Faster submodular optimization for several matroids

Using dynamic data structures to speed up optimization problems

Monika Henzinger (ISTA) Paul Liu (Stanford) Jan Vondrák (Stanford) **Da Wei Zheng** (UIUC)

July 14, 2023

ICALP 2023, Paderborn, Germany

Submodular functions

For a set $\mathcal{N}$, a set function $f : 2^{\mathcal{N}}$ is **submodular** if for any sets S and T:

$$f(S) + f(T) \geq f(S + T) + f(S - T)$$

We only consider monotone submodular functions.

$$f(S) \leq f(T) \quad \text{for any sets } S \subseteq T$$

We assume we have an oracle that computes f.

Matroid

Matroids are set system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ where $\mathcal{I} \subseteq 2^{\mathcal{N}}$.

A set $S \in \mathcal{I}$ is said to be **independent**.

Independent sets satisfy two properties:

Matroid

Matroids are set system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ where $\mathcal{I} \subseteq 2^{\mathcal{N}}$.

A set $S \in \mathcal{I}$ is said to be **independent**.

Independent sets satisfy two properties:

1. Downward closure:

$$\forall S \subseteq T \text{ and } T \in \mathcal{I} \Rightarrow s \in \mathcal{I}$$

Matroid

Matroids are set system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ where $\mathcal{I} \subseteq 2^{\mathcal{N}}$.

A set $S \in \mathcal{I}$ is said to be **independent**.

Independent sets satisfy two properties:

1. Downward closure:

$$\forall S \subseteq T \text{ and } T \in \mathcal{I} \Rightarrow s \in \mathcal{I}$$

2. Exchange property:

$$\forall S, T \in \mathcal{I} \text{ and } |S| < |T| \Rightarrow \exists e \in T \setminus S \text{ such that } S + e \in \mathcal{I}$$

Submodular maximization with matroid constraints

Goal: Given and a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ and a submodular f over $\mathcal{N}$, find:

$$S^* = \arg\max_{S \subseteq \mathcal{I}} f(S)$$

Submodular maximization with matroid constraints

Goal: Given and a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ and a submodular f over $\mathcal{N}$, find:

$$S^* = \arg\max_{S \subseteq \mathcal{I}} f(S)$$

APX-hard to approximate better than $(1 - 1/e)$ so our goal is $(1 - 1/e - \varepsilon)$-approximations.

Submodular maximization with matroid constraints

Goal: Given and a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ and a submodular f over $\mathcal{N}$, find:

$$S^* = \arg\max_{S \subseteq \mathcal{I}} f(S)$$

APX-hard to approximate better than $(1 - 1/e)$ so our goal is $(1 - 1/e - \varepsilon)$-approximations.

Why study this problem?

Submodular maximization with matroid constraints

Goal: Given and a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ and a submodular f over $\mathcal{N}$, find:

$$S^* = \arg\max_{S \subseteq \mathcal{I}} f(S)$$

APX-hard to approximate better than $(1 - 1/e)$ so our goal is $(1 - 1/e - \varepsilon)$-approximations.

Why study this problem?

- Submodular optimization is a fundamental problem in combinatorial optimization, information retrieval, and machine learning.

Submodular maximization with matroid constraints

Goal: Given and a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ and a submodular f over $\mathcal{N}$, find:

$$S^* = \arg\max_{S \subseteq \mathcal{I}} f(S)$$

APX-hard to approximate better than $(1 - 1/e)$ so our goal is $(1 - 1/e - \varepsilon)$-approximations.

Why study this problem?

- Submodular optimization is a fundamental problem in combinatorial optimization, information retrieval, and machine learning.
- Many applications involve combinatorial constraints on subsets, matroids are very general ones that have been well studied.

Sometimes we assume we have an *independence oracle* that we can query with a set S to test if S is independent in $\mathcal{M}$.

Sometimes we assume we have an *independence oracle* that we can query with a set S to test if S is independent in $\mathcal{M}$.

In this work, we assume we instead have an **explicit representation** of the matroid.

Sometimes we assume we have an *independence oracle* that we can query with a set S to test if S is independent in $\mathcal{M}$.

In this work, we assume we instead have an **explicit representation** of the matroid.

We study the following matroids with explicit representations

1. **Graphic** - A graph $G = (V, E)$
   Application: TSP approximations [XR15]

… over specific matroids

Sometimes we assume we have an *independence oracle* that we can query with a set S to test if S is independent in $\mathcal{M}$.

In this work, we assume we instead have an **explicit representation** of the matroid.

We study the following matroids with explicit representations

1. **Graphic** - A graph $G = (V, E)$
   Application: TSP approximations [XR15]
2. **Transversal** - A bipartite graph $G = ((U, V), E)$
   Application: ad placement and matching [BIK07, BHK08]

… over specific matroids

Sometimes we assume we have an *independence oracle* that we can query with a set S to test if S is independent in $\mathcal{M}$.

In this work, we assume we instead have an **explicit representation** of the matroid.

