A Fast Approximation for Maximum Weight Matroid Intersection*

Chandra Chekuri† Kent Quanrud‡

Abstract

We present an approximation algorithm for the maximum weight matroid intersection problem in the independence oracle model. Given two matroids defined over a common ground set $\mathcal{N}$ of n elements, let k be the rank of the matroid intersection and let Q denote the cost of an independence query for either matroid. An exact algorithm for finding a maximum cardinality independent set (the unweighted case), due to Cunningham, runs in $O(nk^{1.5}Q)$ time. For the weighted case, algorithms due to Frank and Brezovec *et al.* run in $O(nk^2Q)$ time. There are also scaling based algorithms that run in $O(n^2\sqrt{k}\log(kW)Q)$ time, where W is the maximum weight (assuming all weights are integers), and ellipsoid-style algorithms that run in $O\big(\big(n^2\log(n)Q + n^3\operatorname{polylog}(n)\big)\log(nW)\big)$ time. Recently, Huang, Kakimura, and Kamiyama described an algorithm that gives a $(1-\epsilon)$-approximation for the weighted matroid intersection problem in $O(nk^{1.5}\log(k)Q/\epsilon)$ time.

We observe that a $(1-\epsilon)$-approximation for the maximum cardinality case can be obtained in $O(nkQ/\epsilon)$ time by terminating Cunningham's algorithm early. Our main contribution is a $(1-\epsilon)$ approximation algorithm for the weighted matroid intersection problem with running time $O(nk\log^2(1/\epsilon)Q/\epsilon^2)$.

1 Introduction

We consider fast *approximation* algorithms for the classical matroid intersection problem defined formally below. An *independence system* consists of a finite *ground set* $\mathcal{N}$ and a non-empty family of sets $\mathcal{I} \subseteq 2^{\mathcal{N}}$ that is closed under taking subsets: $A \in \mathcal{I}$ implies $B \in \mathcal{I}$ for all subsets $B \subset A$. Members of $\mathcal{I}$ are *independent sets*. We let $n = |\mathcal{N}|$ and let $k = \max_{S\in\mathcal{I}}|S|$ denote the *rank* of the independence system. A *matroid* is an independence system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ such that for any two independent sets $A, B \in \mathcal{I}$, if $|A| < |B|$, then there is an element $e \in B \setminus A$ such that $B + e \in \mathcal{I}$. Matroids play a fundamental role in several areas of optimization and mathematics, and are surprisingly ubiquitous despite (or perhaps for) their abstract definition.

Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids over a common ground set $\mathcal{N}$. The *matroid intersection* $\mathcal{M}_1 \cap \mathcal{M}_2$ of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the independence system $(\mathcal{N}, \mathcal{I}_1 \cap \mathcal{I}_2)$. The intersection $\mathcal{M}_1 \cap \mathcal{M}_2$ inherits substantial structure from the matroids $\mathcal{M}_1$ and $\mathcal{M}_2$, but $\mathcal{M}_1 \cap \mathcal{M}_2$ is generally *not* a matroid. The basic algorithmic problem here is to find the rank of $\mathcal{M}_1 \cap \mathcal{M}_2$, or more constructively, a maximum cardinality independent set. When the ground set $\mathcal{N}$ is weighted, the *weighted matroid intersection problem* is to find a maximum weight independent set in $\mathcal{M}_1 \cap \mathcal{M}_2$.

The study of matroid intersection has a rich history, of which we mention a few algorithmic highlights and refer the reader to Schrijver [Sch03, Chapter 41]. Matroid intersection was brought to prominence by Edmonds in 1969 [Edm70]. He gave min-max characterizations for the rank of a matroid intersection, characterized the matroid intersection polyhedron, and derived polynomial-time algorithms for the maximum cardinality and the maximum weight matroid intersection problems. He also extended the theory to the intersection of polymatroids. Matroid intersection is a fundamental achievement in combinatorial optimization and allows one to solve a wide variety of concrete and abstract problems. Canonical problems captured by matroid intersection include bipartite matchings, branchings, and packing spanning trees (via matroid union). There are also several applications of matroid intersection outside of traditional combinatorial optimization [Mur09, Rec13, DFZ11].

Following Edmonds's work there has been considerable activity in algorithms for matroid intersection. We mostly focus our attention on the *oracle* model, where the matroids $\mathcal{M}_1$ and $\mathcal{M}_2$ are exposed via independence oracles: given $S \subseteq \mathcal{N}$ as a query, the independence oracle for a matroid answers whether S is independent in that matroid. There are more specified settings of interest that we discuss later. In stating running times, let Q denote the cost of an independence query. In 1975, Lawler ex-

*Work on this paper is supported in part by NSF grants CCF-1319376 and CCF-1526799.

†Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. chekuri@illnois.edu. http://illinois.edu/~chekuri/

‡Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. quanrud2@illnois.edu. http://illinois.edu/~quanrud2/

tended ideas from the Hungarian algorithm for bipartite matching to obtain a primal-dual algorithm for weighted matroid intersection in $O(nk^2(n^2+Q))$ time. Frank gave a faster and simpler "weight-splitting" algorithm with running time $O(nk^2Q)$ [Fra81] (after low-level optimizations; see also Brezovec, Cornuéjols and Glover [BCG86] and [Sch03, Section 41.3a]). For integral weights bounded by W, a scaling algorithm with a running time of $O(n^2\sqrt{k}\log(kW)Q)$ was obtained by Fujishige and Zhang [FZ95]. Cunningham [Cun86] gave an algorithm to compute the maximum *cardinality* independent set in $\mathcal{M}_1 \cap \mathcal{M}_2$ with running time $O(nk^{1.5}Q)$ independence queries. The above algorithms are all combinatorial. Recently, Sidford, Lee and Wong improved the ellipsoid method to obtain, among many other applications, an $O\big((n^2\log(n)Q + n^3\operatorname{polylog}(n))\log(nW)\big)$ time algorithm for weighted matroid intersection [LSW15].

In recent years, for practical and theoretical reasons, there has been considerable interest in fast *approximation* algorithms for combinatorial optimization problems that already have strongly polynomial time algorithms. Two examples that we wish to highlight are maximum flow [CKM+11, Mą13, KLOS14, LS14, Pen16] and weighted matching [DP14]. We are particularly inspired by the work of Duan and Pettie, who described an algorithm that for any fixed $\epsilon > 0$, obtains a $(1-\epsilon)$-approximation for maximum weight matching in non-bipartite graphs in linear time; more precisely, in time $O(m\log(1/\epsilon)/\epsilon)$, where m is the number of edges. This is faster than the best known exact algorithm for *unweighted* bipartite matching, with running time $O(m\sqrt{n})$, by Hopcroft and Karp [HK73], or running time $\tilde{O}(m^{10/7})$, in the recent breakthrough of Mądry [Mą13].

Bipartite matching is the standard special case for matroid intersection, suggesting the possibility of fast approximations for matroid intersection. A first step in this direction was taken recently by Huang, Kakimura, and Kamiyama [HKK14]. They obtain, among other results, a $(1-\epsilon)$ approximation for weighted matroid intersection using $O(nk^{1.5}\log(k)/\epsilon)$ independence queries. Their general framework uses scaling to obtain a $(1-\epsilon)$-approximation for the weighted case via the use of *exact* algorithms for the unweighted matroid intersection problem; in the oracle model, they employ Cunningham's algorithm.

Our main result is the following theorem for weighted matroid intersection in the oracle model.

THEOREM 1.1. *There is a polynomial-time algorithm that, given two matroids $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ and weights $w : \mathcal{N} \to \mathbb{R}$, outputs a $(1-\epsilon)$-approximation to the maximum weight independent set in $\mathcal{M}_1 \cap \mathcal{M}_2$ using $O(nk\log^2(1/\epsilon)/\epsilon^2)$ independence queries and arithmetic operations.*

For any fixed ϵ, our result improves upon the work of [HKK14] by a $\sqrt{k}$ factor. In the oracle model we essentially need $O(nk)$ time and independence queries to even verify whether a given set I is a maximum weight independent set. The running time we achieve is thus nearly optimal modulo the dependence on ϵ. Perhaps of main importance is our technique. We show that a fast *approximation* algorithm for maximum cardinality matroid intersection problem can be leveraged in a scaling framework for the weighted case. While we are inspired by the approach of Duan and Pettie for matchings, algorithms for (weighted) matroid intersection are generally more involved than those for bipartite matchings. We give a few more details of our ideas, compare them to prior work, and close with a discussion of potential future work in the subsection below.

