A New Dynamic Algorithm for Densest Subhypergraphs

Suman K. Bera
Katana Graph
USA

Sayan Bhattacharya
University of Warwick
UK

Jayesh Choudhari
University of Warwick
UK

Prantar Ghosh
Dartmouth College
USA

ABSTRACT

Computing a dense subgraph is a fundamental problem in graph mining, with a diverse set of applications ranging from electronic commerce to community detection in social networks. In many of these applications, the underlying context is better modelled as a weighted hypergraph that keeps evolving with time.

This motivates the problem of maintaining the densest subhypergraph of a weighted hypergraph in a *dynamic setting*, where the input keeps changing via a sequence of updates (hyperedge insertions/deletions). Previously, the only known algorithm for this problem was due to Hu et al. [HWC17]. This algorithm worked only on unweighted hypergraphs, and had an approximation ratio of $(1+\epsilon)r^2$ and an update time of $O(\text{poly}(r, \log n))$, where r denotes the maximum rank of the input across all the updates.

We obtain a new algorithm for this problem, which works even when the input hypergraph is weighted. Our algorithm has a significantly improved (near-optimal) approximation ratio of $(1+\epsilon)$ that is independent of r, and a similar update time of $O(\text{poly}(r, \log n))$. It is the first $(1+\epsilon)$-approximation algorithm even for the special case of weighted simple graphs.

To complement our theoretical analysis, we perform experiments with our dynamic algorithm on large-scale, real-world data-sets. Our algorithm significantly outperforms the state of the art [HWC17] both in terms of accuracy and efficiency.

CCS CONCEPTS

• **Theory of computation → Dynamic graph algorithms**.

KEYWORDS

hypergraphs, densest subgraph, dynamic algorithms

1 INTRODUCTION

In the weighted densest subhypergraph (WDSH) problem, we are given a weighted hypergraph $G = (V, E, w)$ as input, where $w : E \to \mathbb{R}^+$ is a weight function. The *density* of any subset of vertices $U \subseteq V$ in G is defined as $\rho_G(U) := (\sum_{e \in E[U]} w_e)/|U|$, where $E[U]$ is the set of hyperedges induced by U on G. The goal is to find a subset of vertices $U \subseteq V$ in G with maximum density.

We consider the *dynamic* WDSH problem, where the input hypergraph G keeps changing via a sequence of *updates*. Each update either deletes a hyperedge from G, or inserts a new hyperedge e into G and specifies its weight w_e. In this setting, the *update time* of an algorithm refers to the time it takes to handle an update in G.

We want to design an algorithm that maintains a (near-optimal) densest subhypergraph in G with small update time.

The *rank of a hyperedge* e is the number of vertices incident on e. The *rank of a hypergraph* is the maximum rank among all its hyperedges. Let r denote an upper bound on the *rank* of of the input hypergraph throughout the sequence of updates. Let n be the number of nodes and m be an upper bound on the number of hyperedges over the sequence of updates. Our main result is summarized below.

Theorem 1.1. *(Informal) There is a randomized $(1+\epsilon)$-approximation algorithm for the dynamic WDSH problem with $O(r^2 \cdot \text{polylog}(n, m))$ worst case update time, for every sufficiently small constant $\epsilon > 0$.*

Note that a naive approach for this problem would be to run a static algorithm from scratch after every update, which leads to $\Omega(r \cdot (n+m))$ update time. As r is a small constant in most practical applications, the update time of our dynamic algorithm is *exponentially smaller* than the update time of this naive approach.

1.1 Perspective and Overview

Computing a dense subgraph is a fundamental primitive in graph mining [Gol84, LA07, BKV12, TBG+13, BBC+15, GT15, Tso15, BGP+20]. Over the course of past several decades, it has been found to be useful in a range of different contexts, such as *community detection* [CS12] and *piggybacking* on social networks [GJL+13], *spam detection* in the web [GKT05], *graph compression* [FM95], *expert team formation* [BGKV14], *real-time story identification* [AKSS12], *computational biology* [SHK+10] and *electronic commerce* [LA07]. There are three features that stand out from this diverse list of applications and motivate us to study the more general dynamic WDSH problem. (I) Real-world networks are often *dynamic*, in the sense that they change over time. (II) The underlying real-world context is often easier to capture by making the graph edges *weighted*. (III) It is often more beneficial to model the underlying network as a *hypergraph* rather than a standard graph.

In order to appreciate the significance of these three features, consider two concrete real-world examples.

Example 1: Real-time story identification. The wide popularity of social media yields overwhelming activity by millions of users at all times in the form of, say, tweets, status updates, or blog posts. These are often related to important current events or stories that one might seek to identify in real time. For instance, consider the recent Israel-Palestine conflict in May 2021. After the outbreak of the conflict, multiple incidents occurred in quick succession that are important to be promptly identified. An efficient technique for

real-time story identification is focusing on certain "entities" associated with a story, e.g., famous personalities, places, organizations, products, etc. They consistently appear together in the numerous posts on the related story. In the example of the Israel-Palestine conflict, countless online posts have turned up about the several events, many of which feature the same small set of entities, e.g., *Israel*, *Palestine*, *Hamas*, *Gaza*, *Sheikh Jarrah*, and *airstrike*, or subsets thereof. This correlation can be leveraged: the set of all possible real-world entities (which can be billions) represented by nodes, with an edge connecting each pair iff they appear together in a post, define a graph that changes dynamically over time; maintaining a dense subgraph of this network helps us to identify the group of most strongly-related entities (in the example above, this group might be {*Hamas*, *Gaza*, *Sheikh Jarrah*, *airstrike*}), and in turn, the trending story [AKSS12].

Note the significance of feature (I) here: the number of posts keeps growing rapidly, thus dynamically modifying the underlying graph. Further, a large number of posts gets deleted over time. This is often driven by the proliferation of *fake news* and its eventual removal upon detection. Also notice that feature (II) is crucial for this task. Every minute, millions of entities get mentioned in a small number of posts. The few entities in the story of interest, however, collectively appear in a *massive* number of posts. Therefore, to make them stand out, we can assign to the graph edges *weights* proportional to the number of posts they represent. Thus, the densest subgraph is induced by the union of the entities in the story. Finally, observe the importance of feature (III) in this context. For a post mentioning multiple entities, instead of adding an edge between each pair of them, we can simply include all of them in a single *hyperedge*. The standard graph formulation creates a clique among those nodes, which makes the density of the post proportional to the number of entities mentioned. This is inaccurate for several applications. In contrast, having a single hyperedge represent a post removes this bias. The task of real-time story identification thus reduces to precisely the dynamic WDSH problem.

Example 2: Trending Topics Identification. Consider the setting where we wish to identify a set of recently trendy topics in a website like Stack Overflow. We can model this scenario as a network where each node corresponds to a tag, and there is a hyperedge containing a set of nodes iff there is a post with the corresponding set of tags. The weight of a hyperedge represents the reach of a post, captured by, say, the number of responses it generates. The set of recently trendy topics will be given by the set of tags that form the densest subhypergraph in this network. The network is dynamic: posts are added very frequently and deletions are caused not only by their actual removal but also by our interest in only the ones that appeared (say) within the last few days.

Other applications of the WDSH problem include identifying a group of researchers with the most impact [HWC17] and analysing spectral properties of hypergraphs [CLTZ18].

Previous work. Starting with the work of Angel et al. [AKSS12], in recent years a sequence of papers have dealt with the densest subgraph problem in the dynamic setting. Epasto et al. [ELS15] considered a scenario where the input graph undergoes a sequence of adversarial edge insertions and random edge deletions, and designed a dynamic $(2+\epsilon)$-approximation algorithm with $O(\text{polylog } n)$ update time. In the standard (adversarial) fully dynamic setting, Bhattacharya et al. [BHNT15] gave a $(4+\epsilon)$-approximation algorithm with $O(\text{polylog } n)$ update time. This latter result was recently improved upon by Sawlani and Wang [SW20], who obtained a $(1+\epsilon)$-approximation algorithm with $O(\text{polylog } n)$ update time. All these results, however, hold only on *unweighted* simple graphs (i.e., hypergraphs with rank 2). Our algorithm, in contrast, works for *weighted* rank-r hypergraphs and is the first $(1+\epsilon)$-approximation algorithm with $O(\text{polylog } n)$ update time even for the special case of edge-weighted simple graphs.

For general rank-r hypergraphs, the only dynamic algorithm currently known in the literature was designed by Hu et al. [HWC17]: in the fully dynamic setting, their algorithm has an approximation ratio of $(1+\epsilon)r^2$ and an *amortized* update time of $O(\text{poly}(r, \log n))$. In direct contrast, as summarized in Theorem 1.1, our approximation ratio is near-optimal (and independent of r), and our update time guarantee holds in the *worst case*. Furthermore, our algorithm works even when the hyperedges in the input hypergraph have large weights in $[1, \text{poly}(r, n)]$, whereas the algorithm in [HWC17] needs to assume that the input hypergraph is either unweighted or has very small weights (in $[1, \text{poly}(r, \log n)]$).

