

On Improving the Cohesiveness of Graphs by Merging Nodes: Formulation, Analysis, and Algorithms (Online Appendix)

Fanchen Bu

School of Electrical Engineering, KAIST
Daejeon, South Korea
boqvezen97@kaist.ac.kr

C ON REAL-WORLD EXAMPLES

In this section, we provide more discussions on the real-world examples (bus station reorganization and multiple team formation) used in this work. Specifically, we provide more details on how both merging nodes and k -trusses have realistic meanings.

C.1 Bus station reorganization

Regarding bus station reorganization (or transportation systems in general), we can consider the following specific real-world scenario: we are managing the bus transportation system of a city, and we want to reduce the total number of bus stations to decrease the expenses of maintaining the bus stations due to some financial reasons. We model the transportation system as a graph with bus stations as nodes and routes as edges, and we aim to do so by merging bus stations while maximizing connectivity among the stations. As a measure of (higher-order) connectivity [8, 16, 34], the size of a k -truss is a reasonable choice as a numerical metric for this purpose. Moreover, k -trusses are highly related to triangles, while triangles are important motifs for indicating higher-order connectivity [35, 36] and robustness [9, 11, 27] in transportation/traffic systems. In such a scenario, our proposed algorithm, which aims to maximize the k -truss size by merging nodes, can be used as a solution for finding stations to merge.

C.2 Multiple team formation

Regarding multiple team formation, we can consider the following specific real-world example: as the CEO of a company, we have employees (nodes) and social relations (edges) between them (which corresponds to the input graph), and we want to form multiple small-scale teams (merge nodes) among the employees. We aim to maximize the communication between teams (which is known to be beneficial to teams' performance [3, 15, 23], where we suppose that two teams A and B can communicate well with each other if at least one member in A and one member in B have social connections (i.e., communication is correlated with connectivity). In general, when we merge nodes into teams, the formed teams constitute a social network, where cohesive subgraphs such as k -trusses are indicative of high connectivity and robustness [29]. Moreover, the abundance of triangles and the bounded diameter (i.e., teams can reach each other within a bounded number of hops) indicated by k -trusses are both helpful for better communication between nodes (teams) [10]. In such a scenario, our proposed algorithm, which aims to maximize the k -truss size by merging nodes, can be used as a solution for forming teams.

Kijung Shin

Kim Jaechul Graduate School of AI, KAIST
Seoul, South Korea
kijungs@kaist.ac.kr

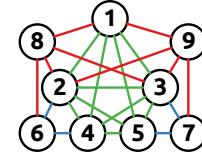


Figure 8: An example for the counterpart problem.

C.3 Limitations and more discussions

Definitely, in real-world scenarios, more conditions and factors might be considered, and we would like to emphasize that we are considering a more general problem, while additional real-world constraints can be considered ad hoc in practical usage. For example, for the bus station reorganization application, where we consider the constraints that only bus stations within a distance threshold can be merged (and indeed we have such information), when we choose candidate pairs (in Algorithms 2 and 3), for each candidate merger, we can simply check the distance between the two nodes (stations), and include the merger in the final returned set of candidates only if the distance is within the distance threshold. See Appendix E for additional experiments on real-world bus station datasets, where we take distance constraints into consideration.

C.4 Additional related work on real-world examples

For designing and optimizing bus/subway networks, many tools and methods have been used, e.g., mixed-integer programming [17], linear programming [21], and genetic algorithm [6, 20]. The multiple team formation problems have been widely studied in the field of operations research, where many methods, such as variable neighborhood local search metaheuristics [14] integer programming [4], and evolutionary algorithm [1], have been used. In this work, we study the multiple team formation problems from the perspective of social networks.

D COUNTERPART PROBLEM USING k -CORES

In this section, we discuss the counterpart problem using k -cores and analyze the technical similarity between this problem and the anchored k -core problem [2].

Recall the problem that we are studying in this work.

PROBLEM 1. (TIMBER: Truss-sIze Maximization By mERgers)

- **Given:** a graph $G = (V, E)$, $k \in \mathbb{N}$, and $b \in \mathbb{N}$,
- **Find:** a set P of up to b mergers in G , i.e., $P \subseteq \binom{V}{2}$ and $|P| \leq b$,
- **to Maximize:** the size of the k -truss after the mergers, i.e.,

$$f(P) = f(P; G, k) = |E(T_k(PM(P; G)))|.$$

It is possible to consider the counterpart problem using k -cores. Note that the size of a k -core is usually defined as the number of nodes in the k -core.

PROBLEM 2. (The counterpart problem of TIMBER using k -cores)

- **Given:** a graph $G = (V, E)$, $k \in \mathbb{N}$, and $b \in \mathbb{N}$,
- **Find:** a set P of up to b node mergers in G , i.e., $P \subseteq \binom{V}{2}$ and $|P| \leq b$,
- **to Maximize:** the size of the k -core after the mergers, i.e.,

$$f(P) = f(P; G, k) = |V(C_k(PM(P; G)))|,$$

where $C_k(G_0)$ is the k -core of a graph G_0 .

We also provide the problem statement of the anchored k -core problem here for the sake of completeness. We first define the anchored k -core.

Definition 6 (anchored k -cores). Given $G = (V, E)$, $k \in \mathbb{N}$ and a set $A \subseteq V$ of anchors, the anchored k -core of G w.r.t the anchor set A is the maximum subgraph $\tilde{C}_k(G; A) = (V', E')$ of G where $A \subseteq V'$ and $d(v'; \tilde{C}_k(G; A)) \geq k, \forall v' \in V' \setminus A$.

PROBLEM 3. (The anchored k -core problem)

- **Given:** a graph $G = (V, E)$, $k \in \mathbb{N}$, and $b \in \mathbb{N}$,
- **Find:** a set A of up to b nodes in G , i.e., $A \subseteq V$ and $|A| \leq b$,
- **to Maximize:** the size of the k -core after anchoring the chosen nodes in A , i.e.,

$$f(A) = f(A; G, k) = |V(\tilde{C}_k(G; A))|.$$

We claim the technical similarity between the counterpart problem of TIMBER using k -cores and the anchored k -core problem, stated as follows.

CLAIM 1. *Problem 2 and Problem 3 are technically similar. Specifically, merging two nodes in Problem 2 is similar to anchoring both of the nodes in Problem 3.*

Formally, given $G = (V, E)$ and $k \in \mathbb{N}$, if two nodes v_1 and v_2 are not in the current k -core and have no common neighbor in the current $(k - 1)$ - and $(k - 2)$ -shell, and after the merger between them, v_1 is the new k -core, then $|V(\tilde{C}_k(G; \{v_1, v_2\}))| = |V(C_k(PM(\{(v_1, v_2)\}; G)))| + 1$, where the difference of one node comes from the merger itself which reduces the number of nodes by one.

See the example in Figure 8. Let $k = 4$, the current k -core contains the five nodes 1, 2, 3, 4, 5. Both merging 6 and 7 and anchoring 6 and 7 brings 8 and 9 into the k -core.

E ADDITIONAL EXPERIMENTS ON REAL-WORLD BUS STATION DATASETS

We conduct additional experiments on real-world bus station datasets, where real-world distance constraints are considered.

E.1 Necessary algorithmic modifications

As mentioned in Appendix C, we can make modifications in Algorithms 2 and 3 to incorporate distance constraints (or other real-world constraints). Here, provide more details.

Suppose that we are given a distance threshold d_{th} and we are allowed to merge stations within the distance threshold only. For each pair of nodes, v_1 and v_2 , let $\text{dist}(v_1, v_2)$ denote the distance

Table 4: The basic statistics of the 21 bus station datasets. Notations: n denotes the number of nodes, m the number of edges, and p_{th} the number of pairs within the 1-km distance threshold.

Dataset	n	m	p_{th}
kuopio	528	676	6695
luxembourg	1,350	1,850	9,382
venice	1,622	2,250	20,177
turku	1,817	2,304	37,094
palermo	2,176	2,559	72,907
nantes	2,206	2,579	40,198
canberra	2,520	2,908	42,143
lisbon	2,730	3,362	181,852
bordeaux	3,212	3,798	64,786
berlin	4,316	5,869	26,764
dublin	4,361	5,271	97,764
prague	4,441	5,862	58,002
winnipeg	5,079	5,846	159,139
detroit	5,683	5,946	140,450
helsinki	6,633	8,592	145,802
adelaide	7,210	8,827	131,111
rome	7,457	9,616	219,972
brisbane	9,279	11,242	181,933
paris	10,644	12,309	372,037
melbourne	17,250	19,071	363,122
sydney	22,659	26,720	555,262

between v_1 and v_2 . In Algorithm 2, after Line 5 and before Line 6, we can check the distance between v_i and v_o , and skip the pair (v_i, v_o) if the distance between the two nodes exceeds the given threshold. Similarly, in Algorithm 3, after Line 2 and before Line 3, we can check the distance between v_1 and v_2 , and skip the pair (v_1, v_2) if the distance between the two nodes exceeds the given threshold. By doing so, we make sure that the candidate mergers all satisfy the distance-threshold constraints.

E.2 Experimental results

We conduct experiments on real-world bus station datasets [19]. Among the 25 available datasets, four datasets are not included because they are too sparse. See Table 4 for the basic statistics of the datasets. Note that the number of possible mergers can be considerably large in real-world bus station networks, and thus it is nontrivial to find good mergers. We set the distance threshold as one kilometer for all the datasets. We take the largest connected component of each dataset.

We compare the proposed method BATMAN with 3 baseline methods:

- **BM (BATMAN):** the proposed method with necessary modifications to consider distance constraints;
- **CR (core):** a method for the counterpart problem considering k -cores (see Appendix F for more details);
- **CS (constraints):** a method considering distance constraints only, and picking pairs satisfying the constraints uniformly at random (the average performance over 10 random trials are reported);
- **CL (closest):** a method greedily merging the pairs with the smallest distance.

Table 5: The results of additional experiments on real-world bus station datasets measured by the Z-score (the larger the better). The best performance is marked in bold.

metric	BM	CR	CS	CL
VB	0.2961	0.4953	0.2996	-1.0910
EB	0.2686	0.5820	0.3109	-1.1615
ER	-0.8604	0.9090	0.6313	-0.6798
SG	0.7637	-0.2035	-0.2932	-0.2670
NC	1.6303	-0.3605	-0.5380	-0.6366
AD	-0.2857	-0.1408	1.2968	-0.8702
TS	1.6837	-0.6469	-0.3103	-0.7266
LC	0.8895	-0.7762	0.8599	-0.9732
average	0.5482	-0.0177	0.2821	-0.8008

Table 6: The results of additional experiments on real-world bus station datasets measured by the average rank (the smaller the better). The best performance is marked in bold.

metric	BM	CR	CS	CL
VB	2.0952	2.0476	2.0952	3.7619
EB	2.0476	2.0000	2.1429	3.8095
ER	3.2857	1.6190	1.8095	3.2857
SG	1.7619	2.5238	2.3810	2.6667
NC	1.0000	2.3333	2.7619	2.9048
AD	2.6190	2.2381	1.2381	3.9048
TS	1.0000	3.3810	2.2381	3.3810
LC	1.5714	3.2381	1.6190	3.5238
average	1.9226	2.4226	2.0357	3.4048

The performance is evaluated by 8 robustness measures that have been considered for transportation networks: (1) VB (average vertex betweenness) [7, 22, 26, 28], (2) EB (average edge betweenness) [7, 22, 26, 28], (3) ER (effective resistance) [30, 32, 33], (4) SG (spectral gap) [5, 12, 30], (5) NC (natural connectivity) [13, 30], (6) AD (average distance) [13, 24], (7) TS (transitivity) [13, 18, 30], and (8) LC (average local clustering coefficient) [13, 18, 30]. The proposed method performs best overall. See Tables 5 and 6 for the detailed results. For each dataset, we apply the mergers output by each considered method, and compute each considered robustness metric of the post-merger graph. We compute for each metric the Z-score (standard score) and the average ranking (the arithmetic mean of the ranking w.r.t the corresponding metric, over all datasets). We also report the average Z-scores and the average (again, the arithmetic mean) of the average rankings over all the considered metrics.

In Figures 9-11, we visualize the results of different methods on several datasets. For each dataset, we draw each node (bus station) with the location information and the topology information (i.e., routes). In each plot, each point represents a node (station), and each black line segment between two nodes represents an edge (route). We also mark the merged nodes (bus stations) in different colors, where the nodes merged together are in the same color. The visualization of graphs helps us obtain a direct and intuitive understanding of the structure of the bus system and helps us to more easily analyze the pattern of merged stations.

Table 7: The correlations between the core size and the robustness measures are clearly weaker than the correlations between the truss size and the robustness measures. For each robustness metric, we list Pearson’s r between the truss size and the metric when we try to maximize the size of a k -truss, and Pearson’s r between the core size and the metric when we try to maximize the size of a k -core.

metric	truss	core
VB	0.99	0.55
EB	0.98	0.57
ER	0.97	0.51
SG	0.99	0.50
NC	0.99	0.55

F WHY IS THE SIZE OF A k -TRUSS A GOOD MEASURE?

In this section, we provide more discussions on why the size of a k -truss is a good measure of cohesiveness and robustness, especially when compared to the size of a k -core.

First, we compare k -trusses with k -cores in measuring cohesiveness and robustness. k -Trusses are theoretically more cohesive than k -cores. As we have mentioned in Section 1, k -trusses require both the engagements of nodes and the interrelatedness of edges while k -cores require only the engagements of the nodes.

Indeed, k -trusses are empirically more indicative of robustness than k -cores. Recall in Section 6.1 (Figure 2), we conduct experiments where we apply our proposed algorithm on the email dataset to enlarge the size of its 10-truss by 100 mergers. We observe that the size of the 10-truss increases and so does the robustness of the whole graph w.r.t multiple robustness measures, where high correlations between the truss size and the robustness measures exist. However, the size (specifically, both the number of nodes and the number of edges) of 11-core (10-truss is a subgraph of 11-core) actually decreases, i.e., the size of a k -core decreases while the robustness of the whole graph increases, surprisingly showing a negative correlation between the size of a k -core and the robustness of the whole graph. We have also tried merging nodes to maximize the size of a k -core (i.e., the counterpart problem using k -cores). We adapt an algorithm for the anchored k -core problem for this purpose because the counterpart problem using k -cores is technically similar to the anchored k -core problem as shown in Appendix D. We observe that the correlations between the core size and the robustness measures (although also positive) are clearly weaker than the correlations between the truss size and the robustness measures (see Table 7).

G REMARKS

In this section, we provide additional remarks to the main text.

G.1 On mergers

REMARK 3. *In this paper, merging two nodes in a graph means identifying the two nodes [25] into one node. and any other node adjacent to either of the two nodes will be connected to the merged node, without adding any self-loop or multi-edge. Since we focus*

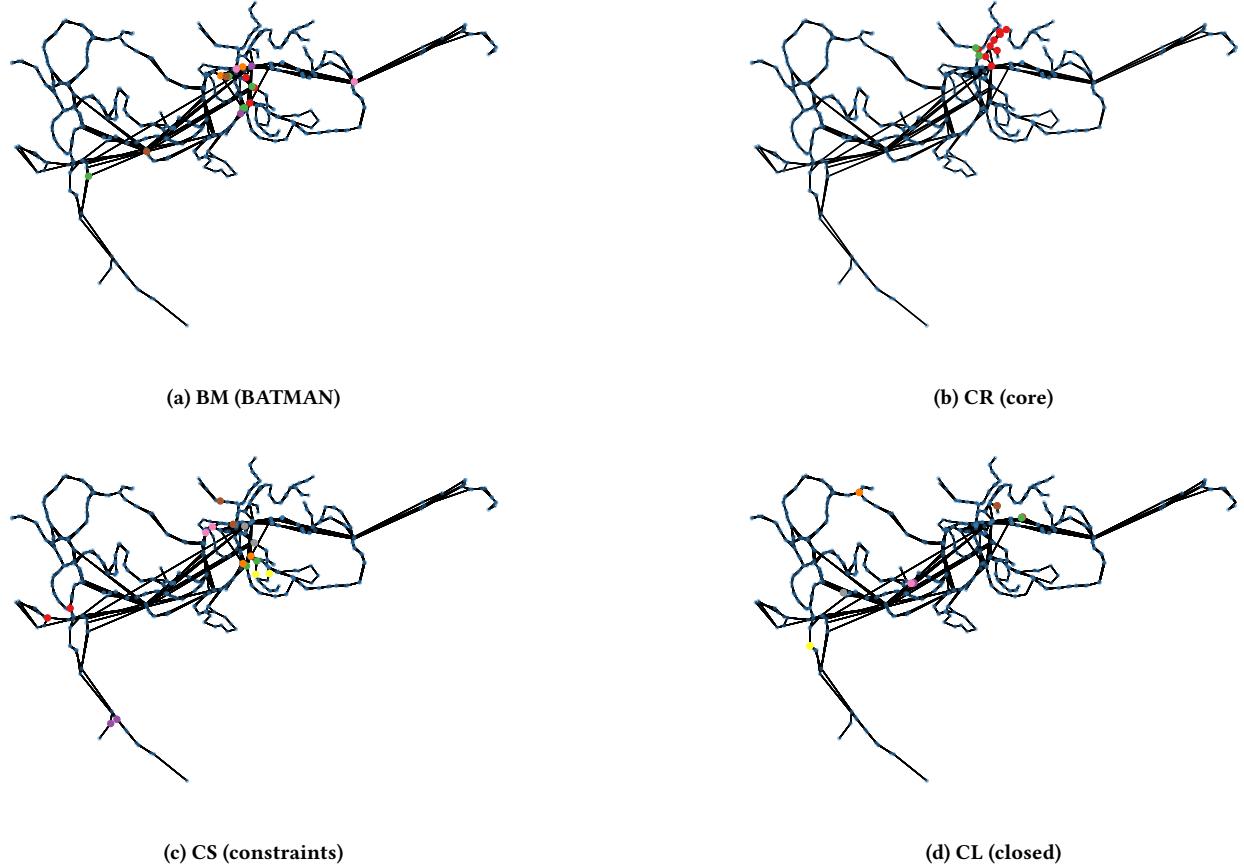


Figure 9: The results on the *kuopio* dataset in the additional experiments on real-world bus station datasets.

on simple graphs, we neither add a multi-edge even if some node is adjacent to both pre-merger nodes, nor add a self-loop even if the two pre-merger nodes are adjacent to each other.

