Improved Multi-Pass Streaming Algorithms for Submodular Maximization with Matroid Constraints

Chien-Chung Huang, Theophile Thiery, Justin Ward

Improved Multi-Pass Streaming Algorithms for Submodular Maximization with Matroid Constraints

Chien-Chung Huang
CNRS, DI ENS, Université PSL, Paris, France
chien-chung.huang@ens.fr

Theophile Thiery
School of Mathematical Sciences, Queen Mary University of London, UK
t.f.thiery@qmul.ac.uk

Justin Ward
School of Mathematical Sciences, Queen Mary University of London, UK
justin.ward@qmul.ac.uk

Abstract

We give improved multi-pass streaming algorithms for the problem of maximizing a monotone or arbitrary non-negative submodular function subject to a general p-matchoid constraint in the model in which elements of the ground set arrive one at a time in a stream. The family of constraints we consider generalizes both the intersection of p arbitrary matroid constraints and p-uniform hypergraph matching. For monotone submodular functions, our algorithm attains a guarantee of $p + 1 + \varepsilon$ using $O(p/\varepsilon)$-passes and requires storing only $O(k)$ elements, where k is the maximum size of feasible solution. This immediately gives an $O(1/\varepsilon)$-pass $(2 + \varepsilon)$-approximation for monotone submodular maximization in a matroid and $(3 + \varepsilon)$-approximation for monotone submodular matching. Our algorithm is oblivious to the choice ε and can be stopped after any number of passes, delivering the appropriate guarantee. We extend our techniques to obtain the first multi-pass streaming algorithms for general, non-negative submodular functions subject to a p-matchoid constraint. We show that a randomized $O(p/\varepsilon)$-pass algorithm storing $O(p^3 k \log(k)/\varepsilon^3)$ elements gives a $(p + 1 + \bar{\gamma}_{\text{off}} + O(\varepsilon))$-approximation, where $\bar{\gamma}_{\text{off}}$ is the guarantee of the best-known offline algorithm for the same problem.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases submodular maximization, streaming algorithms, matroid, matchoid

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.62

Category APPROX

Funding *Chien-Chung Huang*: This work was funded by the grants ANR-19-CE48-0016 and ANR-18-CE40-0025-01 from the French National Research Agency (ANR).
Justin Ward: This work was supported by EPSRC New Investigator Award EP/T006781/1.

1 Introduction

Many discrete optimization problems in theoretical computer science, operations research, and machine learning can be cast as special cases of maximizing a *submodular* function f subject to some constraint. Formally, a function $f : 2^X \to \mathbb{R}_{\geq 0}$ is submodular if and only if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq X$. One reason for the ubiquity of submodularity in optimization settings is that it also captures a natural "diminishing returns" property. Let $f(e \mid A) \triangleq f(A + e) - f(A)$ be the *marginal increase* obtained in f when adding an element e to a set A (where here and throughout we use the shorthands $A + e$

and $A - e$ for $A \cup \{e\}$ and $A \backslash \{e\}$, respectively). Then it is well-known that f is submodular if and only if $f(e \mid B) \leq f(e \mid A)$ for any $A \subseteq B$ and any $e \notin B$. If additionally we have $f(e \mid A) \geq 0$ for all A and $e \notin A$ we say that f is *monotone*.

Here, we consider the problem of maximizing both monotone and arbitrary submodular functions subject to an arbitrary *p-matchoid constraint* on the set of elements that can be selected. Formally, a p-matchoid $\mathcal{M}^p = (\mathcal{I}^p, X)$ on X is given by a collection of matroids $\{\mathcal{M}_i = (X_i, \mathcal{I}_i)\}$ each defined on some subset of X, where each $e \in X$ is present in at most p of these subsets. A set $S \subseteq X$ is then independent if and only if $S \cap X_i \in \mathcal{I}_i$ for each matroid $\mathcal{M}_i$. One can intuitively think of a p-matchoid as a collection of matroids in which each element "participates" in at most p of the matroid constraints. The resulting family of constraints is quite general and captures both intersections of p matroid constraints (by letting $X_i = X$ for all $\mathcal{M}_i$) and matchings in p-uniform hypergraphs (by considering X as a collection of hyperedges and defining a uniform matroid constraint for each vertex, ensuring that at most one hyperedge containing this vertex is selected).

In many applications of submodular optimization, such as summarization [19, 1, 21, 23] we must process datasets so large that they cannot be stored in memory. Thus, there has been recent interest in *streaming* algorithms for submodular optimization problems. In this context, we suppose the ground set X is initially unknown and elements arrive one-by-one in a stream. We suppose that the algorithm has an efficient oracle for evaluating the submodular function f on any given subset of X, but has only enough memory to store a small number of elements from the stream. Variants of standard greedy and local search algorithms have been developed that obtain a constant-factor approximation in this setting, but their approximation guarantees are considerably worse than that of their simple, offline counterparts.

Here, we consider the *multi-pass* setting in which the algorithm is allowed to perform several passes over a stream – in each pass all of X arrives in some order, and the algorithm is still only allowed to store a small number of elements. In the *offline* setting, simple variants of greedy [15] or local search [18, 13] algorithms in fact give the best-known approximation guarantees for maximizing submodular functions subject to the p matroid constraints or a general p-matchoid constraint. However, these algorithms potentially require considering all elements in X *each time a choice is made*. It is natural to ask whether this is truly necessary, or whether we could instead recover an approximation ratio nearly equal to these offline algorithms by using only a constant number of passes through the data stream.

1.1 Our Results

Here we show that for monotone submodular functions, $O(1/\varepsilon)$-passes suffice to obtain guarantees only $(1 + \varepsilon)$ times worse than those guaranteed by the offline local search algorithm. We give an $O(p/\varepsilon)$-pass streaming algorithm that gives a $p + 1 + \varepsilon$ approximation for maximizing a monotone submodular function subject to an arbitrary p-matchoid constraint. This immediately gives us an $O(1/\varepsilon)$ pass streaming algorithm attaining a $2+\varepsilon$ approximation for matroid constraints and a $3 + \varepsilon$ approximation for matching constraints in graphs. Each pass of our algorithm is equivalent to a single pass of the streaming local search algorithm described by Chakrabarti and Kale [6] and Chekuri, Gupta, and Quanrud [7]. However, obtaining a rapid convergence to a $p + 1 + \varepsilon$ approximation requires some new insights. We show that if a pass makes either large or small progress in the value of f, then the guarantee obtained at the end of this pass can be improved. Balancing these two effects then leads a carefully chosen sequence of parameters for each pass. Our general approach is similar to that of Chakrabarti and Kale [6], but our algorithm is *oblivious* to the choice of ε. This

Table 1 Summary of known approximation ratios for maximizing monotone and non-monotone submodular functions. M means that f is monotone, and NN means that f is non-negative. In stating results parameterized by p, we let $o(1)$ denote a term that approaches zero as p increases.

Constraint	Offline	
	M	NN
matroid	$e/(e-1)$ [5]	2.598 [3]
rank p hypergraph b-matching	$p+\varepsilon$ [13]	$\frac{p^2+\varepsilon}{p-1}$ [13]
p matroid intersection	$p+\varepsilon$ [18]	$\frac{p^2+(p-1)\varepsilon}{p-1}$ [18]
p-matchoid	$p+1$ [15]	$\frac{ep}{(1-\varepsilon)(2-o(1))}$ [12, 8]

Constraint	Streaming		Multipass		
	M	NN	M	NN	#-passes
matroid	4 [6, 7, 11]	5.8284 [11]	$2+\varepsilon$ [6]		$O\left(\varepsilon^{-1}\right)$
rank p hypergraph b-matching	$4p$ [7, 11]	$4p+2-o(1)$ [11]	$p+1+\varepsilon$ [6]		$O\left(\frac{p^4\log(p)}{\varepsilon^3}\right)$
p matroid intersection	$4p$ [6, 7, 11]	$4p+2-o(1)$ [11]	$p+1+\varepsilon$ [6]		$O\left(\frac{p^4\log(p)}{\varepsilon^3}\right)$
p-matchoid	$4p$ [7, 11]	$4p+2-o(1)$ [11]			

allows us to give a uniform bound on the convergence of the approximation factor obtained after some number d of passes. This bound is actually available to the algorithm, and so we can certify the quality of the current solution after each pass. In practice, this allows for terminating the algorithm early if a sufficient guarantee has already been obtained. Even in the worst case, however, we improve on the number of passes required by similar previous results by a factor of $O(\varepsilon^{-2})$. Our algorithm only requires storing $O(k)$ elements, where k is the *rank* of the given p-matchoid, defined as the size of the largest independent set of elements.

