FULLY DYNAMIC SUBMODULAR MAXIMIZATION OVER A MATROID WITH POLYLOGARITHMIC UPDATE TIME

Anonymous authors
Paper under double-blind review

ABSTRACT

We study fully dynamic maximization of a normalized monotone submodular function subject to a matroid constraint of rank at most k. For every oblivious sequence of n insertions and deletions and $0 < \epsilon \leq 1/10$, our randomized algorithm maintains a feasible set X_t satisfying $\mathbb{E}[f(X_t)] \geq (1/4 - \epsilon)\mathrm{OPT}_t$ at every fixed time t. The total number of value- and independence-oracle queries is

$$O\Big(\frac{n}{\epsilon}\log^3(k/\epsilon)\log(2k)\log^2(2n)\Big)$$

for every realization of the random choices, giving polylogarithmic amortized query complexity for fixed ϵ. Our approach combines a deletion-tolerant exchange certificate with deterministic hierarchical rebuilding. The certificate charges the optimum to the frozen weights of retained and deleted acceptances. Exact truncated sampling controls expected deletion loss and leaves small residuals for later rebuilding levels, yielding the approximation and query guarantees.

1 INTRODUCTION

Submodularity expresses diminishing returns: the gain from adding a new item cannot increase when more items have already been selected (Nemhauser et al., 1978). The same mathematical structure appears in coverage (Khuller et al., 1999) and influence maximization (Kempe et al., 2003), sensor placement (Krause et al., 2008), data and video summarization (Badanidiyuru et al., 2014; Gygli et al., 2015), and selecting a diverse set of representatives (Lin & Bilmes, 2011). A cardinality bound limits the size of a solution; a matroid also captures structural restrictions, such as a separate capacity for each group or the requirement that selected edges form a forest (Ene & Nguyen, 2019). Offline algorithms assume that all available items remain fixed (Călinescu et al., 2011). In many applications, however, data arrive continuously and may expire, be corrected, or be withdrawn (Lattanzi et al., 2020; Dütting et al., 2023). A previously feasible solution can become invalid after a deletion, and the contribution of an unselected item can increase when previously selected items disappear. Fully dynamic submodular maximization asks for a feasible approximate solution after *every* insertion and deletion, with the cost of repairing it measured by value- and independence-oracle queries (Dütting et al., 2023; Banihashem et al., 2024). Recomputing from scratch after each change would inspect the current ground set repeatedly (Nemhauser et al., 1978).

The central efficiency question is whether the oracle cost per update can be polylogarithmic in both the number of updates and the matroid rank. For a cardinality constraint, this is possible for a $1/2 - \epsilon$ approximation (Lattanzi et al., 2020; Banihashem et al., 2023b). In a general matroid, one must additionally decide whether an arriving element can enter the independent solution and, if necessary, which member of its fundamental circuit to replace. The algorithms of Dütting et al. (2023) and Banihashem et al. (2024) obtain approximately $1/4$ of optimum, but their oracle bounds depend polynomially on the rank k. The $0.3178 - \epsilon$ approximation of Gao et al. (2026) also has polynomial rank dependence. Removing it remains important even at the $1/4$-approximation threshold.

We give, to our knowledge, the first constant-factor fully dynamic algorithm for an arbitrary matroid whose *amortized oracle-query* complexity is polylogarithmic in both n and k. For every oblivious sequence of n updates, it maintains a feasible set X_t with $\mathbb{E}[f(X_t)] \geq (1/4 - \epsilon)\,\mathrm{OPT}_t$ at every fixed

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Fully dynamic *monotone* submodular maximization. The final column bounds expected amortized value- and independence-oracle calls per update. The ICML 2023 cardinality bound marked † is an explicit, nonoptimized upper bound derived from the published analysis.

Result	Constraint	Ratio	Queries per update
Monemizadeh (2020)	cardinality	$1/2-\epsilon$	$O\big(\epsilon^{-3}k^2\log^5(2n)\big)$
Lattanzi et al. (2020) (revised)	cardinality	$1/2-\epsilon$	$O\big(\epsilon^{-4}\log^3(2k)\log(k/\epsilon)\cdot\log(2n)\log(2n/\epsilon)\big)$
Banihashem et al. (2023b)	cardinality	$1/2-\epsilon$	$O\big(\epsilon^{-12}\log^6(2k)\log(k/\epsilon)\cdot\log^2(2nk/\epsilon)\big)^{\dagger}$
Dütting et al. (2023)	matroid	$1/(4+O(\epsilon))$	$O\big(\epsilon^{-1}k^2\log^3(k/\epsilon)\log(2k)\cdot\log^2(2n)\big)$
Banihashem et al. (2024)	matroid	$1/(4+\epsilon)$	$O\big(k\log^3(k/\epsilon)\log(2k)\big)$
Gao et al. (2026)	matroid	$0.3178-\epsilon$	$O\big(\epsilon^{-2}k^3\log^5(k/\epsilon)\log^2(2k)\big)$
This paper	**matroid**	$\mathbf{1/4-\epsilon}$	$O\big(\epsilon^{-1}\log^3(k/\epsilon)\log(2k)\cdot\log^2(2n)\big)$

time, using

$$O\Big(\frac{n}{\epsilon}\log^3(k/\epsilon)\log(2k)\log^2(2n)\Big)$$

value- and independence-oracle calls in total for every random outcome. All oracle queries involve currently available elements. The query bound is amortized. Table 1 locates the result among fully dynamic algorithms, with explicit dependence on ϵ, k, and n in every query bound.