Significance of our results. Given this background, let us now emphasize three aspects of our result as stated in Theorem 1.1.

First, the approximation ratio of our algorithm can be made arbitrarily close to 1, and in particular, it is independent of the rank r of the input hypergraph. For example, if $r = 3$, then [HWC17] can only guarantee that in the worst case, the objective value of the solution maintained by their algorithm is at least $(100/r^2)\% \simeq 11\%$ of the optimal objective value. In contrast, for *any* r, we can guarantee that the objective value of the solution maintained by our algorithm is always within $\simeq 99\%$ of the optimal objective value. In fact, since r can be, in theory, as large as n, the improvement over the approximation ratio is massive.

Second, the update time of our algorithm is $O(r^2 \cdot \text{polylog}(n, m))$. Note that *any* dynamic algorithm for this problem will necessarily have an update time of $\Omega(r)$, since it takes $\Theta(r)$ time to even specify an update. It is not surprising, therefore, that the update time of [HWC17] also had a polynomial dependency on r. Since r is a small constant in most practical applications, our update time is essentially $O(\text{polylog}(n, m))$ in these settings.

Third, our dynamic algorithm works for weighted graphs, which, as noted above, are crucial for applications. Throughout the rest of this paper, we assume that the weight of every hyperedge is a positive integer. This is without loss of generality: if the weights are positive real numbers, then we can scale them appropriately and round them to integers without affecting the approximation factor (see full version [BBCG21] for details). Finally, if the weights of the hyperedges are known to be integers in the range $[1, W]$, then a naive approach would be to make w_e copies of every hyperedge e when it gets inserted, and maintain a near-optimal solution in the resulting unweighted hypergraph. This, however, leads to an update time of $\Theta(W)$. This is prohibitive when W is large. In contrast, our algorithm has polylogarithmic update time for any W.

Overview of Techniques. We obtain the result stated in Theorem 1.1 in two major steps. First, we use random weight scaling to reduce

the weighted version of the problem to the unweighted case, while incurring only a small polylogarithmic overhead in update time (Section 3). Next, to solve the unweighted version, we extend the techniques of [SW20] to handle any general hypergraph (Section 4). Our analysis shows that the approximation factor achieved is $1+\epsilon$ for hypergraphs of any rank r and in particular, *does not* grow with r. See Section 1.2 of the full version of our paper [BBCG21] for a detailed overview of our techniques.

Overview of Experimental Evaluations. We conduct extensive experiments to demonstrate the effectiveness of our algorithm in both fully dynamic and insertion-only settings with weighted and unweighted hypergraphs. We test our algorithm on several real-world temporal hypergraph datasets. For the unweighted case, in both the insertion-only and fully dynamic settings, our algorithm significantly outperforms the state of the art of [HWC17] both in terms of accuracy and efficiency. In comparison against an LP solver for computing the exact solution, our algorithm shows massive speed-up while incurring less than a few percentage points of relative error. See Section 5 of this paper (and Sections 1.3 and 5 of the full version [BBCG21]) for a detailed account of our experimental results.

2 PRELIMINARIES AND NOTATIONS

Let us fix the notations that we use throughout the paper. Our input weighted hypergraph is always a rank-r hypergraph denoted by $G=(V,E,w)$, where $w: E \to \mathbb{N}$ is a weight function. We denote the number of vertices $|V|$ and hyperedges $|E|$ (or an upper bound on it) by n and m respectively. The maximum weight of a hyperedge in G is given by $w_{\max}(G) := \max_{e\in E} w_e$. The *multiplicity* of an edge in a multi-hypergraph is its number of copies in the hypergraph. For a subset of nodes $U \subseteq V$, denote its density in G by $\rho_G(U) := (\sum_{e\in E[U]} w_e)/|U|$, where $E[U]$ is the set of hyperedges induced by U on G. If the hypergraph is unweighted, then the density of U is simply $\rho_G(U) = |E[U]|/|U|$. We denote the maximum density of G by $\rho^*(G) := \max_{U\subseteq V} \rho_G(U)$. We drop the argument G from each of the above when the hypergraph is clear from the context.

We use the shorthands WDSH and UDSH for weighted and unweighted densest subhypergraph respectively. For the dynamic WDSH and UDSH problems, we get two types of queries: (a) max-density query, which asks the value of the maximum density over all subsets of nodes of the hypergraph, and (b) densest-subset query, which asks for a subset of nodes with the maximum density. We say an algorithm *maintains* an α-approximation ($\alpha > 1$) to either of these problems if it answers every max-density query with a value that lies in $[\rho^*/\alpha, \rho^*]$ and every densest-subset query with a subset of nodes whose density lies in $[\rho^*/\alpha, \rho^*]$.

Given any weighted hypergraph G, we denote its *unweighted multi-hypergraph version* by G^{unw}, which is obtained by replacing each edge e having weight w_e by w_e many unweighted copies of e. Note that G and G^{unw} are equivalent in terms of subset densities.

We say that a statement holds *whp* (with high probability) if it holds with probability at least $1 - 1/\text{poly}(n)$.

We use the following version of the Chernoff bound.

Fact 2.1. *(Chernoff bound) Let X be a sum of mutually independent indicator random variables. Let μ and δ be real numbers such that $\mathbb{E}[X] \le \mu$ and $0 \le \delta \le 1$. Then,* $\Pr[|X-\mu| \ge \delta\mu] \le \exp\left(-\mu\delta^2/3\right)$.

3 REDUCTION TO UNWEIGHTED CASE

In this section, we show that we can use an algorithm for the dynamic UDSH problem to obtain one for the dynamic WDSH problem while incurring only a small increase in the update and query times.

3.1 Weight Scaling

Given a weighted hypergraph, we want to scale down the weights to make the max-weight small and simultaneously scale down the max-density by a known factor so that we can retrieve the original density value from the scaled one. Since we want to reduce the problem to the unweighted case, we work with the unweighted multi-hypergraph versions (see Section 2) of the weighted hypergraphs in question. Thus, the maximum edge-weight would correspond to the max-multiplicity of an edge in the unweighted version. Informally, given a weighted hypergraph G on n vertices, we want to obtain an unweighted multi-hypergraph H such that (a) maximum multiplicity of an egde in H is roughly $O(\log n)$ and (b) given $\rho^*(H)$, we can easily obtain an approximate value of $\rho^*(G)$. We achieve these in Lemmas 3.1 and 3.2 respectively.

Given any weighted hypergraph G, we define G_q as the random hypergraph obtained by independently sampling each hyperedge of G^{unw} with probability q.

For a parameter $\widetilde{\rho}$, define $q(\widetilde{\rho}) := \min\left\{c\epsilon^{-2}\cdot\frac{\log n}{\widetilde{\rho}}, 1\right\}$ for some large constant c and an error parameter $\epsilon > 0$.

Our desired multi-hypergraph H will be given by $G_{q(\widetilde{\rho})}$ for some appropriate $\widetilde{\rho}$. The following lemma (proof in the full version [BBCG21]) shows that the max-multiplicity of H is indeed small.

Lemma 3.1. *For $\widetilde{\rho} \ge w_{\max}(G)/r$, let $H = G_{q(\widetilde{\rho})}$. Then, maximum multiplicity of an edge in H is $O(r\epsilon^{-2}\log n)$ whp.*

At the same time, we also need to ensure that we can retrieve the max-density and a densest subset of G from that of H. The next lemma, which follows directly from Theorem 4 of [MPP+15], handles this.

Lemma 3.2. *Given a weighted hypergraph $G=(V,E,w)$, let $H = G_{q(\widetilde{\rho})}$ for a parameter $\widetilde{\rho}$. Then, following hold simultaneously whp:*
(i) $\forall U \subseteq V : \rho_G(U) \ge (1+\epsilon)\widetilde{\rho} \Rightarrow \rho_H(U) \ge c\epsilon^{-2}\log n$
(ii) $\forall U \subseteq V : \rho_G(U) < (1-2\epsilon)\widetilde{\rho} \Rightarrow \rho_H(U) < (1-\epsilon)c\epsilon^{-2}\log n$

It follows from the above lemma that $\rho^*(H) \simeq c\epsilon^{-2}\log n$ iff $\widetilde{\rho}$ is very close to $\rho^*(G)$. We can now make parallel guesses $\widetilde{\rho}$ for $\rho^*(G)$ and find the correct one by identifying the guess that gives the desired value of $\rho^*(H)$. We explain this in detail and prove it formally in the next section.

3.2 Fully Dynamic Algorithm for WDSH using UDSH

We handle the unweighted case UDSH and obtain the following theorem in Section 4.