Building on these ideas, we also give a randomized, multi-pass algorithm that uses $O(p/\varepsilon)$-passes and attains a $p+1+\bar{\gamma}_{\text{off}}+O(\varepsilon)$ approximation for maximizing an arbitrary submodular function subject to a p-matchoid constraint, where $\bar{\gamma}_{\text{off}}$ is the approximation ratio attained by best-known offline algorithm for the same problem. To the best of our knowledge, ours is the first multipass algorithm when the function is non-monotone. In this case, our algorithm requires storing $O(\frac{p^3 k\log k}{\varepsilon^3})$ elements. We remark that to facilitate comparison with existing work, we have stated all approximation guarantees as factors $\gamma\geq 1$. However, we note that if one states ratios of the form $1/\gamma$ less than 1, then our results lead to $1/\gamma-\varepsilon$ approximations in which all dependence on p can be eliminated (by setting simply selecting some $\varepsilon'=p\varepsilon$).

1.2 Related Work

There is a vast literature on submodular maximization with various constraints and different models of computation. In the offline model, the work on maximizing a monotone submodular function goes back to Nemhauser, Wolsey and Fischer [24]. Monotone submodular functions are well studied and many new and powerful results have been obtained since then. The best approximation algorithm under a matroid constraint is due to Calinescu et al. [5] which is the best that can be done using a polynomial number of queries [24] (if f is given as a value oracle) or assuming $P\neq NP$ [9] (if f is given explicitly). For more general constraints, Lee, Sviridenko and Vondrák obtained a $p+\varepsilon$ approximation algorithm under p matroid

Table 2 Summary of our results maximizing monotone and non-monotone submodular functions.

Constraint	Our results		
	monotone	non-negative	#-passes
matroid	$2+\varepsilon$	$4.598+\varepsilon$	$O\left(\frac{1}{\varepsilon}\right)$
rank p hypergraph b-matching	$p+1+\varepsilon$	$p+1+\frac{p^2}{p-1}+O(\varepsilon)$	$O\left(\frac{p}{\varepsilon}\right)$
p matroid intersection	$p+1+\varepsilon$	$p+1+\frac{p^2}{p-1}+O(\varepsilon)$	$O\left(\frac{p}{\varepsilon}\right)$
p-matchoid	$p+1+\varepsilon$	$p+1+\frac{ep}{(1-\varepsilon)(2-o(1))}+O(\varepsilon)$	$O\left(\frac{p}{\varepsilon}\right)$

constraints [18]. Feldman et al. [13] obtained the same approximation ratio for the general class of p-exchange systems. For general p-matchoid constraints, the best approximation ratio is $p+1$, which is attained by the standard greedy algorithm [15].

Non-monotone objectives are less understood even under the simplest assumptions. The current best-known result for maximizing a submodular function under a matroid constraint is 2.598 [3], which is far from the 2.093 hardness result [16]. Table 1 gives the best known bounds for the constraints that we consider in the paper.

Due to the large volume of data in modern applications, there has also been a line of research focused on developing *fast* algorithms for submodular maximization [2, 22]. However, all results we have discussed so far assume that the entire instance is available at any time, which may not be feasible for massive datasets. This has motivated the study of streaming submodular maximization algorithms with low memory requirements. Badaniyuru et al. [1] achieved a $2+\varepsilon$-approximation algorithm for maximizing a monotone submodular function under a cardinality constraint in the streaming setting. This was recently shown to be the best possible bound attainable in one pass with memory sublinear in the size of the instance [14]. Chakrabarti and Kale [6] gave a $4p$-approximation for the intersection of p matroid constraints or p-uniform hypergraph matching. Later, Chekuri et al. [7] generalized their argument to arbitrary p-matchoid constraints, and also gave a modified algorithm for handling non-monotone submodular objectives. A fast, randomized variant of the algorithm of [6] was studied by Feldman, Karbasi and Kazemi [11], who showed that it has the same approximation guarantee when f is monotone and achieves a $2p+2\sqrt{p(p+1)}+1=4p+2-o(1)$-approximation for general submodular function.

When multiple passes through the stream are allowed, less is known and the tradeoff between the approximation guarantee and the number of passes requires more attention. Assuming cardinality constraints, one can obtain a $\frac{e}{e-1}+\varepsilon$-multipass streaming algorithm in $O\left(\varepsilon^{-1}\right)$ passes (see [2, 17, 20, 21, 25]). Huang et al. [17] achieved a $2+\varepsilon$-approximation under a knapsack constraint in $O\left(\varepsilon^{-1}\right)$ passes. For the intersection of p partition matroids or rank p hypergraph matching, the number of passes becomes dependent on p. Chakrabarti and Kale [6][1] showed that if one allows $O\left(\frac{p^4\log(p)}{\varepsilon^3}\right)$-passes, a $p+1+\varepsilon$ approximation is possible. Here we show how to obtain the same guarantee for an arbitrary p-matchoid constraint, while reducing the number of passes to $O(p/\varepsilon)$.

[1] In [6] a bound of $O(\log p/\varepsilon^3)$ is stated. We note that there appears to be a small oversight in their analysis, arising from the fact that their convergence parameter κ in this case is $O(\varepsilon^3/p^4)$. In any case, it seems reasonable to assume that p is a small constant in most cases.

Algorithm 1 The multi-pass streaming local search algorithm.

procedure *MultipassLocalSearch*$(\alpha, \beta_1, \ldots, \beta_d)$
  $S_0 \leftarrow \emptyset$;
  for $i = 1$ **to** d **do**
    Let $\tilde{S}$ be the output of StreamingLocalSearch$(\alpha, \beta_i, S_{i-1})$;
    $S_i \leftarrow \tilde{S}$;
  return S_d;

procedure *StreamingLocalSearch*$(\alpha, \beta, S_{\text{init}})$
  $S \leftarrow S_{\text{init}}$;
  foreach x in the stream **do**
    if $x \in S_{\text{init}}$ **then** discard x;
    $C_x \leftarrow$ Exchange(x, S);
    if $f(x|S) \geq \alpha + (1+\beta)\sum_{c \in C_x} \nu(c, S)$ **then**
      $S \leftarrow S \backslash C_x + x$;
  return S;

Algorithm 2 The procedure Exchange(x, S).

procedure *Exchange*(x, S)
  $C_x \leftarrow \emptyset$;
  foreach $\mathcal{M}_\ell = (X_\ell, \mathcal{I}_\ell)$ with $x \in X_\ell$ **do**
    $S_\ell \leftarrow S \cap X_\ell$;
    if $S_\ell + x \notin \mathcal{I}$ **then**
      $T_\ell \leftarrow \{y \in S_\ell : S_\ell - y + x \in \mathcal{I}_\ell\}$;
      $C_x \leftarrow C_x + \arg\min_{t \in T_\ell} \nu(t, S)$;
  return C_x;

2 The main multi-pass streaming algorithm

Our main multi-pass algorithm is given by the procedure MultipassLocalSearch in Algorithm 1. We suppose that we are given a submodular function $f : 2^X \to \mathbb{R}_{\geq 0}$ and a p-matchoid constraint $\mathcal{M}^p = (\mathcal{I}^p, X)$ on X given as a collection of matroids $\{\mathcal{M}_i = (X_i, \mathcal{I}_i)\}$. Our procedure runs for d passes, each of which uses a modification of the algorithm of Chekuri, Gupta, and Quanrud [7], given as the procedure StreamingLocalSearch. In each pass, procedure StreamingLocalSearch maintains a current solution S, which is initially set to some S_{init}. Whenever an element $x \in S_{\text{init}}$ arrives again in the subsequent stream, the procedure simply discards x. For all other elements x, the procedure invokes a helper procedure Exchange, given formally in Algorithm 2, to find an appropriate set $C_x \subseteq S$ of up to p elements so that $S \backslash C_x + x \in \mathcal{I}$. It then exchanges x with C_x if this gives a significantly improved solution. This improvement is measured with respect to a set of auxiliary weights $\nu(x, S)$ maintained by the algorithm. For $u, v \in X$, let $u \prec v$ denote that "element u arrives before v" in the stream. Then, we define the *incremental value* of an element e with respect to a set T as

$$\nu(e, T) = f(e \mid \{t' \in T : t' \prec e\}) .$$

There is a slight difficulty here in that we must also define incremental values for the elements of S_{init}. To handle this difficulty, we in fact define $\prec$ with respect to a *pretend* stream ordering. Note that in all invocations of the procedure StreamingLocalSearch made by MultipassLocalSearch, the set S_{init} is either $\emptyset$ or the result of a previous application of StreamingLocalSearch. In our pretend ordering ($\prec$) all of S_{init} first arrives in the same relative pretend ordering as the previous pass, followed by all of $X \backslash S_{\text{init}}$ in the same order given by the stream X. We then define our incremental values with respect to this pretend stream ordering.