We study the following matroids with explicit representations

1. **Graphic** - A graph $G = (V, E)$
   Application: TSP approximations [XR15]
2. **Transversal** - A bipartite graph $G = ((U, V), E)$
   Application: ad placement and matching [BIK07, BHK08]
3. **Laminar** - A tree representing the laminar family
   Application: Capacity constraints - YouTube recommendations [WRB+18]

Prior Results and New Results

n is the size of the ground set, m is the size of the matroid representation,

r is the rank of the matroid.

* Assumes access to an independence oracle

Constraint	Runtime	Paper	New
General*	$O_\varepsilon(nr \log^2 n)$	[Badanidiyuru–Vondrák '14]	-

Prior Results and New Results

n is the size of the ground set, m is the size of the matroid representation,

r is the rank of the matroid.

* Assumes access to an independence oracle

Constraint	Runtime	Paper	New
General*	$O_\varepsilon(nr \log^2 n)$	[Badanidiyuru–Vondrák '14]	-
Partition	$\widetilde{O}_\varepsilon(n^{3/2})$	[Buchbinder–Feldman–Schwartz'14]	-
	$O_\varepsilon(n \log^2 n)$	[Ene–Nguyen '19]	

Prior Results and New Results

n is the size of the ground set, m is the size of the matroid representation,

r is the rank of the matroid.

* Assumes access to an independence oracle

Constraint	Runtime	Paper	New
General*	$O_\varepsilon(nr \log^2 n)$	[Badanidiyuru–Vondrák '14]	-
Partition	$\widetilde{O}_\varepsilon(n^{3/2})$	[Buchbinder–Feldman–Schwartz'14]	-
	$O_\varepsilon(n \log^2 n)$	[Ene–Nguyen '19]	
Graphic	$O_\varepsilon(r \log^6 n + n \log^2 n)$	[Ene–Nguyen '19]	$O_\varepsilon(n \log^2 n)$

Prior Results and New Results

n is the size of the ground set, m is the size of the matroid representation,

r is the rank of the matroid.

* Assumes access to an independence oracle

Constraint	Runtime	Paper	New
General*	$O_\varepsilon(nr \log^2 n)$	[Badanidiyuru–Vondrák '14]	-
Partition	$\widetilde{O}_\varepsilon(n^{3/2})$	[Buchbinder–Feldman–Schwartz'14]	-
	$O_\varepsilon(n \log^2 n)$	[Ene–Nguyen '19]	
Graphic	$O_\varepsilon(r \log^6 n + n \log^2 n)$	[Ene–Nguyen '19]	$O_\varepsilon(n \log^2 n)$
Laminar			$O_\varepsilon(n \log^2 n)$

Prior Results and New Results

n is the size of the ground set, m is the size of the matroid representation,

r is the rank of the matroid.

* Assumes access to an independence oracle

Constraint	Runtime	Paper	New
General*	$O_\varepsilon(nr \log^2 n)$	[Badanidiyuru–Vondrák '14]	-
Partition	$\widetilde{O}_\varepsilon(n^{3/2})$ $O_\varepsilon(n \log^2 n)$	[Buchbinder–Feldman–Schwartz'14] [Ene–Nguyen '19]	-
Graphic	$O_\varepsilon(r \log^6 n + n \log^2 n)$	[Ene–Nguyen '19]	$O_\varepsilon(n \log^2 n)$
Laminar			$O_\varepsilon(n \log^2 n)$
Transversal			$O_\varepsilon(m \log n + n \log^2 n)$

Prior Results and New Results

n is the size of the ground set, m is the size of the matroid representation,

r is the rank of the matroid.

* Assumes access to an independence oracle

Constraint	Runtime	Paper	New
General*	$O_\varepsilon(nr\log^2 n)$	[Badanidiyuru–Vondrák '14]	-
Partition	$\widetilde{O}_\varepsilon(n^{3/2})$ $O_\varepsilon(n\log^2 n)$	[Buchbinder–Feldman–Schwartz'14] [Ene–Nguyen '19]	-
Graphic	$O_\varepsilon(r\log^6 n + n\log^2 n)$	[Ene–Nguyen '19]	$O_\varepsilon(n\log^2 n)$
Laminar			$O_\varepsilon(n\log^2 n)$
Transversal			$O_\varepsilon(m\log n + n\log^2 n)$

Takeaway message

Efficient dynamic data structures

Efficient dynamic data structures

$\downarrow\downarrow\downarrow$ $\uparrow\uparrow\uparrow$

Solving optimization problems faster

Our results

1. Improved framework for fast submodular maximization with matroid constraints and a reduction to dynamic matroid independent set problems