Related work. The classical greedy algorithm obtains $1-1/e$ for offline monotone maximization under a cardinality bound (Nemhauser et al., 1978); continuous greedy extends this factor to matroids (Călinescu et al., 2011). In insertion-only streaming, Sieve-Streaming and its variants obtain $1/2-\epsilon$ for cardinality (Badanidiyuru et al., 2014; Kazemi et al., 2019). For a general matroid, Chakrabarti & Kale (2015) use frozen marginal gains and exchanges to obtain $1/4$ in one pass; Feldman et al. (2022) improve the one-pass factor to 0.3178, and obtain $1-1/e-\epsilon$ with $O_\epsilon(1)$ passes. Such an algorithm need not repair its solution after a deletion. A fully dynamic algorithm must also reconsider elements whose earlier rejection was justified by an item that has since been removed.

The first fully dynamic cardinality algorithms were given independently by Monemizadeh (2020) and Lattanzi et al. (2020); the corrected arXiv version of the latter addresses gaps in its original proof, while Banihashem et al. (2023b) gives another polylogarithmic-amortized construction. Chen & Peng (2022) prove that improving on $1/2$ by a constant requires $n^{\widetilde{\Omega}(\epsilon)}/k^3$ amortized queries, ruling out polylogarithmic updates, for example, when k is polylogarithmic in n. The same paper treats insertion-only matroid maximization, obtaining $1-1/e-\epsilon$ with $k^{\widetilde{O}(1/\epsilon^2)}\log n$ amortized queries. For arbitrary matroids with deletions, the results in Table 1 trade approximation factor for dependence on k; this paper focuses on that rank dependence.

For nonmonotone objectives, Banihashem et al. (2023a) obtain a near-$1/8$ approximation under cardinality. Liu & Yang (2025) improve it to approximately $1/6$ and obtain $(1-\epsilon)/(8+e)$ under a general matroid; Banihashem et al. (2025b) also study dynamic p-matchoid constraints. For general matroids, Gao et al. (2026) give a $0.1921-\epsilon$ nonmonotone result. Other directions include dynamic submodular matching (Banihashem et al., 2025a) and algorithms that limit the number of changes to the maintained solution (Dütting et al., 2026). Summaries robust to a fixed deletion budget and dynamic streaming with bounded deletions use different update models: space bounds alone do not ensure cheap repairs under unrestricted insertions and deletions.

Technique overview. To see the algorithmic obstacle, first imagine scanning available items once. As in the matroid streaming algorithm (Chakrabarti & Kale, 2015), maintain an independent set I, an ordered history I' of accepted items, and a weight $w(e)$ equal to the marginal of e when it entered I'. Accept a candidate if it can be added to I; otherwise evict the cheapest exchangeable item only when the new marginal is at least twice its frozen weight. Each exchange uses $O(\log k)$ oracle queries.

Our data structure has $O(\log n)$ *time levels* with fixed rebuilding periods. Given a fixed guess τ of the optimum value, it groups candidates into geometrically descreasing marginal intervals. In each interval it accepts uniformly random eligible items while their number exceeds a cutoff, forwarding the small residual to the next, more frequently rebuilt time level. A whole interval is processed by one random-permutation scan and binary searches over its saved states; only the prefix up to the cutoff is committed. This implements the uniform-sampling rule *exactly* in near-linear oracle queries per reconstruction, instead of rescanning candidates after each acceptance. Each selected item comes from a pool much larger than the possible deletions in that interval, limiting expected lost frozen weight.

Banihashem et al. (2024, Lemmas 17–19) already use random permutations and binary search to obtain near-linear query complexity for a single reconstruction. Their dynamic structure maintains one selection level per accepted element. Uniform selection cancels the candidate-pool size against the rebuilding probability, but the expected query bound still sums over $O(k \log(k/\epsilon))$ selection levels and retains a factor of k. Our hierarchy has only $O(\log n)$ time levels, each of which can commit many acceptances. A level rebuilt every s updates inherits its parent's exchange state and processes only $O((s/\epsilon)\log(k/\epsilon))$ candidates from the parent's residual and recent insertions; the root processes at most s candidates. For fixed ϵ, the query cost of this reconstruction is linear in s up to logarithmic factors in n and k. Rebuilding once every s updates cancels the factor s in amortization. Summing over time levels gives polylogarithmic amortized query complexity in n and k per instance.

Table 2 separates the cost of a reconstruction from its frequency. We charge each direct trigger for the entire suffix it reconstructs. Our data structure maintains $O(\log n)$ time levels. Marginal buckets are formed within each reconstruction and are not counted as additional levels.

Table 2: Data structures and query complexity. Level counts, reconstruction costs, and frequencies refer to one fixed guess with a known update bound. The final column includes parallel enumeration and removal of the update-bound assumption.

Reference	Levels	Queries per level reconstruction	Average rebuilds per level/update	Amortized queries per update
Dütting et al. (2023)	$O(\log(2n))$	$O\big(s_\ell k \log(k/\epsilon) \cdot \log(2k)\big)$	$O\big((k/s_\ell) \cdot \log(k/\epsilon)\big)$	$O\big(\epsilon^{-1}k^2 \log^3(k/\epsilon) \cdot \log(2k)\log^2(2n)\big)$
Banihashem et al. (2024)	$O(k \log(k/\epsilon))$	$O\big(m_i \log(k/\epsilon) \cdot \log(2k)\big)$	$\leq 1/m_i$	$O\big(k \log^3(k/\epsilon) \cdot \log(2k)\big)$
This paper	$O(\log(2n))$	$O\big(\epsilon^{-1}s_\ell \log^2(k/\epsilon) \cdot \log(2k)\log(2n)\big)$	$\leq 1/s_\ell$	$O\big(\epsilon^{-1} \log^3(k/\epsilon) \cdot \log(2k)\log^2(2n)\big)$

Here $s_\ell = 2^{\lceil \log_2 n \rceil - \ell}$ is the capacity or period, and m_i is the current candidate-pool size, before deletion or after insertion. Frequencies count direct triggers: $1/m_i$ is a conditional probability; the other bounds are amortized, in expectation where needed. Reconstruction costs include the entire rebuilt suffix. For our algorithm, costs and frequencies refer to whole time levels, including all their marginal buckets.