Theorem 3.3. *Given an unweighted rank-r (multi-)hypergraph H on n vertices and at most m edges with max-multiplicity at least $w^\star$, there exists a fully dynamic data structure* Udshp *that deterministically maintains a $(1+\epsilon)$-approximation to the densest subhypergraph problem. The worst-case update time is $O(\max\{(64r\epsilon^{-2}\log n)/w^\star, 1\}\cdot r\epsilon^{-4}\log^2 n\log m)$ per edge insertion or deletion. The worst-case query times for max-density and densest-subset queries are $O(1)$ and $O(\beta+\log n)$ respectively, where β is the output-size.*

Here, we describe a way to use the above theorem as a subroutine to efficiently solve the dynamic WDSH problem. For the input weighted hypergraph G, assume that we know the value of $w_{\max}(G)$ and an upper bound m on the number of hyperedges (across all updates) in advance.[1] First, we observe the following.

Observation 3.4. *In a rank-r weighted hypergraph G with at most m edges, we have $w_{\max}(G)/r \le \rho^*(G) \le m w_{\max}(G)$.*

Our algorithm for the dynamic WDSH problem is as follows.

Preprocessing. We keep guesses $\widetilde{\rho}_i = (w_{\max}/r)(1+\epsilon)^i$ for $i = 0, 1, \ldots, \lceil \log_{1+\epsilon}(rm)\rceil$. Note that by Observation 3.4, these are valid guesses for $\rho^*(G)$. For each guess $\widetilde{\rho}_i$ and each $j \in \lceil \log_{1+\epsilon} w_{\max}\rceil$, we construct a data structure SAMPLE(i,j) that, when queried, generates independent samples from the probability distribution $\mathrm{Bin}\big(\lfloor(1+\epsilon)^j\rfloor, q(\widetilde{\rho}_i)\big)$.[2] Each such data structure can be constructed in $O(w_{\max})$ time so that each query is answered in $O(1)$ time ([BP17], Theorem 1.2). Parallel to this, for each i, we have a copy of the data structure for the UDSH problem, given by Udshp(i). The value of $w^\star$ that we set for Udshp(i) is $w_{\max}(G)\cdot q(\widetilde{\rho}_i)/2$.

Update processing. On insertion of the edge e with weight w_e, for each guess $\widetilde{\rho}_i$, query SAMPLE$(i, \lceil \log_{1+\epsilon} w_e\rceil)$ to get a number s, and insert s copies of the unweighted edge e using the data structure Udshp(i). Similarly, on deletion of edge e, for each i, use Udshp(i) to delete all copies of the edge added during its insertion.

Query processing. Denote the value of maximum density returned by Udshp(i) as $\hat{\rho}_i$. Let i^* be the largest i such that $\hat{\rho}_i \ge (1-\epsilon)c\epsilon^{-2}\log n$. On a max-density query for the WDSH problem, we output $\frac{1-2\epsilon}{1+\epsilon}\cdot\widetilde{\rho}_{i^*}$. For the densest-subset query, we output the densest subset returned by Udshp(i^*).

Correctness. Observe that the hypergraph we feed to Udshp(i) is $G'_{q(\widetilde{\rho}_i)}$, where G' is the hypergraph obtained by rounding up each edge weight of G to the nearest power of $(1+\epsilon)$. Thus, $\rho^*(G) \le \rho^*(G') \le (1+\epsilon)\rho^*(G)$.

For simplicity, we write $G'_{q(\widetilde{\rho}_i)}$ as G'_i. Note that the value of $w^\star$ provided to each Udshp(i) satisfies the condition in Theorem 3.3 whp (by the Chernoff bound (Fact 2.1)) since the expected value of max-multiplicity of G'_i is $w_{\max}(G)\cdot q(\widetilde{\rho}_i)$. By Theorem 3.3, Udshp$(i)$ returns value $\hat{\rho}_i$ such that

$$(1-\epsilon)\rho^*(G'_i) \le \hat{\rho}_i \le \rho^*(G'_i)\,.$$

By the definition of i^*, we have $\hat{\rho}_{i^*} \ge (1-\epsilon)c\epsilon^{-2}\log n$. This means $\rho^*(G'_{i^*}) \ge (1-\epsilon)c\epsilon^{-2}\log n$. Then, by Lemma 3.2 (ii), we get $\rho^*(G') \ge (1-2\epsilon)\widetilde{\rho}_{i^*}$. Therefore, we have

$$\widetilde{\rho}_{i^*} \le \frac{\rho^*(G')}{1-2\epsilon} \le \frac{1+\epsilon}{1-2\epsilon}\cdot\rho^*(G)\,. \tag{1}$$

Again, note that $\hat{\rho}_{i^*+1} < (1-\epsilon)c\epsilon^{-2}\log m$. Hence, $\rho^*(G'_{i^*+1}) \le \hat{\rho}_{i^*+1}/(1-\epsilon) < c\epsilon^{-2}\log m$. Then, by Lemma 3.2 (i), it follows that $\rho^*(G') < (1+\epsilon)\widetilde{\rho}_{i^*+1} = (1+\epsilon)^2\widetilde{\rho}_{i^*}$. Hence, we have

$$\widetilde{\rho}_{i^*} > \frac{\rho^*(G')}{(1+\epsilon)^2} \ge \frac{\rho^*(G)}{(1+\epsilon)^2}\,. \tag{2}$$

Thus, from eqs. (1) and (2), we get

$$\rho^*(G) \ge \frac{1-2\epsilon}{1+\epsilon}\cdot\widetilde{\rho}_{i^*} \ge \frac{1-2\epsilon}{(1+\epsilon)^3}\cdot\rho^*(G)\,. \tag{3}$$

Again, let U^* be the densest subset returned by Udshp(i^*). By Lemma 3.2 (ii), we see that

$$\rho_{G'}(U^*) \ge (1-2\epsilon)\widetilde{\rho}_{i^*} \ge \frac{1-2\epsilon}{(1+\epsilon)^2}\cdot\rho^*(G)$$

Therefore, by the definition of G', we have

$$\rho^*(G) \ge \rho_G(U^*) \ge \frac{\rho_{G'}(U^*)}{1+\epsilon} \ge \frac{1-2\epsilon}{(1+\epsilon)^3}\cdot\rho^*(G) \tag{4}$$

Given any $0 < \delta < 1$, we set $\epsilon = \Theta(\delta)$ small enough so that $\frac{1-2\epsilon}{(1+\epsilon)^3} \ge \frac{1}{1+\delta}$. Therefore, by eqs. (3) and (4), the value and the subset that we return on the max-density and densest-subset queries respectively are $(1+\delta)$-approximations to $\rho^*(G)$.

Runtime. As noted before, we feed G'_i to Udshp(i). Fix an i. Let ω_i be the max-multiplicity of an edge in G'_i. When a hyperedge of G is inserted/deleted, we insert/delete at most ω_i unweighted copies of that edge to Udshp(i). Therefore, by Theorem 3.3, the worst case update time for Udshp(i) is $O(\omega_i\cdot\max\{(64r\epsilon^{-2}\log n)/w^\star, 1\}\cdot r\epsilon^{-4}\log^2 n\log m)$. Using the Chernoff bound (Fact 2.1), we have $\omega_i \le 2w_{\max}(G)\cdot q(\widetilde{\rho}_i) = 4w^\star$ whp. Also, since $\widetilde{\rho}_i \ge w_{\max}(G)/r$ for each i, we can apply Lemma 3.1 to get that $\omega_i = O(r\epsilon^{-2}\log n)$. Hence, the expression simplifies to $O(r\epsilon^{-2}\log n\cdot r\epsilon^{-4}\log^2 n\log m) = O(r^2\epsilon^{-6}\log^3 n\log m)$. Finally, accounting for all the $O(\log_{1+\epsilon} rm) = O(\epsilon^{-1}\log m)$ values of i, the total update time is $O(r^2\delta^{-7}\log^3 n\log^2 m)$ (recall that $\delta = \Theta(\epsilon)$). The max-density query for WDSH is answered by binary-searching on the $O(\epsilon^{-1}\log m)$ copies of Udshp, which gives a query time of $O(\log\delta^{-1} + \log\log m)$ by Theorem 3.3. Note that the densest-subset query is made only on the relevant copy i^* after we find it, and hence, by Theorem 3.3, it takes $O(\beta+\log n)$ time, where β is solution-size. Therefore, we obtain the following theorem that captures our main result.

Theorem 3.5. *(Formal version of Theorem 1.1) Given a weighted rank-r hypergraph on n vertices and at most m edges, for any $0<\delta<1$, there exists a randomized fully dynamic algorithm that maintains a $(1+\delta)$-approximation to the densest subhypergraph problem. The worst-case update time is $O(r^2\delta^{-7}\log^3 n\log^2 m)$ per hyperedge insertion or deletion. The worst-case query times for max-density and densest-subset queries are $O(\log\delta^{-1}+\log\log m)$ and $O(\beta+\log n)$ respectively, where β is the output-size. The preprocessing time is $O(w_{\max}\delta^{-2}\log m\log w_{\max})$, where $w_{\max}$ is the max-weight of a hyperedge.*

Now all it remains is to solve the unweighted case and prove Theorem 3.3. We do this in Section 4.