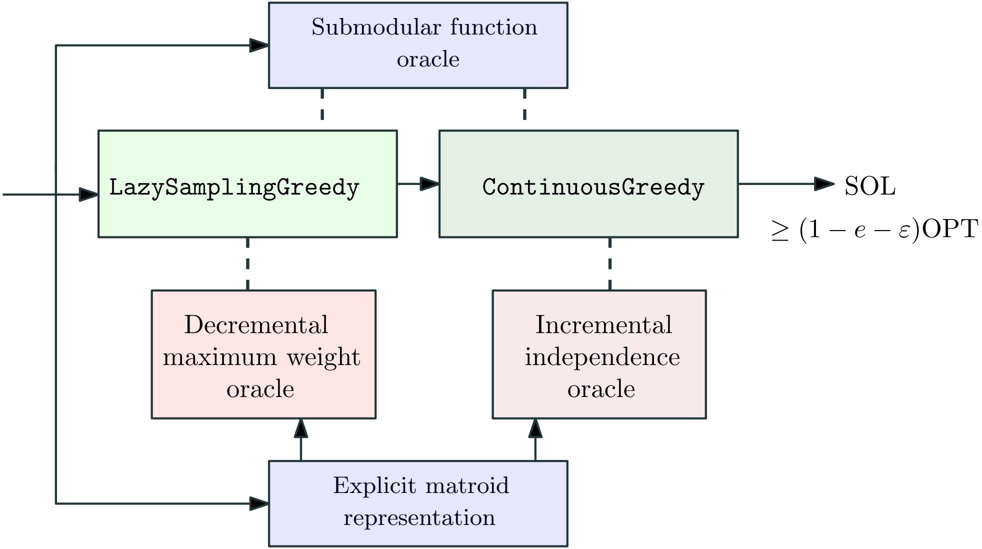
Our results

1. Improved framework for fast submodular maximization with matroid constraints and a reduction to dynamic matroid independent set problems
2. A data structure for laminar matroids undergoing insertion and deletion updates with $O(\log n)$ update time.

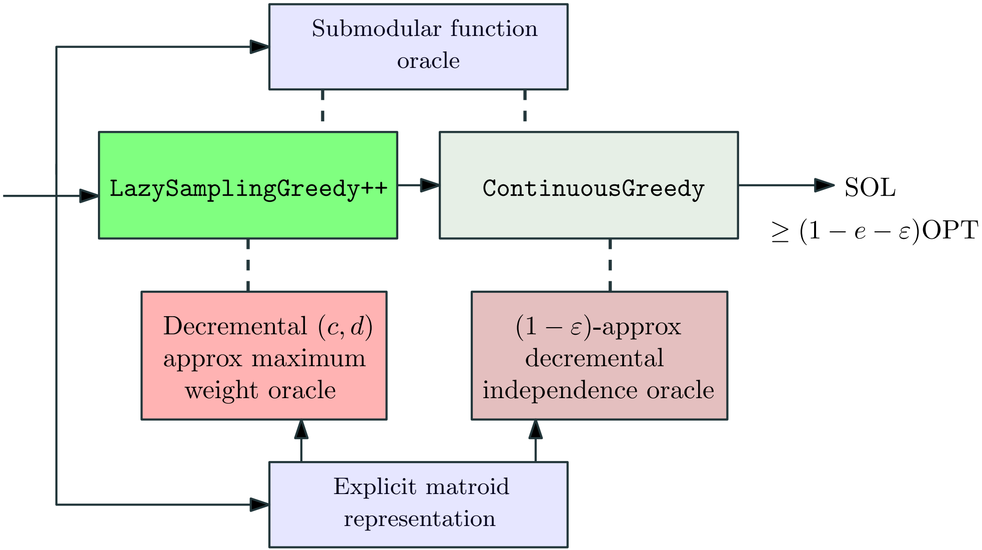
Our results

1. Improved framework for fast submodular maximization with matroid constraints and a reduction to dynamic matroid independent set problems
2. A data structure for laminar matroids undergoing insertion and deletion updates with $O(\log n)$ update time.
3. A data structure for maintaining a $(1-\varepsilon)$–approximate maximum weight matchings in a vertex weighted graph (transversal matroid) with weight decrement operations.

Framework of Ene & Nguyen '19



New Framework



Framework of Ene & Nguyen '19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.

Framework of Ene & Nguyen '19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.
2. Use algorithm `ContinuousGreedy` [BV14] that now runs in near-linear rounds.

Framework of Ene & Nguyen '19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.
2. Use algorithm `ContinuousGreedy` [BV14] that now runs in near-linear rounds.

Two data structures required:

Framework of Ene & Nguyen '19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.
2. Use algorithm `ContinuousGreedy` [BV14] that now runs in near-linear rounds.

Two data structures required:

Decremental maximum weight oracle Given weights w over elements of $\mathcal{N}$ maintain the maximum independent set B and:

- supports decrementing the weight of elements of $\mathcal{N}$.

Framework of Ene & Nguyen '19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.
2. Use algorithm `ContinuousGreedy` [BV14] that now runs in near-linear rounds.