G.2 On the baselines in the main experiments

REMARK 4. We acknowledge that there exist algorithms proposed for maximizing/minimizing the size of a k -truss via adding/deleting nodes/edges, and have discussed them in Section 2. However, since those algorithms greedily find a single node/edge as the objective and do not consider the combination of multiple nodes/edges, those algorithms cannot be directly applied to our problem (cf. Remark 2). Moreover, we compared the best pairs to merge and connect, and the best choices for merging and connecting are very different. Specifically, merging the best pair for connecting usually does not increase the size of the k -truss for the same k .

G.3 On the time complexity (Theorem 4)

REMARK 5. The dominant terms in the time complexity are the two 1.5-order terms from the truss decomposition and checking the truss size after each candidate merger. When we fix n_i and n_o as constants, the time complexity becomes $O(b(|E|^{1.5} + n_c|E(T_{k-1})|^{1.5}))$; If we further fix n_c as a constant, then the time complexity becomes

$O(b|E|^{1.5})$ totally dominated by that of truss decomposition. Empirically, we do observe that for different heuristics, the differences in the running time are small as long as the times checking the truss size are the same (see Section 6.2).

H EXPERIMENTAL DETAILS

H.1 Details regarding Q3

Exclude outside-outside mergers. For each dataset and each $k \in \{5, 10, 15, 20\}$, we randomly sample 10,000 **inside-inside (II)/inside-outside (IO)/outside-outside (OO)** mergers, and report the highest increase in the truss size among all the 10,000 sampled mergers in each of the three cases. For each experimental setting (dataset and k) and each case (II/IO/OO), we do five independent trials and take the average. We do this for one round (i.e., the budget $b = 1$). For each dataset, we compute the truss-size increase and the number of possible mergers of each case averaged over all the considered k values. Compared to II and IO mergers, the number of OO mergers is much higher ($38\times$ on average), while their performance is much lower ($1.8\times$ on average). Thus, excluding OO mergers in our proposed method, BATMAN, for speed is justified.

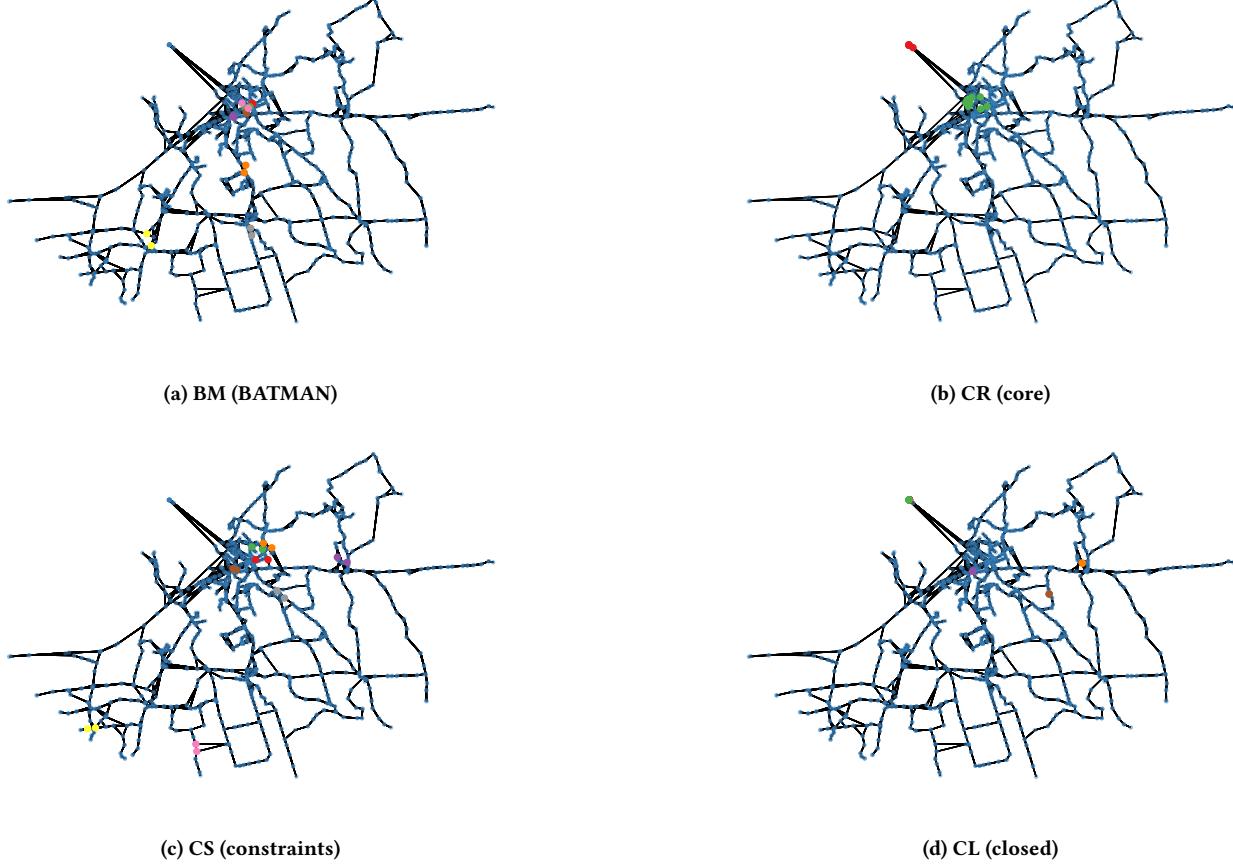


Figure 10: The results on the *venice* dataset in the additional experiments on real-world bus station datasets.

Heuristics for choosing nodes and mergers. Specifically, there are three kinds of heuristics we shall compare: the heuristics for choosing inside nodes, those for choosing IOMs, and those for choosing IIMs. For each kind of heuristics, for each dataset and each $k \in \{5, 10, 15, 20\}$, we use each considered heuristic to choose the inside nodes or mergers, and report the best performance among all the mergers using the chosen inside nodes (or all the chosen mergers). We choose 100 inside nodes using each heuristics for choosing inside nodes, and choose 10 mergers using each heuristic for choosing mergers. We always use 50 outside nodes with most inside neighbors. We do this for one round (i.e., the budget $b = 1$).

We compare the best performance among all the possible IIMs (the 4950 pairs among the 100 chosen inside nodes) or IOMs (the 5000 pairs between the 100 chosen inside nodes and the 50 outside nodes with most inside neighbors) using the 100 inside nodes chosen by each heuristic. On all datasets, IP outperforms the random baseline, with an average superiority of 1.4 \times and 4.3 \times , for IOMs and IIMs, respectively. Overall, the performance of IP is better than that of IN. Hence, our proposed algorithm, BATMAN, employs the heuristic IP to choose the inside nodes. Below, for the heuristics choosing pairs, the inside nodes are chosen by IP.

We compare the best performance among the 10 IIMs/IOMs chosen by each heuristic. We also include the **best (BE)** performance among all possible mergers from the considered inside and outside nodes (i.e., the results of IP in Figure 26(b)), as a reference. In both cases, SE outperforms the other competitors consistently. Notably, in both bases, SE achieves results close (93.2% for IOMs and 96.6% for IIMs) to the best possible performance (BE). Thus, our proposed algorithm, BATMAN, uses the SE heuristic. See Figure 26 for the detailed comparison among the algorithmic choices.

H.2 Main experiments

Experimental settings. For each dataset, we conduct experiments for each $k \in \{5, 10, 15, 20\}$. We use $b = 10$, check 100 inside nodes and 50 outside nodes, and n_c , the number of pairs to check in each round, takes a value in $\{1, 2, 5, 10, 15, 20\}$. We conduct all the experiments on a machine with i9-10900K CPU and 64GB RAM. All algorithms are implemented in C++, and complied by G++ with O3 optimization.

In Tables 9 to 22, we report the full results of each dataset for all the considered parameters.

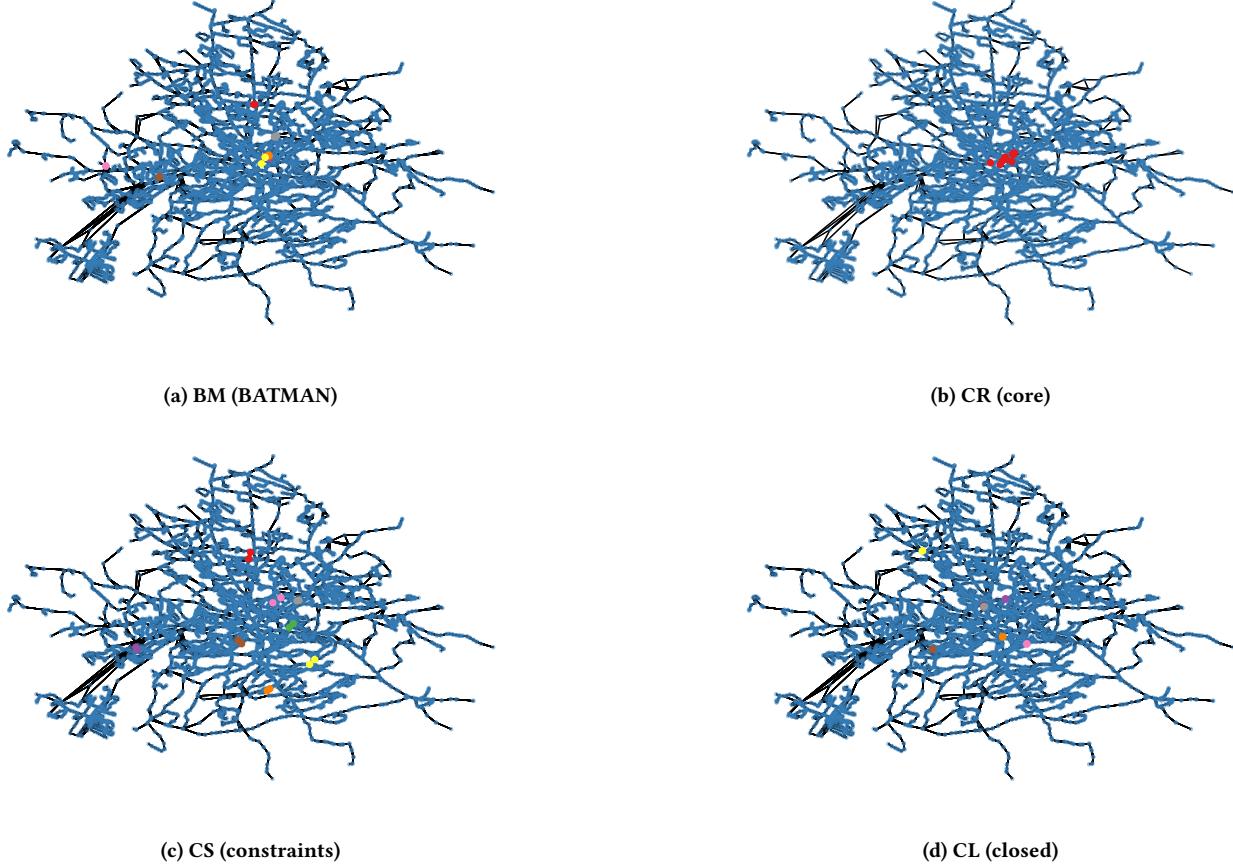


Figure 11: The results on the *rome* dataset in the additional experiments on real-world bus station datasets.

H.3 Sampling experiments

Experimental settings. For each dataset and each $k \in \{5, 10, 15, 20\}$, we randomly sample 10,000 **inside-inside (II)/inside-outside (IO)/outside-outside (OO)** mergers. For each experimental setting (dataset and k) and each case (II/IO/OO), we do five independent trials. We do this for one round (i.e., the budget $b = 1$).

In Figures 12 to 25, we report the full results of each dataset. In each plot, the x -axis represents the sampling size, and for each sampling size $1 \leq x \leq 10000$, the corresponding y value represents the best performance among the first x sampled mergers. We report the mean value and the standard deviation of five independent trials.

H.4 Heuristics experiments

Experimental settings. We study IOMs and IIMs separately. For each dataset and each $k \in \{5, 10, 15, 20\}$, we report the best performance among the mergers using the 100 inside nodes chosen by each node-heuristic (heuristic that chooses inside nodes), and report the best performance among the 10 mergers chosen by each pair-heuristic (heuristic that chooses pairs to be merged). The candidate mergers that each pair-heuristic chooses among are those

using the 100 inside nodes chosen by the node-heuristic IP which performs best among all the considered node heuristics.

In Tables 23 to 36, we report the full results of each dataset.

I Q1 & Q2: SUPPLEMENTARY RESULTS

In this section, we provide more results regarding Q1 & Q2 in Section 6.

I.1 Effectiveness of merging nodes (other random graph models)

Merging nodes is much more effective than adding edges in enhancing graph cohesiveness and robustness. For each measure, we report the change of it when we do 10 times of merging nodes or adding edges, over five independent trials, with the original value below the measure name. For each measure, the first result of merging nodes that is better than adding 10 edges is marked in bold and underlined.

The used random graph models are:

- the Erdős-Rényi model ($n = 50$ and $p = 0.1$),
- the Watts–Strogatz small-world model ($n = 50$, $k = 7$, and $p = 0.1$), and

		(a) The Erdős-Rényi model.									
measure	# operations	1	2	3	4	5	6	7	8	9	10
VB 87.3	merging	83.1	79.2	75.5	71.9	68.4	65.4	62.5	59.9	57.5	55.4
	adding	86.2	85.4	84.7	84.1	83.5	83.0	82.5	82.1	81.6	81.2
EB 25.3	merging	23.3	21.6	20.0	18.6	17.4	16.3	15.2	14.2	13.3	12.4
	adding	24.6	24.1	23.6	23.2	22.8	22.5	22.1	21.7	21.4	21.1
ER 834.8	merging	714.8	633.0	570.2	515.7	468.5	427.0	389.3	355.2	325.1	297.5
	adding	776.1	739.5	712.3	689.9	671.1	654.0	639.7	626.0	613.1	601.3
NC 2.7	merging	3.1	3.6	4.0	4.3	4.7	5.0	5.4	5.7	6.0	6.2
	adding	2.8	2.9	3.0	3.1	3.2	3.4	3.5	3.6	3.7	3.9
SG 2.3	merging	3.1	3.7	4.3	4.7	5.2	5.5	5.9	6.2	6.5	6.8
	adding	2.5	2.7	2.9	3.1	3.2	3.4	3.6	3.7	3.9	4.0

		(b) the Watts-Strogatz small-world model.									
measure	# operations	1	2	3	4	5	6	7	8	9	10
VB 72.58	merging	70.1	67.7	65.5	63.5	61.7	60.0	58.5	56.9	55.3	53.7
	adding	72.3	72.1	71.9	71.7	71.4	71.3	71.1	70.9	70.7	70.5
EB 10.8	merging	10.2	9.6	9.1	8.6	8.1	7.6	7.2	6.8	6.4	6.0
	adding	10.6	10.6	10.5	10.4	10.3	10.2	10.1	10.0	9.9	9.9
ER 369.0	merging	343.1	319.2	296.3	274.3	253.4	233.9	215.3	198.2	182.1	167.2
	adding	364.6	360.7	356.8	353.1	349.6	346.1	342.8	339.7	336.8	333.9
NC 7.5	merging	8.0	8.4	8.8	9.2	9.5	9.8	10.1	10.4	10.7	11.0
	adding	7.6	7.8	7.9	8.0	8.2	8.3	8.4	8.5	8.6	8.7
SG 2.3	merging	3.1	3.7	4.3	4.7	5.2	5.5	5.9	6.2	6.5	6.8
	adding	2.5	2.7	2.9	3.1	3.2	3.4	3.6	3.7	3.9	4.0

		(c) the Holme-Kim powerlaw-cluster model.									
measure	# operations	1	2	3	4	5	6	7	8	9	10
VB 77.5	merging	73.4	69.5	65.8	63.0	60.8	59.1	57.4	55.8	54.2	52.6
	adding	76.9	76.4	76.0	75.6	75.2	74.9	74.5	74.2	73.8	73.5
EB 10.6	merging	9.7	8.8	8.1	7.5	7.0	6.6	6.3	5.9	5.6	5.3
	adding	10.4	10.3	10.2	10.1	10.0	9.8	9.7	9.6	9.5	9.4
ER 300.2	merging	276.1	254.7	236.3	219.8	204.4	190.1	176.7	164.0	152.0	140.7
	adding	296.4	293.1	289.9	287.1	284.3	281.8	279.3	277.0	274.7	272.6
NC 6.5	merging	7.0	7.7	8.3	8.7	9.3	9.7	10.1	10.5	10.9	11.2
	adding	6.6	6.6	6.7	6.8	6.9	6.9	7.0	7.1	7.2	7.3
SG 1.7	merging	2.8	3.6	4.3	4.8	5.6	6.1	6.6	7.1	7.6	8.1
	adding	1.9	2.0	2.1	2.3	2.4	2.5	2.6	2.7	2.9	3.0

Table 8: Results of the effectiveness study using three different random graph models. For each measure, we report the change of it when we do 10 times of merging nodes or adding edges, over five independent trials, with the original value below the measure name. For each measure, the first result of merging nodes that is better than adding 10 edges is marked in bold.

