Locally Defined Independence Systems on Graphs

Yuki Amano[1,a)]

Abstract: The maximization for the independence systems defined on graphs is a generalization of combinatorial optimization problems such as the maximum b-matching, the unweighted MAX-SAT, the matchoid, and the maximum timed matching problems. In this paper, we consider the problem under the local oracle model to investigate the global approximability of the problem by using the local approximability. We first analyze two simple algorithms `FixedOrder` and `Greedy` for the maximization under the model, which shows that they have no constant approximation ratio. Here algorithms `FixedOrder` and `Greedy` apply local oracles with fixed and greedy orders of vertices, respectively. We then propose two approximation algorithms for the k-degenerate graphs, whose approximation ratios are $\alpha + 2k - 2$ and αk, where α is the approximation ratio of local oracles. The second one can be generalized to the hypergraph setting. We also propose an $(\alpha + k)$-approximation algorithm for bipartite graphs, in which the local independence systems in the one-side of vertices are k-systems with independence oracles.

1. Introduction

The maximization for independence system is one of the most fundamental combinatorial optimization problems [4], [16], [17]. An independence system is a pair $(E, \mathcal{I})$ of a finite set E and a family $\mathcal{I} \subseteq 2^E$ that satisfies

$$\mathcal{I} \text{ contains empty set, i.e., } \emptyset \in \mathcal{I}, \text{ and} \tag{1}$$

$$J \in \mathcal{I} \text{ implies } I \in \mathcal{I} \text{ for any } I \subseteq J \subseteq E. \tag{2}$$

Here a member I in $\mathcal{I}$ is called an *independent set*. Property (2) means that $\mathcal{I}$ is downward closed. The maximization problem for an independence system is to find an independent set with the maximum cardinality. This problem includes, as a special case, the maximum independent set of a graph, the maximum matching, the maximum set packing and the matroid (intersection) problems [14], [16], [17].

In this paper, we consider the following independence systems defined on graphs. Let $G = (V, E)$ be a graph with a vertex set V and an edge set E. For a vertex v in V, let E_v denote the set of edges incident to v. In our problem setting, each vertex v has a local independence system $(E_v, \mathcal{I}_v)$, i.e., $\mathcal{I}_v \subseteq 2^{E_v}$, and we consider the independence system $(E, \mathcal{I})$ defined by

$$\mathcal{I} = \{I \subseteq E \mid I \cap E_v \in \mathcal{I}_v \text{ for all } v \in V\}. \tag{3}$$

Namely, $(E, \mathcal{I})$ is obtained by concatenating local independence systems $(E_v, \mathcal{I}_v)$, and is called *an independence system defined on a graph* G. We assume without loss of generality that $\{e\} \in \mathcal{I}$ holds for any $e \in E$, since otherwise the underlying graph G can be replaced by $G' = (V, E \setminus \{e\})$. In this paper, we consider the maximization problem for it, i.e., for a given graph $G = (V, E)$ with local independence systems $(E_v, \mathcal{I}_v)$, our problem is described as

$$\begin{aligned} &\text{maximize} && |I| \\ &\text{subject to} && I \cap E_v \in \mathcal{I}_v \text{ for all } v \in V \\ & && I \subseteq E. \end{aligned} \tag{4}$$

Note that any independence system $(E, \mathcal{I})$ is viewed as an independence system defined on a star. As an example of problem (4), for $b : V \to \mathbb{Z}_+$, let $\mathcal{I}_v = \{I \subseteq E_v \mid |I| \leq b(v)\}$, i.e., $\mathcal{I}_v$ is $b(v)$-bounded for every v in V. Then (3) denotes the family of b-matchings in G. In particular, if $b \equiv 1$, it corresponds to the family of matchings in G [15]. More generally, if $(E_v, \mathcal{I}_v)$ is a matroid for every v in V, then the family (3) is a so-called *matchoid* [7].

