST_G.

Single core and Static random graphs

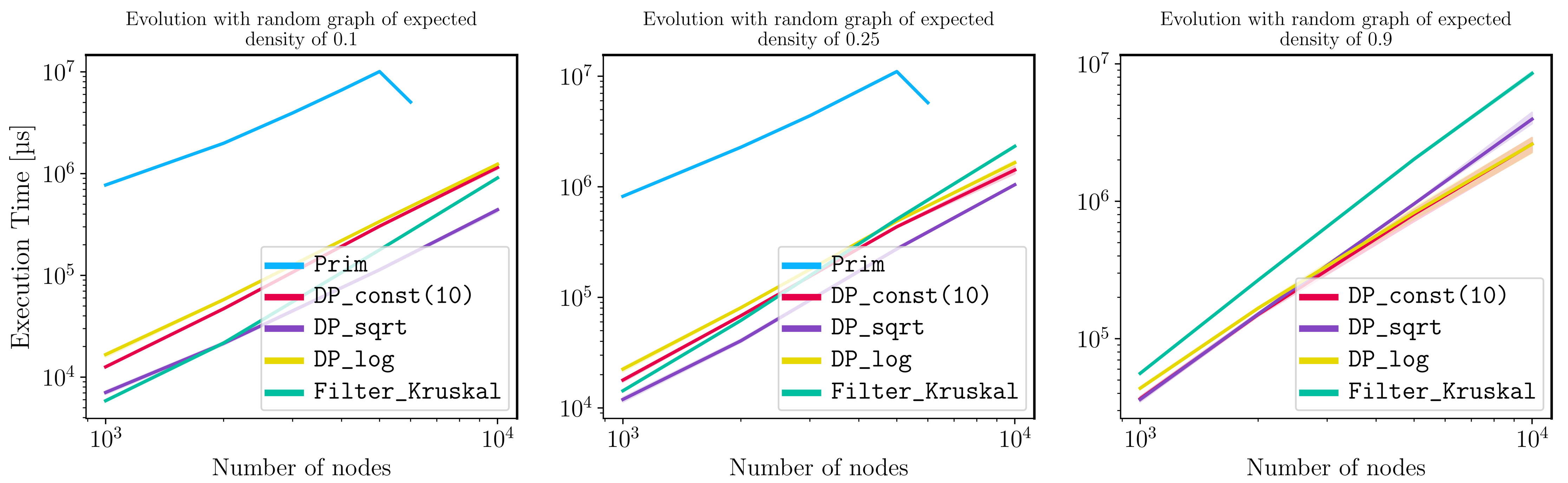


Figure 2: Comparison of `DP_Kruskal` with different root numbers per filter: `DP_const(10)`, `DP_log`, and `DP_sqrt`; and with other algorithms on a single core over random graphs of n vertices. For each size and probability 20 random graphs were generated

We compared `DP_Kruskal` against (i) `Filter_Kruskal` [4], and (ii) a message-passing `Prim` [2] implementation on a single core and randomly generated graphs of n vertices studying the **effect of the number of roots** in each filter of `DP_Kruskal` three options were considered: (a) a constant number of roots `DP_const`, (b) $\log(n)$ roots `DP_log`, and (c) $\sqrt{n}$ roots `DP_sqrt`.

Although for small graphs, `Filter_Kruskal` is faster; as **graph size and density increase**, `DP_sqrt` becomes the **best option** (see Figure 2).

Multi core and Dynamic real graphs

We compare `DP_Kruskal` and `Filter_Kruskal` on realistic dynamic graphs obtained from DynGraphLab [1]. We observe that:

- `DP_Kruskal` is effective for maintaining the MST (See Table 1).
- `DP_Kruskal` has **excellent performance** in a parallel environment, showcasing **significant scalability** (See Figure 3).