1.1 Main techniques and related work

It is well-known that matroid intersection algorithms are related to algorithms for bipartite matching, although there are significant technical differences. Hopcroft and Karp [HK73] obtained an $O(m\sqrt{n})$ time algorithm for maximum cardinality bipartite matching inspired by blocking flow algorithms, and is designed as follows. In linear time, one can find several augmenting paths for a current matching M such that after augmenting along these paths, the shortest augmenting path for the new matching M' is strictly longer than it is for M. Coupled with the observation that if M is a matching for which the length of the shortest augmenting path is at least h, M is a $(1-\frac{1}{h+1})$-approximate matching, we see that most of the augmentations to reach an optimum matching are short. This immediately yields an $O(m/\epsilon)$-time approximation algorithm for maximum cardinality matching in bipartite graphs by running the linear-time subroutine for $O(1/\epsilon)$ iterations.

Combinatorial matroid intersection algorithms also rely on an augmenting path approach via so-called exchange graphs (defined formally in Section 2.1). Readers familiar with matroid intersection algorithms are aware that not all paths in the exchange graph result in a valid augmenting path. Moreover, the exchange graph changes in an opaque fashion with each augmentation, making it difficult to find more than a single augmenting path in a single pass. In an important paper that we make essential use of, Cunningham [Cun86] showed that with technical care, the shortest augmenting path approach of Hopcroft and Karp can still be extended to matroid intersection, leading to an algorithm with running time $O(nk^{1.5}Q)$. Implicit in his work is the following theorem.

THEOREM 1.2. ([CUN86]) *A $(1-\epsilon)$-approximation for maximum cardinality independent set in the intersection of two matroids can be obtained with running time bounded by $O(nk/\epsilon)$ independence queries.*

Scaling is a powerful methodology for solving weighted problems via algorithms for the corresponding unweighted problem. There is a long history of the effectiveness of this approach; we refer the reader to the work of Edmonds and Karp [EK72], Gabow [Gar85], Gabow and Tarjan [GT89, GT91], and many others that followed. Typically, scaling leads to an exact algorithm for the weighted problem whose running time matches that of the unweighted algorithm modulo polylogarithmic factors. Sometimes an approximation algorithm for the weighted case is implicit in these scaling algorithms, but the running time is still proportional to the time of the exact unweighted algorithm. The recent work of Huang *et al.* [HKK14] uses such a scaling approach to obtain a $(1-\epsilon)$-approximation for weighted matroid intersection. We note that theirs is the first work to consider approximate weighted matroid intersection. In particular, they use the weight-splitting approach of Frank [Fra81], scaling, and exact algorithms for unweighted matroid intersection, and our work is influenced by their high-level framework.

Duan and Pettie's algorithm for matching is also a scaling algorithm; however, they do not solve the unweighted problem to optimality. At a high level, we interpret their approach as incorporating a $(1-\epsilon)$-approximation for maximum cardinality matching in a scaling framework to obtain a $(1-\epsilon)$-approximation for weighted matching. (The algorithm in [DP14] is not actually structured in this fashion.) We explicitly use such a decomposition and show that it can be implemented in the setting of weighted matroid intersection via Cunningham's approximate cardinality algorithm. We briefly point out why this is challenging. The main difficulty in matroid intersection is in dealing with the exchange graph for augmentations. Frank's approach substantially simplified previous weighted matroid intersection algorithms by using a weight splitting approach instead of explicitly maintaining dual variables. However, this necessitates the use of, and reasoning by, auxiliary matroids induced by weights on the elements of the ground set. The proof of correctness crucially relies on properties of these derived matroids, and how augmentations in their exchange graphs preserve optimality in the original weighted setting. Unlike Huang *et al.* [HKK14], who run Cunningham's algorithm to optimality, we stop Cunningham's algorithm after a few iterations; our technical task then is to adjust the weights of the elements in such a way that the invariants with respect to the auxiliary matroids are preserved (which helps reason about the correctness of the algorithm), while also ensuring that the adjustment is controlled (such that we still derive a $(1-\epsilon)$-approximation). Whereas Duan and Pettie search once for augmenting paths per iteration, we make $O(1/\epsilon)$ searches per iteration, which leads to the quadratic dependence of $1/\epsilon^2$ in the running time.

There has been substantial work on matroid intersection for specific classes of matroids, including linear matroids [Cun86, GX96, Har09, Har07, CKL13], graphic matroids [GX89], and other classes [FS89]. In specific classes of matroids it is possible to take advantage of the structure of the problem to obtain faster running times; techniques include data structures, modifications to augmenting path algorithms, and algebraic methods (particularly for linear matroids [Har09, CKL13]). Our focus here is on the oracle model, but the high-level approach of incorporating an approximation for the unweighted case in a scaling framework may lead to faster running times for specific settings of interest. The work of Huang *et al.* [HKK14] and ours raises several interesting open questions on approximate matroid intersection which we leave to future work.

1.2 Paper organization

In Section 2, we review basic definitions and results as they are needed for the main approximation algorithm. In the course of reviewing Cunningham's algorithm, we observe that it implies a $(1-\epsilon)$ approximation for the cardinality case. In Section 3, we study an unweighted matroid induced by a weighted matroid. Establishing basic observations for this auxiliary matroid will greatly ease the burden of proof when subsequently presenting the algorithm. In Section 4, we describe and analyze the approximation algorithm `scaled-weight-splitting` for weighted matroid intersection.

2 Preliminaries

A maximum-cardinality independent set of an independence system is a *base*. In a matroid any two bases have the same cardinality. Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid, and $I \in \mathcal{I}$. A set not in $\mathcal{I}$ is dependent. A *circuit* is a minimal dependent set. The *span* of I, written $\text{span}(I)$, is the set of elements $e \in \mathcal{N}$ such that $I+e$ is dependent. An element $e \in \mathcal{N} \setminus I$ such that $I+e \in \mathcal{I}$ is *free*, and we write $\text{free}(I) \stackrel{\text{def}}{=} \mathcal{N} \setminus \text{span}(I)$. An *exchange* is a pair (d, e) of elements $d \in I$ and $e \in \text{span}(I) \setminus I$ such that $I - d + e \in \mathcal{I}$. It is well-known that an exchange is a replacement that preserves span.

2.1 Exchange graphs

Increasing an independent set in the matroid intersection is trickier than in a single matroid. Let $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ be an independent set, and suppose we add an element $e_1 \in \text{free}_1(I)$ that is free in $\mathcal{M}_1$. Then $I + e_1$ may be dependent in $\mathcal{M}_2$, in which case we need to pick an element $d_2 \in I$ such that $I - d_2 + e_1 \in \mathcal{I}_2$. Removing d_2 may free a third element e_2 in $\mathcal{M}_1$ (i.e., $I + e_1 - d_2 + e_3 \in \mathcal{I}_1$) that we add. But now the set may be dependent in $\mathcal{M}_2$, and we have to delete another element, and so on and so forth.

To efficiently find sequences of additions and deletions that improve I, one considers an auxiliary structure called the *exchange graph*. The exchange graph is a directed graph $\mathcal{G}$ whose arcs encode exchanges in either matroid. Its vertex set consists of $\mathcal{N} \cup \{s, t\}$, where s and t are artificial vertices that play the roles of source and sink. The arcs of the exchange graph are of the form

- (s, e) where $e \in \text{free}_1(I)$;
- (d, e) where $d \in I$, $e \in \text{span}_1(I) \setminus I$, and d and e form an exchange in $\mathcal{M}_1$;
- (e, d) where $d \in I$, $e \in \text{span}_2(I) \setminus I$, and d and e form an exchange in $\mathcal{M}_2$; and
- (e, t) where $e \in \text{free}_2(I)$.

An *augmenting path* is a path P from s to t such that $I \Delta (P \setminus \{s, t\})$ is independent in both $\mathcal{M}_1$ and $\mathcal{M}_2$. For ease of notation, we will sometimes write $I \Delta P$ when we really mean $I \Delta (P \setminus \{s, t\})$.

Not all paths from s to t are augmenting paths, because one exchange along a path may invalidate subsequent exchanges. We say a path $v_1 \to v_2 \to \cdots \to v_k$ is *minimal* if it has no shortcuts; that is, there are no arcs of the form (v_i, v_j) where $j > i + 1$, that imply a shorter sub-path from v_1 to v_k.