The maximum b-matching and matchoid problems are well-studied combinatorial optimization problems [7], [15]. It is known that the maximum matchoid problem is NP-hard [11] and it is $3/2$-approximable [6], [9].

The unweighted maximum satisfiability problem (MAX-SAT) is another example of problem (4). The unweighted MAX-SAT is the problem to find a variable assignment that maximizes the number of the satisfied clauses in a given conjunctive normal form (CNF). For a variable set X, let $\varphi = \bigwedge_{c \in C} c$ be a CNF, where C denotes a set of clauses with variables in X. Define a bipartite graph $G = (V = X \cup C, E)$ and local independence systems $(E_v, \mathcal{I}_v)$ by

$$E = \{(x, c) \in X \times C \mid x \text{ is contained in } c\}$$

$$\mathcal{I}_v = \begin{cases} \{I \subseteq E_v \mid \text{either } I \subseteq C_+(v) \text{ or } I \subseteq C_-(v)\} & \text{if } v \in X \\ \{I \subseteq E_v \mid |I| \leq 1\} & \text{if } v \in C, \end{cases}$$

where, for a variable $x \in X$, $C_+(x)$ and $C_-(x)$ respectively denote the sets of clauses in C that contain literals x and $\overline{x}$. Therefore the

[1] Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502, Japan
[a)] ukiamano@kurims.kyoto-u.ac.jp

unweighted MAX-SAT is an example of problem (4). As is well-known, the unweighted MAX-SAT is NP-hard and approximable in ratio 1.255 [2].

(Edge-)temporal graphs [8] were introduced to model dynamic network topologies where the edge set vary with time. Namely, a *temporal graph* is a graph $G = (V, E)$ with given time labels $L_e \subseteq T$ for all e in E, where T is a given finite set of time labels. For a temporal graph, a subset M of E is a *timed matching* if L_e and L_f are disjoint for any adjacent pair of edges e and f in M. By defining local independence systems $(E_v, \mathcal{I}_v)$ by $\mathcal{I}_v = \{I \subseteq E_v \mid L_e \cap L_f = \emptyset \text{ for all } e, f \in I\}$, the maximum timed matching can be formulated as problem (4). It is known that the maximum timed matching problem is NP-hard even if the given graph G is bipartite, while it is solvable in polynomial time if G is a tree and every L_e is represented as an interval for a given order of time labels [12].

In this paper, we consider problem (4) by making use of local oracles $\mathcal{A}_v$ for each v in V. For an independence system $(E, \mathcal{I})$ and a subset $F \subseteq E$, $\mathcal{I}[F]$ denotes the family of independent sets of $\mathcal{I}$ restricted to F, i.e., $\mathcal{I}[F] = \{I \cap F \mid I \in \mathcal{I}\}$. For a vertex $v \in V$ and a subset $F \subseteq E_v$, $\mathcal{A}_v(F)$ is an α-approximate independent set of the maximization for $(F, \mathcal{I}_v[F])$. That is, the oracle $\mathcal{A}_v : 2^{E_v} \to 2^{E_v}$ satisfies

$$\mathcal{A}_v(F) \in \mathcal{I}_v[F] \tag{5}$$

$$\alpha\,|\mathcal{A}_v(F)| \geq \max_{J \in \mathcal{I}_v[F]} |J|. \tag{6}$$

We call $\mathcal{A}_v$ an *α-approximation local* oracle. It is also called an *exact local oracle* if $\alpha = 1$. In this paper, we assume the monotonicity of $\mathcal{A}_v$, i.e., $|\mathcal{A}_v(S)| \leq |\mathcal{A}_v(T)|$ holds for the subsets $S \subseteq T \subseteq E_v$, which is a natural assumption on the oracle since it deals with independence system. We study this oracle model to investigate the global approximability of problem (4) by using the local approximability. The oracle model was used in [3] for the maximum coverage problem with group constraints, where the oracle model is regarded as a generalization of greedy algorithms. It is shown that the maximum coverage problem is $(\alpha + 1)$-approximable in this oracle model. We remark here that a greedy algorithm for the maximum cardinality matching problem can be viewed as a 2-approximation algorithm under the exact local oracle model.