Using these incremental values, StreamingLocalSearch proceeds as follows. When an element $x \notin S_{\text{init}}$ arrives, StreamingLocalSearch computes a set of elements $C_x \subseteq S$ that can be exchanged for x. StreamingLocalSearch replaces C_x with x if and only if the marginal value $f(x \mid S)$ with respect to S is at least $(1+\beta)$ times larger than the sum of the current incremental values $\nu(c, S)$ of all elements $c \in C_x$ plus some threshold α, where $\alpha, \beta > 0$ are given as parameters. In this case, we say that the element x is *accepted*. Otherwise, we say that x is *rejected*. An element $x \in S$ that has been accepted may later be removed from S if $x \in C_y$ for some later element y that arrives in the stream. In this case we say that x is *evicted*.

The approximation ratio obtained by one pass of StreamingLocalSearch depends on the parameter β in two ways, which can be intuitively understood in terms of the standard analysis of the offline local search algorithm for the problem. Intuitively, if β is chosen to be too large, more valuable elements will be rejected upon arrival and so, in the offline setting, our solution would be only approximately locally optimal, leading to a deterioration of the guarantee by a factor of $(1+\beta)$. However, in the streaming setting, the algorithm only attempts to exchange an element upon its arrival, and so the final solution will not necessarily be even $(1+\beta)$-approximately locally optimal – an element x may be rejected because $f(x \mid S)$ is small when it arrives, but the processing of later elements in the stream can evict some elements of S. After these evictions, we could have $f(x \mid S)$ larger. The key observation in the analyses of [6, 7] is that the total value of these evicted elements – and so also the total increase in the marginal value of all rejected elements – can be bounded by $O(\frac{1}{\beta})$ times the final value of $f(S)$ at the end of the algorithm. Intuitively, if β is chosen to be too small, the algorithm will make more exchanges, evicting more elements, which may result in rejected elements being much more valuable with respect to the final solution. Selecting the optimal value of β thus requires balancing these two effects.

Here, we observe that this second effect depends only on the total value of those elements that were accepted *after* an element arrives. To use this observation, we measure the ratio $\delta = f(S_{\text{init}})/f(\tilde{S})$ between the value of the initial solution S_{init} of some pass of StreamingLocalSearch and the final solution $\tilde{S}$ produced by this pass. If δ is relatively small – and so one pass makes a lot of progress – then this pass gives us an improvement of δ^{-1} over the ratio already guaranteed by the previous pass since $f(\tilde{S}) = \delta^{-1} f(S_{\text{init}})$. On the other hand, if δ is relatively large – and so one pass does not make much progress – then the total increase in the value of our rejected elements can be bounded by $\frac{1-\delta}{\beta} f(\tilde{S})$, and so the potential loss due to only testing these elements at arrival is relatively small. Balancing these two effects allows us to set β smaller in each subsequent passes and obtain an improved guarantee.

We now turn to the analysis of our algorithm. Here we focus on a single pass of StreamingLocalSearch. For $T, U \subseteq X$ we let $f(T \mid U) \triangleq f(T \cup U) - f(U)$. Throughout, we use S to denote the current solution maintained by this pass (initially, $S = S_{\text{init}}$). The following key properties of incremental values will be useful in our analysis. We defer the proof to the Appendix.

▶ **Lemma 1.** *For any $T \subseteq U \subseteq X$,*
1. $\sum_{e \in T} \nu(e, T) = f(T) - f(\emptyset)$
2. $\nu(e, U) \leq \nu(e, T)$ *for all* $e \in T$.
3. $f(T \mid U \backslash T) \leq \sum_{t \in T} \nu(t, U)$
4. *At all times during the execution of* StreamingLocalSearch, $\nu(e, S) \geq \alpha$ *for all* $e \in S$.

Let A denote the set of elements *accepted* during the present pass. These are the elements which were present in the solution S at some previous time during the execution of this pass. Initially we have $A = S = S_{\text{init}}$ and whenever an element is added to S, during this pass we also add this element to A. Let $\tilde{A}$ and $\tilde{S}$ denote the sets of elements A and S at the end of this pass. Note that we regard all elements of S_{init} as having been accepted at the start of the pass. The following lemma follows from the analysis of Chekuri, Gupta, and Quanrud [7] in the single-pass setting. We give a complete, self-contained proof in Appendix A. Each element $e \in \tilde{A} \backslash \tilde{S}$ was accepted but later *evicted* by the algorithm. For any such evicted element, we let $\chi(e)$ denote the value of $\nu(e, S)$ at the moment that e was removed from S.

▶ **Lemma 2.** *Let $f : 2^X \to \mathbb{R}_{\geq 0}$ be a submodular function. Suppose $\tilde{S}$ is the solution produced at the end of one pass of* StreamingLocalSearch *and $\tilde{A}$ be the set of all elements accepted during this pass. Then,*

$$f(OPT \cup \tilde{A}) \leq (p + \beta p - \beta) \sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e) + (p + \beta p + 1) f(\tilde{S}) + k\alpha \,.$$

We now derive a bound for the summation $\sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e)$ (representing the value of *evicted* elements) in terms of the total gain $f(\tilde{S}) - f(S_{\text{init}})$ made by the pass, and also bound the total number of accepted elements in terms of $f(OPT)$.

▶ **Lemma 3.** *Let $f : 2^X \to \mathbb{R}_{\geq 0}$ be a submodular function. Suppose that $\tilde{S}$ is the solution produced at the end of one pass of* StreamingLocalSearch *and $\tilde{A}$ is the set of all elements accepted during this pass. Then, $|\tilde{A}| \leq f(OPT)/\alpha$ and*

$$\sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e) \leq \frac{1}{\beta} \left(f(\tilde{S}) - f(S_{\text{init}}) \right) .$$

Proof. We consider the quantity $\Phi(A) \triangleq \sum_{e \in A \backslash S} \chi(e)$. Suppose some element a with $C_a \neq \emptyset$ is added to S by the algorithm, evicting the elements of C_a. Then (as each element can be evicted only once) $\Phi(A)$ increases by precisely $\Delta \triangleq \sum_{e \in C_a} \chi(e)$. Let S_a^-, S_a^+ and A_a^-, A_a^+ be the sets S and A, respectively, immediately before and after a is accepted. Since a is accepted, we must have $f(a \mid S_a^-) \geq \alpha + (1 + \beta) \sum_{e \in C_a} \nu(e, S_a^-)$. Then,

$$\begin{aligned}
f(S_a^+) - f(S_a^-) &= f(S_a^- \backslash C_a + a) - f(S_a^-) \\
&= f(a \mid S_a^- \backslash C_a) - f(C_a \mid S_a^- \backslash C_a) \\
&\geq f(a \mid S_a^-) - f(C_a \mid S_a^- \backslash C_a) && \text{(by submodularity)} \\
&\geq f(a \mid S_a^-) - \sum_{e \in C_a} \nu(e, S_a^-) && \text{(by Lemma 1 (3))} \\
&\geq \alpha + (1 + \beta) \sum_{e \in C_a} \nu(e, S_a^-) - \sum_{e \in C_a} \nu(e, S_a^-) && \text{(since } a \text{ is accepted)} \\
&= \alpha + \beta \sum_{e \in C_a} \chi(e) \,. && \text{(by definition of } \chi(e)) \\
&= \alpha + \beta \Delta
\end{aligned}$$

It follows that whenever $\Phi(A)$ increases by Δ, $f(S)$ must increase by at least $\beta\Delta$. Initially, $\Phi(A) = 0$ and $f(S) = f(S_{\text{init}})$ and at the end of the algorithm, $\Phi(A) = \sum_{e\in\tilde{A}\setminus\tilde{S}} \chi(e)$ and $f(S) = f(\tilde{S})$. Thus, $\beta \sum_{e\in\tilde{A}\setminus\tilde{S}} \chi(e) \leq [f(\tilde{S}) - f(S_{\text{init}})]$.