[1] These assumptions can be removed with very small increase in update time while preserving the approximation ratio (details in the full version [BBCG21])

[2] $\mathrm{Bin}(n,p)$ is the Binomial distribution with parameters n and p.

4 FULLY DYNAMIC ALGORITHM FOR UDSH

Here, due to limited space, we give a sketch of our algorithm and analysis for the dynamic UDSH problem and provide the complete details in the full version [BBCG21].

Our Algorithm and Analysis. We extend the techniques of [SW20] for the densest subgraph problem and take the primal-dual approach to solve the UDSH problem. Recall that the input is an unweighted multi-hypergraph $H = (V, E)$ and we want to find the approximate max-density as well as an approximately densest subset of H. As is standard, we associate a variable $x_v \in \{0, 1\}$ with each vertex v and $y_e \in \{0, 1\}$ with each hyperedge e such that $x_v = 1$ and $y_e = 1$ respectively denote that we include v and e in the solution subset. Relaxing the variables, the primal LP for UDSH (Primal(H)) is given below. Following notations similar to [SW20], for each vertex u and edge e, let $f_e(u)$ and D be the dual variables corresponding to constraints (5) and (6) respectively. Then, the dual program Dual(H) is as follows.

$$\textbf{Primal}(H): \quad \max \sum_{e\in E} y_e$$

$$\text{s.t. } y_e \le x_u \quad \forall u \in e\ \forall e \in E \tag{5}$$

$$\sum_{v\in V} x_v \le 1 \tag{6}$$

$$x_v, y_e \ge 0 \quad \forall v \in V, e \in E \tag{7}$$

$$\textbf{Dual}(H): \quad \min D$$

$$\text{s.t. } \sum_{u\in e} f_e(u) \ge 1 \quad \forall e \in E \tag{8}$$

$$\sum_{e\ni v} f_e(v) \le D \quad \forall v \in V \tag{9}$$

$$f_e(u) \ge 0 \quad \forall u \in e\ \forall e \in E \tag{10}$$

Think of $f_e(u)$ as a "load" that edge e puts on node u. We can thus interpret Dual(H) as a load balancing problem: each hyperedge needs to distribute a unit load among its vertices such that the maximum total load on a vertex due to all its incident edges is minimized. For each $v \in V$, define $\ell(v) := \sum_{e\ni v} f_e(v)$. Note that if for some feasible solution, some edge e assigns $f_e(v) > 0$ to some $v \in e$ and $\ell(v) > \ell(u)$ for some $u \in e \setminus \{v\}$, then we can "transfer" some positive load from $f_e(v)$ to $f_e(u)$ while maintaining constraint (8) and without exceeding the objective value. Therefore, we can always find an optimal solution to Dual(H) satisfying the following "local" property.

$$\forall e \in E : f_e(v) > 0 \Rightarrow \ell(v) \le \ell(u)\ \forall u \in e \setminus \{v\} \tag{11}$$

We can verify that property (11) is also sufficient to get a *global* optimal solution to Dual(H) (see full version [BBCG21]). Next, we show in Theorem 4.1 (proof deferred to Appendix A) that "approximately" maintaining property (11) (see const. (14)) gives us a near-optimal solution to Dual(H), i.e., an approximate value of $\rho^*(H)$. In this regard, we define a system of equations Dual(H, η) as follows.

$$\ell(v) = \sum_{e\ni v} f_e(v) \qquad \forall v \in V \tag{12}$$

$$\sum_{u\in e} f_e(u) = 1 \qquad \forall e \in E \tag{13}$$

$$\ell(v) \le \ell(u) + \eta \qquad \forall u \in e \setminus \{v\},\ \forall e \in E : f_e(v) > 0 \tag{14}$$

$$f_e(u) \ge 0 \qquad \forall u \in e\ \forall e \in E \tag{15}$$

Theorem 4.1. *Given a feasible solution $\langle \hat{f}, \hat{\ell} \rangle$ to Dual(H, η), we have $\rho^*(1-\varepsilon) \le \hat{D}(1-\varepsilon) < \rho^*$, where $\hat{D} = \max_v \hat{\ell}(v)$ and $\varepsilon = \sqrt{\frac{8\eta \log n}{\hat{D}}}$.*

By Theorem 4.1, we see that if we can find $\hat{D}$, i.e., a feasible solution to Dual(H, η), then we can get a $(1+\epsilon)$-approximation to ρ^*, where $\epsilon = \sqrt{\frac{32\eta \log n}{\hat{D}}}$. This means that given ϵ, we initially need to set $\eta = \frac{\epsilon^2 \hat{D}}{32 \log n}$. But we do not know the value of $\hat{D}$ initially, and in fact, that's what we are looking for. However, we shall initially have an estimate $\widetilde{D}$ of $\hat{D}$ such that $\widetilde{D} \le \hat{D} \le 2\widetilde{D}$. We set $\eta := \frac{\epsilon^2 \widetilde{D}}{32 \log n}$. Since $\widetilde{D} \le \hat{D}$, we get $\sqrt{\frac{8\eta \log n}{\hat{D}}} \le \frac{\epsilon}{2}$, which, by Theorem 4.1, implies a $(1+\epsilon)$-approximation to UDSH. To see how we can identify an approximate densest subset (not just the value of its density), see the proof of Theorem 4.1 (Appendix A).

Thus, we focus on finding a feasible solution to Dual(H, η), where $\eta = \frac{\epsilon^2 \widetilde{D}}{32 \log n}$ for a given estimate $\widetilde{D}$ satisfying $\widetilde{D} \le \hat{D} \le 2\widetilde{D}$. Note that if we have $\eta \ge 1$, then we can maintain constraint (14) with some positive slack while having integer loads on the vertices. This means that we are allowed to simply assign the unit load of an edge e entirely on some vertex $u \in e$. Assume that we know a lower bound $w^\star$ on the max-multiplicity of a hyperedge in the graph. If $w^\star \ge 64r\epsilon^{-2} \log n$, then it already implies that $\eta \ge 1$ since $\hat{D} \ge \rho^* \ge 64\epsilon^{-2} \log n$ and hence, $\widetilde{D} \ge \hat{D}/2 \ge 32\epsilon^{-2} \log n$. Otherwise, we duplicate each hyperedge $\lceil (64r\epsilon^{-2} \log n)/w^\star \rceil$ times (hence, this factor appears in the update time of Theorem 3.3), so that we are ensured that $\rho^* \ge 64\epsilon^{-2} \log n$, implying $\eta \ge 1$ as before. Once we have $\eta \ge 1$ and are allowed to assign the entire load of an edge on a single node in it, our problem reduces to the following hypergraph "orientation" problem.

Problem (Hypergraph Orientation). *Given an unweighted multi-hypergraph $H = (V, E)$ and a parameter $\eta \ge 1$, for each edge $e \in E$, assign a vertex $v \in e$ as its head $h(e)$, such that*

$$\forall e \in E : h(e) = v \Rightarrow d_{in}(v) \le d_{in}(u) + \eta\ \forall u \in e \setminus \{v\} \tag{16}$$

where $d_{in}(v) := |\{e \in E : h(e) = v\}|$.

Given a parameter $\widetilde{D}$, we construct a data structure HOP($\widetilde{D}$) that maintains the "oriented" hypergraph satisfying (16) with $\eta = \frac{\epsilon^2 \widetilde{D}}{32 \log n}$ and in turn, solves the UDSH problem. We describe it in detail in Data Structure 1. The following lemmas (see full version for proofs) give the correctness and runtime guarantees of the data structure.

Lemma 4.2. *After each insertion/deletion, the data structure HOP($\widetilde{D}$) maintains constraint (16) with $\eta = \frac{\epsilon^2 \widetilde{D}}{32 \log n}$.*

Lemma 4.3. *If $\widetilde{D} \le \hat{D}$, then the operations querysubset and querydensity of HOP($\widetilde{D}$) return a $(1+\epsilon)$-approximation to the densest-subset and max-density queries respectively.*

Lemma 4.4. *If $\hat{D} \le 2\widetilde{D}$, then the operations insert and delete of HOP($\widetilde{D}$) take $O(r\epsilon^{-4} \log^2 n)$ and $O(r\epsilon^{-2} \log n)$ time respectively. The operation querydensity takes $O(1)$ time and querysubset takes $O(\beta + \log n)$ time, where β is the solution-size.*

Completing the Algorithm. The above lemmas prove Theorem 3.3 as long as we have an estimate $\widetilde{D}$ such that $\widetilde{D} \le \hat{D} \le 2\widetilde{D}$.

For this, we keep parallel data structures HOP($\widetilde{D}$) for $O(\log m)$ guesses of $\widetilde{D}$ in powers of 2. Then, we show that we can maintain an "active" copy of HOP corresponding to the correct guess, from which the solution is extracted. Thus, we incur only an $O(\log m)$ overhead on the total update time for an edge insertion/deletion. This part is very similar to Algorithm 3 of [SW20] and we discuss this in detail in the full version [BBCG21] and formally prove Theorem 3.3.