For the above reductions of the maximum matchoid problem and the unweighted MAX-SAT, their local maximization problems for $(E_v, \mathcal{I}_v)$ can be solved exactly, while their global maximization problems are NP-hard. This means that problem (4) seems to be intractable, even if exact local oracles are given.

In this paper, we first propose two natural algorithms for problem (4), where the first one applies local oracles $\mathcal{A}_v$ in the order of the vertices v that is fixed in advance, while the second one applies local oracles in the greedy order of vertices $v_1, \ldots, v_n$, where $n = |V|$ and

$$v_i \in \operatorname*{arg\,max}_{v \in V \setminus \{v_1, \ldots, v_{i-1}\}} |\mathcal{A}_v(E_v \cap F^{(i)})| \qquad \text{for } i = 1, \ldots, n.$$

Here the subset $F^{(i)} \subseteq E$ is a set of available edges during the i-th iteration.

We show that the first algorithm guarantees an approximation ratio $(\alpha + n - 2)$, and the second algorithm guarantees an approximation ratio $\rho(\alpha, n)$, where ρ is the function of α and n defined as

$$\rho(\alpha, n) = \begin{cases} \alpha + \frac{2\alpha-1}{2\alpha}(n-1) - \frac{1}{2} & \text{if } (\alpha-1)(n-1) \geq \alpha(\alpha+1) \\ \alpha + \frac{\alpha}{\alpha+1}(n-1) & \text{if } \alpha \leq (\alpha-1)(n-1) < \alpha(\alpha+1) \\ \frac{n}{2} & \text{if } (\alpha-1)(n-1) < \alpha. \end{cases}$$

We also show that both of approximation ratios are *almost tight* for these algorithms.

We then consider two subclasses of problem (4). We provide two approximation algorithms for the k-degenerate graphs, whose approximation ratios are $\alpha + 2k - 2$ and αk. Here, a graph is *k-degenerate* if any subgraph has a vertex of degree at most k. This implies for example that the algorithms find an α-approximate independent set for the problem if a given graph is a tree. This is best possible, because the local maximization is not approximable with c $(< \alpha)$. We also show that the second algorithm can be generalized to the hypergraph setting.

We next provide an $(\alpha + k)$-approximation algorithm for the problem when a given graph is bipartite and local independence systems for one side are all k-systems with independence oracles. Here an independence system $(E, \mathcal{I})$ is called a *k-system* if for any subset $F \subseteq E$, any two maximal independent sets I and J in $\mathcal{I}[F]$ satisfy $k|I| \geq |J|$, and its independence oracle is to decide if a given subset $J \subseteq E$ belongs to $\mathcal{I}$ or not.

The rest of the paper is organized as follows. In Section 2, we describe two natural algorithms for problem (4) and analyze their approximation ratios. Section 3 provides approximation algorithms for the problem in which a given graph G has bounded degeneracy. Section 4 also provides an approximation algorithm for the problem in which a given graph G is bipartite, and all the local independence systems of the one side of vertices are k-systems. Section 5 defines independence systems defined on hypergraphs and generalizes algorithms to the hypergraph case.

Due to space constraints, most of the proofs are omitted in this paper, which can be found in [1].

2. Local subpartitions and greedy algorithms

In this section, we first define local subpartitions of edges and analyze two natural algorithms for problem (4). Local subpartitions of edges are used for constructing and analyzing approximation algorithms for the problem.

2.1 Local subpartitions

Definition 1. *For a graph $G = (V, E)$ and subsets $P_v \subseteq E_v$ for all $v \in V$, the collection $\mathcal{P} = \{P_v \mid v \in V\}$ is called a* local subpartition *if $P_u \cap P_v = \emptyset$ for all distinct pairs u and v of vertices. The set $R = E \setminus (\bigcup_{P \in \mathcal{P}} P)$ is called the* residual *edge set of $\mathcal{P}$.*

By definition, for a local subpartition $\mathcal{P} = \{P_v \mid v \in V\}$, the set P_v may be empty for some v in V.