Lemma 2.1. ([Kro75], see also [Law76, Lemma 3.1]) *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids, $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ an independent set, $\mathcal{G} = (\mathcal{N} \cup \{s, t\}, \mathcal{E})$ the exchange graph of I, and P a path from s to t. If P is a minimal path from s to t, then P is an augmenting path.*

Augmenting paths are the primary (if not only) way to increase the cardinality in a matroid intersection. By adding them one by one, we eventually obtain the maximum cardinality independent set. This naive augmenting path algorithm requires $O(nk^2)$ independence queries: we build a new exchange graph after each of k augmentations, and an exchange graph requires $O(nk)$ calls to the independence oracle, one for every possible edge.

For more advanced algorithms, we leverage the fact that an augmenting path does not free an element, in the following well-known sense.

Lemma 2.2. *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids, $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ an independent set, and P an augmenting path for I. Then $\text{free}_1(I \Delta P) \subsetneq \text{free}_1(I)$ and $\text{free}_2(I \Delta P) \subsetneq \text{free}_2(I)$.*

Proof. By symmetry, it suffices to prove that $\text{free}_1(I \Delta P) \subsetneq \text{free}_1(I)$. In $\mathcal{M}_1$, augmenting I by P can be broken into first adding a free element to I, and then performing a sequence of exchanges. Adding a free element only increases the span, while each exchange preserves the span. The span of I is a strict subset of the span of $I \Delta P$ because $I \Delta P$ spans the free element first added to I. ■

2.2 Cunningham's Algorithm

Not unlike Hopcroft and Karp's algorithm for bipartite matching, Cunningham observes an inverse relationship between the cardinality of an independent set and the length of its shortest augmenting path.

Lemma 2.3. ([Cun86, Corollary 2.2]) *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids over a common ground set $\mathcal{N}$, with $\text{rank}(\mathcal{M}_1 \cap \mathcal{M}_2) = k$, and let $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ be an independent set in their intersection. If $|I| < k$, then there exists an augmenting path of length at most $2|I|/(k - |I|) + 1$.*

The core driver of Cunningham's algorithm is a subroutine that we refer to as `batch-augment`. If ℓ is the length of the shortest augmenting path for an independent set $I \in \mathcal{I}$, then `batch-augment` augments I along a *maximal* collection of disjoint augmenting paths of length ℓ, after which the length of the shortest augmenting path has strictly increased.

Lemma 2.4. ([Cun86]) *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids, and let $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ be an independent set in their intersection. Let ℓ be the length of the shortest augmenting path for I. `batch-augment`$(\mathcal{M}_1, \mathcal{M}_2, I)$ computes, in $O(nk)$ independence tests and $O(nk)$ time, a set of disjoint augmenting paths $\mathcal{P}$ such that*

(a) $I' \stackrel{\text{def}}{=} I \Delta (\bigcup_{P \in \mathcal{P}} P) \in \mathcal{I}_1 \cap \mathcal{I}_2$, and
(b) the length of the shortest augmenting path for I' is at least $\ell + 2$.

Cunningham's algorithm calls `batch-augment` about $2\sqrt{k}$ times on an initially empty set $I = \emptyset$. The first $\sqrt{k}$ iterations increases the length of the shortest augmenting path for I to at least $2\sqrt{k}$, which by Lemma 2.3 implies that I has cardinality about $k - \sqrt{k}$. The second $\sqrt{k}$ iterations increase the cardinality of I to k, as desired. The total algorithm takes $O(k^{1.5}n)$ independence queries and running time. If we instead call

`batch-augment` $O(1/\epsilon)$ times, then by Lemma 2.3, I has cardinality at least $(1-\epsilon)k$.

REMARK 2.1. *We do not require the full power of an independence oracle to implement* `batch-augment`. *The subroutine only requires the following queries for each matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$:*

(a) Given an independent set $I \in \mathcal{I}$ and an element $f \notin I$, is f in free(I)*?*

(b) Given an independent set $I \in \mathcal{I}$, element $d \in I$, and element $e \notin I$, is (d, e) an exchange?

Of course, either query is easily implemented in a constant number of calls to an independence oracle. This technicality will be important later, when the proposed algorithm `scaled-weight-splitting` calls `batch-augment` on auxiliary matroids that are not equipped with independence oracles.

2.3 Weight splittings

The algorithm for weighted matroid intersection proposed by Lawler [Law75] is an intricate primal-dual algorithm based on an exponential-size LP. Frank's algorithm [Fra81] is simpler, and is founded on the following basic observation.

LEMMA 2.5. ([FRA81, LEMMA 3]) *Let $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ be an independent set and $w = w_1 + w_2$ a decomposition of weights such that*

(i) I is a maximum weight independent set in $\mathcal{I}_1$ with respect to w_1, and

(ii) I is a maximum weight independent set in $\mathcal{I}_2$ with respect to w_2.

Then I is a maximum w-weight independent set in $\mathcal{I}_1 \cap \mathcal{I}_2$.

We say that two weights w_1 and w_2 are a *weight splitting* if $w_1 + w_2 = w$. Frank's algorithm builds an independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ in conjunction with a weight splitting (w_1, w_2) and terminates when they satisfy Lemma 2.5.

3 A matroid induced by weights

Before diving into the intersection of two weighted matroids, we pause for some structural observations regarding a single weighted matroid. When the objective is to maximize weight over independent sets, there is a natural reduction from weighted matroids to unweighted matroids.

LEMMA 3.1. ([FRA08, PROPOSITION 2]) *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, and let $\mathcal{B}^w$ be the collection of bases of $\mathcal{M}$ with maximum w-weight; that is,*

$$\mathcal{B}^w = \{B \in \mathcal{B} : w(B) \geq w(I) \text{ for all } I \in \mathcal{I}\},$$

where $\mathcal{B}$ denotes the set of bases of $\mathcal{M}$. Then $\mathcal{B}^w$ is the set of bases of a matroid $\mathcal{M}^w = (\mathcal{N}, \mathcal{I}^w)$.

This matroid $\mathcal{M}^w$ is the *w-induced matroid* of $\mathcal{M}$. We use weight-induced matroids to take advantage of Cunningham's techniques for unweighted matroids.

At this point, we know that $\mathcal{M}^w$ is a well-defined matroid but know little of its shape. Building some intuition for $\mathcal{M}^w$ helps motivate the design of our algorithm and eases the burden of proving correctness[1]. Several of these properties seem to be known, but we could not easily find proofs in the literature. We reprove the claims as needed.

LEMMA 3.2. *Let $I \in \mathcal{I}^w$, and let $d \in I$ and $e \in \text{span}(I)$ be an exchange in $\mathcal{M}$. Then $w(d) \geq w(e)$.*

Proof. The inequality is clear when I is a maximum weight base. Otherwise, if B is a maximum weight base containing I, then (d, e) is also an exchange pair in B, and the claim follows. ■

The subroutine `batch-augment`, which we will apply to weight-induced matroids, is ultimately an augmenting path algorithm. The implementation requires some basic and tangible conditions to identify exchanges and free elements with respect to weighted matroids.

LEMMA 3.3. *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with ground set weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, let $I \in \mathcal{I}^w$, and let $d \in I$ and $e \in \text{span}_1(I)$ be an exchange in the (original) matroid $\mathcal{M}$. Then d and e form an exchange in the weight-induced matroid $\mathcal{M}^w$ iff $w(d) = w(e)$.*

Proof. By Lemma 3.2, $w(d) \geq w(e)$. If d and e form an exchange in $\mathcal{I}^w$, then e and d form an exchange with respect to $I - d + e$ (with their roles reversed) in $\mathcal{M}^w$, whence (again) by Lemma 3.2 we have $w(e) \geq w(d)$.