5 EXPERIMENTS

In this section, we present extensive experimental evaluations of our algorithms. We consider weighted and unweighted hypergraphs in both insertion-only and fully dynamic settings, leading to a total of four combinations. However, due to space limitations, we discuss only the fully dynamic setting here and defer the incremental setting to Appendix B. We call our algorithms UDSHP and WDSHP for the unweighted and weighted settings respectively and we compare their accuracy and efficiency to that of the baseline algorithms. Furthermore, we study the trade-off between accuracy and efficiency for UDSHP and WDSHP.

Table 1: Description of our dataset with the key parameters, #vertices(n), #hyperedges(m), maximum size (#hyperedges) in the dynamic setting (m_Δ), #rank(r).

Dataset	n	m	m_Δ	r
dblp-all	2.56M	3.16M	1.99M	449
tag-math-sx	1.6K	558K	21.3K	5
tag-ask-ubuntu	3K	219K	10.4K	5
tag-stack-overflow	50K	12.7M	50.6K	5
dawn	2.5K	834K	11.5K	16
coauth-MAG-geology	1.25M	960K	216.3K	25

Datasets. We collect real-world temporal hypergraphs, as described below. Table 1 presents a summary of these hypergraphs.

Publication datasets. We consider two publication datasets: DBLP [dbl] and Microsoft Academic Graph (MAG) with the geology tag [SSS+15, BAS+18][3]. We encode each author as a vertex and each publication as a hyperedge with the publication year serving as the timestamp. In the fully dynamic case, we maintain a sliding window of 10 years, by removing hyperedges that are older than 10 years. We treat multiple papers by the same set of authors as a single hyperedge and report densest subgraph at the end of each year.

Tag datasets. We consider 3 tag datasets: math exchange [BAS+18][3], stack-overflow [BAS+18][3], and ask-ubuntu [BAS+18][3]. In each of these datasets, a vertex corresponds to a tag, and a hyperedge corresponds to a set of tags on a post or a question in the respective website. In the fully dynamic model, we maintain a sliding window of 3 months. In both insertion only and dynamic settings, we report the densest subgraph at an interval of 3 months.

Drug Abuse Warning Network(DAWN) dataset. This dataset[3] is generated from the national health surveillance system that records drug abuse related events leading to an emergency hospital visit across USA [BAS+18]. We construct a hypergraph where the vertices are the drugs and a hyperedge corresponds to a combination of drugs taken together at the time of abuse. We maintain the most recent 3 months records in fully dynamic and insertion only settings and report the maximum density at an interval of 3 months.

Weighted Datasets. Each of the datasets described above are unweighted. We are not aware of any publicly available weighted temporal hypergraphs. For our weighted settings, we transform the unweighted temporal hypergraph into a weighted temporal hypergraph by the following process. For each edge, we assign it an integer weight sampled uniformly at random from $[1, 100]$.

Implementation Details. The implementation details of our algorithm are given in Data Structure 1.[4] In implementing Algorithm 8, we consider all potential subsets B by ignoring the condition on line 3, and report the subset with the largest density among these choices. We implement all algorithms in C++ and all experiments are run on a workstation with 256 GB memory and Intel Xeon(R) 2.20 GHz processor running Ubuntu 20 operating system.

Baseline Algorithms. We consider two main baselines algorithms. **(1)** The first one is an exact algorithm, denoted as EXACT, that computes the exact value of the densest subhypergraph at every reporting interval of the dataset. We use *google OR-Tools* to implement an LP based solver for the densest subhypergraph [HWC17, PF]. **(2)** Second one is the dynamic algorithm for maintaining densest subhypergraph by Hu et al. [HWC17]; we call it HWC. It takes ϵ_H as an input accuracy parameter and produces a $(1+\epsilon_H)r$ and $(1+\epsilon_H)r^2$-approximate densest subhypergraph in the insertion only and fully dynamic model respectively. For the weighted hypergraphs we modify the HWC implementation – each edge with weight w_e is processed by creating w_e many copies of that edge.

Parameter Settings. Both HWC and our algorithms UDSHP and WDSHP take an accuracy parameter ϵ as an input. However, it is important to note that the accuracy parameter ϵ for both the algorithms are not directly comparable. UDSHP or WDSHP guarantees to maintain a $(1+\epsilon)$-approximate solution in both insertion only and fully dynamic settings, whereas HWC maintains $(1+\epsilon)r$ and $(1+\epsilon)r^2$ approximate solutions for insertion and fully dynamic settings respectively. Thus, for a fair comparison between the algorithms, we run UDSHP (or WDSHP) and HWC with different values of ϵ such that their accuracy is comparable. We use ϵ_H to denote the parameter for HWC to make this distinction clear. For various settings and datasets, we use different parameters and specify them in the corresponding plots. We emphasize here that the motivation behind the choices of the parameters is to compare the update time of UDSHP and WDSHP to that of HWC while ensuring that UDSHP and WDSHP has better accuracy than that of HWC. We restrict our focus to the *small* approximation error regime.

Accuracy and Efficiency Metrics. To measure the accuracy of UDSHP, WDSHP, and HWC, we use *relative error percentage* with respect to EXACT. It is defined as $\frac{|\rho(\text{ALG},t)-\rho(\text{EXACT},t)|}{\rho(\text{EXACT},t)} \times 100\%$, where $\rho(X,t)$ is the density estimate by algorithm X at time interval t. We also compute the average relative error of an algorithm by taking the average of the relative errors over all the reporting intervals. For measuring efficiency we compare the average wall-clock time taken over the operations during each reporting interval and also overall

[3]Source: https://www.cs.cornell.edu/~arb/data/

[4]Our code is available (anonymously): Link to Code Repo

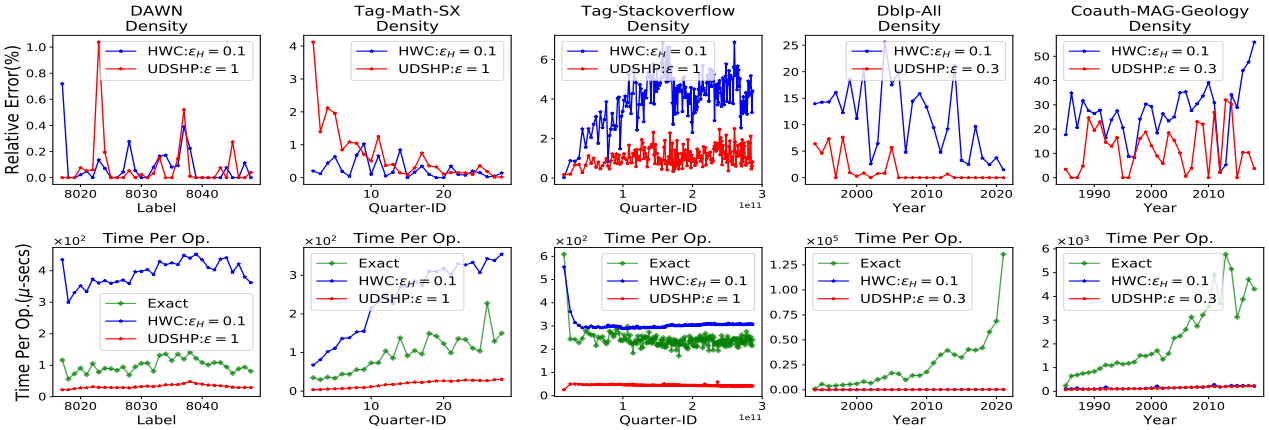


Figure 1: Accuracy and Efficiency Comparison for Unweighted Dynamic Hypergraphs: The top row shows the relative error in the reported maximum density by UDSHP and HWC with respect to EXACT when run with the specified parameters. The bottom row plots the average update time taken by UDSHP, HWC, and EXACT for each reporting intervals. For each dataset (column), the parameter settings are identical.