Let $\mathcal{P} = \{P_v \mid v \in V\}$ be a local subpartition of E, and for every $v \in V$, let $I_v \in \mathcal{I}_v[P_v]$ be a subset of P_v which is independent in $\mathcal{I}_v$. Then the union $I = \bigcup_{v \in V} I_v$ may *not* be an independent set of $\mathcal{I}$. For example, let $G = (V, E)$ be a graph with

$V = \{1,2,3\}$ and $E = \{e_1 = (1,2), e_2 = (2,3)\}$, and for each $i = 1,2,3$, let $\mathcal{I}_i = \{I \subseteq E \mid |I| \leq 1\}$, then for a local partition $\mathcal{P} = \{P_1 = \{e_1\}, P_2 = \emptyset, P_3 = \{e_2\}\}$ of E, $I_1 = \{e_1\}$, $I_2 = \emptyset$, and $I_3 = \{e_2\}$ satisfy $I_i \in \mathcal{I}_i[P_i]$ for all $i = 1,2,3$, while the union $I_1 \cup I_2 \cup I_3 \notin \mathcal{I}$. If the union is independent, we have the following lemma, where we recall that $\mathcal{A}_v$ ($v \in V$) denotes an α-approximate local oracle.

Lemma 2. *For a local subpartition $\mathcal{P} = \{P_v \mid v \in V\}$, let R be the residual edge set of $\mathcal{P}$. If $I = \bigcup_{v\in V} \mathcal{A}_v(P_v)$ is an independent set in $\mathcal{I}$, then it guarantees an approximate ratio of $\alpha + \max_{J\in\mathcal{I}[R]} |J|/|I|$ for problem* (4).

Proof. For an independent set $K \in \mathcal{I}$, the set $I = \bigcup_{v\in V} \mathcal{A}_v(P_v)$ satisfies the following inequalities:

$$\alpha\,|I| = \alpha \sum_{v\in V} |\mathcal{A}_v(P_v)| \geq \sum_{v\in V} |K \cap P_v| = |K| - |K \cap R|.$$

This implies

$$|K| \leq (\alpha + |K \cap R|/|I|)\,|I| \leq (\alpha + \max_{J\in\mathcal{I}[R]} |J|/|I|)\,|I|,$$

which completes the proof. □

All the algorithms proposed in this paper construct local subpartitions of edges, and Lemma 2 is used to analyze their approximation ratio.

Let us then see the following two simple algorithms which cannot guarantee any constant approximation ratio, even if exact local oracles are available.

2.2 Algorithm `FixedOrder`

Our first algorithm called `FixedOrder` makes use of local oracles $\mathcal{A}_v$ in the order of the vertices $v \in V$ that is fixed in advance. Algorithm `FixedOrder` constructs a local subpartition

Algorithm `FixedOrder`

/* $(v_1, \ldots, v_n)$ is a given vertex order. */
$F := E$.
for $v = v_1, \ldots, v_n$ **do**
  $P_v := E_v \cap F$.
  $R_v := \left(\left(\bigcup_{\substack{u\in V:\\ (u,v)\in\mathcal{A}_v(P_v)}} E_u\right) \setminus P_v\right) \cap F$.
  $F := F \setminus (P_v \cup R_v)$.
end for
$I := \bigcup_{v\in V} \mathcal{A}_v(P_v)$.
$R := \bigcup_{v\in V} R_v$.
Output I and halt.

$\mathcal{P} = \{P_v \mid v \in V\}$ and the residual edge set R of $\mathcal{P}$, which will be proven in the next theorem. In order to ensure that $I = \bigcup_{v\in V} \mathcal{A}_v(P_v)$ is independent in Lemma 2, the algorithm maintains F as a candidate edge set during the iteration. The following theorem provides the approximation ratio of the algorithm, which is *almost tight* for the algorithm. In fact, it is tight if α is an integer.