- the Holme-Kim powerlaw-cluster model ($n = 50$, $m = 3$, and $p = 0.1$).

In Tables 8, we provide the results using the three different random graph models.

I.2 A case study on *relato*

We conduct a case study on *relato*, in order to check which nodes are merged together when we apply BATMAN, the proposed algorithm. In the *relato* dataset, each node represents a company (mainly in the IT field) and each edge represents some business-partner relationship between two companies.

We use BATMAN on the *relato* dataset with $(k, b, n_i, n_o, n_c) = (10, 100, 100, 50, 10)$. After $b = 100$ rounds, the size of $(k = 10)$ -truss increases from 89041 to 94944. Only 110 nodes participate in the 100 mergers. There are 7 groups of companies consisting of more than two companies are merged together:

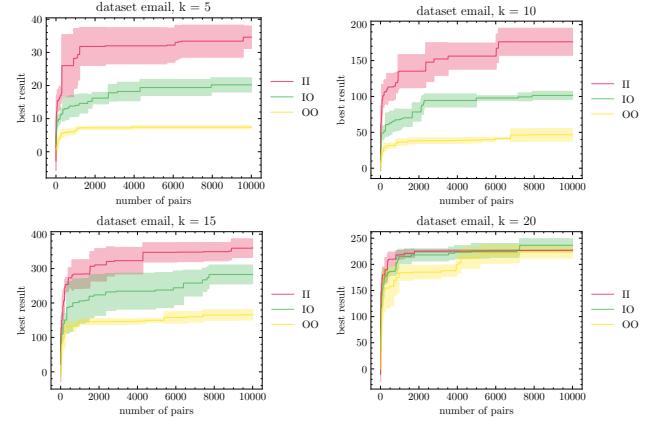


Figure 12: The full results of the sampling experiments on *email*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

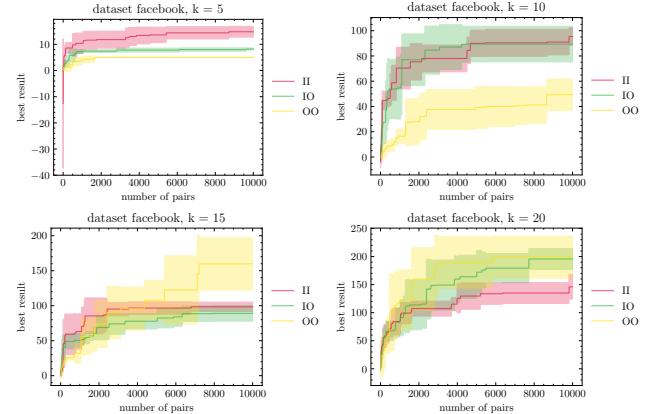


Figure 13: The full results of the sampling experiments on *facebook*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

- (size = 28) Apple Inc., Experian, Ameriprise, Peavey, REC Solar, Salesforce, Cubic, Kirin, RSA, Hanold Associates, Novatek, SKS, Interbrand, Hewlett Packard, Cemex, Beijing Enterprises, space150, NuGen, InSite, Wesco, Thomas & Betts, Bloom Energy, Ashland, Oshkosh Corporation, Azul Systems, ADC, BTG, Palantir; This group consists of a giant company (Apple Inc.) and 27 relatively small companies in various fields; This kind of mergers usually happen in real-world situations.
- (size = 22) SAP, CloudBees, DragonWave, Klocwork, SunTrust, Basho, Merry Maids, Signal, Xcerra, SGI, Veeva, SWIFT, Mitsui & Co, Hologic, Comdata, Martin Agency, Spectra Energy, Zensar Technologies, United Rentals, ThreatMetrix, IMG College, NAVTEQ; Similarly, this group consists of a giant company SAP and 21 relatively small companies in various fields.
- (size = 19) Oracle, Airwatch, Amylin Pharmaceuticals, E2open, BMO Financial Group, Cyber-Ark Software, UC Berkeley, MedImmune, Petco, Piper Aircraft, Wheel Pros, Aker Solutions, Swiss

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(190, 0.163258)	(588, 0.395745)	(1303, 0.452645)	(1527, 0.373819)
	2	(194, 0.297094)	(844, 0.630798)	(1376, 0.682457)	(1528, 0.434685)
	5	(200, 0.433535)	(953, 0.704848)	(1344, 0.748737)	(1445, 0.441133)
	10	(212, 0.536766)	(974, 0.868098)	(1343, 0.780103)	(1634, 0.503709)
	15	(215, 0.753264)	(947, 1.10923)	(1366, 1.040943)	(1634, 0.580716)
	20	(215, 0.939869)	(947, 1.321407)	(1278, 1.090379)	(1201, 0.571081)
EQ	1	N/A	N/A	N/A	N/A
	2	(194, 0.282179)	(844, 0.611814)	(1376, 0.671509)	(1528, 0.423497)
	5	(206, 0.349339)	(895, 0.661047)	(1304, 0.698516)	(1487, 0.456046)
	10	(213, 0.552619)	(896, 0.864929)	(1458, 0.778773)	(1124, 0.480327)
	15	(207, 0.691299)	(911, 1.08249)	(1406, 0.950659)	(1159, 0.512248)
	20	(221, 0.916641)	(974, 1.288419)	(1379, 1.12371)	(1159, 0.565624)
II	1	(190, 0.138759)	(908, 0.306687)	(1303, 0.411716)	(774, 0.333101)
	2	(204, 0.210995)	(908, 0.383291)	(1408, 0.472175)	(757, 0.343574)
	5	(211, 0.315223)	(896, 0.516913)	(1348, 0.546718)	(1178, 0.374554)
	10	(214, 0.484388)	(978, 0.681726)	(1435, 0.634548)	(754, 0.400293)
	15	(213, 0.687604)	(947, 0.908359)	(1448, 0.813922)	(820, 0.470702)
	20	(225, 0.839121)	(947, 1.11379)	(1256, 0.883392)	(820, 0.499459)
IO	1	(139, 0.145697)	(588, 0.35929)	(1417, 0.416889)	(1527, 0.36234)
	2	(138, 0.210198)	(636, 0.430234)	(1290, 0.491398)	(1238, 0.391185)
	5	(138, 0.303721)	(588, 0.563956)	(1411, 0.578495)	(1099, 0.407051)
	10	(145, 0.45026)	(697, 0.731267)	(1320, 0.722222)	(1634, 0.463949)
	15	(145, 0.620213)	(641, 0.958933)	(1452, 0.844707)	(1657, 0.520014)
	20	(145, 0.799318)	(643, 1.167234)	(1452, 0.991405)	(1608, 0.574232)
NT	1	(73, 1.316854)	(428, 1.131434)	(833, 0.762351)	(626, 0.377389)
	2	(72, 2.450766)	(504, 2.046731)	(1072, 1.327413)	(796, 0.54811)
	5	(81, 4.92864)	(465, 4.243704)	(1135, 2.765242)	(1140, 1.070902)
	10	(101, 7.167055)	(587, 6.50248)	(1089, 4.260347)	(1153, 1.559173)
	15	(109, 7.344824)	(590, 6.496875)	(1114, 4.242718)	(1129, 1.541246)
	20	(112, 7.488949)	(595, 6.70923)	(1287, 4.406364)	(1148, 1.600867)
NE	1	(93, 0.074646)	(263, 0.130554)	(394, 0.190378)	(1137, 0.209669)
	2	(93, 0.104544)	(263, 0.168751)	(394, 0.221188)	(1137, 0.21616)
	5	(93, 0.201608)	(263, 0.289639)	(394, 0.31141)	(1137, 0.249993)
	10	(129, 0.353582)	(433, 0.479871)	(888, 0.464709)	(1084, 0.300897)
	15	(93, 0.511208)	(263, 0.682291)	(394, 0.580377)	(1137, 0.359938)
	20	(93, 0.660565)	(263, 0.875819)	(394, 0.736546)	(1137, 0.39993)
RD	1	(35, 0.09364)	(42, 0.202609)	(174, 0.277664)	(351, 0.30549)
	2	(23, 0.124586)	(219, 0.251621)	(313, 0.308926)	(343, 0.307641)
	5	(39, 0.194363)	(264, 0.309078)	(619, 0.360495)	(376, 0.324811)
	10	(66, 0.391954)	(261, 0.557491)	(652, 0.526545)	(654, 0.404832)
	15	(56, 0.514945)	(479, 0.710561)	(660, 0.631788)	(749, 0.446687)
	20	(66, 0.695103)	(401, 0.935773)	(817, 0.815195)	(662, 0.491332)

Table 9: The full results of the main experiments on *email*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

Life, Torch, Brooks Brothers, RWE Group, Bell Mobility, Calabrio, Compal Electronics; Similar to the previous two groups, where the giant company is Oracle.

- (size = 12) Google, Databricks, SimpliVity, Azul, Henry Schein, Apple, Newport News, HCL Technologies America, HP, AES, Hewlett Packard Enterprise, Marshall Aerospace; Similarly, the giant company is Google.
- (size = 11) IBM, Nine Entertainment, Gores, OneSpot, TALX, Kaiser Aluminum, Coty, JC Penney, Scivantage, Ch2m Hill, State

bank of India; Similarly, the giant company is IBM; It is interesting to see State bank of India here.

- (size = 8) Facebook, VCE, SDL, Reval, MAXIM INTEGRATED, NEC, ThyssenKrupp, Commonwealth Bank of Australia; Similarly, the giant company is Facebook; Again, we see a foreign bank here (Commonwealth Bank of Australia).
- (size = 4) Amazon Web Services, Hewlett-Packard, Xignite, Gain-sight; This group is a bit different, both Amazon Web Services and

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(149, 0.506361)	(462, 0.948327)	(505, 1.133406)	(675, 1.171536)
	2	(152, 1.02818)	(620, 1.508909)	(505, 1.773869)	(1095, 1.66875)
	5	(158, 1.483856)	(640, 1.972026)	(631, 2.177783)	(1029, 2.040044)
	10	(163, 2.351635)	(675, 2.764291)	(678, 2.927482)	(1093, 2.70812)
	15	(177, 3.252112)	(675, 3.850927)	(636, 3.978673)	(1124, 3.711568)
	20	(172, 4.169596)	(678, 4.697119)	(678, 4.983967)	(1099, 4.534922)
EQ	1	N/A	N/A	N/A	N/A
	2	(152, 0.987369)	(620, 1.519664)	(505, 1.732861)	(1095, 1.669056)
	5	(158, 1.337278)	(652, 1.808965)	(513, 2.116939)	(1076, 1.996077)
	10	(164, 2.316492)	(675, 2.789305)	(617, 2.940854)	(1176, 2.733471)
	15	(164, 3.114176)	(682, 3.624242)	(617, 3.789719)	(1135, 3.48443)
	20	(171, 4.126044)	(678, 4.729749)	(678, 4.77951)	(1075, 4.421955)
II	1	(149, 0.463815)	(490, 0.796277)	(505, 1.034633)	(675, 1.115816)
	2	(149, 0.820292)	(556, 1.177575)	(612, 1.378685)	(662, 1.40776)
	5	(154, 1.353022)	(550, 1.731343)	(617, 1.878777)	(797, 1.890962)
	10	(171, 2.127139)	(627, 2.483068)	(678, 2.662917)	(877, 2.489292)
	15	(185, 3.103649)	(633, 3.549826)	(678, 3.586211)	(847, 3.418032)
	20	(185, 4.020618)	(616, 4.406482)	(678, 4.429359)	(896, 4.202753)
IO	1	(70, 0.485909)	(462, 0.868455)	(393, 1.137245)	(981, 1.218973)
	2	(84, 0.825116)	(531, 1.240481)	(393, 1.477159)	(1141, 1.487601)
	5	(89, 1.337836)	(653, 1.739939)	(387, 1.996417)	(1266, 1.920456)
	10	(89, 2.119162)	(654, 2.572274)	(352, 2.759679)	(1158, 2.568547)
	15	(90, 3.014986)	(672, 3.503828)	(411, 3.686824)	(1226, 3.380297)
	20	(90, 3.88141)	(657, 4.368629)	(349, 4.635049)	(1200, 4.090064)
NT	1	(25, 4.802132)	(185, 4.553131)	(165, 4.410222)	(212, 4.128859)
	2	(25, 8.323137)	(265, 8.190111)	(175, 7.638757)	(241, 6.939943)
	5	(20, 17.131463)	(276, 16.749262)	(232, 15.494895)	(264, 14.106341)
	10	(20, 27.410854)	(283, 27.03053)	(264, 24.976688)	(289, 22.699999)
	15	(24, 27.212796)	(273, 26.827285)	(296, 24.811599)	(301, 22.458058)
	20	(27, 27.993103)	(219, 27.642417)	(292, 25.626299)	(310, 23.035773)
NE	1	(9, 0.483263)	(0, 0.610785)	(0, 0.757779)	(0, 0.890101)
	2	(9, 0.629466)	(0, 0.787201)	(0, 0.907253)	(0, 1.020575)
	5	(9, 1.168563)	(0, 1.28736)	(0, 1.409836)	(0, 1.496966)
	10	(16, 1.890344)	(0, 2.065769)	(0, 2.246718)	(0, 2.158495)
	15	(9, 2.793643)	(0, 2.996425)	(0, 3.136047)	(0, 3.003932)
	20	(9, 3.632236)	(0, 3.903284)	(0, 3.904875)	(0, 3.791944)
RD	1	(-7, 0.478493)	(-1, 0.722488)	(-14, 0.951511)	(-157, 1.147531)
	2	(1, 0.645246)	(10, 0.931899)	(11, 1.160903)	(5, 1.293247)
	5	(1, 1.021822)	(23, 1.284559)	(9, 1.480335)	(38, 1.562286)
	10	(14, 1.992733)	(32, 2.313069)	(77, 2.51362)	(86, 2.488989)
	15	(12, 2.675058)	(34, 3.037993)	(36, 3.234321)	(46, 3.125762)
	20	(10, 3.640068)	(117, 4.074742)	(115, 4.248889)	(49, 4.021556)

Table 10: The full results of the main experiments on facebook. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

Hewlett-Packard giant companies, while Xignite and Gainsight are two relatively small data and market companies.

- (size = 2) Cisco, YourEncore; This is a combination of Cisco, which corresponds to the node with the highest node degree in the dataset, and a fairly small company YourEncore.
- (size = 2) Intel, Mimecast; This is also a combination between a large company and a small company.
- (size = 2) Amazon, Purdue University; In this dataset, Amazon Web Services and Amazon are two separate nodes; It is interesting

to see that Purdue University can help Amazon through a merger between these two entities.