Conversely, suppose $w(d) = w(e)$. Let B be a maximum weight base containing I. By the augmentation property, we can extend $I - d + e$ to a base C such that $I - d + e \subseteq C \subseteq B + e$. Because e and d form an exchange, $I + e = (I - d + e) + d$ is dependent, and we deduce that $B \setminus C = \{d\}$ and $C \setminus B = \{e\}$. We have,

$$w(C) = w(C - e) + w(e) = w(B - d) + w(d) = w(B).$$

Thus, C is a maximum weight base, and as a subset of C, $I - d + e \in \mathcal{I}^w$. ■

LEMMA 3.4. *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, let $I \in \mathcal{I}^w$, and let $f \in \text{free}(I)$ be a maximum weight free element. Then $I + f \in \mathcal{I}^w$.*

Proof. Let B be a maximum weight base containing I, and suppose B does not contain f. By the augmentation

[1] One can also prove the correctness of Frank's algorithm in terms of weight-induced matroids.

property, we can extend $I + f$ to a base C such that $I + f \subseteq C \subseteq B + f$. The difference $C \setminus B$ consists of a single element b. If b is not in C, then b is not in I, and in particular, $b \in \text{free}(I)$. By choice of f, we have $w(f) \geq w(b)$. Thus,

$$\begin{aligned} w(C) - w(B) &= w(C \setminus B) - w(B \setminus C) \\ &= w(f) - w(b) \\ &\geq 0, \end{aligned}$$

and C is a maximum weight base. As a subset of C, $I + f \in \mathcal{I}^w$. ∎

Lastly, we lay out one set of necessary conditions to assert that an independent set $I \in \mathcal{I}$ is also independent in $\mathcal{I}^w$. This is particularly handy when we start modifying the weights while trying to keep our current solution I independent in the weight-induced matroids.

LEMMA 3.5. *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with ground set weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, and let $I \in \mathcal{I}$. Suppose*

(i) For all $f \in \text{free}(I)$, and $d \in I$, $w(d) \geq w(f)$.

(ii) For all pairs $d \in I$ and $e \in \text{span}(I) \setminus I$ that form an exchange, $w(d) \geq w(e)$.

Then $I \in \mathcal{I}^w$.

Proof. Let $\ell = |I|$. We first show that I is a maximum weight independent ℓ-set. Let $J \in \mathcal{I}^{(\ell)}$ be another independent set with cardinality ℓ. I and J are both bases in the truncation $\mathcal{M}^{(\ell)}$ of $\mathcal{M}$ to independent sets of cardinality ℓ or less. Let $\varphi : I \setminus J \longleftrightarrow J \setminus I$ be a bijection such that for all $e \in I \setminus J$, $I - d + \varphi(d) \in \mathcal{I}$. Fix $d \in I$. We either have

(a) $\varphi(d) \in \text{free}(I)$, in which case $w(d) \geq w(e)$ by assumption (i).

(b) $\varphi(d) \in \text{span}(I) \setminus I$, in which case d and $\varphi(d)$ form an exchange and $w(d) \geq w(\varphi(d))$ by assumption (ii).

Thus,

$$\begin{aligned} w(I) &= w(I \cap J) + w(I \setminus J) \\ &\geq w(I \cap J) + w(\varphi(I \setminus J)) \\ &= w(J), \end{aligned}$$

and I is a maximum weight independent ℓ-set.

Now, let B be a maximum weight base. By the augmentation property, we can extend I to a base C such that $I \subseteq C \subseteq B \cup I$. $B \setminus (C \setminus I)$ is an ℓ-set, so $w(I) \geq w(B \setminus (C \setminus I))$. We have,

$$\begin{aligned} w(C) &= w(I) + w(C \setminus I) \\ &\geq w(B \setminus (C \setminus I)) + w(C \setminus I) \\ &= w(B), \end{aligned}$$

so C is a maximum weight base. As a subset of C, $I \in \mathcal{I}^w$. ∎

3.1 Weighted restrictions

Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with ground set weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$. For $c \in \mathbb{R}$, the *weight c restriction* of $\mathcal{M}$ is the restriction $\mathcal{M}|_{\mathcal{N}_c}$, where $\mathcal{N}_c \stackrel{\text{def}}{=} \{e \in \mathcal{N} : w(e) \geq c\}$ is the set of elements with weight at least c. For a general set $S \subseteq \mathcal{N}$, we denote the intersection of S and $\mathcal{N}_c$ as $S_c \stackrel{\text{def}}{=} S \cap \mathcal{N}_c = \{e \in S : w(e) \geq c\}$.

Restrictions by weight are of particular interest to our scaling architecture. The algorithm maintains a sort of water level c (say $c = 2^i$, for some integer $i \in \mathbb{Z}$), and restricts its attention elements of weight $\geq c$. More precisely, we operate in matroids of the form $(\mathcal{M}^w)|_{\mathcal{N}_c}$. It is desirable to understand the structure of $(\mathcal{M}^w)|_{\mathcal{N}_c}$, particularly in its similarity and contrast to the unrestricted matroid $\mathcal{M}^w$.

Our first lemma is a sort of sanity check to verify that a maximum weight base restricts to a maximum weight base within any $\mathcal{N}_c$.

LEMMA 3.6. *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with ground set weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, and let $c \in \mathbb{R}$. For any maximum weight base $B \in \mathcal{B}^w$, $B \cap \mathcal{N}_c$ is a base of $\mathcal{M}|_{\mathcal{N}_c}$.*

Proof. If not, then let $f \in \mathcal{N}_c \cap \text{free}(B_c)$. By the augmentation property, we can extend $B_c \cap \mathcal{N}_c + f$ to a base C such that $B_c + f \subseteq C \subseteq B + f$. Let d be the unique element in $B \setminus C$. Then $d \notin \mathcal{N}_c$, so $w(f) \geq c > w(d)$. We have,

$$\begin{aligned} w(C) &= w(B - d) + w(f) \\ &> w(B - d) + w(d) \\ &= w(B), \end{aligned}$$

and B is not a maximum weight base, a contradiction. ∎

The following lemma extends Lemma 3.6 to show that the operations of taking weight induced matroids and restricting elements by weight are compatible with one another.

LEMMA 3.7. *Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with ground set weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, and let $c \in \mathbb{R}$. Then $(\mathcal{M}|_{\mathcal{N}_c})^w = (\mathcal{M}^w)|_{\mathcal{N}_c}$.*

Proof. Let $B \in \mathcal{B}^w$ be a maximum weight base, and let C be a base in $\mathcal{M}|_{\mathcal{N}_c}$. By the augmentation property, we can extend C to a base D in $\mathcal{M}$ such that $C \subseteq D \subseteq B \cup C$. Since C spans B_c and $|C| = |B_c|$, we have $D \setminus C = B \setminus B_c$.

Let $\varphi : B \setminus D \longleftrightarrow D \setminus B$ be a bijection such that for all $b \in B \setminus D$, $B - b + \varphi(b) \in \mathcal{I}$. Note that since $B \setminus D = B_c \setminus C$ and $D \setminus B = C \setminus B_c$, φ is a bijection from

```
I ← ∅, w1 ← w, w2 ← 0
for each scale as a power of 2 from W down to 1
  repeat 1/ϵ times:
    restrict N to elements with weight at least the current scale
    run batch-augment on I in M1^{w1} ∩ M2^{w2} a fixed number of times
    shift weight from w1 to w2 by an ϵ-fraction of the current scale
return I
```

Figure 1: Sketch of the algorithm scaled-weight-splitting

$B_c \setminus C$ to $C \setminus B_c$, such that for all $b \in B_c \setminus C$, we have $B - b + \varphi(b) \in \mathcal{I}$. Because B is a maximum weight base, for all $b \in B_c \setminus C$, we have $w(b_c) \geq w(\varphi(b))$. Therefore,

$$\begin{aligned} w(B_c) &= w(B_c \cap C) + w(B_c \setminus C) \\ &\geq w(B_c \cap C) + w(\varphi(B_c \setminus C)) \\ &= w(C), \end{aligned}$$

and B_c is a maximum weight base in $\mathcal{M}|_{\mathcal{N}_c}$. Since any $I \in (\mathcal{I}^w)|_{\mathcal{N}_c}$ is a subset of B_c for some maximum weight base B, this implies that $(\mathcal{I}^w)|_{\mathcal{N}_c} \subseteq (\mathcal{I}|_{\mathcal{N}_c})^w$,

On the other hand, if C is a maximum weight base in $\mathcal{M}|_{\mathcal{N}_c}$, then $w(C) \geq w(B_c)$, and we have

$$\begin{aligned} w(D) &= w(D \setminus C) + w(C) \\ &\geq w(B \setminus B_c) + w(B_c) \\ &= w(B), \end{aligned}$$

so D is a maximum weight base in $\mathcal{M}^w$. As a subset of D, $C \in \mathcal{I}^w$, and more generally, $(\mathcal{I}|_{\mathcal{N}_c})^w \subseteq (\mathcal{I}^w)|_{\mathcal{N}_c}$. ■

4 A scaling approximation for weighted matroid intersection

We first present an approximation algorithm, scaled-weight-splitting, that has a slightly slower running time than our final algorithm but captures the main ideas. The running time depends on $\log W$ where W is the maximum weight of any element. Removing the $\log W$ factor is presented afterwards as an adjustment to scaled-weight-splitting. A high level sketch of scaled-weight-splitting is given in Figure 1; the full algorithm is in Figure 2.