Theorem 3. *Algorithm* `FixedOrder` *computes an $(\alpha + n - 2)$-approximate solution for the maximization for problem* (4) *under the approximate local oracle model, and the approximation ratio of the algorithm is at least $\lfloor\alpha\rfloor + n - 2$.*

2.3 Algorithm `Greedy`

Our second algorithm called `Greedy` makes use of local oracles $\mathcal{A}_v$ in a greedy order of vertices $v_1, \ldots, v_n$, where

$$v_i \in \operatorname*{arg\,max}_{v\in V\setminus\{v_1,\ldots,v_{i-1}\}} |\mathcal{A}_v(E_v \cap F^{(i)})| \qquad \text{for } i = 1, \ldots, n.$$

Here the subset $F^{(i)} \subseteq E$ is the candidate edge set in the i-th round of `Greedy`. For the analysis of `Greedy`, we use the following

Algorithm `Greedy`

$F := E$.
$W := V$.
while $W \neq \emptyset$ **do**
  $v \in \arg\max_{w\in W} |\mathcal{A}_w(E_w \cap F)|$.
  $W := W \setminus \{v\}$.
  $P_v := E_v \cap F$.
  $R_v := \left(\left(\bigcup_{\substack{u\in V:\\ (u,v)\in\mathcal{A}_v(P_v)}} E_u\right) \setminus P_v\right) \cap F$.
  $F := F \setminus (P_v \cup R_v)$.
end while
$I := \bigcup_{v\in V} \mathcal{A}_v(P_v)$
$R := \bigcup_{v\in V} R_v$.
Output I and halt.

lemma which is slightly different from Lemma 2.

Lemma 4. *Let $\mathcal{P} = \{P_v \mid v \in V\}$ be a local subpartition of E, and for the residual R of $\mathcal{P}$, let $\{R_v \mid v \in V\}$ be a partition of R that satisfies $R_v = \emptyset$ for all vertices v with $P_v = \emptyset$. If $I = \bigcup_{v\in V} \mathcal{A}_v(P_v)$ is an independent set in $\mathcal{I}$, then it guarantees approximation ratio β for problem* (4)*, where*

$$\beta = \max_{v\in V:P_v\neq\emptyset} \max_{J\in\mathcal{I}[P_v\cup R_v]} \frac{|J|}{|\mathcal{A}_v(P_v)|}.$$

The following theorem provides the approximation ratio of `Greedy`, which is *almost tight* for the algorithm. In fact, it is tight if $(\alpha-1)(n-1) < \alpha$.

Theorem 5. *Algorithm* `Greedy` *computes a $\rho(\alpha, n)$-approximate solution for problem* (4)*, where ρ is the function of α and n defined as*

$$\rho(\alpha,n) = \begin{cases} \alpha + \frac{2\alpha-1}{2\alpha}(n-1) - \frac{1}{2} & \text{if } (\alpha-1)(n-1) \geq \alpha(\alpha+1) \\ \alpha + \frac{\alpha}{\alpha+1}(n-1) & \text{if } \alpha \leq (\alpha-1)(n-1) < \alpha(\alpha+1) \\ \frac{n}{2} & \text{if } (\alpha-1)(n-1) < \alpha. \end{cases}$$

Moreover, the approximation ratio is at least

$$\varphi(\alpha,n) = \begin{cases} \rho(\alpha,n) - \frac{\alpha}{2} & \text{if } (\alpha-1)(n-1) \geq \alpha(\alpha+1) \\ \rho(\alpha,n) - \left(\frac{3}{2}\alpha - \frac{1}{2}\right) & \text{if } \alpha \leq (\alpha-1)(n-1) < \alpha(\alpha+1) \\ \rho(\alpha,n) & \text{if } (\alpha-1)(n-1) < \alpha. \end{cases}$$