Two data structures required:

Decremental maximum weight oracle Given weights w over elements of $\mathcal{N}$ maintain the maximum independent set B and:

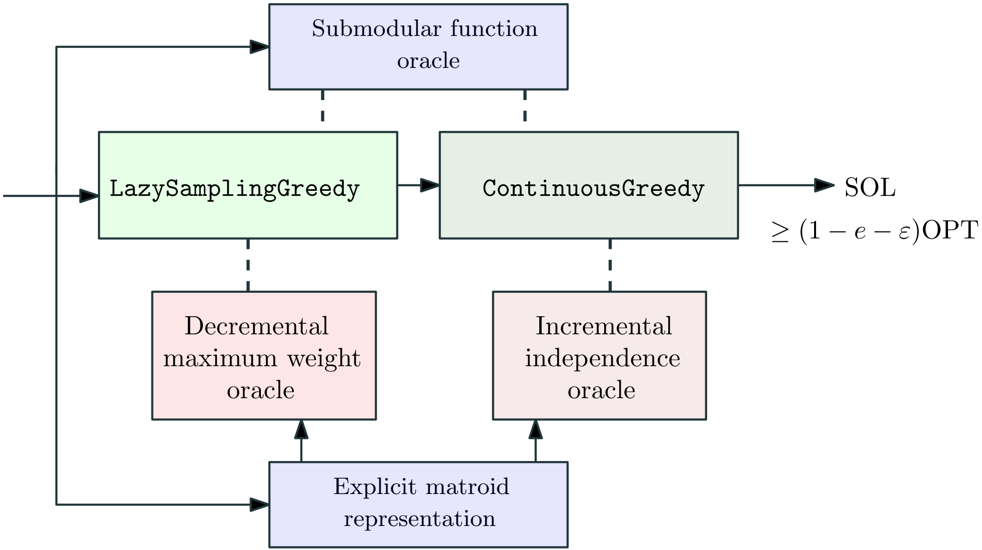
- supports decrementing the weight of elements of $\mathcal{N}$.

Incremental independence oracle Maintains a set B such that we can:

- test adding an element e, outputs if $B + e$ is independent,
- inserts an element e into B.

Laminar Matroids

Framework of Ene & Nguyen '19



We can view the maximum weight independent set of a laminar matroid as follows:

We can view the maximum weight independent set of a laminar matroid as follows:

Given a tree T, capacities c_v for nodes in the tree indicating that we can take at most c_v leaves of the subtree, and weights w on the leaves of the tree.

Goal: the maximum weight set of leaves that respect the capacity constraints.

Laminar

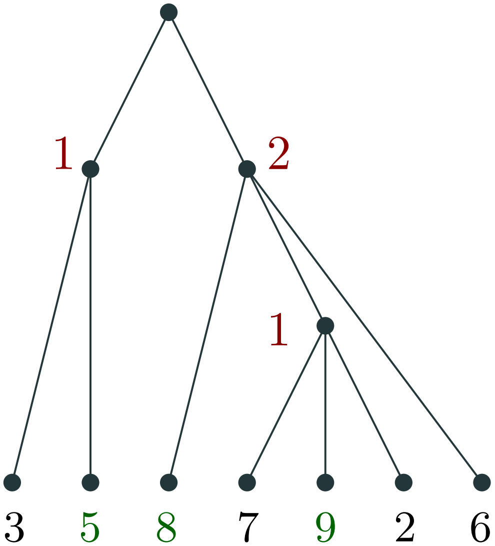
We can view the maximum weight independent set of a laminar matroid as follows:

Given a tree T, capacities c_v for nodes in the tree indicating that we can take at most c_v leaves of the subtree, and weights w on the leaves of the tree.

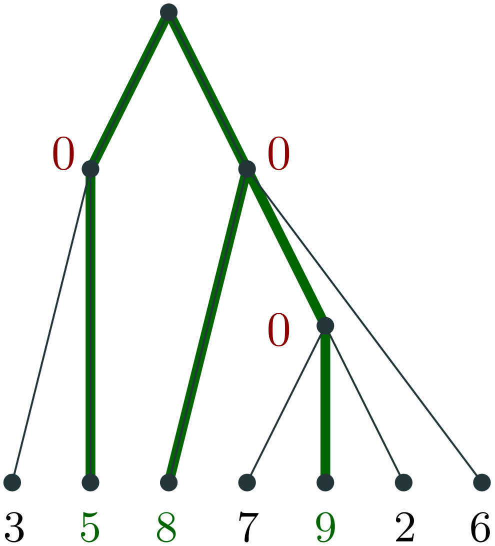
Goal: the maximum weight set of leaves that respect the capacity constraints.

Theorem. There exists a data structure that supports the insertion and deletion of leaves of arbitrary weight that maintains the maximum feasible set of leaves with $O(\log n)$ update time.