It remains to show that $|\tilde{A}| \leq f(OPT)/\alpha$. For this, we note that the above chain of inequalities also implies that every time an element is accepted (and so $|A|$ increases by one), $f(S)$ also increases by at least α. Thus, we have $f(OPT) \geq f(\tilde{S}) \geq \alpha|\tilde{A}|$. ◀

Using Lemma 3 to bound the sum of exit values in Lemma 2 then immediately gives us the following guarantee for each pass performed in MultipassLocalSearch. In the i^{th} such pass, we will have $S_{\text{init}} = S_{i-1}$, $\tilde{S} = S_i$, and $\beta = \beta_i$. We let A_i denote the set of $\tilde{A}$ of all elements accepted during this particular pass.

▶ **Lemma 4.** *Let $f : 2^X \to \mathbb{R}_{\geq 0}$ be a submodular function. Consider the i^{th} pass of StreamingLocalSearch performed by MultipassLocalSearch, and let A_i be the set of all elements accepted during this pass. Then, $|A_i| \leq f(OPT)/\alpha$ and*

$$f(OPT \cup A_i) \leq (p/\beta_i + p - 1)\,[f(S_i) - f(S_{i-1})] + (p + p\beta_i + 1)f(S_i) + k\alpha\,.$$

3 Analysis of the multipass algorithm for monotone functions

We now show how to use Lemma 4 together with a careful selection of parameters α and $\beta_1, \ldots, \beta_d$ to derive guarantees for the solution $f(S_i)$ produced after the i^{th} pass made in MultipassLocalSearch. Here, we consider the case that f is a *monotone* function. In this case, we have $f(OPT) \geq f(OPT \cup A_i)$ for all i. We set $\alpha = 0$ in each pass. In the first pass, we will set $\beta_1 = 1$. Then, since $S_0 = \emptyset$ Lemma 4 immediately gives:

$$f(OPT) \leq f(OPT \cup A_1) \leq (2p - 1)\,[f(S_1) - f(\emptyset)] + (2p + 1)f(S_1) = 4pf(S_1)\,. \tag{1}$$

For passes $i > 1$, we use the following, which relates the approximation guarantee obtained in this pass to that from the previous pass.

▶ **Theorem 5.** *For $i > 1$, suppose that $f(OPT) \leq \gamma_{i-1} \cdot f(S_{i-1})$ and define $\delta_i = \frac{f(S_{i-1})}{f(S_i)}$. Then,*

$$f(OPT) \leq \min\left\{\gamma_{i-1}\delta_i, (\tfrac{p}{\beta_i} + p - 1)(1 - \delta_i) + p + \beta_i p + 1\right\} \cdot f(S_i) + k\alpha\,.$$

Proof. From the definition of γ_{i-1} and δ_i, we have:

$$f(OPT) \leq \gamma_{i-1} f(S_{i-1}) = \gamma_{i-1}\delta_i f(S_i)\,.$$

On the other hand, $f(S_i) - f(S_{i-1}) = (1 - \delta_i)f(S_i)$. Thus, Lemma 4 gives:

$$f(OPT) \leq [(p/\beta_i + p - 1)\,(1 - \delta_i) + p + \beta_i p + 1]\,f(S_i) + k\alpha\,.$$ ◀

Now, we observe that for any fixed guarantee γ_{i-1} from the previous pass, $\gamma_{i-1}\delta_i$ is an increasing function of δ_i and $((p/\beta_i + p - 1)(1 - \delta_i) + p + \beta_i p + 1)$ is an decreasing function of δ_i. Thus, the guarantee we obtain in Theorem 5 is always at least as good as that obtained when these two values are equal. Setting:

$$\gamma_{i-1}\delta_i = (\tfrac{p}{\beta_i} + p - 1)(1 - \delta_i) + p + \beta_i p + 1,$$

and solving for δ_i gives us:

$$\delta_i = \frac{p(1 + \beta_i)^2}{p + \beta_i(\gamma_{i-1} - 1 + p)}\,. \tag{2}$$

In the following analysis, we consider this value of δ_i since the guarantee given by Theorem 5 will always be no worse than that given by this value. The analysis for a single matroid constraint follows from our results for p-matchoids, but the analysis and parameter values obtained are much simpler, so we present it separately, first.

▶ **Theorem 6.** *Suppose we run Algorithm 1 for an arbitrary matroid constraint and monotone submodular function f, with $\beta_i = \frac{1}{i}$. Then $2(1+\frac{1}{i})f(S_i) \geq f(OPT)$ for all $i > 0$. In particular, after $i = \frac{2}{\varepsilon}$ passes, $(2+\varepsilon)f(S_i) \geq f(OPT)$.*

Proof. Let γ_i be the guarantee for our algorithm after i passes. We show, by induction on i, that $\gamma_i \leq \frac{2(i+1)}{i}$. For $i = 1$, we have $\beta_1 = 1$ and so from (1) we have $\gamma_1 = 4$, as required. For $i > 1$, suppose that $\gamma_{i-1} \leq \frac{2i}{i-1}$. Since $p = 1$ and $\beta_i = 1/i$ (2) gives:

$$\delta_i \leq \frac{(1+\frac{1}{i})^2}{1+\frac{1}{i}(\frac{2i}{i-1})} = \frac{\frac{(i+1)^2}{i^2}}{\frac{(i-1)+2}{i-1}} = \frac{(i-1)(i+1)}{i^2}\,.$$

Thus, by Theorem 5, the i^{th} pass of our algorithm has guarantee γ_i satisfying:

$$\gamma_i \leq \gamma_{i-1}\delta_i \leq \frac{2i}{i-1}\frac{(i-1)(i+1)}{i^2} = \frac{2(i+1)}{i}\,,$$

as required. ◀

▶ **Theorem 7.** *Suppose we run Algorithm 1 for an arbitrary p-matchoid constraint and monotone submodular function f, β_1 and*

$$\beta_i = \frac{\gamma_{i-1}-1-p}{\gamma_{i-1}-1+p}$$

for $i > 1$, where γ_i is given by the recurrence $\gamma_1 = 4p$ and

$$\gamma_i = 4p\frac{\gamma_{i-1}(\gamma_{i-1}-1)}{(\gamma_{i-1}-1+p)^2}$$

for $i > 1$. Then $\left(p+1+\frac{4p}{i}\right)f(S_i) \geq f(OPT)$ for all $i > 0$. In particular, after $i = \frac{4p}{\varepsilon}$ passes, $(p+1+\varepsilon)f(S_i) \geq f(OPT)$.

Proof. We first show that approximation guarantee of our algorithm after i passes is given by γ_i. Setting $\beta_1 = 1$, we obtain $\gamma_1 = 4p$ from (1), agreeing with our definition. For passes $i > 1$, let $\beta_i = \frac{\gamma_{i-1}-1-p}{\gamma_{i-1}-1+p}$. As in the case of matroid constraint, Theorem 5 implies that the guarantee for pass i will be at most $\delta_i\gamma_{i-1}$, where δ_i is chosen to satisfy (2). Specifically, if we set

$$\delta_i = \frac{p\left(1+\frac{\gamma_{i-1}-1-p}{\gamma_{i-1}-1+p}\right)^2}{p+\frac{\gamma_{i-1}-1-p}{\gamma_{i-1}-1+p}(\gamma_{i-1}-1+p)} = \frac{p\left(\frac{2(\gamma_{i-1}-1)}{\gamma_{i-1}-1+p}\right)^2}{\gamma_{i-1}-1} = \frac{4p(\gamma_{i-1}-1)}{(\gamma_{i-1}-1+p)^2}\,,$$

then we have $\delta_i\gamma_{i-1} = \gamma_i$.