3. Approximation algorithms based on local oracles

In the previous section, we analyzed two simple algorithms `FixedOrder` and `Greedy` for problem (4). In this section, we focus on the degeneracy of the given graph, and develop approximation algorithms for problem (4). Our first algorithm called

`OrderedApprox` makes use of a linear order $\prec$ of vertices V. Different from `FixedOrder` and `Greedy` in Section 2, the algorithm tries to minimize the size of R. In fact, we have $R = \emptyset$ if G is a tree. More precisely, for each $v \in V$, we initialize $P_v = D_v$, where D_v is the set of downward edges of v, i.e., $D_v = \{(u, v) \in E_v \mid u \prec v\}$, and compute $\mathcal{A}_v(P_v)$ and $\mathcal{A}_v(P_v \cup \{e\})$ for each $e \in U_v$, where U_v is the set of upward edges of v, i.e., $U_v = \{(u, v) \in E_v \mid v \prec u\}$. Based on these outputs of local oracle and their values, we consider four cases, each of which we update P_v and construct $I_v = \mathcal{A}_v(P_v)$ and R_v accordingly. Here P_v and $R = \bigcup_{v \in V} R_v$ correspond to those in Lemma 2. We then modify P_w for vertices w with $v \prec w$, in such a way that the set $I = \mathcal{A}_w(P_w) \cup \bigcup_{u \in V : u \preceq v} I_v$ is an independent set in $\mathcal{I}$. The second algorithm first decomposes edge set E into forests $E_1, \ldots, E_\gamma$, for each forest E_i, applies `OrderedApprox` to compute an independent set I_i, and chooses a maximum independent set among them.

We first define the upward and downward edge sets with respect to a linear order.

Definition 6. *For a graph $G = (V, E)$, let $\prec$ be a linear order of vertices V. For a vertex v, let $U_v = \{(v, w) \in E_v \mid v \prec w\}$ and $D_v = \{(v, w) \in E_v \mid w \prec v\}$ be the sets of* upward *and* downward *edges incident to v, respectively. We define the* width *of G (with respect to a linear order $\prec$) as* $\max_{v \in V} |U_v|$.

The minimum width of the graph $G = (V, E)$ among all linear order of vertices V is called the *degeneracy* of G, and G is called *k-degenerate* if k is at least the minimum width of G [5], [10], [13]. Note that a linear order of vertices V certifying that G is k-degenerate can be obtained by repeatedly choosing vertices with minimum degree in the remaining graph, i.e.,

$$v_i \in \operatorname*{arg\,min}_{v \in V \setminus \{v_1, \ldots, v_{i-1}\}} \deg_{G[V \setminus \{v_1, \ldots, v_{i-1}\}]}(v) \qquad \text{for } i = 1, \ldots, n,$$

where $\deg_H(v)$ denotes the degree of vertex v in the graph H and $G[W]$ denotes the subgraph of G induced by a vertex subset W. Therefore, such a liner order can be computed in linear time. It is also known that the degeneracy of a graph is at most its treewidth.

Algorithm `OrderedApprox` first initializes $P_v = D_v$ for all $v \in V$ and for each i-th iteration of the for-loop, computes an edge set $B_{v_i} \subseteq U_{v_i}$ by

$$B_{v_i} := \{e \in U_{v_i} \mid e \in \mathcal{A}_{v_i}(P_{v_i} \cup \{e\}),\ |\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})| > |\mathcal{A}_{v_i}(P_{v_i})|\}.$$

It separately treats the following four cases as in the description of the algorithm.

Case 1: $U_{v_i} = \emptyset$

Case 2: $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} = \emptyset$

Case 3: $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} \neq \emptyset$

Case 4: $U_{v_i} \neq \emptyset$ and $P_{v_i} = \emptyset$

For all the cases, we show the following two lemmas.

Lemma 7. *Algorithm* `OrderedApprox` *satisfies the following three conditions*

Algorithm `OrderedApprox`$(G, \mathcal{I}, \prec)$

Input: An independence system $(E, \mathcal{I})$ defined on a graph $G = (V, E)$ and a linear order $v_1 \prec v_2 \prec \ldots \prec v_n$ of vertices V.