Here, mergers represent business mergers and the size of each merger represents the scale of the corresponding business merger. We conclude that the main case is that a giant company gets merged with a large number of companies in various fields, which is also the most common case in real-world situations; it is fairly uncommon

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(2660, 7.633889)	(759, 4.289924)	(606, 3.520721)	(738, 3.435266)
	2	(2660, 11.775407)	(759, 5.206878)	(635, 3.837289)	(753, 3.84938)
	5	(2601, 13.85089)	(941, 5.995754)	(749, 4.323002)	(923, 3.913824)
	10	(2606, 17.86887)	(974, 7.244179)	(674, 4.795776)	(967, 4.370866)
	15	(2606, 22.462996)	(940, 8.762238)	(677, 5.658468)	(870, 5.10412)
	20	(2606, 26.806823)	(960, 10.224163)	(705, 6.411012)	(888, 5.76321)
EQ	1	N/A	N/A	N/A	N/A
	2	(2660, 11.641451)	(759, 5.143468)	(635, 3.952877)	(753, 3.644825)
	5	(2644, 13.210472)	(866, 5.820045)	(617, 4.187215)	(818, 3.821646)
	10	(2601, 17.397328)	(937, 7.049637)	(762, 4.711021)	(867, 4.349421)
	15	(2606, 21.439289)	(983, 8.432824)	(705, 5.630066)	(867, 5.113627)
	20	(2606, 26.201712)	(951, 10.179697)	(719, 6.563192)	(870, 5.757685)
II	1	(2660, 7.014586)	(759, 4.099205)	(507, 3.402992)	(535, 3.265395)
	2	(2644, 8.570758)	(866, 4.560648)	(518, 3.426621)	(538, 3.325384)
	5	(2601, 11.002977)	(906, 5.279612)	(617, 3.9874)	(564, 3.80108)
	10	(2606, 15.256073)	(936, 6.663428)	(682, 4.563828)	(571, 4.192416)
	15	(2606, 19.08951)	(954, 7.858194)	(655, 5.383557)	(651, 4.826672)
	20	(2606, 23.448812)	(944, 9.089351)	(682, 6.010336)	(622, 5.58813)
IO	1	(563, 7.180937)	(666, 4.328697)	(606, 3.496552)	(738, 3.371479)
	2	(618, 8.639106)	(414, 4.809814)	(689, 3.638419)	(912, 3.561971)
	5	(598, 10.839214)	(610, 5.62101)	(772, 4.184578)	(921, 3.808747)
	10	(604, 14.174811)	(595, 6.497019)	(796, 4.620337)	(923, 4.280902)
	15	(618, 18.288144)	(594, 7.977383)	(807, 5.632297)	(837, 5.012554)
	20	(612, 22.374368)	(640, 9.567872)	(807, 6.256049)	(837, 5.788372)
NT	1	(532, 7.518926)	(371, 4.91902)	(409, 4.087247)	(380, 3.841569)
	2	(530, 12.03291)	(287, 6.96289)	(443, 5.47965)	(419, 4.904233)
	5	(577, 22.764435)	(300, 12.170004)	(470, 8.886947)	(416, 7.767748)
	10	(613, 39.233378)	(332, 21.191422)	(523, 14.484679)	(445, 12.271197)
	15	(613, 38.239648)	(359, 18.641235)	(590, 13.312159)	(515, 11.42225)
	20	(613, 41.894058)	(411, 19.597006)	(629, 13.746571)	(542, 11.971981)
NE	1	(219, 2.906677)	(78, 2.720494)	(116, 2.747374)	(184, 2.789192)
	2	(219, 3.663202)	(78, 2.963913)	(116, 2.88292)	(184, 2.912347)
	5	(219, 5.929757)	(78, 3.733847)	(116, 3.341918)	(184, 3.293982)
	10	(316, 9.966511)	(170, 5.009166)	(191, 4.191981)	(346, 3.908425)
	15	(219, 13.353948)	(78, 6.313105)	(116, 4.734477)	(184, 4.500818)
	20	(219, 17.166411)	(78, 7.551934)	(116, 5.43375)	(184, 5.182752)
RD	1	(3, 4.93581)	(50, 3.678218)	(-68, 3.118525)	(-2, 3.084485)
	2	(9, 5.627936)	(2, 3.963403)	(115, 3.250154)	(54, 3.154718)
	5	(6, 7.074868)	(77, 4.426632)	(181, 3.55516)	(271, 3.391911)
	10	(27, 11.565817)	(111, 6.226378)	(165, 4.524875)	(335, 4.239357)
	15	(17, 14.382752)	(167, 6.908416)	(112, 5.006554)	(282, 4.594821)
	20	(52, 19.584597)	(129, 8.378082)	(238, 5.881316)	(348, 5.450063)

Table 11: The full results of the main experiments on *brightkite*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

for two relatively large companies to be merged together, but not totally impossible.

J TESTS FOR STATISTICAL SIGNIFICANCE

We conduct tests for the statistical significance of the empirical superiority of our proposed method BATMAN, especially over the variants. We generate 400 random graphs using the Watts-Strogatz small-world random graph model [31] with $n = 1000$ nodes, $d \in$

$\{10, 20, 30, 40\}$, and $p = 0.5$, where d is the number of nearest neighbors each node is joined with, and p is the probability of rewiring each edge. For each d value, we generate 100 random graphs. We apply the proposed method (BM) over the variants (EQ, II, and IO) to the 400 random graphs and obtain the 400 final truss sizes $(s_1^M, s_2^M, \dots, s_{400}^M)$ for each method $M \in \{\text{BM}, \text{EQ}, \text{II}, \text{IO}\}$. We apply the proposed method and the variants to maximize ($k = 5$)-truss with $b = 10$ mergers, $n_i = 100$ candidate inside nodes, $n_o = 50$ candidate outside nodes, and $n_c = 100$ pairs to check in each round.

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(511, 4.331406)	(2251, 5.445484)	(2293, 5.092611)	(4206, 5.190955)
	2	(511, 7.368159)	(2251, 7.866471)	(2486, 6.35433)	(4054, 6.097935)
	5	(544, 9.032235)	(2371, 8.981611)	(2590, 7.032122)	(4067, 6.437749)
	10	(526, 11.510036)	(2419, 11.134309)	(2668, 8.266657)	(3971, 6.609456)
	15	(552, 15.222338)	(2419, 14.188718)	(2738, 9.63729)	(4320, 7.399298)
	20	(556, 18.698971)	(2464, 16.68068)	(2738, 11.13879)	(4326, 7.999017)
EQ	1	N/A	N/A	N/A	N/A
	2	(511, 7.321111)	(2251, 7.676253)	(2486, 6.339604)	(4054, 6.03925)
	5	(554, 8.473394)	(2260, 8.626599)	(2509, 7.061166)	(4270, 6.04146)
	10	(561, 11.800697)	(2391, 11.205999)	(2673, 8.185404)	(3982, 6.583055)
	15	(526, 14.552206)	(2419, 13.813283)	(2668, 9.529306)	(4116, 7.424883)
	20	(526, 18.264272)	(2419, 16.406695)	(2738, 11.096055)	(4220, 7.607183)
II	1	(511, 3.870919)	(2251, 4.976404)	(2293, 4.810573)	(4206, 4.784768)
	2	(554, 5.19664)	(2260, 5.952932)	(2587, 5.369132)	(4178, 4.922209)
	5	(561, 7.048542)	(2391, 7.489798)	(2736, 5.913927)	(3951, 5.258277)
	10	(526, 10.603894)	(2419, 10.044706)	(2696, 7.635753)	(4324, 5.678987)
	15	(542, 13.509703)	(2419, 12.478366)	(2696, 8.871025)	(4297, 6.356354)
	20	(557, 17.138216)	(2464, 14.813504)	(2803, 10.09049)	(4297, 7.013217)
IO	1	(253, 3.62462)	(807, 5.110506)	(1941, 5.200724)	(2661, 5.386959)
	2	(267, 4.718587)	(838, 5.855962)	(1810, 5.676982)	(3195, 5.39356)
	5	(303, 6.34697)	(919, 7.50673)	(1756, 6.367354)	(3077, 5.783836)
	10	(298, 8.802689)	(946, 9.613448)	(1975, 7.534384)	(3732, 6.110118)
	15	(288, 11.671551)	(953, 11.961914)	(2002, 9.285292)	(3706, 6.872089)
	20	(278, 14.307012)	(933, 14.2551)	(2009, 10.585216)	(3873, 7.553795)
NT	1	(134, 10.738631)	(783, 7.442617)	(1648, 5.438749)	(3330, 4.277926)
	2	(142, 18.263571)	(826, 11.940534)	(1749, 7.689251)	(3470, 5.097733)
	5	(238, 36.935501)	(895, 21.992599)	(1924, 12.950219)	(3465, 7.417933)
	10	(242, 53.261851)	(875, 35.932222)	(1949, 21.180753)	(3888, 11.740368)
	15	(242, 58.864717)	(927, 34.930516)	(1959, 19.943574)	(3457, 10.411915)
	20	(244, 61.634726)	(928, 37.404548)	(1888, 21.231685)	(3629, 10.944137)
NE	1	(96, 1.883449)	(333, 2.670745)	(369, 3.156782)	(1273, 3.440137)
	2	(96, 2.50555)	(333, 3.098938)	(369, 3.420798)	(1273, 3.596383)
	5	(96, 3.918953)	(333, 4.518043)	(369, 4.203173)	(1273, 4.062471)
	10	(150, 6.508428)	(473, 6.744301)	(677, 5.571919)	(1337, 4.780549)
	15	(96, 9.349192)	(333, 9.112854)	(369, 6.814711)	(1273, 5.572274)
	20	(96, 11.92908)	(333, 11.537116)	(369, 8.080567)	(1273, 6.342468)
RD	1	(2, 2.086758)	(16, 3.947758)	(73, 4.294525)	(218, 4.13741)
	2	(7, 2.612672)	(103, 4.312337)	(215, 4.549018)	(1276, 4.31771)
	5	(2, 3.625463)	(131, 5.261591)	(594, 5.061123)	(1770, 4.634283)
	10	(10, 6.822703)	(300, 8.060247)	(772, 6.506863)	(1994, 5.512082)
	15	(12, 9.075867)	(284, 10.132303)	(825, 7.534221)	(2504, 5.927042)
	20	(23, 12.33712)	(360, 12.98275)	(845, 9.166356)	(2724, 6.702388)

Table 12: The full results of the main experiments on *enron*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

For each variant $T \in \{EQ, II, IO\}$, we compute the differences in the final truss size $D^T = (s_1^{BM} - s_1^T, s_2^{BM} - s_2^T, \dots, s_{400}^{BM} - s_{400}^T)$, and apply a one-tailed one-sample t -test to this population D^T with the null hypothesis H_0 : the mean value of $D_T \leq 0$. See Table 37 for the details (t -statistics and p -values) of the tests, where we observe that BATMAN outperforms the variants with statistical significance.

Note that the only difference between the proposed method BATMAN and the variants is the adaptive candidate-distribution

component. Intuitively, such a candidate-distribution component enhances the stability of our method since a wider range of candidate mergers is considered (as shown in Fig 6), which is validated by the cases when the variants (EQ, II, and IO) significantly perform worse than the proposed method. The candidate-distribution component is used mainly to improve the worst-case performance, and such a target is achieved as shown by the cases where the variants without such a component perform much worse than BATMAN with such a component.

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(1880, 10.71228)	(3239, 15.911818)	(1202, 13.304726)	(1309, 12.044654)
	2	(1880, 19.355977)	(3239, 21.000184)	(1423, 13.881173)	(1154, 12.016041)
	5	(1914, 25.084593)	(3210, 24.484485)	(1487, 14.438804)	(1230, 11.795518)
	10	(1897, 34.753535)	(3305, 31.267398)	(1668, 14.691768)	(1282, 11.975049)
	15	(1897, 46.433482)	(3305, 38.903798)	(1668, 15.861955)	(1419, 12.384224)
	20	(1897, 57.769216)	(3305, 45.976823)	(1735, 17.265859)	(1327, 12.78748)
EQ	1	N/A	N/A	N/A	N/A
	2	(1880, 18.920002)	(3239, 20.863257)	(1423, 13.529899)	(1154, 11.746036)
	5	(1865, 23.131213)	(3239, 23.511837)	(1727, 13.829663)	(1225, 11.902334)
	10	(1914, 34.562731)	(3305, 30.660947)	(1670, 14.639725)	(1289, 11.906666)
	15	(1897, 43.864536)	(3305, 37.108482)	(1573, 15.868421)	(1247, 12.38159)
	20	(1897, 55.831186)	(3305, 45.19937)	(1716, 17.017798)	(1377, 13.189475)
II	1	(1880, 9.928412)	(3239, 15.159316)	(1494, 12.704985)	(974, 11.709136)
	2	(1865, 14.008636)	(3239, 17.550303)	(1670, 13.081662)	(969, 11.524561)
	5	(1914, 20.544462)	(3305, 21.620116)	(1779, 13.34531)	(993, 12.081781)
	10	(1897, 31.228379)	(3305, 27.702994)	(1579, 14.490075)	(1037, 11.943175)
	15	(1897, 41.566679)	(3305, 34.459385)	(1591, 15.280399)	(1037, 12.185434)
	20	(1897, 51.697639)	(3305, 41.698888)	(1678, 16.12399)	(986, 12.690062)
IO	1	(446, 10.507681)	(1486, 15.723185)	(1202, 13.027482)	(1309, 11.839805)
	2	(487, 14.628238)	(1884, 18.08194)	(1472, 13.024111)	(1384, 11.741507)
	5	(464, 21.155257)	(1663, 22.202272)	(1597, 13.652119)	(1239, 11.755369)
	10	(486, 30.477748)	(1694, 27.723429)	(1651, 14.424961)	(1440, 11.875024)
	15	(500, 40.967156)	(2122, 34.057724)	(1651, 15.465994)	(1494, 12.399112)
	20	(500, 52.151216)	(1935, 42.542172)	(1797, 16.509463)	(1431, 12.686913)
NT	1	(185, 15.613714)	(1459, 14.178798)	(1011, 12.249921)	(773, 11.548451)
	2	(263, 25.216267)	(1498, 18.081384)	(838, 12.964268)	(776, 11.626199)
	5	(374, 49.21995)	(1889, 28.206763)	(1070, 14.966518)	(717, 12.372918)
	10	(491, 86.139788)	(2015, 51.823022)	(902, 23.426422)	(967, 17.74904)
	15	(463, 88.918728)	(2016, 47.755671)	(1401, 19.107009)	(996, 13.495021)
	20	(477, 99.358816)	(2015, 54.490456)	(1352, 19.638155)	(828, 13.64916)
NE	1	(12, 7.222833)	(0, 10.950282)	(20, 11.108081)	(428, 11.147636)
	2	(12, 9.055707)	(0, 12.183232)	(20, 11.29506)	(428, 11.223747)
	5	(12, 15.460855)	(0, 15.839832)	(20, 11.986463)	(428, 11.414039)
	10	(148, 25.873818)	(138, 22.346785)	(16, 12.837359)	(90, 11.690283)
	15	(12, 35.600186)	(0, 29.10255)	(20, 13.885689)	(428, 12.032483)
	20	(12, 46.373334)	(0, 34.820893)	(20, 14.854479)	(428, 12.373706)
RD	1	(0, 8.070257)	(0, 13.770937)	(0, 12.127693)	(88, 11.294168)
	2	(7, 9.89358)	(1, 15.024687)	(8, 12.293329)	(96, 11.355414)
	5	(1, 13.864795)	(0, 17.633792)	(3, 12.70848)	(273, 11.566043)
	10	(13, 26.739019)	(27, 25.426077)	(74, 13.92805)	(174, 11.872816)
	15	(25, 34.082305)	(204, 31.451282)	(138, 14.873115)	(401, 12.145065)
	20	(22, 46.099643)	(190, 38.863589)	(260, 16.012712)	(440, 12.447638)

Table 13: The full results of the main experiments on *hepph*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

It is definitely possible that in some scenarios, e.g., inside-inside mergers (IIMs) are consistently better than inside-outside mergers (IOMs). In such cases, BATMAN would increase the proportion of IIMs in the candidate mergers and can perform comparably to II (as we observe in the experimental results). We will further clarify this in the revised manuscript.

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(6502, 14.009076)	(7422, 14.679457)	(5780, 16.213886)	(4647, 15.751298)
	2	(6502, 22.344013)	(7422, 20.710244)	(6210, 19.447721)	(4934, 16.959438)
	5	(6637, 29.08989)	(7533, 25.647605)	(6681, 23.093492)	(5281, 17.980626)
	10	(6637, 40.696907)	(7533, 34.454681)	(6437, 26.957407)	(4942, 20.196656)
	15	(6637, 52.941491)	(7598, 44.065532)	(6437, 31.977536)	(4901, 22.260134)
	20	(6637, 64.841984)	(7598, 53.258305)	(6424, 37.207439)	(4818, 24.122128)
EQ	1	N/A	N/A	N/A	N/A
	2	(6502, 22.395535)	(7422, 20.669226)	(6210, 19.259234)	(4934, 17.382814)
	5	(6566, 26.95458)	(7383, 24.252107)	(6296, 21.941935)	(4760, 17.675723)
	10	(6637, 40.199188)	(7533, 34.247148)	(6258, 26.508954)	(5099, 20.104714)
	15	(6637, 50.061666)	(7533, 42.551266)	(6437, 31.300332)	(4868, 22.052029)
	20	(6637, 64.309491)	(7598, 53.074237)	(6437, 37.134823)	(5091, 24.256497)
II	1	(6502, 13.020298)	(7422, 14.011352)	(5780, 15.102904)	(4647, 15.371281)
	2	(6566, 17.572612)	(7383, 17.345369)	(6198, 16.74011)	(4505, 15.681111)
	5	(6637, 24.936549)	(7533, 22.787584)	(6498, 20.076558)	(4522, 16.746233)
	10	(6637, 36.842822)	(7598, 32.665774)	(6444, 24.675284)	(4522, 18.867458)
	15	(6637, 48.663534)	(7598, 41.064545)	(6601, 29.358617)	(4573, 20.916193)
	20	(6637, 60.973164)	(7598, 50.015291)	(6512, 34.012869)	(4567, 22.750986)
IO	1	(724, 12.581297)	(3232, 14.817602)	(5554, 16.286631)	(4633, 16.595315)
	2	(724, 16.798537)	(3302, 18.140384)	(5114, 17.774026)	(4654, 16.344556)
	5	(731, 23.663116)	(3363, 23.349389)	(5266, 20.962841)	(4906, 17.199864)
	10	(733, 33.785853)	(3429, 31.778832)	(5325, 25.200098)	(5058, 19.132696)
	15	(739, 46.322793)	(3509, 41.170955)	(5029, 30.604151)	(5198, 22.107354)
	20	(730, 56.169985)	(3338, 50.081576)	(5029, 35.782734)	(5165, 23.562279)
NT	1	(675, 24.167564)	(3003, 20.321986)	(3953, 17.597147)	(4306, 15.468346)
	2	(687, 39.750607)	(3061, 30.566673)	(4127, 23.683506)	(4380, 18.519992)
	5	(751, 78.414224)	(2945, 55.958911)	(4683, 38.628292)	(4697, 26.570887)
	10	(757, 127.379362)	(3214, 93.788406)	(4968, 65.970313)	(4836, 43.972665)
	15	(781, 131.775808)	(3350, 93.01329)	(4982, 60.001404)	(4954, 37.727108)
	20	(781, 142.780931)	(3180, 102.161783)	(5155, 64.922102)	(4925, 39.824714)
NE	1	(275, 7.370404)	(874, 10.173047)	(1324, 11.814216)	(491, 12.521565)
	2	(275, 9.37843)	(874, 11.923304)	(1324, 12.848376)	(491, 12.910815)
	5	(275, 15.794254)	(874, 17.237137)	(1324, 15.893406)	(491, 14.346997)
	10	(357, 26.582149)	(1279, 26.279711)	(1653, 21.400361)	(774, 16.745987)
	15	(275, 37.006772)	(874, 34.932669)	(1324, 25.826337)	(491, 19.091083)
	20	(275, 47.663125)	(874, 43.770436)	(1324, 30.826765)	(491, 21.424672)
RD	1	(15, 9.700132)	(53, 13.060758)	(251, 14.402305)	(333, 14.10091)
	2	(40, 11.891144)	(204, 14.965532)	(870, 15.397589)	(1703, 14.681986)
	5	(117, 16.139847)	(850, 18.510415)	(1505, 17.313076)	(2159, 15.575501)
	10	(118, 29.474276)	(908, 29.621664)	(1907, 23.615062)	(2917, 18.531446)
	15	(139, 37.421587)	(829, 36.361822)	(2366, 27.304857)	(2919, 19.847579)
	20	(135, 50.370877)	(1083, 46.770155)	(2524, 32.542717)	(3014, 22.616209)