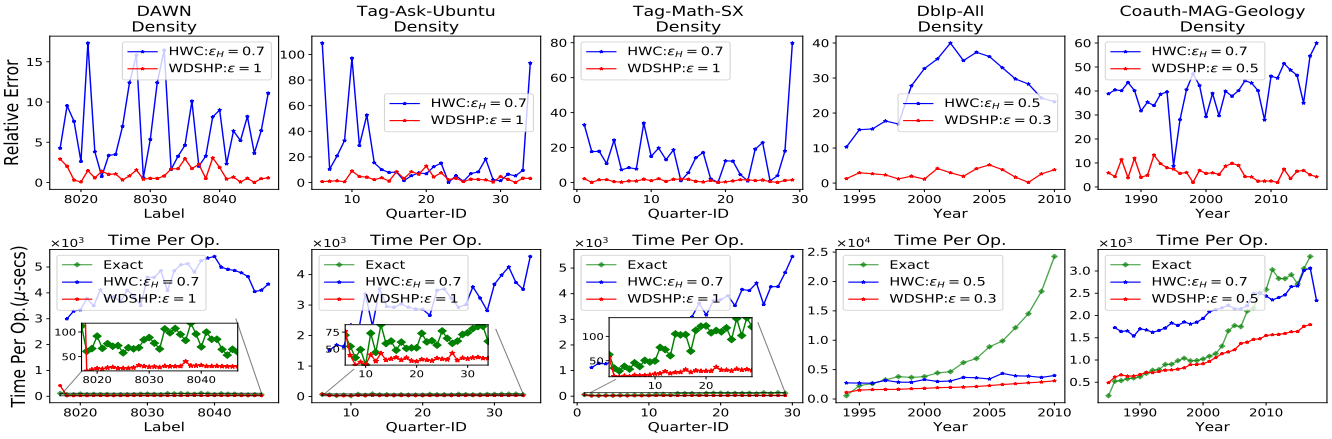
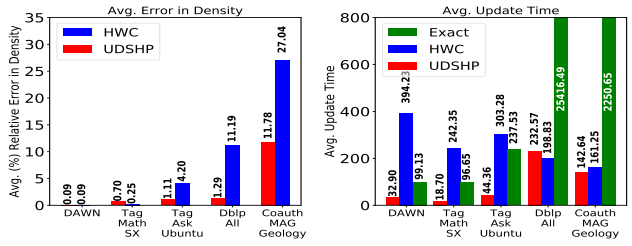
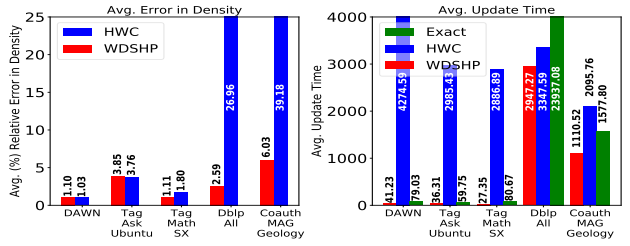


Figure 2: Accuracy and Efficiency Comparison for Weighted Dynamic Hypergraphs: The top row shows the relative error in the reported maximum density by WDSHP and HWC with respect to EXACT when run with the specified parameters. The bottom row plots the average update time taken by WDSHP, HWC, and EXACT for each reporting intervals. For each dataset (column), the parameter settings are identical.



(a) UDSHP for Unweighted Dynamic Hypergraphs



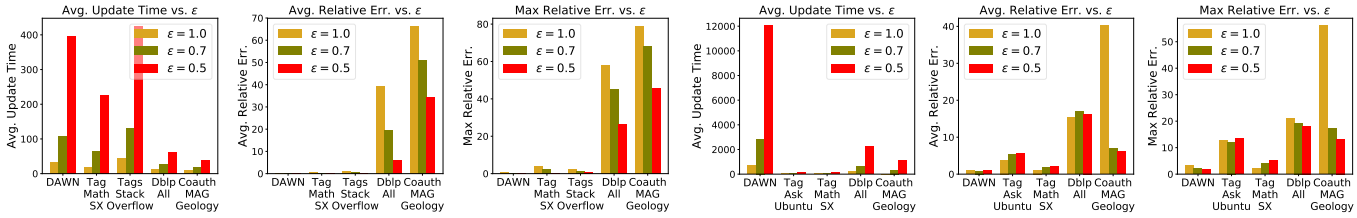
(b) WDSHP for Weighted Dynamic Hypergraphs

Figure 3: Average Accuracy and Efficiency Comparison: In Figure 3a, on the left, we plot the average relative error of UDSHP and HWC over all the reporting intervals for each dataset, and on the right, we compare the average update time of UDSHP, HWC, and EXACT over the entire duration. In Figure 3b, we give analogous plots comparing WDSHP with HWC and EXACT.

average time (taken over all reporting intervals) as an efficiency comparison metric.

5.1 Fully Dynamic Case

In this section, we consider the fully dynamic setting where the hyperedges can be both inserted and deleted. We perform experiments for hypergraphs with unweighted as well as weighted edges.



(a) UDSHP for Unweighted Dynamic Hypergraphs

(b) WDSHP for Weighted Dynamic Hypergraphs

Figure 4: Accuracy vs Efficiency Trade-off: In Figure 4a, we give trade-offs for UDSHP for Unweighted Dynamic Hypergraphs: On the left, we plot the average update time for different settings of ϵ. In the middle and right, we show the effect of ϵ on the average relative error and maximum relative error (over the reporting intervals). In Figure 4b, we give analogous trade-off plots for WDSHP.

For both the cases, we first compare the accuracy and the efficiency of our algorithm against the baselines. And then we analyze the accuracy vs efficiency trade-off of UDSHP and WDSHP.

UNWEIGHTED HYPERGRAPHS: We first discuss our findings for the unweighted case.

Accuracy and Efficiency Comparison. In Figure 1, we compare the accuracy and efficiency of UDSHP against the baselines for the unweighted hypergraphs. In the top row, we compare the accuracy of UDSHP and HWC in terms of relative error percentage with respect to EXACT. In the bottom row, we plot the average time taken per operation by EXACT, UDSHP, and HWC during each reporting interval. For each dataset, the parameters are identical for the top row and bottom row plots. We reiterate that the input parameters for UDSHP and HWC are chosen to compare UDSHP and HWC in the low relative error regime. We highlight our main findings below.

We observe that for smaller hypergraphs (DAWN, tag-math-sx), UDSHP and HWC achieve impressive accuracy, however UDSHP is consistently more than 10x faster than HWC. In fact, HWC is several times slower compared to EXACT. On the other hand, UDSHP is 3x-5x times faster compared to EXACT. As the sizes of the hypergraphs increase, EXACT gets much slower compared to UDSHP and HWC as LP solvers are known to have scaling issues.[5] For larger datasets, UDSHP maintains a clear edge in terms of accuracy over HWC even when their update times are almost identical or better for UDSHP, as demonstrated by the last three columns. To quantify the gain further, in Figure 3a, we compare the performance of UDSHP against HWC and EXACT in terms of average relative error and average update time, where the average is taken over all the reporting intervals. We make several interesting observations. (1) UDSHP is 3x-5x faster than EXACT for small hypergraphs; the gain is massive (10x-15x) for larger graphs. (2) Compared against HWC, the avg. update time for UDSHP can be 10x-12x smaller (DAWN and tag-math-sx) while maintaining almost the same average relative error of less than 1%. (3) At the other end of the spectrum, for almost the same average update time, UDSHP can offer 55%-90% improvement in accuracy over HWC (Coauth-MAG and DBLP-All). (4) HWC performs worse than EXACT for smaller datasets, being slower by 3x-5x factors (DAWN and tag-math-sx).

[5]Note that, although tag-stack-overflow hypergraph has overall more edges than Coauth-MAG, the reporting interval for the latter is much longer than the former. Thus at any given interval, the latter contains more edges leading to larger update times.

Accuracy vs Efficiency trade offs for UDSHP. In Figure 4a we plot *average update time*, and *average and max relative error* for UDSHP for different values of ϵ. The max relative error is the maximum of the relative error over all the reporting intervals. As expected, when ϵ decreases, the update time increases and the average and maximum relative error incurred by UDSHP decreases.

We observe that for the hypergraphs with high density values ($\Omega(\log n)$), e.g., DAWN, tag-math-sx, tag-stack-overflow, the average and maximum relative errors are quite low ($< 2-5\%$). Thus, we recommend using UDSHP with larger values of ϵ (like $\epsilon = 1$) for them. Note that reduction in update time is quite dramatic (~8x) when increasing ϵ from 0.5 to 1.0 for these graphs. For the hypergraphs with low density values ($o(\log n)$) the relative errors can go well above $30\% - 40\%$ for larger values of ϵ. Thus, we recommend using UDSHP with smaller values of ϵ (like $\epsilon = 0.3$) for more accurate solutions, as for hypergraphs like Coauth-MAG, reducing ϵ from 1.0 to 0.5 reduces the average relative error from 70% to 30% (albeit at the cost of a 3-fold increase in average update time).

WEIGHTED HYPERGRAPHS: For the weighted case in Figure 2, we consider similar settings as in Figure 1. In the top row, we compare the relative error percentage of WDSHP and HWC, and the bottom row, shows the average update times of WDSHP, HWC, and EXACT with same parameters (for each hypergraph). For a detailed discussion on the accuracy and efficiency comparison and tradeoffs, please refer to the full version [BBCG21].

REFERENCES

[AKSS12] Albert Angel, Nick Koudas, Nikos Sarkas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, 2012.

[BAS+18] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proc. of the National Academy of Sciences*, 2018.

[BBC+15] Oana Denisa Balalau, Francesco Bonchi, T-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM)*, page 379–388, 2015.