Output: An independent set in $\mathcal{I}$.

1: $X := \emptyset$.
2: $P_v := D_v$ for $v \in V$.
3: **for** $i = 1, \ldots, n$ **do**
4: $\quad B_{v_i} := \{e \in U_{v_i} \mid e \in \mathcal{A}_{v_i}(P_{v_i} \cup \{e\}),\ |\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})| > |\mathcal{A}_{v_i}(P_{v_i})|\}$.
5: $\quad$ **if** $U_{v_i} = \emptyset$ **then**
6: $\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i})$.
7: $\quad\quad R_{v_i} := \emptyset$.
8: $\quad$ **else if** $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} = \emptyset$ **then**
9: $\quad\quad$ Choose an edge $e = (v_i, v_j) \in U_{v_i}$ arbitrarily.
10: $\quad\quad$ **if** $e \notin \mathcal{A}_{v_i}(P_{v_i} \cup \{e\})$ **then**
11: $\quad\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i} \cup \{e\})$.
12: $\quad\quad$ **else** (i.e., $|\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})| = |\mathcal{A}_{v_i}(P_{v_i})|$)
13: $\quad\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i})$. /* $|I_{v_i}| = |\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})|$ */
14: $\quad\quad$ **end if**
15: $\quad\quad P_{v_i} := P_{v_i} \cup \{e\}$.
16: $\quad\quad R_{v_i} := U_{v_i} \setminus \{e\}$.
17: $\quad\quad P_{v_j} := P_{v_j} \setminus \{e\}$.
18: $\quad$ **else if** $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} \neq \emptyset$ **then**
19: $\quad\quad$ Choose an edge $b \in B_{v_i}$ arbitrarily.
20: $\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i} \cup \{b\}) \setminus \{b\}$. /* $I_{v_i} \subseteq P_{v_i}$ and $|I_{v_i}| = |\mathcal{A}_{v_i}(P_{v_i})|$ */
21: $\quad\quad R_{v_i} := U_{v_i} \setminus \{b\}$.
22: $\quad$ **else** (i.e., $U_{v_i} \neq \emptyset$ and $P_{v_i} = \emptyset$)
23: $\quad\quad I_{v_i} := \emptyset$.
24: $\quad\quad R_{v_i} := \emptyset$. /* R_{v_i} might be updated in line 29 */
25: $\quad\quad X := X \cup \{v_i\}$.
26: $\quad$ **end if**
27: $\quad$ **for** $(v_l, v_i) \in I_{v_i}$ **do**
28: $\quad\quad$ **if** $v_l \in X$ **then**
29: $\quad\quad\quad R_{v_l} := \{(v_l, v_j) \in U_{v_l} \mid j > i\}$.
30: $\quad\quad\quad X := X \setminus \{v_l\}$.
31: $\quad\quad$ **end if**
32: $\quad$ **end for**
33: $\quad$ **for** $j = i + 1, \ldots, n$ **do**
34: $\quad\quad P_{v_j} := P_{v_j} \setminus (\bigcup_{l \leq i} R_{v_l})$.
35: $\quad$ **end for**
36: **end for**
37: Output $I = \bigcup_{v \in V} I_v$ and halt.

(i) $I_{v_j} \subseteq P_{v_j} \subseteq E_{v_j}$ *with* $|I_{v_j}| = |\mathcal{A}_{v_j}(P_{v_j})|$ *for all* $j \leq i$,

(ii) $R_{v_j} \subseteq U_{v_j}$ *for all* $j \leq i$, *and*

(iii) $\mathcal{P} = \{P_v \mid v \in V\}$ *is a local subpartition of* E *with residual* $R = \bigcup_{j \leq i} R_{v_j}$

at the end of the i-th iteration of the for-loop.