Table 14: The full results of the main experiments on *epinions*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(1399, 7.953749)	(1806, 7.376449)	(1302, 8.720494)	(1193, 8.159371)
	2	(1399, 13.250584)	(1806, 11.222614)	(1859, 10.938343)	(1332, 9.812224)
	5	(1404, 16.606493)	(1821, 13.930346)	(1920, 13.012133)	(1400, 11.523038)
	10	(1435, 21.827349)	(1872, 18.522814)	(1766, 15.586085)	(1303, 13.396541)
	15	(1467, 28.570891)	(1872, 24.503237)	(1574, 20.184689)	(1312, 16.084518)
	20	(1467, 34.881025)	(1876, 29.810441)	(1662, 23.744555)	(1326, 18.67953)
EQ	1	N/A	N/A	N/A	N/A
	2	(1399, 13.108872)	(1806, 10.868075)	(1859, 10.784598)	(1332, 10.09833)
	5	(1407, 15.511006)	(1821, 12.975058)	(1905, 12.289557)	(1279, 11.266874)
	10	(1435, 22.817317)	(1834, 18.198366)	(1952, 15.547785)	(1329, 13.307694)
	15	(1435, 27.209416)	(1806, 23.174056)	(1820, 19.458249)	(1323, 16.130359)
	20	(1445, 35.29857)	(1872, 29.688204)	(1590, 23.624911)	(1336, 18.755269)
II	1	(1399, 7.347652)	(1806, 7.081155)	(1728, 7.402999)	(1193, 7.493956)
	2	(1407, 9.912281)	(1821, 9.027593)	(1649, 8.69315)	(1224, 8.236692)
	5	(1435, 13.420627)	(1821, 12.348087)	(1724, 10.95328)	(1272, 9.71036)
	10	(1445, 22.163684)	(1872, 17.916443)	(1645, 14.745402)	(1316, 12.522046)
	15	(1467, 25.681819)	(1876, 22.858826)	(1645, 18.432552)	(1324, 14.675866)
	20	(1467, 32.023161)	(1876, 28.102126)	(1709, 22.085486)	(1342, 17.1649)
IO	1	(344, 6.962134)	(734, 7.338649)	(1302, 8.383445)	(1078, 8.590261)
	2	(327, 9.15968)	(729, 9.38972)	(1293, 9.778422)	(1198, 9.346424)
	5	(373, 12.85268)	(811, 12.347115)	(1361, 11.731662)	(1247, 10.700964)
	10	(363, 17.719778)	(742, 16.525877)	(1436, 15.063044)	(1230, 12.823346)
	15	(352, 23.680394)	(809, 21.543639)	(1524, 19.290959)	(1133, 16.408843)
	20	(354, 29.439376)	(809, 26.545)	(1524, 22.326641)	(1295, 18.101346)
NT	1	(273, 17.075997)	(608, 12.542701)	(1186, 10.383663)	(974, 9.07537)
	2	(264, 29.446356)	(752, 20.999337)	(1090, 16.217407)	(1282, 13.211717)
	5	(278, 57.006561)	(774, 41.079849)	(1189, 30.793082)	(1438, 23.824083)
	10	(284, 95.389934)	(854, 69.297485)	(1335, 52.808828)	(1500, 41.780687)
	15	(284, 93.791758)	(879, 66.696617)	(1377, 50.005423)	(1513, 38.528761)
	20	(286, 97.925998)	(880, 71.977105)	(1389, 53.819828)	(1615, 40.939716)
NE	1	(166, 3.621131)	(533, 4.608468)	(656, 5.246678)	(878, 5.60472)
	2	(166, 4.709424)	(533, 5.584783)	(656, 5.87103)	(878, 6.110478)
	5	(166, 7.966682)	(533, 8.348632)	(656, 7.968597)	(878, 7.644592)
	10	(240, 13.274469)	(557, 13.109462)	(789, 11.400203)	(1073, 10.386018)
	15	(166, 18.819954)	(533, 18.035512)	(656, 14.98909)	(878, 12.661607)
	20	(166, 24.10867)	(533, 22.786093)	(656, 18.558893)	(878, 15.229757)
RD	1	(35, 4.760348)	(109, 6.44393)	(108, 7.065213)	(94, 7.287121)
	2	(42, 5.93591)	(147, 7.444798)	(305, 7.930402)	(347, 7.920304)
	5	(59, 7.972565)	(256, 9.458135)	(514, 9.439179)	(723, 8.825496)
	10	(82, 14.790045)	(261, 15.718018)	(653, 13.987663)	(807, 12.326806)
	15	(113, 18.800661)	(315, 18.947982)	(658, 16.484998)	(900, 13.760862)
	20	(108, 25.363641)	(340, 25.863805)	(566, 20.477926)	(806, 16.893236)

Table 15: The full results of the main experiments on *relato*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(18200, 20.066584)	(1525, 10.278747)	(1093, 9.362024)	(732, 8.8609)
	2	(18200, 30.911228)	(1631, 10.966134)	(1081, 9.880414)	(601, 8.934576)
	5	(17886, 37.100711)	(1769, 11.868651)	(1170, 10.131934)	(727, 9.160244)
	10	(17886, 47.582482)	(1815, 12.89966)	(1228, 10.930069)	(724, 9.874507)
	15	(17886, 59.291739)	(1634, 14.067592)	(1221, 12.1711)	(737, 10.838254)
	20	(17886, 71.037556)	(1634, 15.437461)	(1228, 13.320627)	(756, 11.759433)
EQ	1	N/A	N/A	N/A	N/A
	2	(18200, 31.367987)	(1631, 10.950923)	(1081, 9.631465)	(601, 8.898312)
	5	(18200, 35.502089)	(1777, 11.500195)	(1111, 10.015313)	(675, 9.169323)
	10	(17886, 47.871221)	(1618, 12.671074)	(1259, 10.910955)	(753, 9.767921)
	15	(17886, 57.456869)	(1815, 14.061171)	(1236, 11.888506)	(727, 10.982649)
	20	(17886, 71.596376)	(1634, 15.462811)	(1236, 12.980841)	(748, 11.536748)
II	1	(18200, 18.671986)	(1525, 9.813701)	(860, 9.093311)	(510, 8.63311)
	2	(18200, 23.279586)	(1698, 10.157537)	(862, 9.126792)	(537, 8.703942)
	5	(17886, 29.692579)	(1514, 10.802643)	(1083, 9.729115)	(570, 9.060151)
	10	(17886, 40.454163)	(1508, 12.003224)	(979, 10.872554)	(571, 9.79806)
	15	(17886, 52.286336)	(1586, 13.291565)	(1210, 11.522062)	(573, 10.516353)
	20	(17886, 62.886783)	(1537, 15.036476)	(1187, 12.597883)	(553, 11.15737)
IO	1	(3932, 19.492647)	(1398, 10.281061)	(1093, 9.361074)	(732, 8.828502)
	2	(3731, 23.300827)	(1554, 10.593149)	(1080, 9.357446)	(770, 9.018518)
	5	(3725, 29.477057)	(1542, 11.187701)	(1155, 9.99313)	(798, 9.323371)
	10	(3725, 39.646281)	(1567, 12.319004)	(1170, 10.702775)	(738, 9.72672)
	15	(3725, 50.968981)	(1443, 13.853482)	(1170, 11.932157)	(788, 10.766915)
	20	(3725, 61.487066)	(1499, 15.024706)	(1173, 12.932639)	(802, 11.435084)
NT	1	(3713, 15.530225)	(1048, 9.655758)	(777, 8.969255)	(495, 8.575624)
	2	(3838, 22.892191)	(1259, 11.657735)	(759, 10.429413)	(554, 9.588718)
	5	(3906, 42.422152)	(1374, 16.689715)	(717, 13.836441)	(652, 12.155182)
	10	(3751, 77.338054)	(1407, 25.30496)	(694, 20.404779)	(697, 17.628668)
	15	(3792, 77.208547)	(1445, 23.562442)	(815, 18.741732)	(736, 16.094562)
	20	(3792, 88.151785)	(1566, 24.759263)	(746, 19.567629)	(744, 16.827505)
NE	1	(719, 8.593711)	(610, 7.798539)	(375, 7.781)	(657, 7.841317)
	2	(719, 10.628516)	(610, 8.030336)	(375, 7.939821)	(657, 7.975527)
	5	(719, 16.820136)	(610, 8.773347)	(375, 8.54328)	(657, 8.386603)
	10	(1565, 27.174709)	(823, 9.991018)	(539, 9.470157)	(755, 9.209895)
	15	(719, 37.641159)	(610, 11.284551)	(375, 10.396356)	(657, 9.836579)
	20	(719, 47.937585)	(610, 12.481733)	(375, 11.305846)	(657, 10.579446)
RD	1	(26, 13.311316)	(53, 9.054355)	(45, 8.633299)	(56, 8.333059)
	2	(133, 15.341061)	(203, 9.317372)	(202, 8.856714)	(109, 8.478147)
	5	(1028, 19.566863)	(348, 9.954583)	(347, 9.187858)	(207, 8.818307)
	10	(1123, 32.759762)	(403, 11.688949)	(537, 10.80347)	(229, 9.698891)
	15	(1417, 40.345749)	(503, 12.404703)	(482, 11.102026)	(282, 10.363468)
	20	(879, 52.582941)	(518, 13.748626)	(502, 12.081631)	(336, 11.073083)

Table 16: The full results of the main experiments on slashdot. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(3078, 9.137355)	(6872, 23.077852)	(3216, 28.016744)	(1601, 31.364135)
	2	(3078, 15.931743)	(6872, 35.356373)	(3216, 34.519191)	(2532, 34.329742)
	5	(3078, 22.583234)	(7438, 46.274978)	(3717, 41.788776)	(2658, 39.501804)
	10	(2942, 33.817354)	(7410, 67.306994)	(3689, 52.481098)	(3066, 47.082249)
	15	(2942, 46.665795)	(7376, 89.841317)	(3760, 66.716082)	(3237, 55.769771)
	20	(2942, 57.968247)	(7487, 111.159931)	(3786, 79.026608)	(3260, 66.239228)
EQ	1	N/A	N/A	N/A	N/A
	2	(3078, 16.044073)	(6872, 35.108201)	(3216, 33.769681)	(2532, 33.472693)
	5	(3078, 20.728486)	(6977, 43.846302)	(3393, 39.830286)	(2466, 37.913488)
	10	(2942, 33.712189)	(7387, 65.986137)	(3705, 52.399195)	(3076, 47.011571)
	15	(2942, 43.736657)	(7410, 83.202126)	(3877, 64.562998)	(2991, 55.601181)
	20	(2942, 57.613948)	(7376, 109.244064)	(3694, 78.403743)	(3278, 66.61477)
II	1	(3078, 8.383145)	(6872, 21.465917)	(3216, 26.426679)	(2381, 29.018141)
	2	(3078, 13.202961)	(6977, 29.534145)	(3393, 30.959716)	(2445, 31.939913)
	5	(2942, 20.100729)	(7387, 41.521061)	(3705, 38.231511)	(2989, 36.817633)
	10	(2942, 31.696554)	(7376, 61.768423)	(3694, 51.451934)	(3278, 45.83766)
	15	(2942, 44.12425)	(7487, 82.986386)	(3786, 62.135343)	(3260, 53.732327)
	20	(2942, 56.538565)	(7487, 104.331364)	(3819, 75.139627)	(3260, 63.18732)
IO	1	(419, 9.144208)	(2292, 23.211475)	(2594, 27.894258)	(1601, 30.808521)
	2	(425, 13.140243)	(2464, 30.368479)	(2583, 31.953489)	(2105, 33.214502)
	5	(439, 20.094935)	(2403, 43.179821)	(2345, 38.853809)	(2286, 37.464285)
	10	(447, 30.193815)	(2458, 60.519335)	(2342, 50.122045)	(2339, 45.529253)
	15	(409, 41.46253)	(2458, 82.493371)	(2373, 64.779021)	(2129, 57.230257)
	20	(409, 52.13128)	(2435, 99.477884)	(2252, 76.79444)	(2127, 65.92331)
NT	1	(365, 25.020149)	(2017, 32.740586)	(2035, 32.208022)	(1695, 32.031095)
	2	(380, 41.080853)	(2017, 46.1374)	(2170, 41.072028)	(1874, 38.400952)
	5	(391, 81.016286)	(2240, 81.933398)	(2385, 63.984136)	(2047, 55.264581)
	10	(391, 133.656234)	(2378, 138.864884)	(2562, 105.211061)	(1881, 88.877944)
	15	(412, 136.512992)	(2381, 145.23193)	(2699, 103.561808)	(1920, 84.221371)
	20	(415, 147.788968)	(2519, 163.972721)	(2715, 115.110821)	(1935, 92.756394)
NE	1	(146, 8.032294)	(116, 20.242908)	(142, 23.902179)	(196, 25.865214)
	2	(146, 10.182026)	(116, 23.973655)	(142, 26.298077)	(196, 27.51254)
	5	(146, 16.575301)	(116, 35.209148)	(142, 33.098032)	(196, 32.672331)
	10	(241, 27.0287)	(492, 54.028916)	(581, 45.354465)	(388, 41.016239)
	15	(146, 38.067553)	(116, 72.680811)	(142, 55.833599)	(196, 49.720338)
	20	(146, 48.803032)	(116, 91.74155)	(142, 67.307069)	(196, 58.03318)
RD	1	(66, 8.209677)	(99, 21.610461)	(11, 26.46422)	(2, 28.758133)
	2	(56, 10.330515)	(121, 25.326023)	(122, 28.75007)	(67, 30.611359)
	5	(113, 14.586695)	(243, 32.846758)	(275, 33.366299)	(79, 33.651221)
	10	(174, 28.085976)	(332, 57.08309)	(297, 47.721802)	(137, 44.519811)
	15	(176, 36.432143)	(385, 70.733631)	(278, 56.186387)	(369, 50.485354)
	20	(185, 49.585751)	(625, 94.054384)	(429, 69.665654)	(338, 60.714386)