The remaining difficulty is correctness after a deletion. Removing an item from I' can increase other items' submodular marginals and can invalidate a previous exchange rejection. In the proof alone, we keep a record of accepted items deleted since the relevant constructions; adding them to the current independent set yields a weighted-span certificate that survives deletions. It charges the optimum to four times the total frozen weight of current and deleted items; the sampling bound makes the deleted part small in expectation.

Paper structure. Section 2 formulates the problem. Section 3 describes the algorithmic operations, Section 4 presents the main invariants and bounds, and the appendix provides detailed proofs. Section 5 concludes.

2 Preliminaries

Matroid. Let $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ be a fixed matroid of rank at most a known $k \geq 1$; thus $\mathcal{I} \subseteq 2^{\mathcal{V}}$ is downward closed and, for $I, J \in \mathcal{I}$ with $|I| < |J|$, some $e \in J \setminus I$ satisfies $I + e \in \mathcal{I}$. Here and below $A + e = A \cup \{e\}$ and $A - e = A \setminus \{e\}$. We write $\textsc{Span}(A) = \{e \in \mathcal{V} : \mathrm{rank}(A + e) = \mathrm{rank}(A)\}$

for the matroid span, defined on the fixed ground set even when A contains a subsequently deleted item. If $I \in \mathcal{I}$ and $I + e \notin \mathcal{I}$, then $C_I(e)$ is the unique fundamental circuit in $I + e$.

Dynamic submodular maximization. Let $f : 2^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ be fixed, normalized ($f(\varnothing) = 0$), monotone, and submodular. In particular, for $A \subseteq B$ and $e \notin B$, its marginal obeys $f(e \mid A) := f(A + e) - f(A) \geq f(e \mid B) \geq 0$. At update t, an element is inserted into or removed from the active set $V_t \subseteq \mathcal{V}$, initially $V_0 = \varnothing$. Every operation is legal, and the update sequence is fixed independently of the algorithm's random choices. The output $X_t \subseteq V_t$ must satisfy $X_t \in \mathcal{I}$. Define $I_t^* \in \arg\max\{f(I) : I \subseteq V_t,\ I \in \mathcal{I}\}$ and $\mathrm{OPT}_t = f(I_t^*)$. There are at most n available elements after any prefix of n updates. A value query $f(A)$ and an independence query $\mathbf{1}\{A \in \mathcal{I}\}$ are allowed only for $A \subseteq V_t$; a set used in a value query need not be independent. We measure query complexity by the number of these oracle calls, including evaluation of the reported solution. We initially assume that a deleted element is never reinserted; Appendix G handles reinsertion by assigning a fresh parallel copy to each insertion. If $\mathrm{OPT}_t = 0$, the empty set suffices.

3 Algorithm

We first assume that a suitable guess τ of the optimum value is supplied and describe the data structure for this fixed guess. Its organization has two indices: $O(\log n)$ time levels ℓ and $O(\log(k/\epsilon))$ marginal-gain buckets j within each level. Algorithm 5 initializes the time levels and handles insertions and deletions by rebuilding a scheduled suffix, with geometrically decreasing periods. Each level stores an independent set, an acceptance history with frozen marginal weights, and a residual candidate set. To rebuild level ℓ, Algorithm 5 calls Level-Construct (Algorithm 4), which traverses the marginal buckets from high to low and calls Bucket-Construct (Algorithm 3) for each bucket. Thus Algorithms 5 and 4 together implement the time-level and marginal-bucket organization. Within a bucket, a fresh random-permutation scan and binary searches over saved states exactly simulate successive uniform draws until the eligible pool falls below a prescribed cutoff. Individual acceptance and exchange decisions use Promote and Accept (Algorithms 1 and 2). The terminal time level is rebuilt after every assigned update. Parallel enumeration, described at the end of this section, runs copies of this entire structure to remove the supplied-guess assumption.

A supplied guess of the optimum. For a supplied $\tau > 0$ and fixed time t with $\mathrm{OPT}_t > 0$, assume

$$\mathrm{OPT}_t \leq \tau < 2\,\mathrm{OPT}_t\,. \tag{1}$$

An *instance* is the complete data structure for one fixed τ. The parameter stays fixed throughout that instance's execution; the accuracy condition is needed only at the observation time being analyzed. The update rules below are defined for every $\tau > 0$. The instance receives exactly the elements with $\{e\} \in \mathcal{I}$ and $\eta \leq f(\{e\}) \leq \tau$, where the minimum marginal threshold is

$$\eta = \epsilon\tau/k, \qquad B = 1 + \lceil \log_2(k/\epsilon) \rceil, \qquad a_j = 2^j \eta \quad (j = 0, \ldots, B-1). \tag{2}$$

The B *value intervals* $[a_j, 2a_j)$ are processed in the order $j = B-1, B-2, \ldots, 0$. Their index j is different from the time-level index ℓ introduced below. An element's marginal is at most its singleton value, so $\tau < 2^B \eta$ covers every acceptable marginal in the instance. Even with $\tau = \mathrm{OPT}_t$, the unchanged ratio $\tau/\eta = k/\epsilon$ leaves B marginal buckets per time level.