We now show by induction on i that $\gamma_i \leq p+1+\frac{4p}{i}$. In the case $i = 1$, we have $\gamma_1 = 4p$ and the claim follows immediately from $p \geq 1$. In the general case $i > 0$, and we may assume without loss of generality that $\gamma_{i-1} \geq 1$. Otherwise the theorem holds immediately, as each subsequent pass can only increase the value of the solution. Then, we note (as shown in Appendix B) that for $p \geq 1$ and $\gamma_{i-1} \geq 1$, γ_i is an increasing function of γ_{i-1}. By the induction hypothesis, $\gamma_{i-1} \leq p+1+\frac{4p}{i-1}$. Therefore:

$$\gamma_i \leq \frac{4p\left(p+1+\frac{4p}{i-1}\right)\left(p+\frac{4p}{i-1}\right)}{\left(2p+\frac{4p}{i-1}\right)^2} \leq p+1+\frac{4p}{i}\,,$$

as required. The last inequality above follows from straightforward but tedious algebraic manipulations, which can be found in Appendix B. ◀

Algorithm 3 The randomized multi-pass streaming algorithm.

procedure MultipassRandomizedLocalSearch($\alpha, \beta_1, \ldots, \beta_d, m$)
  $S_0 \leftarrow \emptyset, S'_0 \leftarrow \emptyset$;
  for $i = 1$ **to** d **do**
    Let $(\tilde{S}, S')$ be the output of RandomizedLocalSearch($S_{i-1}, \alpha, \beta_i, m$);
    $S_i \leftarrow \tilde{S}$, $S'_i \leftarrow \arg\max\{f(S'_{i-1}), f(S')\}$;
  return $\bar{S} = \arg\max\{f(S_d), f(S'_d)\}$;

procedure RandomizedLocalSearch($S_{\text{init}}, \alpha, \beta, m$)
  $S \leftarrow S_{\text{init}}$; $B \leftarrow \emptyset$;
  foreach x in the stream **do**
    if $f(x \mid S) \geq \alpha + (1+\beta)\sum_{e \in C_x} \nu(e, S)$ **then**
      $B \leftarrow B + x$;
    if $|B| = m$ **then**
      $x \leftarrow$ uniformly random element from B;
      $C_x \leftarrow$ Exchange(x, S);
      $B \leftarrow B - x$; $S \leftarrow S + x - C_x$;
      foreach x' in B **do**
        $C_{x'} \leftarrow$ Exchange(x', S);
        if $f(x' \mid S) < \alpha + (1+\beta)\sum_{e \in C_{x'}} \nu(e, S)$ **then**
          $B \leftarrow B - x'$;
  $S' \leftarrow$ Offline(B);
  return (S, S');

4 A multi-pass algorithm for general submodular functions

In this section, we show that the guarantees for monotone submdodular maximization can be extended to non-monotone submodular maximization even when dealing with multiple passes. Our main algorithm is given by procedure MultipassRandomizedLocalSearch in Algorithm 3. In each pass, it calls a procedure RandomizedLocalSearch, which is an adaptation of StreamingLocalSearch, to process the stream. Note that each such pass produces a pair of feasible solutions S and S', which we now maintain throughout MultipassRandomizedLocalSearch. The set S is maintained similarly as before and gradually improves by exchanging "good" elements into a solution throughout the pass. The set S' will be maintained by considering the best output of an offline algorithm that we run after each pass as described in more detail below.

To deal with non-monotone submodular functions, we will limit the probability of elements being added to S. Instead of exchanging good elements on arrival, we store them in a buffer B of size m. When the buffer becomes full, an element is chosen uniformly at random and added to S. Adding a new element to the current solution may affect the quality of the remaining elements in the buffer and thus we need to re-evaluate them and remove the elements that are no longer good. As before, we let A denote the set of elements that were previously added to S during the current pass of the algorithm. Note that we do not consider an element to be accepted until it has actually been added to S from the buffer. For any fixed set of random choices, the execution of RandomizedLocalSearch can be considered

as the execution of StreamingLocalSearch on the following stream: we suppose that an element x arrives whenever it is selected from the buffer and accepted into S. All elements that are discarded from the buffer after accepting x then arrive, and will also be rejected by StreamingLocalSearch. Any elements remaining in the buffer after the execution of the algorithm do not arrive in the stream. Applying Lemma 4 with respect to this pretend stream ordering allows us to bound $f(\tilde{S})$ with respect to $f(OPT \setminus B)$ (that is, the value of the part of OPT that does not remain in the buffer B) after a single pass of RandomizedLocalSearch. Formally, let $\tilde{B}_i$ be the value of the buffer after the i^{th} pass of our algorithm. Then, applying Lemma 4 to the set $OPT \setminus \tilde{B}_i$, and taking expectation, gives:

$$\mathbb{E}[f(A_i \cup (OPT \setminus \tilde{B}_i))] \leq (p/\beta + p - 1)\,(\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})]) + (p + \beta p + 1)\,\mathbb{E}[f(S_i)] + \alpha k\,. \quad (3)$$

In order to bound the value of the elements in $\tilde{B}_i$, we apply any offline $\bar{\gamma}_{\text{off}}$-approximation algorithm Offline to the buffer at the end of the pass to obtain a solution S'. In MultipassRandomizedLocalSearch, we then remember the best such offline solution S'_i computed across the first i passes. Then, in the i^{th} pass, we have

$$\mathbb{E}[f(OPT \cap \tilde{B}_i)] \leq \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S')] \leq \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)]\,. \quad (4)$$

From submodularity of f and $A_i \cap \tilde{B}_i = \emptyset$ we have $f(A_i \cup OPT) \leq f(A_i \cup (OPT \setminus \tilde{B}_i)) + f(OPT \cap \tilde{B}_i)$. Thus, combining (3) and (4) we have:

$$\mathbb{E}[f(A_i \cup OPT)] \leq (p/\beta + p - 1)\,(\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})]) + (p + \beta p + 1)\,\mathbb{E}[f(S_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)] + \alpha k\,. \quad (5)$$

To relate the right-hand side to $f(OPT)$ we use the following result from Buchbinder et al. [4]:

▶ **Lemma 8** (Lemma 2.2 in [4]). *Let $f\colon 2^X \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function. Suppose that A is a random set where no element $e \in X$ appears in A with probability more than p. Then, $\mathbb{E}[f(A)] \geq (1-p)\,f(\emptyset)$. Moreover, for any set $Y \subseteq X$, it follows that $\mathbb{E}[f(Y \cup A)\,] \geq (1-p) f(Y)$.*

We remark that a similar theorem also appeared earlier in Feige, Mirrokni, and Vondrák [10] for a random set that contains each element *independently* with probability *exactly* p. Here, the probability that an element occurs in A_i is delicate to handle because such an element may either originate from the starting solution S_{i-1} or be added during the pass. Thus, we use a rougher estimate. By definition $A_i \subseteq A_i \cup A_{i-1} \cup \ldots \cup A_1$. Thus, $\Pr[e \in A_i] \leq \Pr[e \in A_i \cup \ldots \cup A_1]$. The number of selections during the j^{th} pass is at most $|A_j|$ and by Lemma 4 (applied to the set $OPT \setminus \tilde{B}_j$ due to our pretend stream ordering in each pass j), $|A_j| \leq f(OPT \setminus \tilde{B}_j)/\alpha \leq f(OPT)/\alpha$ in any pass. Here, the second inequality follows from the optimality of OPT, and the fact that any subset of the feasible solution OPT is also feasible for our p-matchoid constraint. Thus, the total number of selections in the first i passes at most $\sum_{j=1}^{i} |A_j| \leq i \cdot f(OPT)/\alpha$. We select an element only when the buffer is full, and each selection is made independently and uniformly at random from the buffer. Thus, the probability that any given element is selected when the algorithm makes a selection is at most $1/m$ and by a union bound, $\Pr[e \in A_i \cup \ldots \cup A_1] \leq i \cdot f(OPT)/(m\alpha)$. Let d be the number of passes that the algorithm makes and suppose we set $\alpha = \varepsilon f(OPT)/2k$ (in Appendix C we show that this can be accomplished approximately by guessing $f(OPT)$, which can be done at the expense of an extra factor $O(\log k)$ space). Finally, let $m = 4dk/\varepsilon^2$. Then, applying Lemma 8, after $i \leq d$ passes we have:

$$\mathbb{E}[f(A_i \cup OPT)] \geq (1 - d \cdot f(OPT)/(m\alpha))\,f(OPT) \geq (1 - \varepsilon/2)\,f(OPT)\,. \quad (6)$$

Our definition of α also implies that $\alpha k \leq (1-\varepsilon/2)f(OPT)$. Using this and (6) in (5), we obtain:

$$(1-\varepsilon)f(OPT) \leq (p/\beta + p - 1)(\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})]) + (p + \beta p + 1)\,\mathbb{E}[f(S_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)]\,. \quad (7)$$

As we show in Appendix C, the rest of the analysis then follows similarly to that in Section 3, using the fact that $f(\bar{S}) = \max\{f(S_d), f(S'_d)\}$.