Table 17: The full results of the main experiments on syracuse. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(13584, 42.918833)	(6208, 37.637487)	(4041, 33.792637)	(3162, 31.89478)
	2	(13584, 67.972448)	(6208, 46.788955)	(4041, 37.013729)	(3455, 30.918941)
	5	(13575, 86.737187)	(6752, 57.790389)	(3823, 39.597373)	(3831, 32.481303)
	10	(13599, 119.925492)	(6726, 73.331638)	(4003, 45.031304)	(3831, 34.853393)
	15	(13599, 157.43262)	(6609, 92.880868)	(4032, 53.166402)	(3883, 37.619268)
	20	(13599, 194.824569)	(6768, 108.651844)	(3854, 59.337779)	(3883, 39.708514)
EQ	1	N/A	N/A	N/A	N/A
	2	(13584, 68.083202)	(6208, 46.390997)	(4041, 37.020529)	(3455, 30.987662)
	5	(13534, 81.61071)	(6677, 54.475581)	(3832, 39.988916)	(3885, 32.167532)
	10	(13575, 119.566966)	(6752, 73.701732)	(3896, 44.986285)	(3831, 34.600187)
	15	(13599, 152.357478)	(6707, 87.71731)	(4003, 52.026007)	(4122, 37.404323)
	20	(13599, 194.678042)	(6609, 111.009044)	(4032, 58.400001)	(3885, 40.207563)
II	1	(13584, 39.784097)	(6208, 35.406014)	(4041, 33.207643)	(3155, 30.838721)
	2	(13534, 53.167858)	(6677, 40.995739)	(3832, 33.814285)	(3050, 29.701283)
	5	(13575, 73.246415)	(6752, 51.249537)	(3896, 37.390047)	(3634, 31.497672)
	10	(13599, 106.95152)	(6609, 68.252545)	(4032, 44.07337)	(3231, 34.835858)
	15	(13599, 141.721769)	(6823, 85.316382)	(4032, 49.657822)	(3667, 35.776236)
	20	(13599, 177.902717)	(6823, 102.361751)	(3837, 55.366453)	(3667, 38.565606)
IO	1	(1180, 36.660868)	(1394, 34.937324)	(2062, 34.445082)	(3162, 32.699253)
	2	(1475, 49.889538)	(1401, 41.541774)	(2236, 34.817455)	(2604, 31.520299)
	5	(1475, 69.707858)	(1732, 51.103882)	(2536, 39.362832)	(3156, 32.904191)
	10	(1579, 99.048102)	(1743, 67.193724)	(2516, 43.292258)	(3219, 33.990909)
	15	(1556, 137.172782)	(1855, 87.183607)	(2375, 50.947088)	(3754, 37.311674)
	20	(1556, 172.371989)	(1698, 104.317272)	(2593, 56.394751)	(3859, 39.280838)
NT	1	(1577, 47.474689)	(2267, 35.795039)	(2200, 31.486294)	(2080, 28.850227)
	2	(2078, 74.812402)	(2283, 47.830792)	(2509, 36.376688)	(2133, 30.553379)
	5	(2188, 141.600142)	(2510, 76.865257)	(2699, 48.753152)	(2611, 35.919703)
	10	(2188, 230.624679)	(2515, 122.091206)	(2866, 72.79768)	(2667, 50.070531)
	15	(2035, 248.787257)	(2583, 127.339392)	(2809, 69.564209)	(2794, 44.51512)
	20	(2038, 280.394172)	(2566, 143.537395)	(2907, 74.808112)	(2757, 47.333103)
NE	1	(329, 21.047036)	(438, 24.928693)	(300, 26.296863)	(176, 26.673974)
	2	(329, 28.007259)	(438, 28.143183)	(300, 27.413802)	(176, 27.1633)
	5	(329, 45.821058)	(438, 37.908263)	(300, 30.91087)	(176, 28.71001)
	10	(574, 78.649986)	(702, 54.212184)	(917, 37.518702)	(439, 31.280486)
	15	(329, 110.359272)	(438, 70.787691)	(300, 42.592056)	(176, 34.128857)
	20	(329, 139.094946)	(438, 88.008542)	(300, 48.424284)	(176, 36.65451)
RD	1	(0, 28.390049)	(1, 31.537561)	(116, 30.250711)	(72, 28.421338)
	2	(6, 34.359303)	(75, 34.768408)	(9, 31.355581)	(493, 28.907215)
	5	(7, 46.523486)	(82, 41.281639)	(557, 33.869331)	(983, 29.929439)
	10	(21, 86.319679)	(158, 62.431811)	(428, 40.812047)	(989, 33.312353)
	15	(22, 108.426765)	(197, 73.398591)	(418, 44.921136)	(1245, 35.22558)
	20	(37, 146.327001)	(333, 92.227997)	(849, 51.733232)	(1628, 37.525779)

Table 18: The full results of the main experiments on *gowalla*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(2156, 17.250765)	(4639, 34.457268)	(4558, 50.228279)	(3348, 62.148935)
	2	(2156, 28.167689)	(4639, 51.946555)	(4558, 68.217324)	(3348, 79.537184)
	5	(2169, 39.767643)	(4828, 70.161341)	(4815, 88.415418)	(3600, 95.969509)
	10	(2209, 59.279109)	(4991, 104.108203)	(5175, 122.547026)	(4051, 127.31719)
	15	(2169, 80.004062)	(4991, 136.618916)	(5372, 159.781998)	(4617, 160.744555)
	20	(2181, 101.828541)	(4991, 172.50932)	(5372, 199.099672)	(4652, 189.724944)
EQ	1	N/A	N/A	N/A	N/A
	2	(2156, 28.096701)	(4639, 52.097043)	(4558, 68.635538)	(3348, 78.514117)
	5	(2169, 36.565866)	(4639, 64.364222)	(4720, 81.717093)	(3480, 90.34128)
	10	(2169, 59.696621)	(4656, 102.948816)	(5221, 121.257137)	(3851, 125.697294)
	15	(2209, 76.146773)	(4991, 129.479459)	(5175, 152.444814)	(3853, 152.075571)
	20	(2241, 99.555031)	(4991, 168.191196)	(5372, 196.499618)	(4051, 188.770842)
II	1	(2156, 15.817059)	(4639, 33.14413)	(4558, 48.953266)	(3348, 61.198515)
	2	(2169, 23.972167)	(4639, 45.983909)	(4720, 63.230449)	(3480, 73.074145)
	5	(2169, 36.35377)	(4656, 65.563888)	(5221, 84.586214)	(3851, 91.057811)
	10	(2241, 60.956302)	(4991, 98.907459)	(5372, 121.011317)	(4051, 122.64463)
	15	(2173, 77.744099)	(4991, 132.748716)	(5372, 158.346635)	(4652, 154.625164)
	20	(2340, 100.499021)	(4991, 168.338879)	(5372, 195.343929)	(4599, 185.53489)
IO	1	(585, 15.390641)	(1665, 33.078106)	(2011, 49.502759)	(2408, 62.065986)
	2	(588, 23.13957)	(1653, 44.773951)	(1850, 62.257377)	(2217, 73.81874)
	5	(584, 33.783655)	(1524, 62.245573)	(2022, 82.289333)	(2352, 91.868419)
	10	(574, 51.673385)	(1516, 90.121998)	(2173, 113.223071)	(2593, 119.825747)
	15	(580, 70.769903)	(1516, 121.838574)	(2169, 145.921236)	(2342, 150.819693)
	20	(582, 88.914154)	(1517, 149.250357)	(2095, 177.999262)	(2400, 180.99605)
NT	1	(476, 58.926643)	(1558, 64.954823)	(1915, 71.927896)	(1748, 77.397144)
	2	(545, 99.766209)	(1596, 101.376652)	(2067, 102.452668)	(2105, 100.385736)
	5	(589, 200.149661)	(1748, 191.729642)	(2353, 177.370222)	(2209, 159.370979)
	10	(589, 334.970778)	(1870, 319.254124)	(2590, 288.175274)	(2651, 250.22854)
	15	(603, 325.665102)	(1940, 317.799222)	(2678, 293.884366)	(2481, 256.016475)
	20	(607, 346.781807)	(1940, 349.985538)	(2790, 324.063543)	(2600, 285.047393)
NE	1	(309, 13.788904)	(216, 28.110017)	(190, 43.50338)	(113, 55.991149)
	2	(309, 17.40175)	(216, 33.555561)	(190, 49.712142)	(113, 62.696089)
	5	(309, 28.33458)	(216, 50.846547)	(190, 68.710448)	(113, 79.404135)
	10	(329, 47.057335)	(695, 79.058022)	(434, 102.175082)	(392, 108.591515)
	15	(309, 65.575881)	(216, 108.238793)	(190, 132.052858)	(113, 137.83236)
	20	(309, 83.175236)	(216, 137.192202)	(190, 163.539437)	(113, 166.78048)
RD	1	(0, 14.388134)	(0, 31.767857)	(0, 48.771893)	(20, 62.0023)
	2	(1, 18.149702)	(5, 37.393978)	(55, 55.114615)	(116, 67.78141)
	5	(2, 25.468295)	(12, 48.794362)	(66, 67.796522)	(26, 79.191418)
	10	(12, 47.65298)	(19, 85.499047)	(100, 108.456509)	(81, 115.707457)
	15	(13, 61.772505)	(46, 105.917868)	(104, 130.881632)	(270, 137.866356)
	20	(17, 84.348169)	(32, 140.406237)	(201, 168.625494)	(217, 172.471614)

Table 19: The full results of the main experiments on *twitter*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(5792, 32.22977)	(25273, 32.191346)	(12588, 38.239932)	(1068, 44.567476)
	2	(5406, 51.697749)	(25523, 48.436832)	(14172, 52.23859)	(993, 54.257068)
	5	(4919, 69.780089)	(25445, 66.888133)	(14820, 68.372353)	(1085, 66.340954)
	10	(6309, 105.836748)	(25574, 100.419997)	(14715, 101.620759)	(149189, 100.941726)
	15	(7288, 135.667083)	(25574, 132.941585)	(15129, 130.15164)	(149188, 124.15587)
	20	(7494, 170.204321)	(25672, 168.079383)	(17087, 165.087614)	(149176, 156.316675)
EQ	1	N/A	N/A	N/A	N/A
	2	(5406, 51.869004)	(25523, 48.432708)	(14172, 52.378278)	(993, 54.443817)
	5	(6741, 64.559927)	(25445, 60.918173)	(15350, 63.883068)	(1507, 63.765601)
	10	(6986, 101.530793)	(26344, 98.349682)	(14347, 101.317042)	(148768, 99.605697)
	15	(6992, 129.077426)	(25586, 126.146423)	(14669, 129.259442)	(1929, 107.28004)
	20	(7047, 168.380577)	(25586, 166.89998)	(18161, 161.833963)	(2362, 134.152945)
II	1	(5792, 30.174098)	(25273, 31.20066)	(12588, 37.185525)	(1887, 43.346764)
	2	(5792, 43.317874)	(25453, 43.40582)	(14225, 48.705107)	(2014, 50.856149)
	5	(5948, 63.137128)	(25465, 64.157894)	(19593, 65.183179)	(1495, 65.033025)
	10	(6385, 96.344019)	(25398, 97.407013)	(22378, 99.228458)	(119139, 88.479309)
	15	(6380, 129.943815)	(25692, 133.076064)	(17087, 127.554944)	(1930, 110.292199)
	20	(7090, 162.720907)	(25684, 167.108386)	(14767, 158.833008)	(119139, 135.438184)
IO	1	(4675, 29.207645)	(8406, 31.45901)	(13057, 36.980751)	(1068, 43.652551)
	2	(3658, 41.505399)	(20216, 42.844049)	(13337, 47.190616)	(1611, 51.012466)
	5	(5096, 59.139864)	(20619, 61.069498)	(14106, 63.702171)	(148768, 69.591456)
	10	(5091, 88.221983)	(8406, 91.471204)	(15410, 90.407044)	(149176, 94.918306)
	15	(5087, 119.665234)	(20720, 121.399374)	(17017, 117.710842)	(149188, 121.505219)
	20	(5264, 150.64538)	(8406, 155.302002)	(16280, 146.199353)	(149190, 147.688692)
NT	1	(1347, 85.938826)	(2766, 75.619618)	(11628, 66.656098)	(0, 59.299755)
	2	(1347, 121.679605)	(1690, 86.804609)	(8219, 82.54612)	(930, 81.934731)
	5	(1447, 206.327432)	(3680, 135.998975)	(4050, 110.855094)	(743, 124.472368)
	10	(1706, 314.501682)	(6224, 243.396519)	(10790, 230.891864)	(710, 190.639052)
	15	(1992, 390.721497)	(6225, 302.550387)	(27415, 267.379108)	(1697, 203.172245)
	20	(2121, 417.767729)	(6225, 342.265757)	(10872, 291.925606)	(847, 224.827101)
NE	1	(18, 23.140345)	(1605, 30.113673)	(15, 35.068699)	(0, 41.998658)
	2	(18, 29.351086)	(1605, 36.695096)	(15, 40.527794)	(0, 47.790536)
	5	(18, 48.526995)	(1605, 54.255724)	(15, 56.555583)	(0, 60.634051)
	10	(649, 80.425189)	(2740, 84.403829)	(15, 82.86777)	(0, 83.096467)
	15	(18, 109.869827)	(1605, 114.536891)	(15, 109.623321)	(0, 105.489152)
	20	(18, 140.707342)	(1605, 148.581678)	(15, 137.350974)	(0, 127.577672)
RD	1	(0, 24.450804)	(1, 31.233218)	(-11, 36.223551)	(-48, 41.869068)
	2	(0, 30.485499)	(2, 37.26326)	(0, 41.262597)	(25, 45.953401)
	5	(0, 42.50117)	(2, 49.479786)	(0, 51.695204)	(0, 54.720054)
	10	(1, 80.653146)	(1, 85.758497)	(17, 84.920984)	(9, 82.029572)
	15	(0, 102.999965)	(4, 108.338868)	(111, 104.065065)	(62, 98.417199)
	20	(1, 139.453087)	(10, 146.111188)	(3, 135.508874)	(25, 124.279489)

Table 20: The full results of the main experiments on stanford. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(34727, 96.815317)	(13903, 88.077645)	(7313, 79.695829)	(7933, 74.58187)
	2	(34727, 163.800204)	(13903, 101.660277)	(7358, 84.292301)	(8619, 75.905248)
	5	(35472, 210.667127)	(13712, 116.792159)	(7474, 90.294164)	(8364, 79.933056)
	10	(35483, 296.104172)	(13549, 141.735323)	(6773, 101.146874)	(9058, 85.75348)
	15	(35483, 379.843322)	(13549, 167.357328)	(6773, 112.453282)	(8655, 92.784557)
	20	(35483, 470.990498)	(13549, 193.362098)	(6773, 123.598105)	(8801, 99.734185)
EQ	1	N/A	N/A	N/A	N/A
	2	(34727, 162.052522)	(13903, 101.409627)	(7358, 84.475508)	(8619, 75.585538)
	5	(35057, 196.236969)	(13650, 112.538271)	(6727, 89.331761)	(8552, 79.23621)
	10	(35472, 291.866031)	(13549, 141.026136)	(6823, 100.507107)	(8891, 85.741432)
	15	(35483, 363.928129)	(13549, 163.431182)	(6761, 110.469261)	(8861, 91.665476)
	20	(35483, 467.948006)	(13549, 193.633128)	(6773, 123.979598)	(8967, 99.959984)
II	1	(34727, 91.237944)	(13903, 86.955482)	(7313, 79.52332)	(7933, 74.978634)
	2	(35057, 126.056803)	(13650, 96.646623)	(7361, 82.315159)	(8332, 75.345899)
	5	(35472, 175.673164)	(13549, 111.462079)	(7773, 89.368702)	(8364, 78.324265)
	10	(35483, 259.633478)	(13549, 135.378938)	(7078, 100.353824)	(8260, 86.502816)
	15	(35483, 347.251763)	(13549, 164.680759)	(7212, 112.436771)	(8294, 92.507716)
	20	(35442, 437.978777)	(13549, 188.161939)	(7212, 123.091461)	(8625, 99.140762)
IO	1	(5661, 89.014214)	(7077, 84.894401)	(5199, 83.422277)	(8145, 75.072201)
	2	(5479, 121.469296)	(7999, 92.596306)	(5880, 82.015747)	(8581, 73.922073)
	5	(5502, 167.262149)	(8450, 107.182301)	(5533, 91.749191)	(8682, 78.988132)
	10	(5520, 246.015041)	(8261, 129.768289)	(5921, 98.238124)	(8921, 84.525209)
	15	(5796, 333.021418)	(8261, 154.97954)	(5878, 113.489638)	(9052, 91.156537)
	20	(5796, 413.023137)	(7483, 177.147368)	(6062, 121.723597)	(9059, 98.88832)
NT	1	(3382, 103.389162)	(5724, 79.779424)	(2916, 74.370836)	(5361, 71.278303)
	2	(3859, 151.221716)	(6139, 92.829653)	(3099, 79.956918)	(5771, 72.274644)
	5	(4147, 270.213349)	(7012, 123.380309)	(3480, 93.147245)	(5908, 79.698591)
	10	(4179, 466.206248)	(7186, 185.257056)	(4242, 127.623758)	(6223, 97.864621)
	15	(4418, 517.675057)	(7995, 189.778014)	(4095, 122.447309)	(6103, 93.907214)
	20	(4498, 597.433512)	(7995, 216.309984)	(4110, 133.193601)	(6092, 100.494656)
NE	1	(139, 63.555624)	(595, 70.558657)	(371, 69.325438)	(3312, 68.733193)
	2	(139, 79.795067)	(595, 73.237299)	(371, 71.77277)	(3312, 71.158174)
	5	(139, 127.806795)	(595, 87.703684)	(371, 78.094574)	(3312, 75.459431)
	10	(506, 208.08281)	(1191, 111.561699)	(814, 90.293125)	(4510, 81.541043)
	15	(139, 290.510849)	(595, 136.452374)	(371, 99.434331)	(3312, 86.7421)
	20	(139, 371.255455)	(595, 161.463131)	(371, 110.222559)	(3312, 96.512952)
RD	1	(0, 64.169277)	(31, 81.413712)	(11, 76.234492)	(0, 70.665266)
	2	(121, 79.942868)	(250, 85.861006)	(545, 78.40747)	(3264, 72.485616)
	5	(140, 112.099574)	(462, 95.675977)	(635, 82.484578)	(4441, 75.22764)
	10	(37, 210.442689)	(697, 126.084332)	(1648, 95.920044)	(4859, 86.439338)
	15	(208, 271.602123)	(727, 141.739601)	(1807, 103.595016)	(4414, 88.147885)
	20	(197, 367.440253)	(1038, 172.267636)	(1989, 116.029053)	(4467, 95.818715)