First and foremost, scaled-weight-splitting is a *scaling* algorithm. A *scale*, in this setting, is a natural number between 1 and $\log W$, where W is the maximum weight. At each scale i, we only consider elements e with $w(e) = \Omega(2^i)$. To some extent, the scaling algorithm reduces the weighted problem to an unweighted one at each scale.

Within a scale i, there is an inner loop that repeats $1/\epsilon$ times. Henceforth, an *iteration* will mean a pair (i, j), where i is the scale and j is the index of the inner loop. There are $\ell = O(\log(W/\epsilon)/\epsilon)$ total iterations over the algorithm.

Each iteration of the inner loop tries to augment I with the approximate version of Cunningham's algorithm, calling `batch-augment` only ℓ times. (Running Cunningham's exact algorithm to completion, of course, forfeits any chance of improving on Cunningham's running time.) We then shift $\epsilon 2^i$ weight per element from w_1 to w_2, by a similar Hungarian-style technique as Frank's algorithm.

Interrupting Cunningham's algorithm at a fixed length results in an incomplete Hungarian tree, forcing some loss in approximation. We truncate the tree further at a carefully selected depth, and then "cheat" by adding a small amount of extra weight to elements in I that are cut off. This extra weight is stored in a map denoted excess : $\mathcal{N} \to \mathbb{R}_{\geq 0}$, which has value 0 for any element not in I. The excess weight helps maintain certain invariants for the sake of basic correctness, but hurts the approximation ratio. The amount of excess added per element cut-off, $\epsilon 2^i$ at an iteration (i, j), is an $O(\epsilon)$ fraction of any element in play at scale i. Because the excess is built by these relatively small increments, and the stunted Hungarian trees are sufficiently deep, we can bound the total amount of excess over the entire algorithm to an $O(\epsilon)$ fraction of $w(I)$.

Note that we do *not* maintain an exact weight splitting: in general, $w_1(e) + w_2(e)$ may not equal $w(e)$ for any element $e \in \mathcal{N}$.

THEOREM 4.1. *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids over a common ground set $\mathcal{N}$, weighted by $w : \mathcal{N} \to [1, W]$. Then scaled-weight-splitting $(\mathcal{N}, \mathcal{I}_1, \mathcal{I}_2, w, \epsilon)$ returns an independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ such that for all $J \in \mathcal{I}_1 \cap \mathcal{I}_2$, $w(I) \geq (1 - O(\epsilon))w(J)$. Its running time is bounded by $O(nk \log^2(W/\epsilon)/\epsilon^2)$ calls to an independence oracle.*

```
scaled-weight-splitting(𝒩, ℐ₁, ℐ₂, w, ε)
  // assume 1/ε is an integer and min_{e∈𝒩} w(e) = 1
  I ← ∅
  b₁ ← ⌈log max_{e∈𝒩} w(e)⌉,  b₂ ← ⌈log ε⌉,  ℓ ← (b₁ + 1 − b₂)/ε
  for all e ∈ 𝒩
    w₂(e) ← 0,  excess(e) ← 0
    δ ← ε2^{⌊log w(e)⌋}
    w₁(e) ← δ · ⌊w(e)/δ⌋
  for i = b₁ down to b₂
    // * start of scale i *
    δ_i ← ε2^{i−1}
    for j = 1, ..., 1/ε
      // * start of iteration (i, j) *
      𝒩_{i,j} ← {e ∈ 𝒩 : w₁(e) ≥ 2^i − (j − 1)δ_i}
      repeat ℓ times
        I ← batch-augment(ℳ₁^{w₁}|_{𝒩_{i,j}}, ℳ₂^{w₂}|_{𝒩_{i,j}}, I)
      for each m ∈ ℕ, let S_m ⊆ 𝒩_{i,j} be the set of elements whose shortest
        path from s in the exchange graph of (ℳ₁^{w₁} ∩ ℳ₂^{w₂})|_{𝒩_{i,j}} has length m
      let m* ∈ {2, 4, 6, ..., 2ℓ} minimize w(S_{m*}) // S_{m*} ⊆ I
      for all e ∈ S₁ ∪ S₂ ∪ ··· ∪ S_{m*−1}
        w₁(e) ← w₁(e) − δ_i,  w₂(e) ← w₂(e) + δ_i
      for all e ∈ S_{m*}
        w₂(e) ← w₂(e) + δ_i,  excess(e) ← excess(e) + δ_i
      for all e ∈ 𝒩 \ I // reset excess for all e ∉ I
        w₁(e) ← w₁(e) − excess(e),  excess(e) ← 0
    // * end of iteration (i, j) *
  // * end of scale i *
  return I
```

Figure 2: Full details of the algorithm scaled-weight-splitting

4.1 Correctness

We first prove the correctness of Theorem 4.1. The goal of the algorithm is to approximate the winning conditions of Frank's algorithm (Lemma 2.5). Formally, we target the following.

LEMMA 4.1. *Let $\mathcal{M}_1 = (\mathcal{N}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{N}, \mathcal{I}_2)$ be two matroids over a common ground set $\mathcal{N}$, weighted by $w : \mathcal{N} \to \mathbb{R}_{\geq 1}$, and let $\epsilon > 0$. Let $w_1, w_2 : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be two sets of weights and $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ an independent set such that*

(i) w_1 and w_2 are approximately a weight-splitting of w; specifically, $w_1(e) + w_2(e) \geq (1 - O(\epsilon))w(e)$ for all $e \in \mathcal{N}$, and $w_1(I) + w_2(I) \leq (1 + O(\epsilon))w(I)$.

(ii) The free elements have nearly zero weight; namely, for all $f \in \text{free}_1(I)$, $w_1(f) \leq \epsilon$, and for all $g \in \text{free}_2(I)$, $w_2(g) = 0$.

(iii) $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.

Then I is a $(1 - O(\epsilon))$-approximation for the maximum weight independent set in the intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$.

Proof. Let $J \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a competing independent set. We have,

$$\begin{aligned} w(J) &\leq (1 + O(\epsilon))(w_1(J) + w_2(J)) && \text{by (i)}, \\ &= (1 + O(\epsilon))\big[w_1(J \cap \text{span}_1(I)) \\ &\qquad + w_1(J \cap \text{free}_1(I)) \\ &\qquad + w_2(J)\big]. \end{aligned} \tag{4.1}$$

We analyze each of the three terms $w_1(J \cap \text{span}_1(I))$, $w_1(J \cap \text{free}_1(I))$, and $w_2(J)$ individually. If $I \in \mathcal{I}_1^{w_1}$, then by Lemma 3.2, we have $w_1(J \cap \text{span}_1(I)) \leq w_1(I)$. As per the second term, for each $f \in J \cap \text{free}_1(I)$, we have $w_1(f) \leq \epsilon/2$ by (ii). Assuming $w(e) \geq 1$ and hypothesis (i), we have $w_2(f) \geq 1 - O(\epsilon)$ for each f, and in sum, $w_1(J \cap \text{free}_1(I)) \leq O(\epsilon w_2(J \cap \text{free}_1(I)))$. Finally, if $I \in \mathcal{I}_2^{w_2}$ and $w_2(\text{free}_2(I)) = 0$, then I is a maximum

w_2-weight independent set in $\mathcal{M}_2$ and $w_2(J) \le w_2(I)$. Plugging these inequalities into equation (4.1), we have,

$$\begin{aligned} w(J) &\le (1+O(\epsilon))(w_1(I) + O(\epsilon)w_2(I) + w_2(I)) \\ &\le (1+O(\epsilon))(w_1(I) + w_2(I)). \end{aligned}$$

By assumption (i), $w_1(I) + w_2(I) \le (1+O(\epsilon))w(I)$, so we have $w(J) \le (1+O(\epsilon))w(I)$, as desired. ■

With Lemma 4.1 in mind, we now begin to analyze the algorithm. We first observe that all relevant quantities at a scale i are integer multiples of δ_i.