▶ **Theorem 9.** *Let $\mathcal{M}^p = (X, \mathcal{I})$ be a p-matchoid of rank k and let $f\colon 2^X \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function. Suppose there exists an algorithm for the offline instance of the problem with approximation factor $\bar{\gamma}_{\text{off}}$. For any $\varepsilon > 0$, the randomized streaming local-search algorithm returns a solution $\bar{S} \in \mathcal{I}$ such that*

$$f(OPT) \leq (p + 1 + \bar{\gamma}_{\text{off}} + O(\varepsilon))\,\mathbb{E}[f(\bar{S})]$$

using a total space of $O\Big(\frac{p^3 k \log_2 k}{\varepsilon^3}\Big)$ and $O\big(\frac{p}{\varepsilon}\big)$-passes.

References

1 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *Proc. ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 671–680, 2014.

2 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1497–1514, 2013.

3 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.

4 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1433–1452, 2014.

5 Gruia Calinescu, 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.

6 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.

7 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 318–330, 2015.

8 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.

9 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

10 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

11 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 732–742, 2018.

12 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *Proc. IEEE Symposium on Foundations of Computer Science, (FOCS)*, pages 570–579, 2011.

13 Moran Feldman, Joseph (Seffi) Naor, Roy Schwartz, and Justin Ward. Improved approximations for k-exchange systems. In *Proc. European Symposium on Algorithms (ESA)*, pages 784–798, 2011.

14 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 1363–1374, 2020.

15 M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions II. *Mathematical Programming Study*, 8:73–87, 1978.

16 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1098–1116, 2011.

17 Chien-Chung Huang and Naonori Kakimura. Multi-pass streaming algorithms for monotone submodular function maximization. *CoRR*, abs/1802.06212, 2018. `arXiv:1802.06212`.

18 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4):795–806, 2010.

19 Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Proc. Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 912–920, 2010.

20 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63(7):1595–1619, 2019.

21 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Aamin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proc. International Conference on Machine Learning (ICML)*, pages 1358–1367, 2016.

22 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, pages 1812–1818, 2015.

23 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, pages 1379–1386, 2018.

24 G L Nemhauser and L A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

25 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrović, Amir Zandieh, Aida Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *Proc. International Conference on Machine Learning (ICML)*, pages 3826–3835, 2018.

A Proof of Lemma 2

Here, we give a self-contained analysis of the single-pass algorithm of Chekuri, Gupta, and Quanrud [7], corresponding to Algorithm 1 initialized with $S_{\text{init}} = \emptyset$. First, we prove Lemma 1, which concerns properties of the incremental values maintained by Algorithm 1.

▶ **Lemma 1.** *For any $T \subseteq U \subseteq X$,*

1. $\sum_{e \in T} \nu(e, T) = f(T) - f(\emptyset)$
2. $\nu(e, U) \leq \nu(e, T)$ *for all* $e \in T$.
3. $f(T \mid U \backslash T) \leq \sum_{t \in T} \nu(t, U)$
4. *At all times during the execution of* StreamingLocalSearch, $\nu(e, S) \geq \alpha$ *for all* $e \in S$.

Proof. Property (1) follows directly from the telescoping summation

$$\sum_{e\in T}\nu(e,T)=\sum_{e\in T}[f(e\cup\{t'\in T:t'\prec e\})-f(\{t'\in T:t'\prec e\}]=f(T)-f(\emptyset).$$

Property (2) follows from submodularity since $T\subseteq U$ implies that $\{t'\in T:t'\prec e\}\subseteq\{t'\in U:t'\prec e\}$.

For property (3), we note that:

$$f(T\mid U\backslash T)=\sum_{t\in T}f(t\mid U\backslash T\cup\{t'\in T:t'\prec t\})\leq\sum_{t\in T}f(t\mid\{u'\in U:u'\prec t\})=\sum_{t\in T}\nu(t,U)\,,$$

where the first equation follows from a telescoping summation, and the inequality follows from submodularity, since $\{u'\in U:u'\prec t\}\subseteq U\setminus T\cup\{t'\in T:t'\prec t\}$.

We prove property (4) by induction on the stream of elements arriving. Initially $S=\emptyset$. Thus, the first time that any element x is accepted, we must have $C_x=\emptyset$ and so $f(x\mid S)\geq\alpha$. After this element is accepted, we have $\nu(x,S)=\nu(x,\{x\})=f(x\mid\emptyset)=\alpha$. Proceeding inductively, then, let S_x^- and S_x^+ be the set of elements in S before and after some new element x arrives and is processed by Algorithm 1, and suppose that $\nu(s,S_x^-)\geq\alpha$ for all $s\in S_x^-$. Then, if x is rejected, we have $S_x^+=S_x^-$ and so $\nu(s,S_x^+)=\nu(s,S_x^-)\geq\alpha$ for all $s\in S_x^+$. If x is accepted, then $S_x^+=S\backslash C_x+x$ and $f(x\mid S_x^-)\geq\alpha+(1+\beta)\sum_{e\in C_x}\nu(e,S_x^-)$. Thus,

$$\nu(x,S_x^+)\geq f(x\mid S_x^+-x)\geq f(x\mid S_x^-)\geq\alpha+(1+\beta)|C_x|\alpha\geq\alpha\,,$$

where the first inequality follows from property (2) of the lemma, the second from submodularity, and the third from the induction hypothesis and the assumption that x is accepted. For any other $s\in S_x^+$, we have $\{t'\in S\backslash C_x:t'\prec s\}\subseteq\{t'\in S:t'\prec s\}$ and so by property (3) of the lemma, $\nu(s,S_x^+)\geq\nu(s,S_x^-)\geq\alpha$, as required. ◀

In our analysis we will use the following structural lemma from Chekuri et al. [7] (here, restated in our notation). This lemma applies to the execution of our algorithm StreamingLocalSearch when $S_{\text{init}}=\emptyset$, and so no element is discarded upon arrival due to $x\in S_{\text{init}}$. However, we note that the execution of our algorithm is in fact exactly the same as this algorithm executed on the pretend stream ordering introduced in Section 2 to define the incremental values ν. Specifically, in each pass of our algorithm, the set S_{init} is a feasible solution produced by the preceding pass and in the pretend stream ordering, all elements of S_{init} arrive in our pretend ordering in the same relative (pretend) order as this preceding pass. It follows that whenever $x\in S_{\text{init}}$ arrives in our pretend ordering for the present pass, we have $C_x=\emptyset$ and $\nu(x,S)=\nu(x,S_{\text{init}})\geq\alpha$ by Lemma 1 (4), since x was present in the feasible solution $S=S_{\text{init}}$ at the end of the preceding pass. Thus, each $x\in S_{\text{init}}$ will first be accepted in our pretend stream ordering, and then the rest of $X\backslash S_{\text{init}}$ is processed, exactly as in StreamingLocalSearch.

Recall that we let $\tilde{A}$ be the set of all elements that were accepted by this pass of StreamingLocalSearch (and so at some point appeared in S). For each element $x\in X$, we let S_x^- be the current set S at the moment that x arrives and S_x^+ the set after x is processed. For an element e that is accepted but later *evicted* from S, let $\chi(e)$ be the incremental value $\nu(e,S)$ of e at the moment that e was evicted.

▶ **Lemma 10** (Lemma 9 of [7]). *Let $T\in\mathcal{I}$ be a feasible solution disjoint from $\tilde{A}$, and $\tilde{S}$ be the output of the streaming algorithm. There exists a mapping $\varphi:T\to 2^{\tilde{A}}$ such that:*

1. *Every $s \in \tilde{S}$ appears in the set $\varphi(t)$ for at most p choices of $t \in T$.*
2. *Every $e \in \tilde{A} \backslash \tilde{S}$ appears in the set $\varphi(t)$ for at most $p-1$ choices of $t \in T$.*
3. *For each $t \in T$:*

$$\sum_{c \in C_t} \nu(c, S_t^-) \leq \sum_{e \in \varphi(t) \backslash \tilde{S}} \chi(e) + \sum_{s \in \varphi(t) \cap \tilde{S}} \nu(s, \tilde{S}) \,.$$

Using this charging argument, we can now prove Lemma 2 directly.