Exchange state and oracle tests. A *working state* is $\sigma = (I', I, w)$: I' is an ordered history of accepted *live* elements, $I \subseteq I'$ is independent, and $w(e)$ is the marginal frozen when e was accepted. Exchanging an element out of I leaves it in I' while it remains live. On accepting e, let $g = f(e \mid I')$; if $I + e$ is independent, add e, otherwise replace the minimum-weight exchangeable element $\hat{e}$ of $C_I(e) - e$ only when $g \geq 2w(\hat{e})$. An element of marginal below η is rejected. We call this decision Promote, following the promote operation of Banihashem et al. (2024). All ties use a fixed order.

Algorithm 1 PROMOTE

```
function PROMOTE(e, σ)
  if e ∈ I′ then return OLD
  g ← f(I′ + e) − f(I′)
  if g < η then return LOW
  if I + e ∈ 𝓘 then
    ê ← ⊥
  else
    ê ← arg min_{y ∈ C_I(e) − e} w(y)
  if ê ≠ ⊥ and g < 2w(ê) then
    return BLOCK
  return (PASS, g, ê)
```

Algorithm 2 ACCEPT

```
function ACCEPT(e, σ, p)
  (PASS, g, ê) ← p
  Append e to I′; set w(e) ← g
  if ê = ⊥ then
    I ← I + e
  else
    I ← I − ê + e
  return (I′, I, w)
```

The lightest circuit member needs only $O(\log(2k))$ independence queries (Dütting et al., 2023, Lemma 5.4): sort I by frozen weight and binary-search the first prefix whose removal makes $(I \setminus \text{prefix}) + e$ independent. This is the only oracle-costly part of the exchange test besides two value queries. The chosen $\hat{e}$ is the first circuit member in that order. ACCEPT (Algorithm 2) keeps the accepted element in I' even if it later leaves I, so its weight remains available for the analysis. Every set in these tests is live; the proof of the binary-search claim is in Appendix A.

One interval and an exact truncated sampler. Suppose every candidate in G currently passes PROMOTE and has marginal in $[a, 2a)$. For any state reached by acceptances in this construction, let $\chi_a(e, \sigma)$ indicate that e still passes PROMOTE and has $f(e \mid I') \geq a$, and put $P(\sigma) = \{e \in G : \chi_a(e, \sigma) = 1\}$. A conceptual procedure repeatedly accepts a uniform member of $P(\sigma)$ as long as $|P(\sigma)| \geq q$. Upon stopping it returns the residual $P(\sigma)$, which has fewer than the q elements. Every such acceptance has frozen weight in $[a, 2a)$.

Lemma 1 (Persistence within one reconstruction)**.** *Along any sequence of acceptances without intervening updates, $\chi_a(e, \sigma)$ may change from 1 to 0 but never back to 1.*

Indeed, the marginal $f(e \mid I')$ decreases as I' grows, and the weighted spans of the current independent set do not shrink under accepted exchanges. The latter observation keeps a blocked element blocked even if a different item is replaced. A later *deletion* can reverse an actual rejection, which is why this lemma applies only within one reconstruction (see Appendix A).

BUCKET-CONSTRUCT implements the conceptual uniform rule without rescanning the pool after each acceptance. A full random-order scan yields saved speculative states $\sigma_0, \ldots, \sigma_h$. The lemma permits a binary search for the first state in which each element becomes ineligible. We then discard the suffix beyond the first state with fewer than q eligible elements. The discarded states influence the query complexity but not the committed random choices.

Lemma 2 (Exact truncated sampling)**.** *If $m = |G|$, Algorithm 3 has exactly the law of successive uniform draws from the currently eligible pool, stopped when its size is below q. It returns fewer than q candidates and uses $O(m \log(2m) \log(2k))$ oracle queries for $m \geq 1$.*

Conditioned on the previously *committed* draws, eligible unscanned elements retain a uniformly random relative order. Thus the next scanned eligible element is uniform in the current pool; the saved states make the exact stopping point available without resampling. Future constructions see only the returned state and R and use fresh permutations. The full proof, including the oracle count, is Lemma 21. After the complete scan, each member of G has either been accepted or has permanently failed; hence its first failure position $d(e)$ and the stopping prefix r^* in Algorithm 3 exist.

Time levels and saved data. Within the instance for the fixed guess τ, let u count updates assigned to it, and let V be its currently live elements. First suppose the global update bound $n \geq 1$ is known. Set $N = 2^{\lceil \log_2 n \rceil}$ and $L = \log_2 N$. At time level $\ell \in \{0, \ldots, L\}$, define

$$s_\ell = N/2^\ell, \qquad q_\ell = \begin{cases} \lceil 2s_\ell/\epsilon \rceil, & \ell < L, \\ 1, & \ell = L. \end{cases} \tag{3}$$

We call level 0 the *root*: it has period $s_0 = N$ and starts from an empty exchange state. Level L is the *terminal level*: it has period $s_L = 1$, cutoff $q_L = 1$, and is reconstructed after every update assigned to the instance. If $L = 0$, these are the same level. A larger index means a shorter reconstruction period. The level stores a snapshot $\sigma_\ell = (I'_\ell, I_\ell, w_\ell)$, a residual R_ℓ, and its last reconstruction time r_ℓ; initially these are empty with $r_\ell = 0$. The state is a saved snapshot: an element may have been deleted after its construction. We first copy it and remove all elements outside V from I' and I (retaining the weights of survivors), and only then make oracle calls. Write $J_{>r}(u) = \{e : r < b(e) \le u\}$ for insertion records born after r; $b(e)$ is the instance's assigned-update count when e was inserted. We always intersect those records with V before they become candidates.