[BBCG21] Suman K. Bera, Sayan Bhattacharya, Jayesh Choudhari, and Prantar Ghosh. A new dynamic algorithm for densest subhypergraphs. https://drive.google.com/file/d/1P8rysfIgGnuChrJ_bsZ8l8UXfPBObx5O/view, 2021.

[BGKV14] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. Core decomposition of uncertain graphs. In *The 20th ACM SIGKDD Intl. Conf. on KDD*, pages 1316–1325, 2014.

[BGP+20] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos E. Tsourakakis, Di Wang, and Junxing Wang. Flowless: Extracting densest subgraphs without flow computations. In *The Web Conference (WWW)*, pages 573–583, 2020.

[BHNT15] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proc. 47th annual ACM*

symposium on Theory of computing, pages 173–182, 2015.

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proc. VLDB Endow.*, 5(5):454–465, 2012.

[BP17] Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. *Algorithmica*, 79(2):484–508, 2017.

[CLTZ18] T.-H. Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. Spectral properties of hypergraph laplacian and approximation algorithms. *J. ACM*, 65(3):15:1–15:48, 2018.

[CS12] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *IEEE TKDE*, 24(7):1216–1230, 2012.

[dbl] Dblp dataset. https://dblp.uni-trier.de/xml/. 2020.

[ELS15] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *The 24th International Conf. on World Wide Web, (WWW)*, pages 300–310, 2015.

[FM95] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, 1995.

[GJL+13] Aristides Gionis, Flavio Junqueira, Vincent Leroy, Marco Serafini, and Ingmar Weber. Piggybacking on social networks. *Proc. VLDB Endow.*, 6(6):409–420, 2013.

[GKT05] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proc. 31st International Conference on Very Large Data Bases (VLDB)*, pages 721–732, 2005.

[Gol84] Andrew V Goldberg. *Finding a maximum density subgraph.* University of California Berkeley, 1984.

[GT15] Aristides Gionis and Charalampos E. Tsourakakis. Dense subgraph discovery:kdd 2015 tutorial. In *Proc. 21st Annual SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 2313–2314, 2015.

[HWC17] Shuguang Hu, Xiaowei Wu, and TH Hubert Chan. Maintaining densest subsets efficiently in evolving hypergraphs. In *Proc. of 2017 ACM on Conf. on Information and Knowledge Management*, pages 929–938, 2017.

[LA07] Kevin J. Lang and Reid Andersen. Finding dense and isolated submarkets in a sponsored search spending graph. In *Proc. 16th ACM Conf. on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, pages 613–622. ACM, 2007.

[MPP+15] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proc. 21st Annual SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 815–824, 2015.

[PF] Laurent Perron and Vincent Furnon. Or-tools.

[SHK+10] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Research in Computational Molecular Biology (RECOMB)*, volume 6044, pages 456–472, 2010.

[SSS+15] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. An overview of microsoft academic service (MAS) and applications. In *Proc. 24th Proceedings, International World Wide Web Conference (WWW)*, 2015.

[SW20] Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proc. 52nd Annual ACM Symposium on the Theory of Computing*, pages 181–193. ACM, 2020.

[TBG+13] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proc. 19th Annual SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 104–112, 2013.

[Tso15] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proc. 24th Intl. Conf. on World Wide Web*, pages 1122–1132, 2015.

Data Structure 1: The algorithms for HOP($\widetilde{D}$) that solves the hypergraph orientation problem

Input: Unweighted hypergraph $H = (V, E)$, parameters $\epsilon, \widetilde{D}$

$n \leftarrow |V|$

$\eta \leftarrow \frac{\epsilon^2 \widetilde{D}}{32 \log n}$

Indegrees $\leftarrow$ Balanced binary search tree for $\{d_{in}(v) : v \in V\}$

Each hyperedge e maintains a list of vertices that it contains and has a pointer $h(e)$ to the head vertex.

Each vertex v maintains the following data structures:

- $d_{in}(v)$: Number of hyperedges e such that $h(e) = v$
- $\text{In}(v)$: List of hyperedges (labels only) where v is the head
- $\text{Out}(v)$: Max-priority queue of $\{e \in E : h(e) \neq v\}$, indexed by $d_{in}^{(v)}(h(e))$
- $d_{in}^{(v)}(u)$ ($\forall e \in \text{Out}(v)\ \forall u \in e$): $d_{in}(u)$ from v's perspective

Algorithm 1 ROTATE(e, v)

1: $z \leftarrow h(e)$
2: remove e from $\text{In}(z)$
3: remove e from $\text{Out}(u)$ for each $u \in e \setminus \{z\}$
4: $h(e) \leftarrow v$; add e to $\text{In}(v)$
5: add e to $\text{Out}(u)$ for each $u \in e \setminus \{v\}$

Algorithm 2 TIGHTINEDGE(v)

1: **for** $e \in \{\text{next } 4d_{in}(v)/\eta \text{ edges in } \text{In}(v)\}$ **do**
2: $u \leftarrow \arg\min_{z \in e} d_{in}(z)$
3: **if** $d_{in}(u) \leq d_{in}(v) - \eta/2$ **then**
4: return e
5: return null

Algorithm 3 INSERT(e)

1: $v \leftarrow \arg\min_{z \in e} d_{in}(z)$
2: $h(e) \leftarrow v$
3: add e to $\text{In}(v)$
4: add e to $\text{Out}(u)$ for each $u \in e \setminus \{v\}$
5: **while** tightinedge(v) $\neq$ null **do**
6: $f \leftarrow$ tightinedge(v)
7: $v \leftarrow \arg\min_{z \in f} d_{in}(z)$; rotate($f, v$)
8: increment(v)

Algorithm 4 INCREMENT(v)

1: $d_{in}(v) \leftarrow d_{in}(v) + 1$
2: Update $d_{in}(v)$ in Indegrees
3: **for** $e \in \{\text{next } 4d_{in}(v)/\eta \text{ edges in } \text{In}(v)\}$ **do**
4: **for** $u \in e$ **do**
5: $d_{in}^{(u)}(v) \leftarrow d_{in}(v)$

Algorithm 5 TIGHTOUTEDGE(v)

1: $e \leftarrow \text{Out}(v)$.max
2: **if** $d_{in}^{(v)}(h(e)) \geq d_{in}(v) + \eta/2$ **then**
3: return e
4: return null

Algorithm 6 DELETE(e)

1: $v \leftarrow h(e)$
2: remove e from $\text{In}(v)$
3: remove e from $\text{Out}(u)$ for each $u \in e \setminus \{v\}$
4: **while** tightoutedge(v) $\neq$ null **do**
5: $f \leftarrow$ tightoutedge(v); $z \leftarrow h(f)$
6: rotate(f, v); $v \leftarrow z$
7: decrement(v)

Algorithm 7 DECREMENT(v)

1: $d_{in}(v) \leftarrow d_{in}(v) - 1$
2: Update $d_{in}(v)$ in Indegrees
3: **for** $e \in \{\text{next } 4d_{in}(v)/\eta \text{ edges in } \text{In}(v)\}$ **do**
4: **for** $u \in e$ **do**
5: $d_{in}^{(u)}(v) \leftarrow d_{in}(v)$

Algorithm 8 DENSESTSUBSET(γ)

1: $\hat{D} \leftarrow$ Indegrees.max; $A \leftarrow \{v : d_{in}(v) \geq \hat{D}\}$
2: $B \leftarrow \{v : d_{in}(v) \geq \hat{D} - \eta\}$
3: **while** $|B|/|A| \geq 1 + \gamma$ **do**
4: $\hat{D} \leftarrow \hat{D} - \eta$; $A \leftarrow B$
5: $B \leftarrow \{v : d_{in}(v) \geq \hat{D} - \eta\}$
6: return B

Algorithm 9 QUERYSUBSET()

1: $\hat{D} \leftarrow$ Indegrees.max; $\gamma \leftarrow \sqrt{2\eta \log n / \hat{D}}$
2: return densestsubset(γ)

Algorithm 10 QUERYDENSITY()

1: return (Indegrees.max)$\cdot(1 - \frac{\epsilon}{2})$

A MISSING PROOF

Proof of Theorem 4.1. Since $\langle \hat{f}, \hat{\ell} \rangle$ is a feasible solution to Dual(H, η), we see that $\langle \hat{f}, \hat{D} \rangle$ is a feasible solution to Dual(H). Since ρ^* is an optimal solution to Dual(H), we have $\hat{D} \geq \rho^*$ and the left inequality follows.

Define $S_i := \{v : \hat{\ell}(v) \geq \hat{D} - \eta i\}$ for $i \geq 0$. For some parameter $0 < \gamma < 1$, let k be the maximal number such that $|S_i| \geq (1+\gamma)|S_{i-1}|$ for all $i \in [k]$. Thus, $|S_{k+1}| < (1+\gamma)|S_k|$.