Table 21: The full results of the main experiments on youtube. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

algorithm	n_c	$k = 5$	$k = 10$	$k = 15$	$k = 20$
BM	1	(69841, 184.947758)	(15865, 109.131924)	(6154, 125.102625)	(1973, 116.184614)
	2	(69841, 269.715721)	(16639, 156.3468)	(6487, 148.578884)	(2187, 136.397296)
	5	(70392, 333.232194)	(17250, 188.865752)	(6246, 172.212999)	(2336, 159.929409)
	10	(70184, 447.916909)	(17482, 263.394664)	(6246, 217.438621)	(2347, 198.744257)
	15	(70184, 559.750126)	(17482, 321.388279)	(5906, 267.584617)	(2299, 226.794706)
	20	(70184, 681.211728)	(17482, 392.373696)	(4467, 312.21477)	(2357, 269.380681)
EQ	1	N/A	N/A	N/A	N/A
	2	(69841, 270.361965)	(16639, 156.262437)	(6487, 149.014505)	(2187, 135.776025)
	5	(70289, 314.912242)	(17348, 175.188496)	(6508, 164.977351)	(2258, 150.627091)
	10	(69737, 444.372591)	(17482, 260.021884)	(6246, 215.380874)	(2342, 196.503726)
	15	(70184, 534.436413)	(17482, 310.851462)	(6246, 259.94217)	(2269, 222.622928)
	20	(70184, 670.27082)	(17482, 388.571692)	(5906, 317.5184)	(2257, 270.485292)
II	1	(69841, 179.602117)	(15865, 108.033626)	(6154, 124.044298)	(1973, 117.409596)
	2	(70289, 225.902993)	(16368, 134.849478)	(6237, 140.792145)	(1983, 132.427966)
	5	(69737, 285.287187)	(16158, 176.135094)	(6246, 169.299593)	(2217, 152.173485)
	10	(70184, 399.478906)	(16168, 246.115413)	(5906, 213.597636)	(2274, 193.611218)
	15	(70184, 526.861574)	(16216, 314.135171)	(4467, 258.704611)	(2329, 226.436015)
	20	(70184, 642.020926)	(16455, 375.499729)	(4620, 308.995176)	(2280, 262.455593)
IO	1	(2295, 115.656188)	(3453, 119.008855)	(3767, 112.720193)	(2021, 122.897098)
	2	(2208, 156.583749)	(3866, 141.172058)	(3597, 130.649717)	(2239, 140.72643)
	5	(1885, 217.510045)	(3806, 176.122573)	(3686, 156.575334)	(2248, 162.098473)
	10	(1710, 316.557355)	(3614, 232.498359)	(3711, 198.373934)	(2441, 194.604057)
	15	(818, 420.451756)	(3561, 293.285089)	(3590, 242.467404)	(2391, 232.632324)
	20	(710, 522.844322)	(3934, 349.380345)	(3640, 285.66923)	(2419, 267.734644)
NT	1	(3361, 154.290558)	(3616, 144.460126)	(2800, 137.186695)	(3539, 142.928956)
	2	(3271, 233.801483)	(3625, 188.470627)	(2784, 169.520057)	(3677, 168.023174)
	5	(3220, 439.065302)	(3811, 298.813281)	(3030, 251.15891)	(3874, 233.807214)
	10	(3494, 705.59448)	(3906, 473.593645)	(3109, 375.332806)	(3990, 342.674057)
	15	(3783, 804.524135)	(3967, 495.260585)	(3123, 389.246175)	(4078, 351.200492)
	20	(4008, 915.755914)	(4093, 554.369119)	(3147, 429.889279)	(4118, 386.540142)
NE	1	(682, 86.673368)	(648, 101.997978)	(1405, 109.180708)	(1452, 118.312682)
	2	(682, 106.525254)	(648, 118.811623)	(1405, 117.111967)	(1452, 125.247142)
	5	(682, 167.174618)	(648, 152.96458)	(1405, 142.058964)	(1452, 146.397903)
	10	(1344, 267.953063)	(947, 215.08867)	(1510, 181.730074)	(1819, 179.622152)
	15	(682, 371.214326)	(648, 279.089291)	(1405, 224.449483)	(1452, 216.560952)
	20	(682, 470.687674)	(648, 343.055558)	(1405, 265.112105)	(1452, 250.082739)
RD	1	(23, 89.965633)	(16, 102.005447)	(121, 112.442735)	(155, 122.04761)
	2	(51, 109.182972)	(366, 116.9455)	(966, 124.456227)	(779, 130.313734)
	5	(119, 150.46222)	(453, 142.383985)	(936, 139.941693)	(1387, 145.472471)
	10	(105, 275.956563)	(971, 226.873622)	(1564, 196.010391)	(1854, 196.772464)
	15	(159, 353.856702)	(996, 262.244178)	(1770, 232.126078)	(2292, 219.906269)
	20	(137, 469.818807)	(1264, 337.926391)	(2056, 290.098063)	(2632, 265.957037)

Table 22: The full results of the main experiments on *wikitalk*. For each algorithm, each considered k value, and each n_c , we report the performance (left) and the running time (right).

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	21	150	331	255
		IN	21	150	331	255
		RD	18.7	97.0	289.3	255.0
	pair-heuristics	SE	21	150	196	255
		NN	21	27	195	0
		RD	7.0	39.3	124.0	133.0
IIMs	node-heuristics	IP	46	192	385	227
		IN	47	200	381	227
		RD	27.7	127.3	320.7	227.0
	pair-heuristics	SE	30	175	255	227
		AE	18	150	291	206
		RD	14.0	78.0	181.7	129.0

Table 23: The full results of the heuristics experiments on *email*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	12	128	142	227
		IN	5	93	54	61
		RD	8.3	100.0	105.0	188.7
	pair-heuristics	SE	12	128	128	165
		NN	0	16	29	151
		RD	3.0	7.0	48.7	85.7
IIMs	node-heuristics	IP	21	100	151	171
		IN	7	17	24	18
		RD	12.3	73.7	92.3	116.7
	pair-heuristics	SE	17	46	151	169
		AE	15	74	50	72
		RD	7.0	22.3	36.0	14.3

Table 24: The full results of the heuristics experiments on *facebook*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	79	100	204	195
		IN	79	100	126	151
		RD	34.0	77.0	181.7	150.7
	pair-heuristics	SE	79	72	204	150
		NN	62	24	76	88
		RD	38.7	34.7	36.3	72.3
IIMs	node-heuristics	IP	452	161	238	181
		IN	452	161	72	37
		RD	54.0	81.7	159.3	113.7
	pair-heuristics	SE	452	161	136	71
		AE	452	161	136	58
		RD	130.3	51.3	36.3	85.0

Table 25: The full results of the heuristics experiments on *brightkite*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	104	145	461	1058
		IN	104	149	456	1058
		RD	13.3	126.0	413.0	971.0
	pair-heuristics	SE	104	122	364	806
		NN	10	82	101	666
		RD	10.7	53.3	162.7	659.7
IIMs	node-heuristics	IP	139	315	445	1224
		IN	97	315	429	1224
		RD	10.3	131.0	348.7	992.0
	pair-heuristics	SE	139	315	330	1115
		AE	57	315	330	1115
		RD	59.0	128.7	215.0	606.3

Table 26: The full results of the heuristics experiments on *enron*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	71	441	409	430
		IN	73	441	409	430
		RD	30.0	272.0	321.3	428.7
	pair-heuristics	SE	71	441	409	430
		NN	19	61	23	0
		RD	10.7	182.0	15.7	198.7
IIMs	node-heuristics	IP	267	531	328	407
		IN	227	556	231	115
		RD	34.0	156.7	219.7	399.3
	pair-heuristics	SE	267	531	312	108
		AE	267	531	328	115
		RD	22.3	108.0	22.3	21.0

Table 27: The full results of the heuristics experiments on *hepph*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	102	550	770	1069
		IN	102	550	770	1205
		RD	45.0	385.3	705.7	1081.3
	pair-heuristics	SE	102	550	770	1054
		NN	49	243	457	445
		RD	39.3	237.3	417.0	487.7
IIMs	node-heuristics	IP	1177	1113	1105	1067
		IN	1177	1113	1129	914
		RD	117.7	543.3	725.7	858.3
	pair-heuristics	SE	1177	1113	1105	958
		AE	1093	1113	1105	841
		RD	333.0	585.7	590.7	778.0

Table 28: The full results of the heuristics experiments on *epinions*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	62	115	246	260
		IN	62	115	246	311
		RD	28.3	82.0	185.3	264.0
	pair-heuristics	SE	59	89	246	186
		NN	20	57	90	165
		RD	33.3	76.0	81.0	113.3
IIMs	node-heuristics	IP	265	259	255	236
		IN	265	248	410	287
		RD	29.0	123.3	228.7	241.7
	pair-heuristics	SE	265	259	244	236
		AE	224	259	196	236
		RD	79.7	68.7	107.3	83.0

Table 29: The full results of the heuristics experiments on *relato*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	621	171	192	154
		IN	621	179	189	154
		RD	329.3	166.0	176.3	142.0
	pair-heuristics	SE	621	150	185	135
		NN	237	89	171	83
		RD	201.0	64.0	81.3	81.7
IIMs	node-heuristics	IP	2737	186	233	136
		IN	2737	177	233	151
		RD	495.7	136.3	173.3	124.7
	pair-heuristics	SE	2737	186	136	124
		AE	2737	153	233	136
		RD	918.3	65.7	120.3	68.7

Table 30: The full results of the heuristics experiments on *slashdot*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	67	259	287	859
		IN	67	259	287	722
		RD	32.7	171.3	212.0	790.3
	pair-heuristics	SE	67	240	249	859
		NN	57	259	249	216
		RD	11.3	65.3	72.7	72.0
IIMs	node-heuristics	IP	525	973	634	809
		IN	525	973	634	414
		RD	60.0	226.0	231.7	400.7
	pair-heuristics	SE	525	973	463	709
		AE	525	973	634	414
		RD	98.3	382.3	129.0	231.7

Table 31: The full results of the heuristics experiments on *syracuse*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	334	496	376	757
		IN	334	496	470	757
		RD	28.0	147.0	297.3	735.0
	pair-heuristics	SE	334	427	376	757
		NN	63	161	270	593
		RD	42.7	133.3	219.7	233.7
IIMs	node-heuristics	IP	2802	853	601	779
		IN	2802	853	601	676
		RD	28.3	144.0	319.3	574.3
	pair-heuristics	SE	2802	810	601	529
		AE	2802	853	437	529
		RD	460.7	399.0	207.0	273.7

Table 32: The full results of the heuristics experiments on *gowalla*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	214	345	443	540
		IN	214	345	520	483
		RD	12.0	84.7	193.0	340.0
	pair-heuristics	SE	214	345	306	327
		NN	147	251	197	167
		RD	18.3	84.0	147.7	113.7
IIMs	node-heuristics	IP	632	955	815	787
		IN	632	955	815	787
		RD	12.0	62.7	146.7	280.3
	pair-heuristics	SE	632	955	815	414
		AE	632	955	815	414
		RD	62.0	332.3	355.3	315.7

Table 33: The full results of the heuristics experiments on *twitter*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	1212	1853	4036	704
		IN	423	1853	4036	676
		RD	680.3	1363.7	2435.3	260.3
	pair-heuristics	SE	1212	1853	4036	704
		NN	0	1165	0	0
		RD	344.3	215.7	462.7	215.7
IIMs	node-heuristics	IP	2407	4879	4054	648
		IN	1454	2868	4054	144
		RD	244.7	896.0	607.7	55.0
	pair-heuristics	SE	2407	4879	4054	59
		AE	1826	2868	4054	14
		RD	317.3	1355.0	40.7	0.0

Table 34: The full results of the heuristics experiments on *stanford*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	854	995	973	2964
		IN	854	995	887	2964
		RD	394.7	669.0	942.7	2929.3
	pair-heuristics	SE	854	995	973	2809
		NN	54	512	724	1902
		RD	185.3	395.7	430.0	2151.3
IIMs	node-heuristics	IP	5363	2145	952	2853
		IN	5363	2145	821	2853
		RD	239.0	688.0	808.7	2771.0
	pair-heuristics	SE	5363	2145	952	2813
		AE	4447	2137	949	2813
		RD	923.7	814.3	458.7	1677.0

Table 35: The full results of the heuristics experiments on *youtube*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

IOMs/IIMs	nodes/pairs	heuristic	$k = 5$	$k = 10$	$k = 15$	$k = 20$
IOMs	node-heuristics	IP	778	1595	930	344
		IN	778	1595	930	687
		RD	84.0	701.7	553.0	551.7
	pair-heuristics	SE	117	1595	930	317
		NN	73	118	194	287
		RD	207.3	193.0	166.3	221.7
IIMs	node-heuristics	IP	30577	2687	1025	287
		IN	30577	2121	1144	954
		RD	121.0	687.3	555.0	615.3
	pair-heuristics	SE	30577	2687	1025	214
		AE	30577	1570	1025	210
		RD	2892.0	1123.0	204.3	204.7

Table 36: The full results of the heuristics experiments on *wikitalk*. For each algorithm, each considered k value, we report the performance of each considered node-heuristic and pair-heuristic.

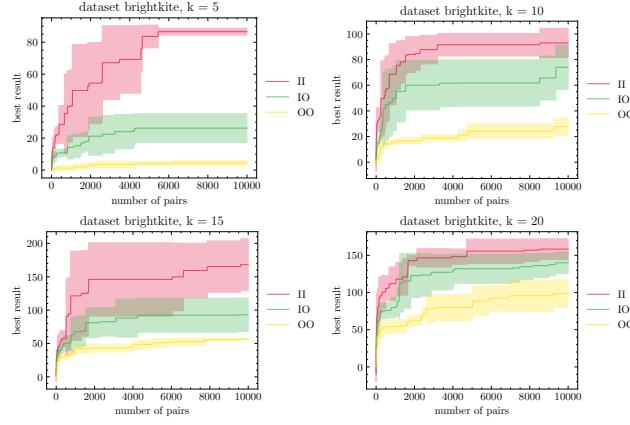


Figure 14: The full results of the sampling experiments on *brightkite*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

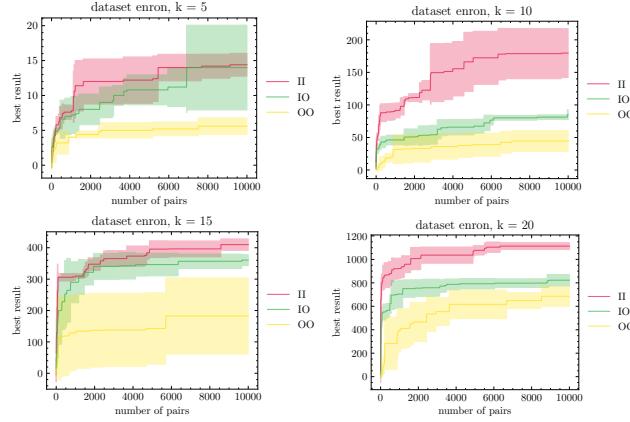


Figure 15: The full results of the sampling experiments on *enron*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

Table 37: The results of the tests for statistical significance. Smaller p-values indicate the superiority of BATMAN over the corresponding variant is more obvious. For each variant, the null hypothesis is clearly rejected with any reasonable significance level.

variant	t -statistic	p -value
EQ	3.28	5.65e-04
II	4.75	1.41e-06
IO	25.85	1.40e-87

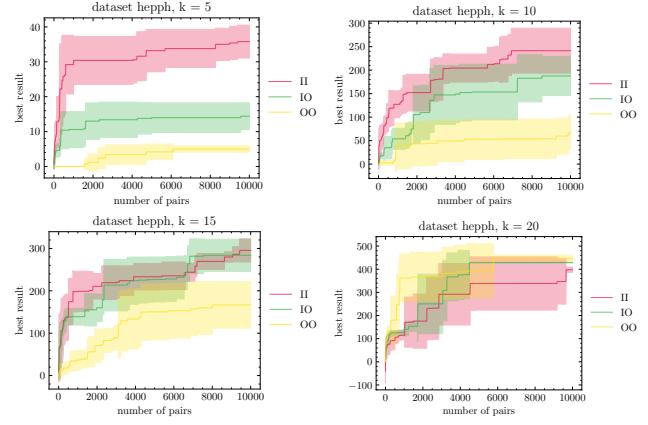


Figure 16: The full results of the sampling experiments on *hepph*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

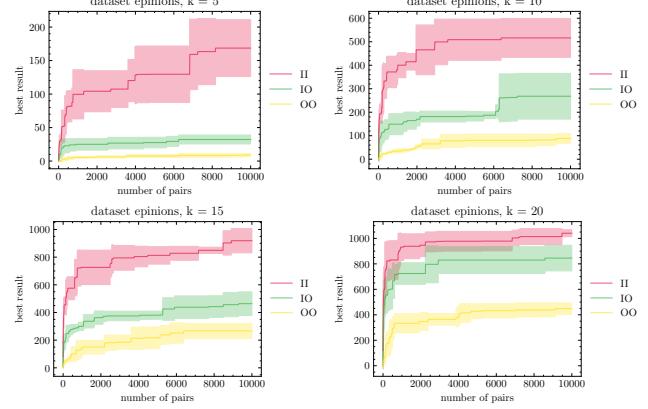


Figure 17: The full results of the sampling experiments on *opinions*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

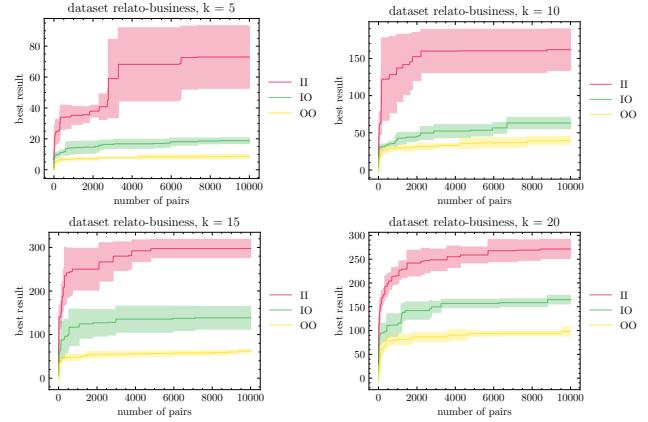


Figure 18: The full results of the sampling experiments on *relato*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