▶ **Lemma 2.** *Let $f : 2^X \to \mathbb{R}_{\geq 0}$ be a submodular function. Suppose $\tilde{S}$ is the solution produced at the end of one pass of* StreamingLocalSearch *and $\tilde{A}$ be the set of all elements accepted during this pass. Then,*

$$f(OPT \cup \tilde{A}) \leq (p + \beta p - \beta) \sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e) + (p + \beta p + 1) f(\tilde{S}) + k\alpha \,.$$

Proof. Let $R = OPT \backslash \tilde{A}$. Since $S_r^- \subseteq \tilde{A}$ for all r, submodularity of f implies that

$$\sum_{r \in R} f(r \mid S_r^-) \geq \sum_{r \in R} f(r \mid \tilde{A}) \geq f(R \cup \tilde{A}) - f(\tilde{A}) = f(OPT \cup \tilde{A}) - f(\tilde{A}) \,. \tag{8}$$

For any $r \in R$, since r was rejected upon arrival,

$$f(r \mid S_r^-) \leq (1 + \beta) \sum_{c \in C_r} \nu(c, S_r^-) + \alpha \,. \tag{9}$$

Thus, applying Lemma 10 we obtain:

$$\begin{aligned}
\sum_{r \in R} f(r \mid S_r^-) &\leq (1+\beta) \sum_{r \in R} \sum_{c \in C_r} \nu(c, S_r^-) + k\alpha && \text{(by (9) and } |R| \leq k) \\
&\leq \sum_{r \in R} (1+\beta) \left[\sum_{e \in \varphi(r) \backslash \tilde{S}} \chi(e) + \sum_{s \in \varphi(r) \cap \tilde{S}} \nu(s, \tilde{S}) \right] + k\alpha && \text{(by Lemma 10 (3))} \\
&\leq (1+\beta) \left[(p-1) \sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e) + p \sum_{s \in \tilde{S}} \nu(s, \tilde{S}) \right] + k\alpha && \text{(by Lemma 10 (1) and (2))}
\end{aligned}$$

where in the last inequality we have also used Lemma 1 (4), which implies that each $\chi(e)$ and $\nu(s, \tilde{S})$ is non-negative. Combining the above inequality with (8), we obtain

$$f(OPT \cup \tilde{A}) \leq (1+\beta) \left[(p-1) \sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e) + p \sum_{s \in \tilde{S}} \nu(s, \tilde{S}) \right] + f(\tilde{A}) + k\alpha \,. \tag{10}$$

We now bound $f(\tilde{A})$ in terms of the values $\nu(s, \tilde{S})$ and $\chi(e)$. Since $S \subseteq \tilde{A}$ at all times during the algorithm, and $\chi(e) = \nu(e, S)$ at the moment e was evicted, we have $\chi(e) \geq \nu(e, \tilde{A})$ by Lemma 1 (2). Thus,

$$f(\tilde{A}) - f(\emptyset) = \sum_{a \in \tilde{A}} \nu(a, \tilde{A}) = \sum_{s \in \tilde{S}} \nu(s, \tilde{A}) + \sum_{e \in \tilde{A} \backslash \tilde{S}} \nu(e, \tilde{A}) \leq \sum_{s \in \tilde{S}} \nu(s, \tilde{S}) + \sum_{e \in \tilde{A} \backslash \tilde{S}} \chi(e) \,, \tag{11}$$

where the first equation follows from Lemma 1 (1), and the last inequality follows from Lemma 1 (2).

Combining (10) and (11) we have:

$$f(OPT \cup \tilde{A}) \leq ((1+\beta)(p-1)+1) \sum_{e \in \tilde{A} \setminus \tilde{S}} \chi(e) + ((1+\beta)p+1) \sum_{e \in \tilde{S}} \nu(s, \tilde{S}) + f(\emptyset) + k\alpha$$
$$= (p + p\beta - \beta) \sum_{e \in \tilde{A} \setminus \tilde{S}} \chi(e) + (p + \beta p + 1) \sum_{s \in \tilde{S}} \nu(s, \tilde{S}) + f(\emptyset) + k\alpha \,. \tag{12}$$

By Lemma 1 (1), we have the following bound for the second summation in (12):

$$(p + \beta p + 1) \sum_{e \in \tilde{S}} \nu(e, \tilde{S}) + f(\emptyset) = (p + \beta p + 1)[f(\tilde{S}) - f(\emptyset)] + f(\emptyset) \leq (p + \beta p + 1) f(\tilde{S}) \,.$$

Combining this and (12) we obtain:

$$f(OPT \cup \tilde{A}) \leq (p + p\beta - \beta) \sum_{e \in \tilde{A} \setminus \tilde{S}} \chi(e) + (p + \beta p + 1) f(\tilde{S}) + k\alpha$$ ◀

B Calculations for the proof of Theorem 7

We recall that

$$\gamma_i = \gamma_{i-1}\delta_i = \frac{4p\gamma_{i-1}(\gamma_{i-1}-1)}{(\gamma_{i-1}-1+p)^2} \,.$$

Then, to see that γ_i is an increasing function of γ_{i-1} for $p \geq 1$ and $\gamma_{i-1} \geq 1$, we note that:

$$\frac{d}{d\gamma_{i-1}}\gamma_i = \frac{4p(\gamma_{i-1}-1) + 4p\gamma_{i-1}}{(\gamma_{i-1}-1+p)^2} - \frac{8p\gamma_i(\gamma_{i-1}-1)}{(\gamma_{i-1}-1+p)^3}$$

$$= \frac{4p(\gamma_{i-1}-1)(\gamma_{i-1}-1+p) + 4p\gamma_i(\gamma_{i-1}-1+p) - 8p\gamma_{i-1}(\gamma_{i-1}-1)}{(\gamma_{i-1}-1+p)^3} \geq 0$$

Where the final inequality follows from $\gamma_{i-1} \geq 1$ and $p \geq 1$.

We now verify the following inequality used at the end of Theorem 7:

$$\frac{4p\left(p+1+\frac{4p}{i-1}\right)\left(p+\frac{4p}{i-1}\right)}{\left(2p+\frac{4p}{i-1}\right)^2} \leq p+1+\frac{4p}{i} \,.$$

Rearranging both sides and placing over a common denominator gives:

$$\frac{4p\left(p+1+\frac{4p}{i-1}\right)\left(p+\frac{4p}{i-1}\right)}{\left(2p+\frac{4p}{i-1}\right)^2} = \frac{4p\left((p+1)(i-1)+4p\right)\left(p(i-1)+4p\right)}{\left(2p(i-1)+4p\right)^2}$$
$$= \frac{4p\left((p+1)(i-1)+4p\right)\left(p(i-1)+4p\right)}{\left(2p(i+1)\right)^2}$$
$$= \frac{\left((i-1)(p+1)+4p\right)(i+3)}{(i+1)^2}$$
$$= \frac{(i-1)(i+3)i(p+1)+i(i+3)4p}{i(i+1)^2}$$
$$= \frac{\left(i^2+2i-3\right)i(p+1)+(i^2+3i)4p}{i(i+1)^2}$$

and

$$\begin{aligned} p+1+\tfrac{4p}{i} &= \frac{(p+1)i+4p}{i} \\ &= \frac{i(i+1)^2(p+1)+(i+1)^2 4p}{i(i+1)^2} \\ &= \frac{\left(i^2+2i+1\right)i(p+1)+\left(i^2+2i+1\right)4p}{i(i+1)^2}. \end{aligned}$$

Then, since $p \geq 1$ and $i \geq 1$,

$$\left(p+1+\tfrac{4p}{i}\right) - \frac{4p\left(p+1+\frac{4p}{i-1}\right)\left(p+\frac{4p}{i-1}\right)}{\left(2p+\frac{4p}{i-1}\right)^2} = \frac{4i(p+1)-4(i-1)p}{i(i+1)^2} \geq 0.$$

C Additional Details for the Non-Monotone Case

C.1 Guessing the value of $f(OPT)$

Guessing the value of $f(OPT)$ is a common technique in streaming submodular function maximization. Badanidiyuru et al. [1] showed how to approximate $f(OPT)$ within a constant factor using $O(\log(k))$ space in a single pass. To avoid extra complications, we show how to guess $f(OPT)$ in two passes and refer the reader to [1] for an approximation of $f(OPT)$ on the fly. Let $\tau = \max_{e \in X} f(e)$. Using submodularity, it is easy to see that $\tau \leq f(OPT) \leq k\tau$. Consider the set

$$\Lambda = \left\{2^i \mid i \in \mathbb{Z},\ \tau \leq 2^i \leq k \cdot \tau\right\}.$$

Then there exists a value $\lambda \in \Lambda$ such that $\frac{f(OPT)}{2} \leq \lambda \leq f(OPT)$. Setting the parameter $\alpha = \varepsilon\lambda/(2k)$, we get that $\alpha \in [\varepsilon f(OPT)/4k; \varepsilon f(OPT)/2k]$. The defined range of α is sufficient for the analysis[2]. Unfortunately, it is still not possible to know which $\lambda \in \Lambda$ satisfies the property. However, it suffices to run the randomized local-search algorithm for every $\lambda \in \Lambda$ in parallel and output the best solution of all the copies. This operation increases the space complexity by a multiplicative $O(\log_2 k)$ factor, and adds one additional pass to find τ.