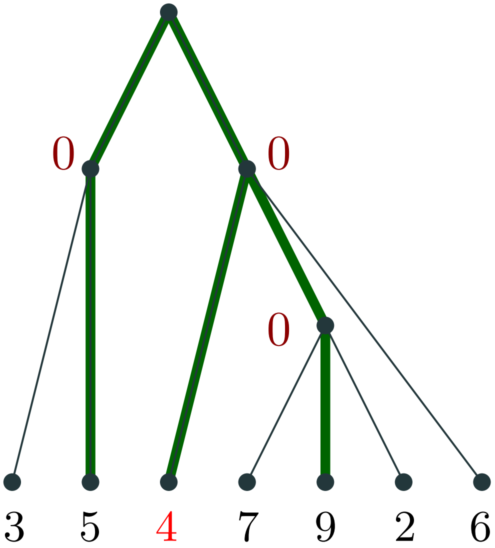
Laminar example



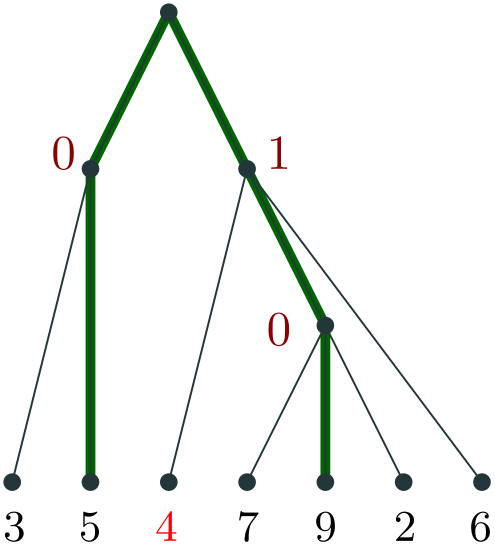
Laminar example



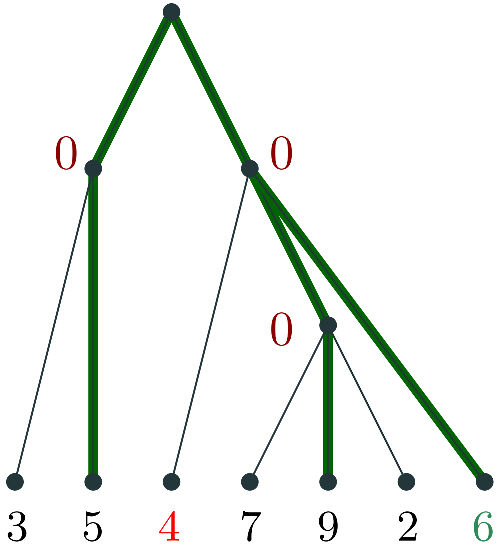
Laminar example



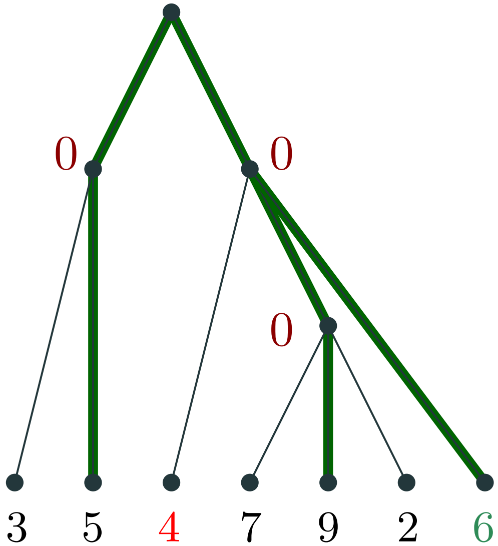
Laminar example



Laminar example



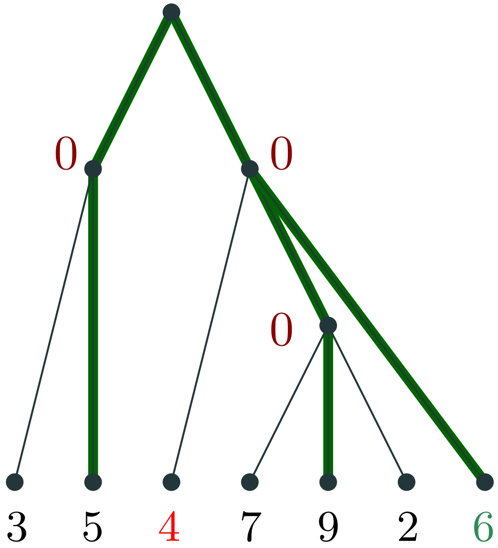
Laminar example



$O(n)$ time per update -

Not too hard

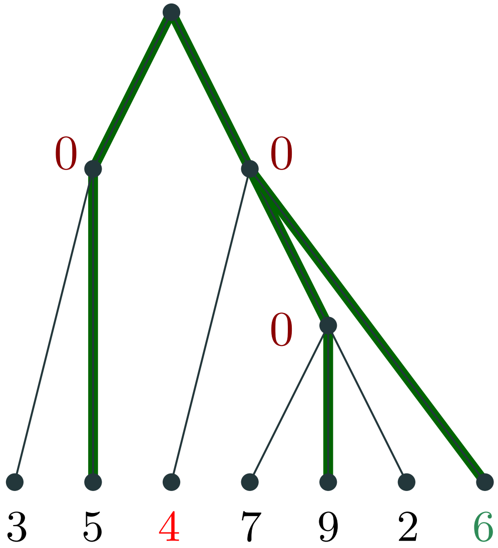
Laminar example



$O(n)$ time per update - Not too hard

$O(\log^2 n)$ time per update - Heavy-light decomposition

Laminar example



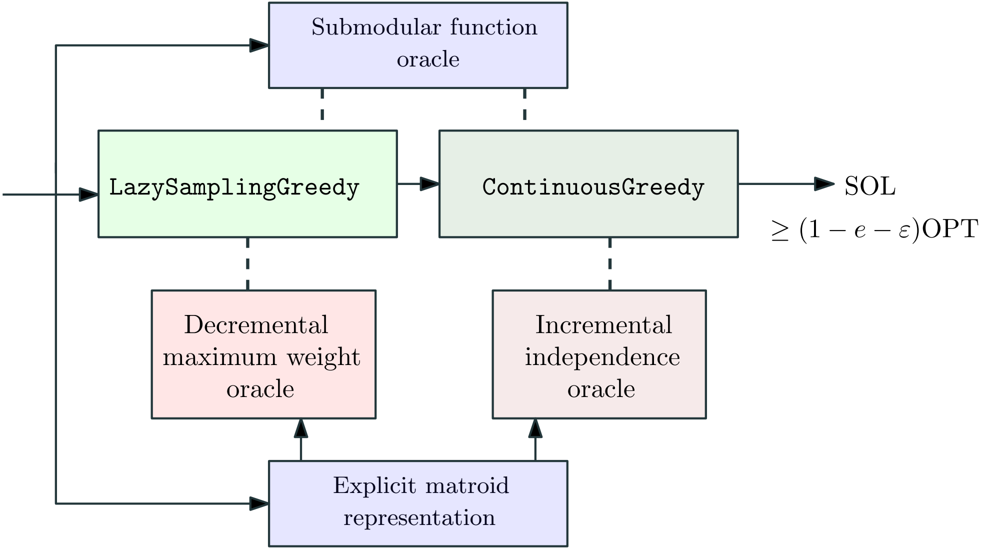
$O(n)$ time per update - Not too hard

$O(\log^2 n)$ time per update - Heavy-light decomposition

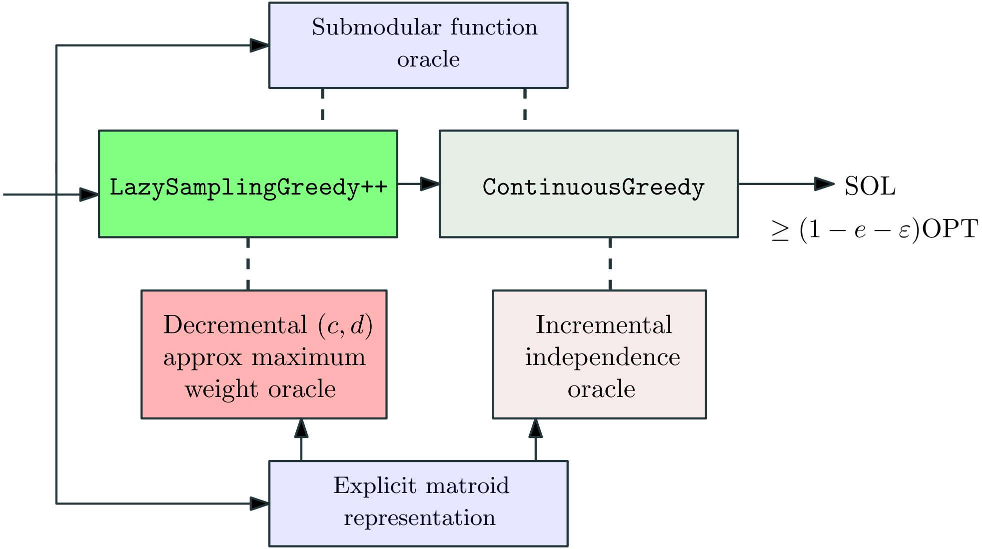
$O(\log n)$ time per update - Top tree framework

Graphic Matroids

Framework of Ene & Nguyen ’19



New Framework



Improved Framework

Old framework of Ene & Nguyen '19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.

Improved Framework

Old framework of Ene & Nguyen ’19

1. Use algorithm `LazySamplingGreedy` to sample from maximum weight base $O_\varepsilon(\log n)$ times as a preconditioning step.

New framework

1. Use algorithm `LazySamplingGreedy++` to sample from constant approximate independent set of constant size $O_\varepsilon(\log n)$ times as a preconditioning step.

Data structure differences

Old framework

Decremental maximum weight oracle

Given weights w over elements of $\mathcal{N}$ maintain the maximum independent set B and:

- supports decrementing the weight of elements of $\mathcal{N}$.

Data structure differences

Old framework

Decremental maximum weight oracle
Given weights w over elements of $\mathcal{N}$ maintain the maximum independent set B and:

- supports decrementing the weight of elements of $\mathcal{N}$.