Algorithm 3 BUCKET-CONSTRUCT

Input: σ_0, G, a, q
G is eligible in $[a, 2a)$.
if $|G| < q$ **then return** (σ_0, G)
Draw a fresh uniform order π of G
Save σ_0; $h \leftarrow 0$
for e in order π **do**
  if $\chi_a(e, \sigma_h) = 1$ **then**
    $p \leftarrow$ PROMOTE(e, σ_h)
    $\sigma_{h+1} \leftarrow$ copy of σ_h
    $\sigma_{h+1} \leftarrow$ ACCEPT(e, σ_{h+1}, p)
    $h \leftarrow h + 1$
for $e \in G$ **do**
  $d(e) \leftarrow$ first $r \in \{1, \dots, h\}$ with $\chi_a(e, \sigma_r) = 0$, or $h + 1$ if none
Find the first $r^* \in \{0, \dots, h\}$ with $|\{e \in G : d(e) > r^*\}| < q$
$R \leftarrow \{e \in G : d(e) > r^*\}$
Discard speculative states and decisions after r^*
return (σ_{r^*}, R)

Algorithm 4 LEVEL-CONSTRUCT

Input: ℓ, σ, C, q_ℓ
$P \leftarrow C$; $R \leftarrow \varnothing$
for $j = B - 1, B - 2, \dots, 0$ **do**
  $G \leftarrow \varnothing$
  for $e \in P$ **do**
    $p \leftarrow$ PROMOTE(e, σ)
    if $p = \mathsf{BLOCK}$ **or** $p = \mathsf{LOW}$ **then**
      $P \leftarrow P - e$
    else if $p = (\mathsf{PASS}, g, \hat{e})$ **and** $g \ge a_j$ **then**
      $G \leftarrow G + e$; $P \leftarrow P - e$
  $(\sigma, R_j) \leftarrow$ BUCKET-CONSTRUCT(σ, G, a_j, q_ℓ)
  $P \leftarrow P \cup \big(G \setminus (I'(\sigma) \cup R_j)\big)$
  $R \leftarrow R \cup R_j$
return (σ, R)

LEVEL-CONSTRUCT goes from high to low marginal intervals. In each interval it removes candidates that permanently fail the exchange test, applies the truncated sampler to candidates in that interval, and passes its residual to the next time level. After each batch, the candidates in $G \setminus (I'(\sigma) \cup R_j)$ are returned to P, using the committed state σ returned by the batch. Thus a candidate that still passes PROMOTE but whose marginal has fallen below a_j proceeds to a lower interval. Candidates accepted during an interval are not returned in R_j; at most $q_\ell - 1$ candidates per interval remain. The scan is exact even though eligibility changes during the scan, since earlier failures persist until the end of this construction (Lemma 1).

Lemma 3 (One level's output)**.** *For an input of m live candidates C and a positive integer cutoff q_ℓ,* LEVEL-CONSTRUCT *returns a live state and a residual $R \subseteq C$ with $|R| \le B(q_\ell - 1)$. Every candidate in $C \setminus R$ either belongs to the returned history I' or fails* PROMOTE *in the returned state. Its committed acceptances within each value interval are uniform choices from a pool of at least q_ℓ elements; the call makes $O(mB \log(2m) \log(2k))$ oracle queries for $m \ge 1$.*

The residual bound follows from retaining fewer than q_ℓ eligible items in each of the B disjoint intervals. An item rejected during the call stays rejected by Lemma 1; the final interval begins at $a_0 = \eta$, so no passing candidate can remain outside R. The proof of the interval bounds and the query charge appears in Lemma 22. In particular, $q_L = 1$ makes the terminal residual empty.

UPDATE-SCALE rebuilds levels $h, \dots, L$ in increasing order, where h is the smallest index with $s_h \mid u$. The root processes V; each other level copies and cleans its parent snapshot, then processes the parent's residual and insertions since that snapshot, restricted to live elements outside the inherited history. The updated terminal set is $X^{(\tau)} = I_L$, whose value is recorded with one query. Reinsertion and an unknown update bound are handled in Appendices G and H.

Algorithm 5 Initialization, insertion, and deletion for one fixed guess τ.

Input: A fixed guess τ; updates to elements with $\eta \le f(\{e\}) \le \tau$

function REMOVE-DELETED($\sigma = (I', I, w), V$)
  $I' \leftarrow I' \cap V$; $I \leftarrow I \cap V$; $w \leftarrow w|_{I'}$
  return (I', I, w)
function INITIALIZATION(τ, n)
  $u \leftarrow 0$; $V \leftarrow \varnothing$; set N, L, s_ℓ, q_ℓ by (3)
  for $\ell = 0, \dots, L$ **do**
    $\sigma_\ell \leftarrow (\varnothing, \varnothing, \varnothing)$; $R_\ell \leftarrow \varnothing$; $r_\ell \leftarrow 0$
  $X^{(\tau)} \leftarrow \varnothing$; $v^{(\tau)} \leftarrow 0$
function INSERTION(e)
  $u \leftarrow u + 1$; $V \leftarrow V \cup \{e\}$; $b(e) \leftarrow u$; UPDATE-SCALE(τ)
function DELETION(e)
  $u \leftarrow u + 1$; $V \leftarrow V \setminus \{e\}$; UPDATE-SCALE($\tau$)
function UPDATE-SCALE(τ)
  $h \leftarrow \min\{\ell \in \{0, \dots, L\} : s_\ell \mid u\}$
  for $\ell = h, h + 1, \dots, L$ **do** ▷ Scheduled time levels
    if $\ell = 0$ **then**
      $\sigma \leftarrow (\varnothing, \varnothing, \varnothing)$; $C \leftarrow V$
    else
      $\sigma \leftarrow$ REMOVE-DELETED(copy of $\sigma_{\ell-1}, V$)
      $C \leftarrow ((R_{\ell-1} \cup J_{>r_{\ell-1}}(u)) \cap V) \setminus I'(\sigma)$
    $(\sigma_\ell, R_\ell) \leftarrow$ LEVEL-CONSTRUCT(ℓ, σ, C, q_ℓ); $r_\ell \leftarrow u$
  $X^{(\tau)} \leftarrow I_L$; query $v^{(\tau)} \leftarrow f(X^{(\tau)})$