LEMMA 4.2. *At any point in iteration (i,j), for all $e \in \mathcal{N}_{i,j}$, $w_1(e) \in \delta_i \cdot \mathbb{Z}$ and $w_2(e) \in \delta_i \cdot \mathbb{Z}$.*

Proof. Let $e \in \mathcal{N}_{i,j}$. Observe that for all preceding scales $i' < i$, $\delta_{i'}$ is an even multiple of δ_i, because the δ_i's simply half from one scale to the next. At the beginning of the algorithm, $w_1(e) \in \delta_i' \cdot \mathbb{Z}$ and $w_2(e) = 0$ for some $i' \le i$. All subsequent adjustments to $w_1(e)$ and $w_2(e)$ were in units of $\delta_{i''}$ for various scales i'' between i' and i, which are each divisible by δ_i. Thus, $w_1(e) \in \delta_i \cdot \mathbb{Z}$ and $w_2(e) \in \delta_i \cdot \mathbb{Z}$, as desired. ■

Next, we set a foundation for how $w_1(e)$ and $w_2(e)$ generally develop across iterations. The following lemma shows that the maximum w_1-weight for free elements in $\mathcal{M}_1$ is gradually decreasing, all the way down to $\epsilon/2$ at the end of the algorithm. This addresses the first half of condition (ii) in Lemma 4.1.

LEMMA 4.3. *(a) At the beginning of iteration (i,j), $w_1(e) \le 2^i + (1-j)\delta_i$ for all $e \in \text{free}_1(I)$.*
(b) At the end of iteration (i,j), $w_1(e) \le 2^i - j\delta_i$ for all $e \in \text{free}_1(I)$.
(c) At the end of the algorithm, $w_1(e) \le \epsilon/2$ for all $e \in \mathcal{N} \setminus \text{span}_1(I)$.

Proof. We prove (a) and (b) together in one large induction argument on (i,j). Condition (c) simply restates (b) at the end of the final iteration.

At the beginning of the first iteration $(b_1, 1)$, we have $w_1(e) \le 2^{b_1}$ for all e. For any subsequent scale (i,j), condition (a) matches condition (b) at the beginning of the previous iteration (either $(i, j-1)$ or $(i-1, 1/\epsilon)$, depending on j).

During iteration (i,j), we grow I only by augmenting paths, so by Lemma 2.2, $\text{free}_1(I)$ does not increase. We decrement $w_1(e)$ by δ_i for all $e \in \mathcal{N}_{i,j} \cap \text{free}_1(I)$ because these are precisely the elements in layer S_1. Subtracting δ_i from condition (a) at the beginning of iteration (i,j) gives $w_1(e) \le 2^i + (1-j)\delta_i - \delta_i = 2^i - j\delta_i$ at the end of iteration (i,j), hence (b). ■

The story for $\mathcal{M}_2$-free elements is simpler.

LEMMA 4.4. *At any point in the algorithm, for all $e \in \text{free}_2(I)$, $w_2(e) = 0$.*

Proof. Initially, w_2 is zero everywhere. By Lemma 2.2, the augmentation loop of each iteration does not free any elements in $\mathcal{M}_2$. In the weight-update step of each iteration, the algorithm only increases w_2 on elements whose distance from the source s in the exchange graph is less than or equal to 2ℓ. By Lemma 2.4, running `batch-augment` for ℓ iterations ensures that all augmenting paths have length strictly greater than 2ℓ. Since any $\mathcal{M}_2$-free element e would be the endpoint of an augmenting path, this puts any $\mathcal{M}_2$-free elements out of reach of the weight update. ■

The next lemma shows that the elements in I always have larger w_1-weight than any $\mathcal{M}_1$-free elements. This is particularly relevant in light of Lemma 3.5.

LEMMA 4.5. *(a) At the beginning of iteration (i,j), $w_1(d) \ge 2^i + (1-j)\delta_i$ for all $d \in I$.*
(b) After each call to `batch-augment` in iteration (i,j), we have $w_1(d) \ge 2^i + (1-j)\delta_i$ for all $d \in I$.
(c) At the end of scale i, $w_1(d) \ge 2^i$ for all $d \in I$.

Proof. At the beginning of the first iteration $(b_1, 1)$, $I = \emptyset$, so (a) is satisfied vacuously. By induction, for subsequent iterations (i,j), (a) is satisfied at the beginning of iteration (i,j) because (b) holds at the end of the previous iteration (either $(i, j-1)$ or $(i-1, 1/\epsilon)$).

When we run `batch-augment` in scale (i,j), we restrict the subroutine to elements in $\mathcal{N}_{i,j}$. Any element added to I, by membership in $\mathcal{N}_{i,j}$, has $w_1(e) \ge 2^i - (j-1)\delta_i$.

After ℓ iterations of `batch-augment` in scale (i,j), the algorithm proceeds to shift weight from w_1 to w_2. Subtracting the decrement of δ_i leaves $w_1(d) \ge 2^i - j\delta_i$ at the end of scale (i,j) for all $d \in I$, as desired. ■

The next task is to verify condition (iii) in Lemma 4.1, which is that $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.

LEMMA 4.6. *(a) At the beginning of iteration (i,j), $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.*
(b) After each call to `batch-augment` in iteration (i,j), we have $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.
(c) At the end of iteration (i,j), $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.
(d) At the end of the algorithm $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.

Proof. At the beginning of the first iteration $(b_1, 1)$, $I = \emptyset$ is independent in $\mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$. At the beginning of each subsequent iteration, condition (a) follows from condition (b) at the end of the previous iteration.

Before we call `batch-augment`, by Lemma 4.5 and Lemma 3.7, $I \in (\mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2})|_{\mathcal{N}_{i,j}}$. The subroutine

preserves independence, hence $I \in (\mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2})|_{\mathcal{N}_{i,j}}$ after it is called. By Lemma 3.7, this implies $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$.

Before the end of iteration (i, j), and after all calls to `batch-augment`, we shift weight from w_1 to w_2. If we had repeated `batch-augment` until I no longer improves (and effectively run all of Cunningham's algorithm), then we can use Frank's updating scheme, which guarantees to preserve $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$ because there are no augmenting paths. This is not quite the case here, where ℓ iterations of `batch-augment` only ensures that each augmenting path has length at least 2ℓ.

To this end, we cut off the Hungarian tree at some layer $m \leq 2\ell$, and we halt all updates in layers beyond layer m. All elements in layers $m' < m$ are updated as usual, and in layer m (which is an even layer, containing only elements of I), we do not decrease w_1 as we normally would. At a high level, the shortcuts we take strictly help I with respect to independence in $\mathcal{M}_1^{w_1} \cap \mathcal{M}_2^{w_2}$. Ultimately, one can verify case by case that this cheating preserves Lemma 3.5 directly, and $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$ at the end of iteration (i, j).

Let $d \in I$ and $e \notin I$ form an exchange in $\mathcal{M}_1$. If $w_1(d) > w_1(e)$, then by Lemma 4.2, the difference $w_1(d) - w_1(e)$ is at least δ_i, hence $w_1(d) \geq w_1(e)$ even after subtract δ_i from $w_1(e)$.

Otherwise, by Lemma 3.2, $w_1(d) = w_1(e)$. By Lemma 4.5, $e \in \mathcal{N}_{i,j}$, and by Lemma 3.3, (d, e) is an arc in the exchange graph of I in $\mathcal{M}_1^{w_1} \cap \mathcal{M}_2^{w_2}|_{\mathcal{N}_{i,j}}$. Therefore, the distance from the source s to e in the exchange graph is at most one more than the distance from s to d. If $w_1(d)$ is decreased, then $d \in S_{m'}$ for some layer $m' < m$. If $S_{m''}$ is the layer containing e, then $m'' \leq m' + 1 < m$. Therefore $w_1(e)$ is also decreased, by the same amount δ_i. After updating weights, we still have $w_1(d) \geq w_1(e)$.

Now, let $d \in I$ and $e \notin I$ form an exchange in $\mathcal{M}_2$. If $w_2(d) > w_2(e)$, then by Lemma 4.2, the difference between the weights is at least δ_i. Therefore, even if we increase $w_2(e)$ and not $w_2(d)$, we still have $w_2(e) \leq w_2(d)$.

Suppose instead that $w_2(d) = w_2(e)$. By Lemma 3.3, (e, d) is an arc in the exchange graph of $\mathcal{M}_1^{w_1} \cap \mathcal{M}_2^{w_2}$. If $w_2(e)$ is incremented, then e is in layer $S_{m'}$ for some $m' < m$. Because (e, d) is an arc, d is in layer $S_{m'+1}$ or earlier. Since $m' + 1 \leq m$, $w_2(d)$ is also incremented. Thus $w_2(d) \geq w_2(e)$ after the update.