C.2 Proof of Theorem 12

Here we give a full proof of the following theorem from Section 4:

▶ **Theorem 9.** *Let $\mathcal{M}^p = (X, \mathcal{I})$ be a p-matchoid of rank k and let $f\colon 2^X \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function. Suppose there exists an algorithm for the offline instance of the problem with approximation factor $\bar{\gamma}_{\text{off}}$. For any $\varepsilon > 0$, the randomized streaming local-search algorithm returns a solution $\bar{S} \in \mathcal{I}$ such that*

$$f(OPT) \leq (p+1+\bar{\gamma}_{\text{off}}+O(\varepsilon))\,\mathbb{E}[f(\bar{S})]$$

using a total space of $O\left(\frac{p^3 k \log_2 k}{\varepsilon^3}\right)$ and $O\left(\frac{p}{\varepsilon}\right)$-passes.

[2] Equation (6) and the bound $\alpha k \leq \varepsilon f(OPT)$ are where we need the exact value of α, using upper and lower bounds for α yield the same result up to the hidden constant in the term $O(\epsilon)$.

In the same spirit as in Section 3, we show that we can derive a guarantee with respect to the solution $\mathbb{E}[f(S_i)]$ produced after the i^{th} pass even when the function is non-monotone. In fact, we show that the analysis of the non-monotone case reduces to the monotone case as shown in the following theorem.

▶ **Theorem 11.** *Let f be a non-negative submodular function. Assume $\alpha = \varepsilon f(OPT)/2k$ and let $d \geq i > 1$. Suppose that at the start of the i^{th} iteration of the randomized local-search algorithm with a buffer of size $m = 4dk/\varepsilon^2$ we have $(1-\varepsilon)f(OPT) \leq \gamma_{i-1}\,\mathbb{E}[f(S_{i-1})] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_{i-1})]$. Then,*

$$(1-\varepsilon)f(OPT) \leq \min\left\{\gamma_{i-1}\delta_i, \left(\frac{p}{\beta_i} + p - 1\right)(1-\delta_i) + p + \beta_i p + 1\right\} \cdot \mathbb{E}[f(S_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)],$$

where $\delta_i = \frac{\mathbb{E}[f(S_{i-1})]}{\mathbb{E}[f(S_i)]}$.

Proof. From the definition of γ_{i-1} and δ_i, it follows that,

$$(1-\varepsilon)f(OPT) \leq \gamma_{i-1}\,\mathbb{E}[f(S_{i-1})] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_{i-1})] \leq \gamma_{i-1}\delta_i\,\mathbb{E}[f(S_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)]\,. \quad (13)$$

where in the last inequality we have used the definition of δ_i and the fact that $f(S'_i) \geq f(S'_{i-1})$, which follows from the way S'_i is defined in Algorithm 3.

On the other hand, $\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})] = (1-\delta_i)\,\mathbb{E}[f(S_i)]$. Thus, by (7) we also have:

$$\begin{aligned}
&(1-\varepsilon)f(OPT)\\
&\quad\leq \left(\tfrac{p}{\beta_i} + p - 1\right)(\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})]) + (p + \beta p + 1)\,\mathbb{E}[f(S_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)]\\
&\quad= \left(\left(\tfrac{p}{\beta_i} + p - 1\right)(1-\delta_i) + p + \beta_i p + 1\right)\mathbb{E}[f(S_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)]\,. \qquad (14)
\end{aligned}$$

Since the right-hand side of equation 13 is an increasing function of δ_i and the right-hand side of equation 14 is a decreasing function of δ_i, the guarantee we obtain is always at least as good as that obtained when these two values are equal. ◀

As in the monotone case, the lemma enables us to derive values of β so as to minimize the value of the approximation ratio. The following follows directly from the same calculations as in Section 3 and Appendix B.

▶ **Theorem 12.** *Suppose we run Algorithm 3 with a buffer of size $m = 4dk/\varepsilon^2$ on a arbitrary p-matchoid constraint and a submodular function, with $\alpha = \varepsilon f(OPT)/2k$, $\beta_1 = 1$ and $\beta_i = \frac{\gamma_{i-1}-1-p}{\gamma_{i-1}-1+p}$ where γ_i is given by the recurrence, $\gamma_1 = 4p$ and $\gamma_i = \frac{4p\gamma_{i-1}(\gamma_{i-1}-1)}{(\gamma_{i-1}-1+p)^2}$. Then,*

$$(1-\varepsilon)f(OPT) \leq \left(p + 1 + \frac{4p}{i}\right)\mathbb{E}[f(\tilde{S}_i)] + \bar{\gamma}_{\text{off}}\,\mathbb{E}[f(S'_i)].$$

In particular after $d = \frac{4p}{\varepsilon}$ passes,

$$(1-\varepsilon)f(OPT) \leq (p + 1 + \bar{\gamma}_{\text{off}} + \varepsilon)\,\mathbb{E}[f(\bar{S}_d)]\,.$$

Under a matroid constraint, Algorithm 3 with $\alpha = \varepsilon f(OPT)/2k$, $\beta_i = 1/i$ and $d = 2\varepsilon^{-1}$ passes outputs a solution $\bar{S}$ such that,

$$(1-\varepsilon)f(OPT) \leq (2 + \bar{\gamma}_{\text{off}} + \varepsilon)\,\mathbb{E}[f(\bar{S})]\,,$$

where $\bar{\gamma}_{\text{off}}$ is the approximation ration of the best offline algorithm for maximizing f under a matroid constraint.

We are now ready to complete the proof of Theorem 9. We assume that we know the value of $f(OPT)$ before hand, which can be accomplished approximately as in Section C.1. Let $\varepsilon' = \varepsilon/p$ with $1/2 \geq \varepsilon' > 0$ and let $\alpha = \varepsilon' f(OPT)/2k$. We want to obtain an additive error term instead of a multiplicative error term as stated in Theorem 12. By Theorem 12,

$$(1-\varepsilon')f(OPT) \leq \left(p+1+\bar{\gamma}_{\text{off}}+\frac{4p}{d}\right)\mathbb{E}[f(\bar{S}_d)] = (p+1+\bar{\gamma}_{\text{off}})\left(1+O\left(d^{-1}\right)\right)\mathbb{E}[f(\bar{S}_d)]\,.$$

Using the fact that $(1-\varepsilon')^{-1} \leq 1+2\varepsilon'$ for $\varepsilon' \in (0, 1/2]$, we get that,

$$f(OPT) \leq (p+1+\bar{\gamma}_{\text{off}})\left(1+O\left(d^{-1}\right)\right)(1+2\varepsilon')\,\mathbb{E}[f(\bar{S}_d)]\,. \tag{15}$$

Since $\varepsilon' = \varepsilon/p$, setting $d = O(p/\varepsilon)$ we finally obtain the desired result:

$$f(OPT) \leq (p+1+\bar{\gamma}_{\text{off}})(1+O(\varepsilon/p))(1+2\varepsilon/p)\,\mathbb{E}[f(\bar{S}_d)] \leq (p+1+\bar{\gamma}_{\text{off}}+O(\varepsilon))\,\mathbb{E}[f(\bar{S}_d)].$$

For the space complexity, we note that the randomized local-search algorithm stores the buffer B and maintains two past solutions $S_i, S_i' \in \mathcal{I}$, together with the current solution $S \in \mathcal{I}$. Hence, the total space needed is equal to $O(|B| + |S_i'| + |S_i| + |S|) = O(m+3k) = O\left(p^3 k \varepsilon^{-3}\right)$, times an additional factor of $O(\log k)$ for guessing $f(OPT)$. The number of passes is $d = O(p/\varepsilon)$.