Removing the guess assumption. We maintain instances for dyadic guesses $\tau = 2^z$, $z \in \mathbb{Z}$, creating each empty on its first assigned insertion. On insertion, test $\{e\} \in \mathcal{I}$ and, if feasible, query $c_e = f(\{e\})$; for $c_e > 0$, send the update to all instances with $c_e \le \tau \le kc_e/\epsilon$ and record this list for the eventual deletion. After each update, report the solution with largest recorded value, or the empty set if no instance exists. For the largest current feasible singleton value m, we have $m \le \mathrm{OPT}_t \le km$; whenever $\mathrm{OPT}_t > 0$, the input-determined dyadic guess satisfying (1) obeys $m \le \tau < 2km \le km/\epsilon$ and is therefore present. Its guarantee (6) carries over to the reported solution. Each update affects at most B instances, adding a factor B to the query bound.

Theorem 4 (Fully dynamic matroid maximization)**.** *For $0 < \epsilon \le 1/10$, a matroid of rank at most k and an oblivious sequence of $n \ge 1$ insertions and deletions, the algorithm outputs a live independent X_t at every time t such that $\mathbb{E}[f(X_t)] \ge (1/4 - \epsilon)\,\mathrm{OPT}_t$. Its total number of value- and independence-oracle calls is*

$$O\Big(\frac{n}{\epsilon}\log^3(k/\epsilon)\log(2k)\log^2(2n)\Big)$$

for every realization of its randomness. Every query uses only live elements. The result extends to reinsertion and an unknown update bound (see Appendices G and H).

4 ANALYSIS

Throughout this section we analyze one instance with a fixed parameter τ. For the approximation guarantee, fix an observation time t with $\mathrm{OPT}_t > 0$ and assume (1); the guess need not have been accurate at earlier updates. The structural and query bounds hold for every fixed $\tau > 0$. The instance receives updates only to elements with $\eta \le f(\{e\}) \le \tau$, with local counter u and live set V. Its update sequence is fixed independently of the random permutations. Complete proofs, including oracle legality and the auxiliary exchange facts, appear in the appendix.

4.1 ANALYSIS OF THE APPROXIMATION GUARANTEE

The streaming $1/4$-approximation proof of Chakrabarti & Kale (2015) provides useful background: we extend its exchange accounting to handle deletions.

After a τ instance's uth assigned update, its snapshots have a *retained ancestry*: their committed reconstruction calls follow chronological level order, excluding speculative computations and superseded snapshots. Let A denote the resulting ordered set of accepted elements, including those that were later deleted, and retain their frozen weights w for analysis. Put

$$D = A \setminus V, \qquad K = I_L \cup D, \qquad W = w(K) = w(I_L) + w(D),$$
$$K_{\geq b} = \{e \in K : w(e) \geq b\}.$$

The sets A, D, and K are proof objects. In particular, the old accepted elements in A must not be confused with the live terminal history I'_L.

Definition 5 (Analytical coverage)**.** In this definition, A, V, I take their values at the point under consideration, and $K = I \cup (A \setminus V)$; at the terminal point, $I = I_L$. At a point along the retained ancestry, an element e is *covered* if $e \in A$. For $e \notin A$, it is covered if $f(e \mid A) < \eta$, or if there exists $b_e > 0$ such that

$$e \in \textsc{Span}(K_{\geq b_e}) \quad \text{and} \quad f(e \mid A) < 2b_e.$$

An accepted element satisfies the first condition. An element rejected by PROMOTE due to a low marginal or a matroid exchange satisfies the second or third condition, respectively.

Lemma 6 (Deletion-resistant accounting)**.** *Along a retained ancestry, let I be the current independent set and $K = I \cup (A \setminus V)$. Then $\textsc{Span}(K_{\geq b})$ never shrinks for any fixed $b > 0$, and $w(A) \leq 2w(K)$. At the terminal endpoint, $f(I_L) \geq w(I_L)$.*

When an accepted element is deleted from the current independent set, it enters D, so the proof set K retains its weight and span. If an exchange removes $\hat{e}$ from I, the entering element has at least twice its weight; the circuit relation preserves the weighted span, and the extra weight pays for the earlier accepted element. Finally, surviving accepted elements have current submodular marginals at least their frozen marginals, yielding $f(I_L) \geq w(I_L)$. Appendix B gives all three statements with proofs. Coverage persists through later acceptances and deletions by the same span invariant, although an actual PROMOTE rejection need not persist after a deletion.

Lemma 7 (Retained ancestry and full coverage)**.** *The retained level-ℓ snapshot was built at the deterministic time $r_\ell = s_\ell \lfloor u/s_\ell \rfloor$, and these times are nondecreasing in ℓ, with $r_L = u$. Every element of V is covered at the terminal state (Definition 5).*

The root processes every item available when constructed, or places all live items in its initialized residual when the horizon is doubled. Between a parent's reconstruction and its child's, any older live item is either already covered or is in the parent's residual; newer live items appear in the insertion records $J_{>r_{\ell-1}}(u)$. The child processes both groups, and Lemma 3 covers all its inputs outside its new residual. Coverage survives both subsequent acceptances and deletions by Lemma 6. At level L, the residual is empty because $q_L = 1$. Nested periods ensure that each retained child copied the currently retained parent snapshot; Appendix D gives the induction.

Lemma 8 ($1/4$-approximation certificate)**.** *Let O be an independent set whose elements are all analytically covered by the retained trajectory. Then*

$$f(O) \leq 4W + k\eta, \qquad f(I_L) \geq w(I_L). \tag{4}$$

For covered independent O, the small-marginal and weighted-span bounds give $f(O) \leq w(A) + k\eta + 2W \leq 4W + k\eta$ by Lemma 6. See Appendix B for the proof, including rank integration.