Dataset	n	op.	`Filter_Kruskal`	`DP_Kruskal`
`as-caida`	31379	119468	1h 30min	1h 19min
`movielens10m`	49847	384585	1h 39min	1h 20min
`simplewiki`	100312	889016	17h 8min	11h 29min

Table 1: Time with 1 core and 10 roots per filter

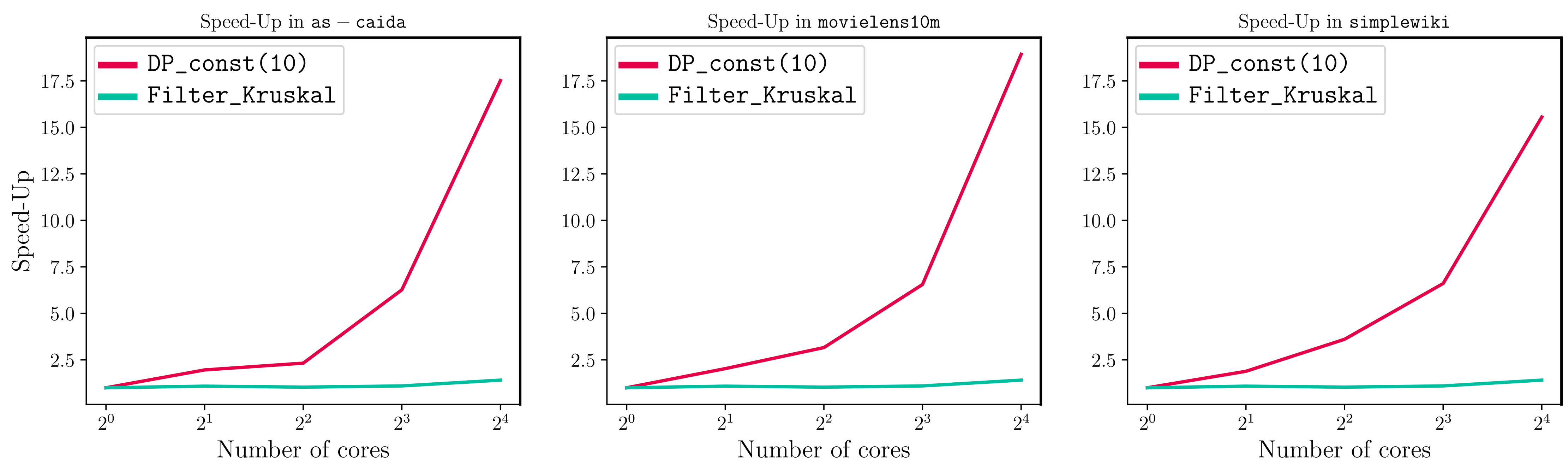


Figure 3: Multicore comparison of speed-ups

Final Remarks

Experiments on random and real graphs demonstrate that the `DP_Kruskal` algorithm is particularly **competitive for dense graphs**, showing improved performance over the `Filter_Kruskal` algorithm for graphs with over $2 \cdot 10^3$ vertices. It also **scales well**, enhancing efficiency and speed up to 16 cores. Future work includes for instance to compare `DP_Kruskal` against a `MapReduce`-based competitor [3]; to evaluate other MST algorithms within the `DP_Kruskal` model; to assess different implementation languages; to experiment with larger datasets and other dynamic graph models.

Source Code

References

[1] Kathrin Hanauer, M.H., Schulz, C.: Recent advances in fully dynamic graph algorithms – a quick reference guide (2022)

[2] Lončar, V., Škrbić, S., Balaž, A.: Parallelization of minimum spanning tree algorithms using distributed memory architectures (2014)

[3] Nowicki, K., Onak, K.: Dynamic graph algorithms with batch updates in the massively parallel computation model (2021)

[4] Osipov, V., Sanders, P., Singler, J.: The filter-kruskal minimum spanning tree algorithm (2009)

[5] Pasarella, E., Vidal, M.E., Zoltan, C., Royo Sales, J.P.: A computational framework based on the dynamic pipeline approach (2024)