For an edge e incident on $v \in S_k$, consider $u \in e \setminus \{v\}$. We have

$$u \notin S_{k+1} \Rightarrow \hat{\ell}(u) < \hat{D} - \eta(k+1) \leq \hat{\ell}(v) - \eta \Rightarrow \hat{f}_e(v) = 0$$

where the last implication is by (14). Hence, we get the following.

Observation A.1. *For $v \in S_k$, we have $\sum_{e \ni v} \hat{f}_e(v) = \sum_{\substack{e \ni v: \\ e \subseteq S_{k+1}}} \hat{f}_e(v)$.*

We try to get a lower bound on $\rho(S_{k+1})$. We see that

$$(\hat{D} - \eta k)|S_k| \leq \sum_{v \in S_k} \hat{\ell}(v) = \sum_{v \in S_k} \sum_{e \ni v} \hat{f}_e(v) = \sum_{v \in S_k} \sum_{\substack{e \ni v: \\ e \subseteq S_{k+1}}} \hat{f}_e(v)$$
$$\leq \sum_{v \in S_{k+1}} \sum_{\substack{e \ni v: \\ e \subseteq S_{k+1}}} \hat{f}_e(v) = \sum_{e \subseteq S_{k+1}} \sum_{v \in e} \hat{f}_e(v) = |E(S_{k+1})| .$$

The second equality follows by Obs. A.1 and the last one by (13). Therefore, by definition of k, we get

$$\rho(S_{k+1}) = \frac{|E(S_{k+1})|}{|S_{k+1}|} \geq \frac{(\hat{D} - \eta k)|S_k|}{|S_{k+1}|} > \frac{\hat{D} - \eta k}{1 + \gamma} > (\hat{D} - \eta k)(1 - \gamma) .$$

Again, since $|S_k| \geq (1+\gamma)^k |S_0| \geq (1+\gamma)^k$, we have $k \leq \log_{1+\gamma} |S_k| \leq \log_{1+\gamma} n \leq 2 \log n / \gamma$. Therefore, we have

$$\rho(S_{k+1}) > \left(\hat{D} - \frac{2\eta \log n}{\gamma}\right)(1 - \gamma) = \hat{D}\left(1 - \frac{2\eta \log n}{\gamma \hat{D}}\right)(1 - \gamma) .$$

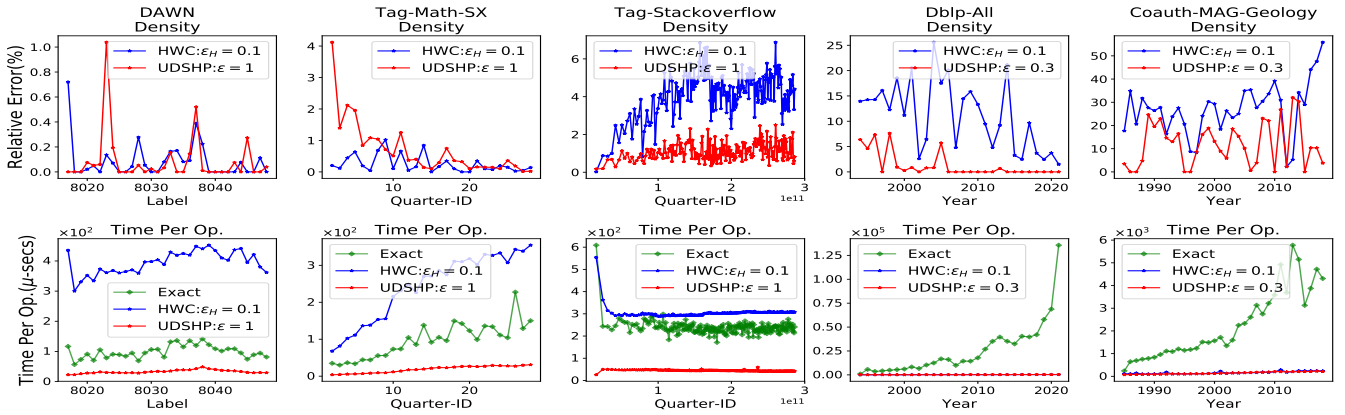


Figure 5: Accuracy and Efficiency Comparison for Unweighted Insertion-only Hypergraphs: The top row shows the relative error in the maximum density by UDSHP and HWC with respect to EXACT when run with the specified parameters. The bottom row plots the average update time taken by UDSHP, HWC, and EXACT for each reporting intervals. For each dataset (column), the parameter settings are identical.

We set γ so as to maximize the RHS. Clearly, it is maximized when $\gamma = \frac{2\eta \log n}{\gamma \hat{D}}$, and so, we set $\gamma := \sqrt{\frac{2\eta \log n}{\hat{D}}}$. Hence, we get

$$\rho^* \geq \rho(S_{k+1}) > \hat{D}\,(1-\gamma)^2 > \hat{D}(1-2\gamma) = \hat{D}\left(1 - \sqrt{\frac{8\eta \log n}{\hat{D}}}\right). \quad \square$$

B EXPERIMENTS: INSERT-ONLY CASE

Here, we give an account of our experiments for the insert-only setting with unweighted hyperedges. We defer the discussion on the weighted incremental setting to the full version [BBCG21].

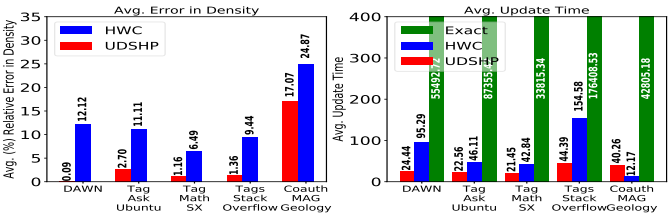


Figure 6: Avg. Accuracy and Efficiency Comparison for Unweighted Incremental Setting: On the left, we plot avg. relative err. of UDSHP and HWC, and on the right, we compare the avg. update time of UDSHP, HWC, and EXACT. (Average is taken over the entire duration)

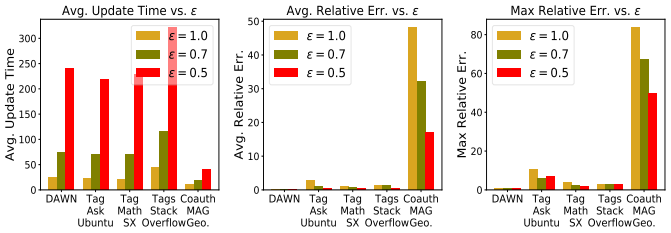


Figure 7: Accuracy vs Efficiency Trade-off for Unweighted Incremental Hypergraphs (UDSHP): We plot the avg. update time (left), avg. relative err. (middle), and max. relative err. (right) over the reporting intervals for different settings of ϵ.

Accuracy and Efficiency Comparison. In Figure 5 top row we compare the accuracy of UDSHP and HWC with respect to EXACT. And in the bottom row, we plot the average time taken per operation by EXACT, UDSHP, and HWC during each reporting interval. To further quantify the gain of UDSHP, in Figure 6, we compare the performance of UDSHP against HWC and EXACT in terms of average relative error and average update time. We highlight some of our main findings below.

(1) Performance of HWC fluctuates quite a lot over time as evident from the *saw-tooth* behaviour in the relative error and the update time curves for HWC in Figure 5. Thus, even if the average case update time for HWC is low, the worst-case update time could be very high. In contrast, UDSHP exhibits a much more stable behavior over time, making it more suitable for practical use. Note that this is consistent with the theoretical results for the respective algorithms since HWC only guarantees small *amortized* update time while UDSHP guarantees small *worst-case* update time.

(2) For the first four datasets, on average UDSHP has 70% better accuracy while being 2x-4x faster (on average) compared to HWC (Figure 6). For the largest dataset `Coauth-MAG`, HWC indeed has an edge over UDSHP in terms of average update time while both incurring comparable loss in accuracy (Figure 6). However, as we noted before, the *saw-tooth* behavior of HWC implies a higher worst-case update time for HWC compared to UDSHP (Figure 5).

(3) EXACT performs extremely poorly in the incremental settings, as one would expect. The sizes of the hypergraphs are much larger compared to the dynamic settings, making EXACT extremely unsuitable for any practical purpose.

Accuracy vs Efficiency trade offs. As similar to that in the dynamic setting, in Figure 7, we analyze the change in the *average update time* and the *average and max relative error* for UDSHP for different values of $\epsilon \in \{1.0, 0.7, 0.5\}$. We observe that even if the update time is sensitive to change in ϵ, the average and maximum relative error for all the high density ($\Omega(\log n)$) hypergraphs (`DAWN`, `tag-ask-ubuntu`, `tag-math-sx`, `tag-stack-overflow`) is low ($< 10\%$). And thus we recommend, using UDSHP with high value of ϵ (like $\epsilon = 1$) for these hypergraphs. On the other hand for the low density ($o(\log n)$) hypergraphs (like `Coauth-MAG`), we recommend using UDSHP with low value of ϵ (like $\epsilon = 0.5$ or 0.3).