The preceding case analysis shows that elements of I continues to dominate all exchanges in $(\mathcal{M}_1 \cap \mathcal{M}_2)|_{\mathcal{N}_{i,j}}$ after the weight update. By Lemma 4.3, and Lemma 4.5, we have $w_1(d) \geq w_1(e)$ for all $d \in I$ and $e \in \text{free}_1(I)$, and by Lemma 4.4, $w_2(d) \geq 0 = w_2(e)$ for all $d \in I$ and $e \in \text{free}_2(I)$. By Lemma 3.5, $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$ at the end of iteration (i, j), as desired.

Finally, condition (d) is just condition (c) at the end of the last iteration $(b_2, 1/\epsilon)$. ■

$I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$ is preserved in Lemma 4.6 because we give elements in I *extra* weight, stored in the vector excess. Cheating the weight splitting weakens the approximation ratio, and we have to show that the sum of violations is controlled in the aggregate.

LEMMA 4.7. *At the end of the algorithm, we have* $\text{excess}(I) \leq \epsilon w(I)$.

Proof. Consider an iteration (i, j), where we increase $\text{excess}(d)$ by δ_i for all $d \in S_{m^\star}$. By membership $d \in I$ at iteration (i, j), we have $w(d) \geq 2^i + (1 - j)\delta_i \geq 2^{i-1}$, and so $\delta_i \leq 2\epsilon w(d)$. Additionally, by choice of $m^\star$, $w(S_{m^\star}) \leq w(I)/\ell$, where ℓ is the total number of iterations. Thus, $\text{excess}(I)$ increases by at most

$$\delta_i \cdot |S_{m^\star}| \leq \epsilon \sum_{d \in S_{m^*}} w(d) \leq \epsilon w(I)/\ell$$

at iteration (i, j). Summed over all ℓ iterations, we have $\text{excess}(I) \leq \epsilon w(I)$, as desired. ■

We conclude the proof of correctness of Theorem 4.1 by showing that the algorithm meets the sufficient conditions of Lemma 4.1.

Proof of Theorem 4.1. Condition (ii), regarding small free weights, follows by Lemma 4.3 (e) and Lemma 4.4. Condition (iii), that $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$, follows Lemma 4.6. It remains to verify (i), which states that w_1 and w_2 are approximately a weight splitting of w.

Consider an element $e \in \mathcal{N}$. Let $w'(e) = \delta \lfloor w(e)/\delta \rfloor$, where $\delta = \epsilon 2^{\lfloor \log w(e) \rfloor}$. The discretized weight $w'(e)$ rounds $w(e)$ down inside a bucket of width $\delta \leq \epsilon w(e)$, so $w'(e) \geq (1 - \epsilon)w(e)$. The algorithm maintains the invariant that $w_1(e) + w_2(e) = w'(e) + \text{excess}(e)$. Since $\text{excess}(e)$ is non-negative, we have $w_1(e) + w_2(e) \geq w'(e) \geq (1 - \epsilon)w(e)$, as desired. On the other hand, by Lemma 4.7, we have $w_1(I) + w_2(I) = w'(I) + \text{excess}(I) \leq (1 + \epsilon)w(I)$, as desired. ■

4.2 Efficiency

Now we turn to proving the running time of $O(nk \log^2 W/\epsilon^2)$ calls to an independence oracle. The algorithm is clearly bound by $\ell^2 = \log^2(W)/\epsilon^2$ calls to `batch-augment`. It remains to prove that `batch-augment` runs in $O(nk)$ independence queries. This is not immediate because we are not given independence oracles for $\mathcal{M}_1^{w_1}$ and $\mathcal{M}_2^{w_2}$. The following lemma shows that we can nevertheless implement `batch-augment` efficiently.

LEMMA 4.8. *The subroutine* `batch-augment` *can be implemented with running time bounded by* $O(nk)$ *calls to independence oracles for* $\mathcal{M}_1$ *and* $\mathcal{M}_2$.

Proof. By Remark 2.1, it suffices to show that we can verify free elements and verify exchanges.

Suppose we call `batch-augment` with arguments $(\mathcal{M}_1^{w_1}|_{\mathcal{N}_{i,j}}, \mathcal{M}_2^{w_2}|_{\mathcal{N}_{i,j}}, I)$. By Lemma 4.3, all free elements in $\mathcal{M}_1^{w_1}|_{\mathcal{N}_{i,j}}$ have equal w_1-weight $(2^i - (j-1)\delta_i)$, and by Lemma 4.4, all free elements in $\mathcal{M}_2^{w_2}|_{\mathcal{N}_{i,j}}$ have 0 w_2-weight. By Lemma 3.4, checking if an element is free in the weight-induced matroids is reduced to an independence query in the underlying matroids.

For exchange queries, by Lemma 3.3, verifying an exchange in the weight-induced matroids is only a comparison of weights beyond verifying the exchange in the underlying matroids.

Thus, each query can be implemented with a constant number of independence queries to the underlying matroids. ■

4.3 Removing the dependency on W

A low-level adjustment removes the dependency on W in the running time of `scaled-weight-splitting`, achieving the bounds of Theorem 1.1. The adjustment is similar to that by Duan and Pettie for matching [DP14, Section 3.3], and we limit ourselves to a sketch. The full algorithm is given in Figure 3.

The adjustments are made on a per-scale basis. At scale i, we ignore all elements with weight greater than $\epsilon 2^i$. Let

$$\mathcal{N}_i^- = \{e \in \mathcal{N} : w(e) \le 2^i/\epsilon\}, \quad \mathcal{N}^+ = \mathcal{N} \setminus \mathcal{N}^-,$$
$$I^- = I \cap \mathcal{N}_i^-, \quad \text{and} \quad I^+ = I \setminus I^i$$

identify which elements have weight above and below the threshold $2^i/\epsilon$. Let $\mathcal{M}_{1,i} = \mathcal{M}_1/I^+$ and $\mathcal{M}_{2,i} = \mathcal{M}_2/I^+$ contract I^+ in each matroid. At scale i, we will augment $I^- \in \mathcal{M}_{1,i} \cap \mathcal{M}_{2,i}$, and combine it with I^+ at the end of the scale to recover $I = I^+ \cup I^- \in \mathcal{M}_1 \cap \mathcal{M}_2$. For each iteration (i,j), we redefine

$$\mathcal{N}_{i,j} = \{e \in \mathcal{N}_i : w(e) \ge 2^i - (j-1)\delta_i\}.$$

The work at iteration (i,j) is essentially the same, except we restrict $\mathcal{N}_{i,j}$, operate in the contracted matroids $\mathcal{M}_{1,i}$ and $\mathcal{M}_{2,i}$, and augment $I^- \in \mathcal{M}_{1,i}^{w_1} \cap \mathcal{M}_{2,i}^{w_2}$ instead of all of I. Contracting I^+ is trivial to implement and does not complicate the oracle model.

In this variation, an element participates in $(1/\epsilon)\log(1/\epsilon)$ rounds between the moment it first appears in $\mathcal{N}_{i,j}^+$ and the last time it appears in $\mathcal{N}_{i,j}^-$. We replace ℓ with $(1/\epsilon)\log(1/\epsilon)$ accordingly. This reduces the running time to meet the bound claimed by Theorem 1.1. It remains to address the correctness of the algorithm.

The restriction to $\mathcal{N}_i^-$ removes larger elements from participating in the augmentation loop or dual update. The same effect could be achieved by incrementing $w_1(e)$ by 2^i for all $e \in I^+$ and decreasing $w_1(e)$ by 2^i for all $e \in \mathcal{N}_i^+ \setminus I^+$ once at beginning of scale i, for then none of these elements would be reachable from the source s in the exchange graph. Adding extra weight to elements in I and removing extra weight from elements outside I only strengthens the invariant that $I \in \mathcal{I}_1^{w_1} \cap \mathcal{I}_2^{w_2}$ (Lemma 4.6).