Let Z_ℓ denote the accepted elements of the retained reconstruction at level ℓ. These sets are disjoint, and $A = Z_0 \,\dot\cup \cdots \dot\cup\, Z_L$; only these committed acceptances enter the proof history.

Lemma 9 (Deletion loss within a level)**.** *For each nonterminal level $\ell < L$ and any fixed observation time u,*

$$\mathbb{E}[w(Z_\ell \setminus V)] \leq \epsilon \mathbb{E}[w(Z_\ell)].$$

No member of the terminal set Z_L has been deleted at time u.

The level-ℓ snapshot was last built at $r_\ell = s_\ell \lfloor u/s_\ell \rfloor$. Fewer than s_ℓ items live at r_ℓ can be deleted before u. This deletion set is fixed independently of the algorithm's random choices: membership in the fixed instance depends only on the matroid and singleton values, and the update sequence is oblivious. Immediately before a *committed* acceptance from interval $[a_j, 2a_j)$, condition on the

accepted prefix, the eligible pool P, and the interval; call this information $\mathcal{F}$. Lemma 2 makes the next element uniform in P, where $|P| \geq q_\ell \geq 2s_\ell/\epsilon$. Therefore

$$\mathbb{E}[w(e)\mathbf{1}\{e \text{ is later deleted}\} \mid \mathcal{F}] \leq 2a_j \frac{s_\ell}{q_\ell} \leq \epsilon a_j \leq \epsilon \mathbb{E}[w(e) \mid \mathcal{F}].$$

The interval bounds require no independence between weight and deletion. Summing over committed acceptances proves the lemma. Since $r_L = u$, terminal acceptances remain live.

Corollary 10 (Total deletion loss)**.** *For the above A, D, W at a predetermined time,*

$$\mathbb{E}[w(D)] \leq \epsilon \mathbb{E}[w(A)] \leq 2\epsilon \mathbb{E}[W], \qquad \mathbb{E}[f(I_L)] \geq (1-2\epsilon)\mathbb{E}[W]. \tag{5}$$

Sum Lemma 9 over disjoint Z_ℓ and use Lemma 6 for $w(A) \leq 2W$ and $f(I_L) \geq w(I_L) = W - w(D)$. Conditioning on the full speculative scan would be invalid; the filtration here contains only the committed prefix. Complete proofs are in Appendix E.

At the fixed observation time t, let O_t be an optimal independent set in the global live set. Each $e \in O_t$ has $f(\{e\}) \leq \mathrm{OPT}_t \leq \tau$. If $f(\{e\}) \geq \eta$, it is assigned to this instance and covered by Lemma 7; otherwise, $f(e \mid A) \leq f(\{e\}) < \eta$, so it is also analytically covered. Lemma 8 thus applies directly to O_t, counting all small-marginal contributions in a single $k\eta$ term. Since $k\eta = \epsilon\tau < 2\epsilon\,\mathrm{OPT}_t$, Lemma 8 and Corollary 10 give the single-instance guarantee

$$\mathbb{E}[f(X^{(\tau)})] = \mathbb{E}[f(I_L)] \geq \frac{1-2\epsilon}{4}(\mathrm{OPT}_t - k\eta) \geq \frac{(1-2\epsilon)^2}{4}\,\mathrm{OPT}_t \geq (1/4-\epsilon)\,\mathrm{OPT}_t\,. \tag{6}$$

4.2 Polylogarithmic Query Complexity Analysis

Lemma 3 bounds the query cost of Level-Construct. We now derive the fixed instance's amortized insertion and deletion costs by bounding reconstruction inputs and frequencies.

Lemma 11 (Small reconstruction inputs)**.** *For $\ell > 0$, a level-ℓ reconstruction receives at most $(4B/\epsilon + 2)s_\ell$ candidates; the root receives at most $N = s_0$. All candidates are live.*

By Lemma 3, the parent residual has fewer than $2Bs_{\ell-1}/\epsilon$ elements and at most $s_{\ell-1}$ new insertions occur before its next reconstruction; since $s_{\ell-1} = 2s_\ell$, their sum is $(4B/\epsilon + 2)s_\ell$. Full proofs are in Appendix D. This is the point where the residual-size bound makes a large historical live set irrelevant to the query cost of a frequently rebuilt level.

Lemmas 3 and 11 bound one level-ℓ reconstruction by $O((s_\ell/\epsilon)B^2 \log(2n)\log(2k))$ queries. At local update u, let $h = \min\{\ell : s_\ell \mid u\}$. Algorithm 5 rebuilds precisely levels $h, \ldots, L$.

Insertion cost. An assigned insertion updates V, records $b(e) = u$, and rebuilds levels $h, \ldots, L$. Summing their query bounds with $\sum_{\ell=h}^{L} s_\ell < 2s_h$ gives

$$O\Big(\frac{s_h}{\epsilon} B^2 \log(2k)\log(2n)\Big) \tag{7}$$

queries, including evaluation of the updated output.

Deletion cost. An assigned deletion advances the same counter and rebuilds the same suffix. Cleaning copied states and filtering candidates against V require no oracle queries and preserve the input-size bound of Lemma 11, so (7) also bounds the deletion cost.

Over n_τ updates received by this instance, level ℓ is rebuilt exactly $\lfloor n_\tau/s_\ell \rfloor$ times. Charging each reconstruction to its own level cancels the factor s_ℓ and gives $O((n_\tau/\epsilon)B^2\log(2k)\log(2n))$ queries per level. Summing over the $L+1 = O(\log(2n))$ levels yields

$$O\Big(\frac{n_\tau}{\epsilon} B^2 \log(2k)\log^2(2n)\Big)\,. \tag{8}$$

The amortized cost is $O(\epsilon^{-1}\log^2(k/\epsilon)\log(2k)\log^2(2n))$ for every outcome (see Appendix F).