New framework

Decremental (c, d)-approximate maximum weight oracle
Given weights w over elements of $\mathcal{N}$, constants $c < 1$ and $d < 1$, maintain an independent set B that is:

- a c-approximate of the maximum independent set,
- has at least dr elements,
- supports decrementing the weight of elements of $\mathcal{N}$.
- supports "freezing" an element (required to be in B)

Graphic approximate maximum weight oracle

Dynamic $(1/2, 1/2)$-approximate maximum weight oracle

Given weights w over edges E of a graph $G = (V, E)$ maintain a maximum spanning tree B that is:

- a $(1/2)$-approximate of the maximum spanning tree,
- has at least $|V|/2$ edges,
- supports decrementing weight of an edge.
- supports contracting an edge

Graphic approximate maximum weight oracle

Dynamic $(1/2, 1/2)$-approximate maximum weight oracle

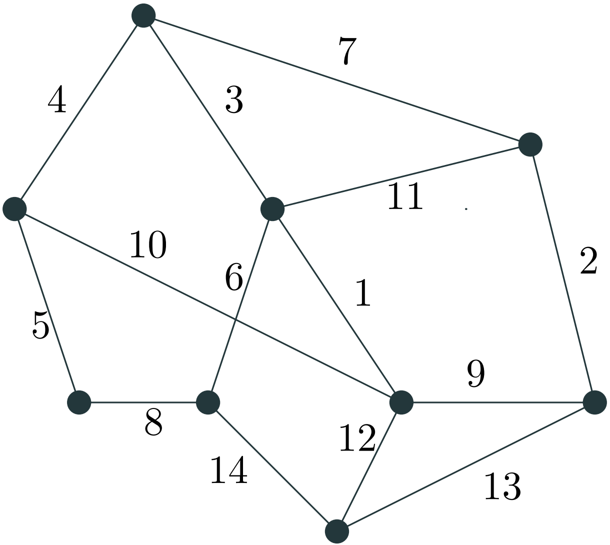
Given weights w over edges E of a graph $G = (V, E)$ maintain a maximum spanning tree B that is:

- a (1/2)-approximate of the maximum spanning tree,
- has at least $|V|/2$ edges,
- supports decrementing weight of an edge.
- supports contracting an edge

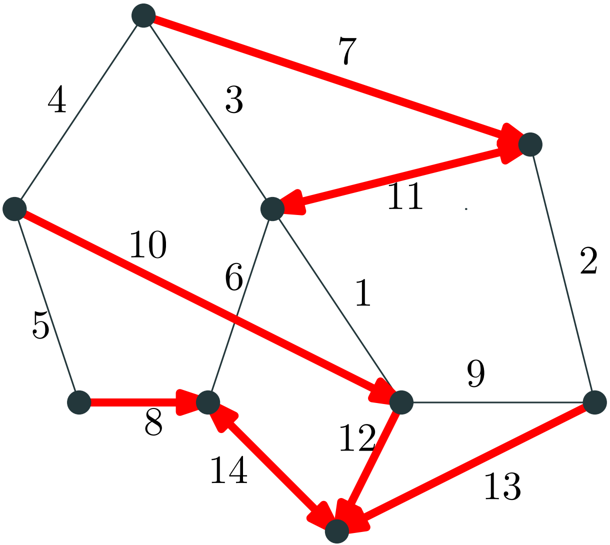
Simple data structure with $O(\log n)$ update time

- Maintain for every vertex a sorted list of incident edges,
- $B =$ union of maximum weight incident edge to each vertex
- On decrement, fix sorted list at endpoints (heap).
- On contract, merge lists (heaps)

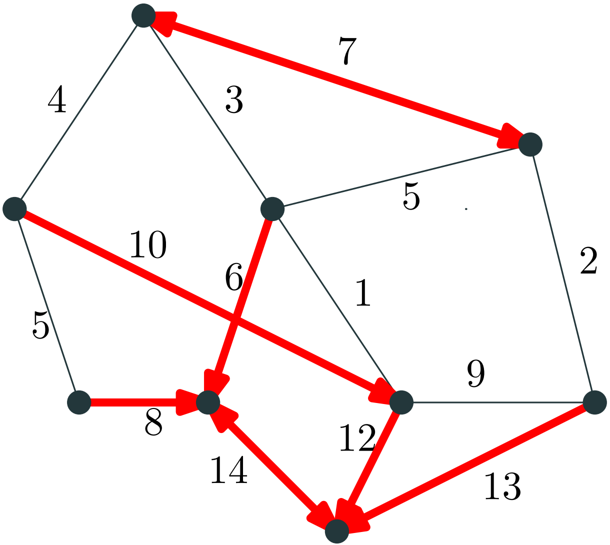
Graphic matroid example



Graphic matroid example



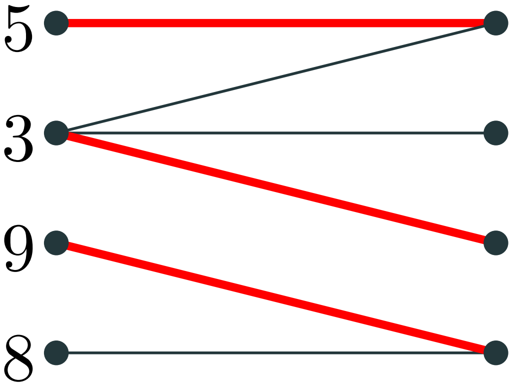
Graphic matroid example



Transversal matroids

Transversal matroids = Matchable U of a bipartite graph $G = (U \cup V, E)$

Let $m = |E|$.