We interpret the restriction to $\mathcal{N}_i^-$ as simulating these extra adjustments to w_1. To prove correctness, we need to show that for each element e, the sum of these virtual adjustments to $w_1(e)$ is relatively small. Suppose an element $e \in I$ leaves $\mathcal{N}_{i,j}^-$ at scale i, and accrues extra w_1-weight in increments of $2^{i'}$ for each scale $i' \ge i$. Because the increments are geometrically decreasing, and $2^i \le \epsilon w(e)$, the sum of virtual increments is at most

$$\sum_{i'=i}^{\infty} 2^{i'} = 2^i \sum_{m=0}^{\infty} \frac{1}{2^m} \le 2\epsilon\, w(e).$$

Similarly, we subtract at most $O(\epsilon w(e))$ from $w_1(e)$ for each $e \in \mathcal{N} \setminus I$. It follows that these virtual adjustments have a negligible effect on the quality of (w_1, w_2) as an approximate weight splitting.

Although we have decreased the length ℓ against which we amortize the excess cost, the number of times an element participates in the weight update stage is decreased by equal proportion. This allows us to repeat the proof of Lemma 4.7 for the smaller value of ℓ and obtain the same bounds on the excess.

All accounted for, restricting to $\mathcal{N}_i^-$ and filtering out large elements at each scale i has a negligible cumulative effect on the conditions required by Lemma 4.1, and the new algorithm still has a $(1 - O(\epsilon))$ approximation ratio.

```
lazy-scaled-weight-splitting(𝒩, ℐ₁, ℐ₂, w, ε)
  // we assume 1/ε is an integer and min_{e∈𝒩} w(e) = 1
  I ← ∅
  b₁ ← ⌈log max_{e∈𝒩} w(e)⌉,  b₂ ← ⌈log ε⌉,  ℓ ← ⌈log 1/ε⌉/ε
  for all e ∈ 𝒩
    w₂(e) ← 0,  excess(e) ← 0
    δ ← ε2^{⌊log w(e)⌋}
    w₁(e) ← δ⌊w(e)/δ⌋
  for i = b₁ down to b₂
    // * start of scale i *
    δᵢ ← ε2^{i−1}
    // exclude sufficiently large elements
    𝒩ᵢ⁻ ← {e ∈ 𝒩 : w(e) ≤ 2^i/ε},  𝒩⁺ ← 𝒩 \ 𝒩ᵢ
    I⁻ ← I ∩ 𝒩ᵢ⁻,  I⁺ ← I ∩ 𝒩ᵢ⁺
    let ℳ_{1,i} = ℳ₁/I⁺,  ℳ_{2,i} = ℳ₂/I⁺
    for j = 1, ..., 1/ε
      // * start of iteration (i, j) *
      𝒩_{i,j} ← {e ∈ 𝒩ᵢ⁻ : w(e) ≤ (1/ε)(2^i − (j − 1)δᵢ)}
      repeat ℓ times
        I⁻ ← batch-augment(ℳ^{w₁}_{1,i}|_{𝒩_{i,j}}, ℳ^{w₂}_{2,i}|_{𝒩_{i,j}}, I⁻)
      for each m ∈ ℕ, let S_m ⊆ 𝒩_{i,j} be the set of elements whose shortest
        path from s in the exchange graph of (ℳ^{w₁}_{1,i} ∩ ℳ^{w₂}_{2,i})|_{𝒩_{i,j}} has length m
      let m* ∈ {2, 4, 6, ..., 2ℓ} minimize w(S_{m*})  // S_{m*} ⊆ I⁻
      for all e ∈ S₁ ∪ S₂ ∪ ··· ∪ S_{m*−1}
        w₁(e) ← w₁(e) − δᵢ,  w₂(e) ← w₂(e) + δᵢ
      for all e ∈ S_{m*}
        w₂(e) ← w₂(e) + δᵢ,  excess(e) ← excess(e) + δᵢ
      for all e ∈ 𝒩ᵢ \ I⁻  // reset excess for all e ∈ 𝒩ᵢ \ I⁻
        w₁(e) ← w₁(e) − excess(e),  excess(e) ← 0
    // * end of iteration (i, j) *
    I ← I⁻ ∪ I⁺
  // * end of scale i *
  return I
```

Figure 3: Full details of the algorithm lazy-scaled-weight-splitting

References

[BCG86] C. Brezovec, G. Cornuéjols, and F. Glover. Two algorithms for weighted matroid intersection. *Math. Prog.*, 36(1):39–53, 1986.

[CKL13] Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25, October 2013.

[CKM+11] P. Christiano, J. Kelner, A. Madry, D. Spielman, and S.H. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proc. 43th Annu. ACM Symp. Theory Comput.* (STOC), pages 273–282, 2011.

[Cun86] W. H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986.

[DFZ11] Randall Dougherty, Chris Freiling, and Kenneth Zeger. Network coding and matroid theory. *Proceedings of the IEEE*, 99(3):388–405, 2011.

[DP14] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *J. Assoc. Comput. Mach.*, 61(1), January 2014.

[Edm70] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Proc. of the Calgary Int. Conf. on Combinatorial Structures and their Applications, 1969*, pages 69–92, 1970.

[EK72] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19(2):248–264, 1972.

[Fra81] A. Frank. A weighted matroid intersection algorithm. *J. Algorithms*, 2:328–336, 1981.

[Fra08] A. Frank. A quick proof of the matroid intersection weight-splitting theorem. *EGRES Quick-Proofs*, 3, 2008.

[FS89] G. Frederickson and M. Srinivas. Algorithms and data structures for an expanded family of matroid intersection problems. *SIAM J. Comput.*, 18(1):112–138, 1989.

[FZ95] S. Fujishige and X. Zhang. An efficient cost scaling algorithm for the independent assignment problem. *J. Oper. Res. Soc. Japan*, 38(1):124–136, 1995.

[Gar85] Harold N. Garbow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148 – 168, 1985.

[GT89] H. Gabow and R. E. Tarhan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[GT91] H. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. Assoc. Comput. Mach.*, 38(4):815–853, 1991.

[GX89] H. N. Gabow and Y. Xu. Efficient algorithms for independent assignment on graphic and linear matroids. In *Proc. 30th Annu. IEEE Symp. Found. Comput. Sci.* (FOCS), pages 106–111, 1989.

[GX96] Harold N Gabow and Ying Xu. Efficient theoretic and practical algorithms for linear matroid intersection problems. *Journal of Computer and System Sciences*, 53(1):129–147, 1996.

[Har07] N. Harvey. An algebraic algorithm for weighted linear matroid intersection. In *Proc. 18th ACM-SIAM Symp. Discrete Algs.* (SODA), pages 444–453, 2007.

[Har09] N. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM J. Comput.*, 39(2):679–702, 2009. Preliminary version in Proc. 47th Annu. IEEE Symp. Found. Comput. Sci. *(FOCS)*, 2009.

[HK73] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, pages 225–231, 1973.

[HKK14] C. Huang, N. Kakimura, and N. Kamiyama. Exact and approximation algorithms for weighted matroid intersection. *MI Preprint Series, Math. for Industry, Kyushu University*, November 2014. To appear in Proc. 27th ACM-SIAM Symp. Discrete Algs. *(SODA)*, 2016.

[KLOS14] J. Kelner, Y.T. Lee, L. Orecchia, and A. Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalization. In *Proc. 25th ACM-SIAM Symp. Discrete Algs.* (SODA), pages 217–226, 2014.

[Kro75] S. Krogdahl. A combinatorial proof of Lawler's matroid intersection algorithm. unpublished manuscript, 1975.

[Law75] E. Lawler. Matroid intersection algorithms. *Math. Prog.*, 9:31–56, 1975.

[Law76] E. Lawler. *Combinatorial optimization: networks and matroids*. Dover Publications, 1976.

[LS14] Y. Lee and A. Sidford. Path finding methods for linear programming. In *Proc. 55th Annu. IEEE Symp. Found. Comput. Sci.* (FOCS), pages 424–433, 2014.

[LSW15] Y.T. Lee, A. Sidford, and S.C. Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Proc. 56th Annu. IEEE Symp. Found. Comput. Sci.* (FOCS), 2015.

[Mur09] Kazuo Murota. *Matrices and matroids for systems analysis*, volume 20. Springer Science & Business Media, 2009.

[Mą13] A. Mądry. Navigating central path with electrical flows: from flows to matchings, and back. In *Proc. 54th Annu. IEEE Symp. Found. Comput. Sci.* (FOCS), pages 253–262, 2013.

[Pen16] R. Peng. Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time. To appear in Proc. 27th ACM-SIAM Symp. Discrete Algs. *(SODA)*, 2016.

[Rec13] András Recski. *Matroid theory and its applications in electric network theory and in statics*, volume 6. Springer Science & Business Media, 2013.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.