5 Conclusion

We obtain a $(1/4-\epsilon)$ expected approximation for fully dynamic monotone submodular maximization over matroids with polylogarithmic amortized query complexity, using only live elements in oracle queries. Whether a stronger approximation can retain this query complexity remains open.

AI USE STATEMENT

We used generative AI tools to help examine and refine research ideas and proof arguments, including identifying potential gaps and counterexamples, and to assist with drafting and revising the manuscript. The authors take full responsibility for the content of this paper, including its mathematical claims, proofs, and citations.

REFERENCES

Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 671–680, 2014. doi: 10.1145/2623330.2623637. URL https://las.inf.ethz.ch/files/badanidiyuru14streaming.pdf.

Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic non-monotone submodular maximization. In *Advances in Neural Information Processing Systems 36*, 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/387982dbf23d9975c7fc45813dd3dabc-Abstract.html.

Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, Mohammadtaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic constrained submodular optimization with polylogarithmic update time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 1660–1691. PMLR, 2023b. URL https://proceedings.mlr.press/v202/banihashem23a.html.

Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic algorithms for matroid submodular maximization. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 3485–3533. SIAM, 2024. doi: 10.1137/1.9781611977912.125. URL https://epubs.siam.org/doi/10.1137/1.9781611977912.125.

Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic algorithms for submodular matching. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, volume 334 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 19:1–19:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025a. doi: 10.4230/LIPIcs.ICALP.2025.19. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2025.19.

Kiarash Banihashem, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Non-monotone submodular optimization: p-matchoid constraints and fully dynamic setting. In *Advances in Neural Information Processing Systems 38*, 2025b. URL https://papers.nips.cc/paper_files/paper/2025/hash/963acdcdf87fec84916b10949cbee255-Abstract-Conference.html.

Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011. doi: 10.1137/080733991. URL https://epubs.siam.org/doi/10.1137/080733991.

Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids, and more. *Mathematical Programming*, 154(1–2):225–247, 2015. doi: 10.1007/s10107-015-0900-7. URL https://link.springer.com/article/10.1007/s10107-015-0900-7.

Xi Chen and Binghui Peng. On the complexity of dynamic submodular maximization. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1685–1698. ACM, 2022. doi: 10.1145/3519935.3519951. URL https://arxiv.org/abs/2111.03198.

Paul Dütting, Federico Fusco, Silvio Lattanzi, Ashkan Norouzi-Fard, and Morteza Zadimoghaddam. Fully dynamic submodular maximization over matroids. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8821–8835. PMLR, 2023. URL https://proceedings.mlr.press/v202/duetting23a.html.

Paul Dütting, Federico Fusco, Silvio Lattanzi, Ashkan Norouzi-Fard, Ola Svensson, and Morteza Zadimoghaddam. A general framework for dynamic consistent submodular maximization, 2026. URL https://arxiv.org/abs/2606.04946. Accepted at ICML 2026.

Alina Ene and Huy L. Nguyen. Towards nearly-linear time algorithms for submodular maximization with a matroid constraint. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 54:1–54:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi: 10.4230/LIPIcs.ICALP.2019.54. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2019.54.

Moran Feldman, Paul Liu, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. Streaming submodular maximization under matroid constraints. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 59:1–59:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPIcs.ICALP.2022.59. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2022.59.

Yiwei Gao, Jialin Zhang, and Zhijie Zhang. Improved fully dynamic submodular maximization under matroid constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40 (43):36928–36936, 2026. doi: 10.1609/aaai.v40i43.41020. URL https://ojs.aaai.org/index.php/AAAI/article/view/41020.

Michael Gygli, Helmut Grabner, and Luc Van Gool. Video summarization by learning submodular mixtures of objectives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3090–3098. IEEE, 2015. doi: 10.1109/CVPR.2015.7298928. URL https://openaccess.thecvf.com/content_cvpr_2015/html/Gygli_Video_Summarization_by_2015_CVPR_paper.html.

Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3311–3320. PMLR, 2019. URL https://proceedings.mlr.press/v97/kazemi19a.html.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 137–146. ACM, 2003. doi: 10.1145/956750.956769. URL https://www.cs.cornell.edu/home/kleinber/kdd03-inf.pdf.

Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. doi: 10.1016/S0020-0190(99)00031-9. URL https://doi.org/10.1016/S0020-0190(99)00031-9.

Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(8):235–284, 2008. URL https://jmlr.org/papers/v9/krause08a.html.

Silvio Lattanzi, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakub Tarnawski, and Morteza Zadimoghaddam. Fully dynamic algorithm for constrained submodular optimization. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://arxiv.org/abs/2006.04704. Corrected analysis: arXiv:2006.04704v2, May 2023.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 510–520. Association for Computational Linguistics, 2011. URL https://aclanthology.org/P11-1052/.

Yuanyang Liu and Wenguo Yang. Dynamic algorithms for non-monotone submodular maximization. In *Combinatorial Optimization and Applications (COCOA 2024), Part II*, volume 15435 of *Lecture Notes in Computer Science*, pp. 119–131. Springer, 2025. doi: 10.1007/978-981-96-4448-3_10. URL https://link.springer.com/chapter/10.1007/978-981-96-4448-3_10.

Morteza Monemizadeh. Dynamic submodular maximization. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/6fbd841e2e4b2938351a4f9b68f12e6b-Abstract.html.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978. doi: 10.1007/BF01588971. URL https://link.springer.com/article/10.1007/BF01588971.