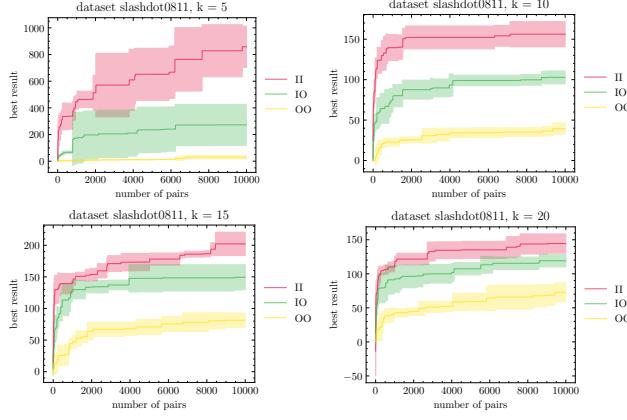


Figure 19: The full results of the sampling experiments on *slashdot*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

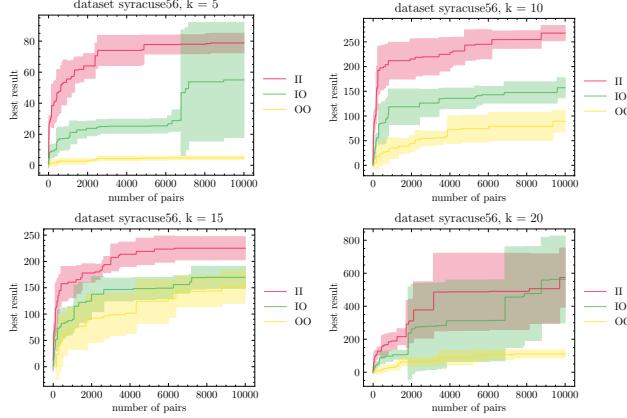


Figure 20: The full results of the sampling experiments on *syracuse*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

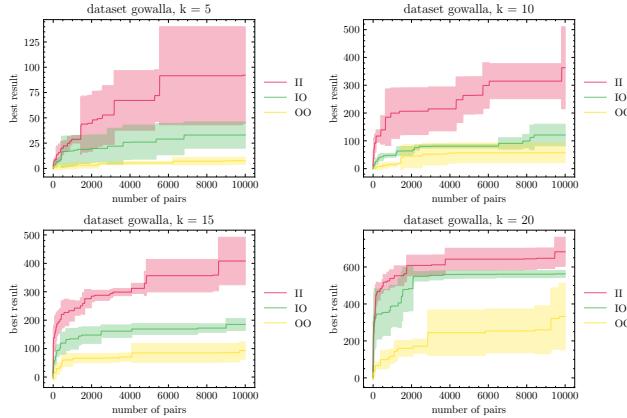


Figure 21: The full results of the sampling experiments on *gowalla*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

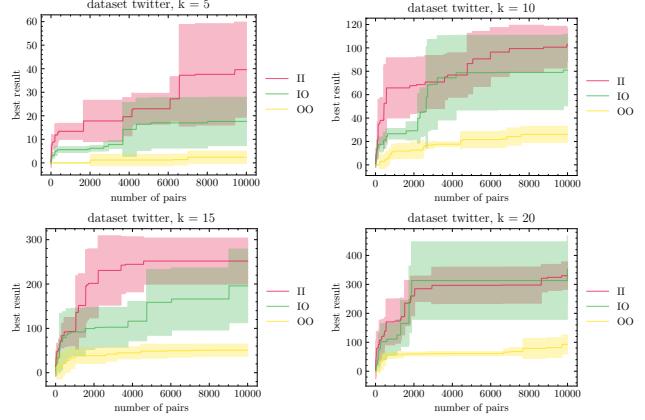


Figure 22: The full results of the sampling experiments on *twitter*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

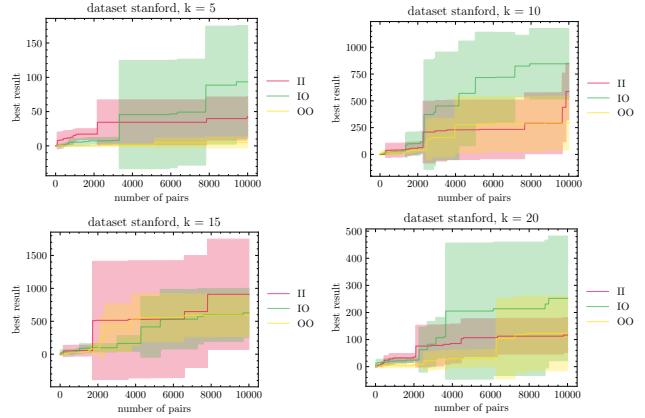


Figure 23: The full results of the sampling experiments on *stanford*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

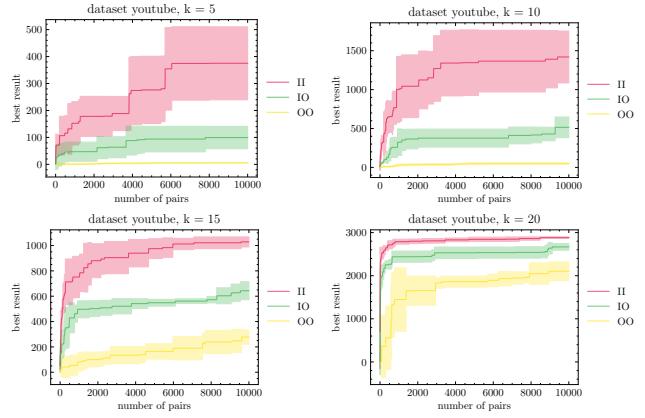


Figure 24: The full results of the sampling experiments on *youtube*. For each considered k value and each case, we report the best performance among the first x sampled mergers.

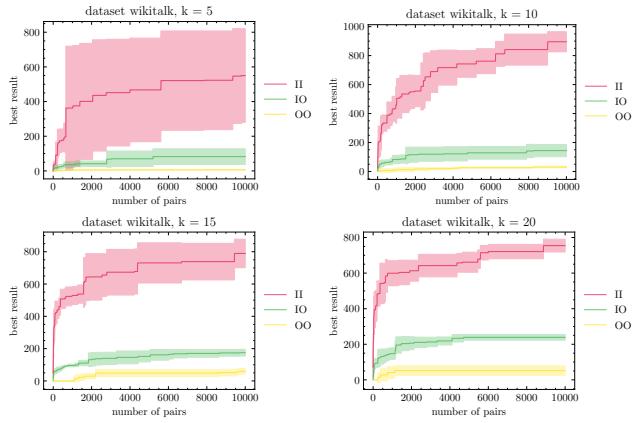


Figure 25: The full results of the sampling experiments on wikitalk. For each considered k value and each case, we report the best performance among the first x sampled mergers.

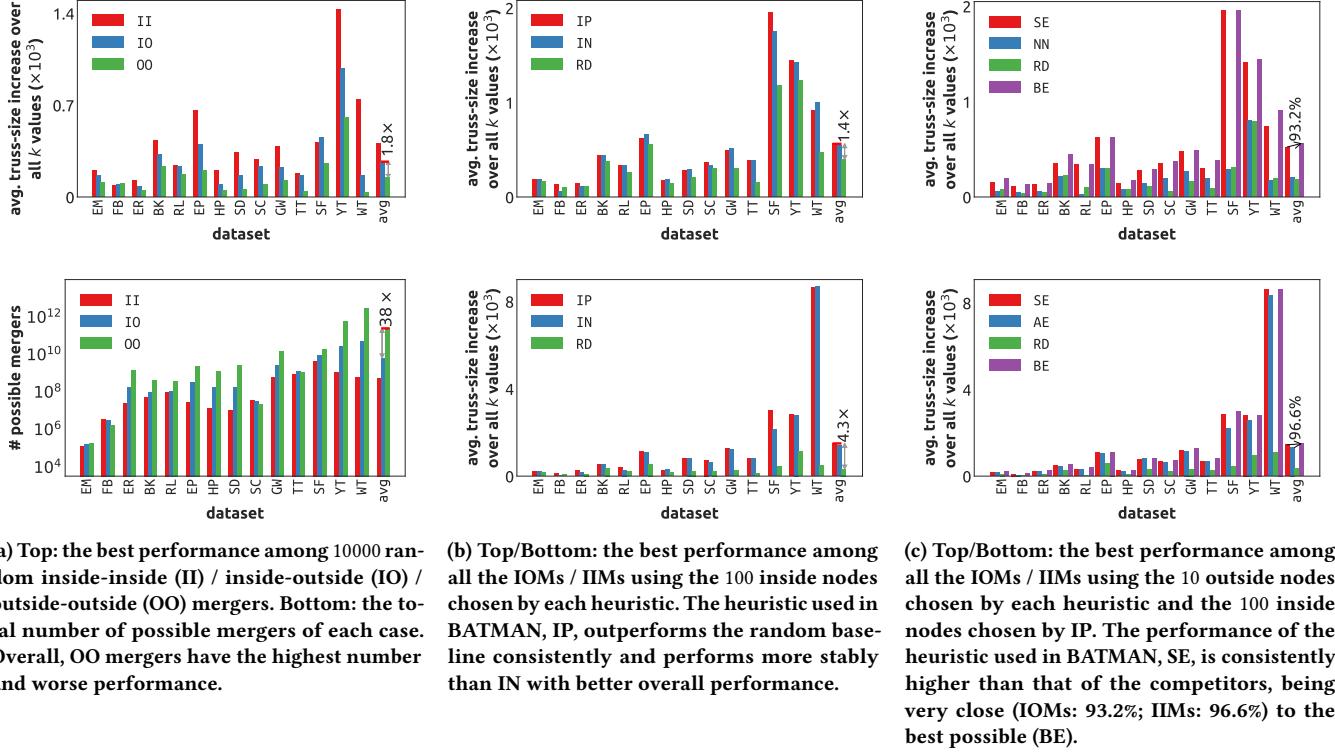


Figure 26: Empirical support of our algorithmic choices: excluding outside-outside mergers and the proposed heuristics.

REFERENCES

- [1] Vivek Singh Baghel and S Durga Bhavani. 2018. Multiple team formation using an evolutionary approach. In *IC3*.
- [2] Kshipra Bhawalkar, Jon Kleinberg, Kevin Lewi, Tim Roughgarden, and Aneesh Sharma. 2015. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics* 29, 3 (2015), 1452–1475.
- [3] Edward C Brewer and Terence L Holmes. 2016. Better communication= better teams: A communication exercise to improve team performance. *IEEE Transactions on Professional Communication* 59, 3 (2016), 288–298.
- [4] Manoel Campelo and Tatiane Fernandes Figueiredo. 2021. Integer programming approaches to the multiple team formation problem. *Computers & Operations Research* 133 (2021), 105354.
- [5] Antonio Candelieri, Bruno G Galuzzi, Ilaria Giordani, and Francesco Archetti. 2019. Vulnerability of public transportation networks against directed attacks and cascading failures. *Public Transport* 11 (2019), 27–49.
- [6] Yi Cao, Dandan Jiang, and Shan Wang. 2022. Optimization for feeder bus route model design with station transfer. *Sustainability* 14, 5 (2022), 2780.
- [7] Oded Cats and Panchamy Krishnakumari. 2020. Metropolitan rail network robustness. *Physica A: Statistical Mechanics and its Applications* 549 (2020), 124317.
- [8] Lijun Chang and Lu Qin. 2019. Cohesive subgraph computation over large sparse graphs. In *ICDE*.
- [9] Sybil Derrible and Christopher Kennedy. 2010. The complexity and robustness of metro networks. *Physica A: Statistical Mechanics and its Applications* 389, 17 (2010), 3678–3691.
- [10] Nurcan Durak, Ali Pinar, Tamara G Kolda, and C Seshadhri. 2012. Degree relations of triangles in real-world networks and graph models. In *CIKM*.
- [11] LK Eraso-Hernandez, AP Riascos, TM Michelitsch, and J Wang-Michelitsch. 2021. Evolution of transport under cumulative damage in metro systems. *arXiv preprint arXiv:2111.14979* (2021).
- [12] Ernesto Estrada. 2006. Network robustness to targeted attacks. The interplay of expandability and degree distribution. *The European Physical Journal B-Condensed Matter and Complex Systems* 52 (2006), 563–574.
- [13] Elisa Frutos Bernal and Angel Martín del Rey. 2019. Study of the structural and robustness characteristics of madrid metro network. *Sustainability* 11, 12 (2019), 3486.
- [14] Jimmy H Gutiérrez, César A Astudillo, Pablo Ballesteros-Pérez, Daniel Mora-Melià, and Alfredo Candia-Véjar. 2016. The multiple team formation problem using sociometry. *Computers & Operations Research* 75 (2016), 150–162.
- [15] Janet Hillier and Linda M Dunn-Jensen. 2013. Groups meet... teams improve: Building teams that learn. *Journal of Management Education* 37, 5 (2013), 704–733.
- [16] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*.
- [17] Jian Gang Jin, Loon Ching Tang, Lijun Sun, and Der-Horng Lee. 2014. Enhancing metro network resilience via localized integration with bus services. *Transportation Research Part E: Logistics and Transportation Review* 63 (2014), 17–30.
- [18] Ralihalizara Julliard, PJ Yun, and Ranaivoson Rado. 2015. Robustness of transportation networks: The case of Madagascar's road network. *International Journal of Advancements in Research & Technology* 4, 5 (2015), 1–4.
- [19] Rainer Kujala, Christoffer Weckström, Richard K Darst, Miloš N Mladenović, and Jari Saramäki. 2018. A collection of public transport network data sets for 25 cities. *Scientific data* 5, 1 (2018), 1–14.
- [20] Ming Li, Hongwei Wang, and Huashen Wang. 2019. Resilience assessment and optimization for urban rail transit networks: A case study of beijing subway network. *IEEE Access* 7 (2019), 71221–71234.
- [21] Jinpeng Liang, Jianjun Wu, Ziyou Gao, Huijun Sun, Xin Yang, and Hong K Lo. 2019. Bus transit network design with uncertainties on the basis of a metro network: A two-step model framework. *Transportation Research Part B: Methodological* 126 (2019), 115–138.
- [22] Oriol Lordan, Jose M Sallan, Pep Simó, and David Gonzalez-Prieto. 2014. Robustness of the air transport network. *Transportation Research Part E: Logistics and Transportation Review* 68 (2014), 155–163.
- [23] Gretchen A Macht, David A Nembehard, Jung Hyup Kim, and Ling Rothrock. 2014. Structural models of extraversion, communication, and team performance. *International Journal of Industrial Ergonomics* 44, 1 (2014), 82–91.
- [24] Mary Luz Mouronte and ROSA BENITO. 2012. Structural properties of urban bus and subway networks of Madrid. *Networks & Heterogeneous Media* 7, 3 (2012).
- [25] James G Oxley. 2006. *Matroid theory*.
- [26] Peng Peng, Shifen Cheng, Jinhai Chen, Mengdi Liao, Lin Wu, Xiliang Liu, and Feng Lu. 2018. A fine-grained perspective on the robustness of global cargo ship transportation networks. *Journal of Geographical Sciences* 28 (2018), 881–889.
- [27] Li Ping, Xiong Xing, Qiao Zhong-Liang, Yuan Gang-Qiang, Sun Xing, and Wang Bing-Hong. 2006. Topological properties of urban public traffic networks in Chinese top-ten biggest cities. *Chinese physics letters* 23, 12 (2006), 3384.
- [28] Sebastian Wandelt, Xing Shi, and Xiaoqian Sun. 2021. Estimation and improvement of transportation network robustness by exploiting communities. *Reliability Engineering & System Safety* 206 (2021), 107307.
- [29] Ji Wang and James Cheng. 2012. Trust decomposition in massive networks. *Proceedings of the VLDB Endowment* 5, 9 (2012), 812–823.
- [30] Xiangrong Wang, Yakup Koç, Sybil Derrible, Sk Nasir Ahmad, Willem JA Pino, and Robert E Kooy. 2017. Multi-criteria robustness analysis of metro networks. *Physica A: Statistical Mechanics and its Applications* 474 (2017), 19–31.
- [31] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of ‘small-world’ networks. *nature* 393, 6684 (1998), 440–442.
- [32] Kazuyuki Yamashita and Hiroyuki Ohsaki. 2021. Effective node resistance and its implication on network robustness. In *ICOIN*.
- [33] Changpeng Yang, Jianfeng Mao, Xiongwen Qian, and Peng Wei. 2018. Designing robust air transportation networks via minimizing total effective resistance. *IEEE Transactions on Intelligent Transportation Systems* 20, 6 (2018), 2353–2366.
- [34] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *KDD*.
- [35] Na Zhang, Xuefeng Guan, Jun Cao, Xinglei Wang, and Huayi Wu. 2019. Wavelet-HST: A wavelet-based higher-order spatio-temporal framework for urban traffic speed prediction. *IEEE Access* 7 (2019), 118446–118458.
- [36] Difeng Zhu, Guojiang Shen, Jingjing Chen, Wenfeng Zhou, and Xiangjie Kong. 2022. A higher-order motif-based spatiotemporal graph imputation approach for transportation networks. *Wireless Communications and Mobile Computing* 2022 (2022), 1–16.