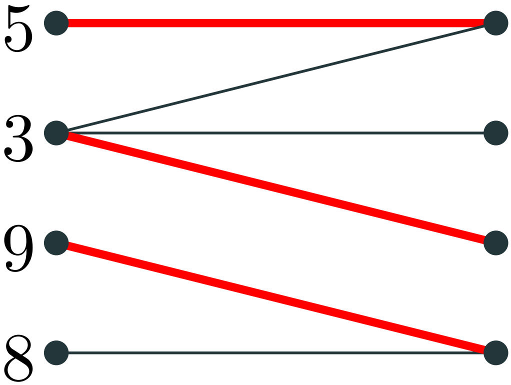
Transversal matroids

Transversal matroids = Matchable U of a bipartite graph $G = (U \cup V, E)$

Let $m = |E|$.

Dynamic maximum weight oracle:

Maintain maximum vertex weight matching with changing vertex weights.



Transverse matroids

Transversal matroids = Matchable U of a bipartite graph $G = (U \cup V, E)$

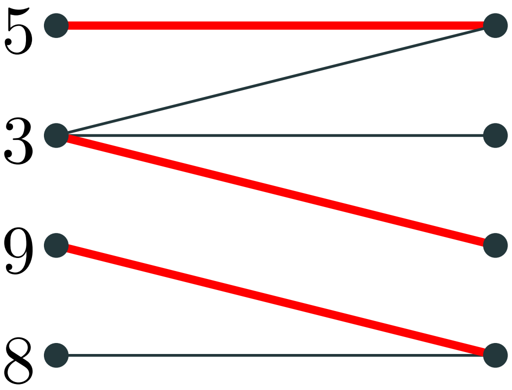
Let $m = |E|$.

Dynamic maximum weight oracle:

Maintain maximum vertex weight matching with changing vertex weights.

Incremental independence oracle:

Vertex incremental matching.



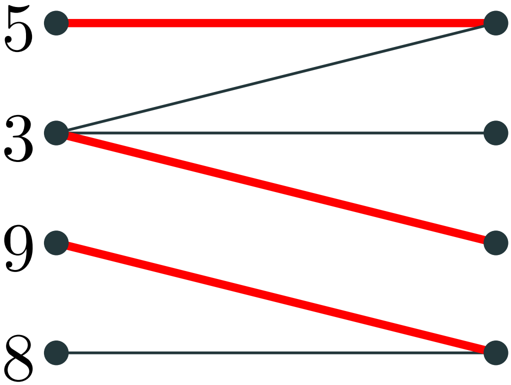
Transversal matroids

Transversal matroids = Matchable U of a bipartite graph $G = (U \cup V, E)$
Let $m = |E|$.

Dynamic maximum weight oracle:
Maintain maximum vertex weight matching with changing vertex weights.

Incremental independence oracle:
Vertex incremental matching.

Fast algorithm would imply fast matching.
Both problems have *conditional lower bounds* of $\Omega(m^{3/2-\varepsilon})$ update time.



Comparison of Framework

Old framework of Ene & Nguyen '19

Incremental independence oracle

Maintains a set B such that we can:

- test adding an element e, outputs if $B + e$ is independent,
- inserts an element e into B.

Comparison of Framework

Old framework of Ene & Nguyen '19

Incremental independence oracle

Maintains a set B such that we can:

- test adding an element e, outputs if $B + e$ is independent,
- inserts an element e into B.

New framework

$(1 - \varepsilon)$-**approximate decremental independent set data structure**

Maintains a set B of size $(1 - \varepsilon)r$ such that we can:

- delete one element from $\mathcal{N}$.

Transversal dynamic approximate maximum weight oracle

Dynamic $(1-\varepsilon, 1/2)$-approximate maximum weight oracle

Transversal dynamic approximate maximum weight oracle

Dynamic $(1-\varepsilon, 1/2)$**-approximate maximum weight oracle**

High level idea:

Transversal dynamic approximate maximum weight oracle

Dynamic $(1-\varepsilon, 1/2)$**-approximate maximum weight oracle**

High level idea:

Use variation of multiplicative auction algorithm of Z. and Henzinger (2023) to support vertex weight decrements with $O(\log n)$ update time, and ensure maximality.

Dynamic $(1-\varepsilon, 1/2)$-approximate maximum weight oracle

High level idea:

Use variation of multiplicative auction algorithm of Z. and Henzinger (2023) to support vertex weight decrements with $O(\log n)$ update time, and ensure maximality.

$(1-\varepsilon)$**-approximate decremental independent set data structure**

Transversal dynamic approximate maximum weight oracle

Dynamic $(1-\varepsilon, 1/2)$**-approximate maximum weight oracle**

High level idea:

Use variation of multiplicative auction algorithm of Z. and Henzinger (2023) to support vertex weight decrements with $O(\log n)$ update time, and ensure maximality.

$(1-\varepsilon)$**-approximate decremental independent set data structure**

High level idea:

Use vertex decremental algorithm of Bosek, Leniowski, Sankowski, and Zych (2014)

Summary

1. Improved framework for fast submodular maximization with matroid constraints and a reduction to dynamic matroid independent set problems
2. A data structure for laminar matroids undergoing insertion and deletion updates with $O(\log n)$ update time.
3. A data structure for maintaining a $(1 - \varepsilon)$–approximate maximum weight matchings in a vertex weighted graph (transversal matroid) with weight decrement operations.

New Framework