Proof. Since I_{v_j} and P_{v_j} are never modified after the j-th iteration of the for-loop in Line 3, it is enough to show that $I_{v_i} \subseteq P_{v_i} \subseteq E_{v_i}$ and $|I_{v_i}| = |\mathcal{A}_{v_i}(P_{v_i})|$ at the end of the i-th iteration of the for-loop to prove Condition (i). We can see that P_{v_i} is initialized to D_{v_i}, and P_{v_i} is modified in Lines 15, 17, and 34. In Lines 17 and 34, no edge is added to P_{v_i}, and in Line 15, some edge $e \in U_{v_i}$ is added to P_{v_i}. These show that $P_{v_i} \subseteq E_{v_i}$. Moreover, since I_{v_i} is constructed in Lines 6, 11, 13, 20, and 23, we have $I_{v_i} \subseteq P_{v_i}$ and $|I_{v_i}| = |\mathcal{A}_{v_i}(P_{v_i})|$, which implies Condition (i).

Since R_{v_j} is never modified after the j-th iteration for all the

cases except for Case 4, Condition (ii) is satisfied if Case 4 is not satisfied in the j-th iteration of the for-loop. On the other hand, if Case 4 is satisfied in the j-th iteration of the for-loop, then R_{v_j} might be updated in Line 29, which again satisfies $R_{v_j} \subseteq U_{v_j}$. This implies Condition (ii).

Let us finally show Condition (iii). Before the first iteration of the for-loop, $\mathcal{P} = \{P_v = D_v \mid v \in V\}$ is a local subpartition of E with the residual $R = \emptyset$. Assuming that Condition (iii) is satisfied in the beginning of the i-th iteration, we show that Condition (iii) is satisfied at the end of the i-th iteration. Note that P_{v_i} is modified in Lines 15, 17, and 34. In Lines 17 and 34, no edge is added to P_{v_j} for any j. If some edge $e = (v_i, v_j) \in U_{v_i}$ is added to P_{v_i} in Line 15, e is removed from P_{v_j}. These imply that $\mathcal{P} = \{P_v \mid v \in V\}$ is a subpartition of E. Moreover, in any iteration, no edge is added to $\bigcup_{P_v \in \mathcal{P}} P_v$, and when R_v is constructed in Lines 7, 16, 21, 24, and 29, all the edges in such an R_v are deleted from the corresponding P_{v_j} in Line 34. Thus Condition (iii) is satisfied at the end of the i-th iteration, which completes the proof of the lemma. □

Lemma 8. *Algorithm* `OrderedApprox` *satisfies that*

$$I \cup \bigcup_{j<i} I_{v_j} \in \mathcal{I} \text{ for any independent set } I \in \mathcal{I}[P_{v_i}] \qquad (7)$$

in the beginning of the i-th iteration of the for-loop.

Proof. For an index j, consider the end of the j-th iteration of the for-loop in Line 3. We have $I_{v_j} \cup \{e\} \in \mathcal{I}_{v_j}$ for any $e \in U_{v_j} \setminus (P_{v_j} \cup R_{v_j})$. If the j-th iteration of the for-loop falls into Cases 1, 2, or 3, then $U_{v_j} \cap (\bigcup_{k>j} P_{v_k}) = U_{v_j} \setminus (P_{v_j} \cup R_{v_j})$ contains at most one edge. Otherwise (i.e., Case 4), we have $U_{v_j} \subseteq \bigcup_{k>j} P_{v_k}$, and R_{v_j} will be updated to $\{(v_j, v_l) \in U_{v_j} \mid l > k\}$ once $(v_j, v_k) \in U_{v_j}$ is chosen by I_{v_k} in the k-th iteration for some $k > j$. Therefore, (7) is satisfied in the beginning of the i-th iteration of the for-loop. □

Theorem 9. *Algorithm* `OrderedApprox` *computes an $(\alpha + 2\gamma - 2)$-approximate independent set I in $\mathcal{I}$ in polynomial time, where γ is the width of a given graph G with respect to a given linear order of vertices V.*

Proof. For any v in V, let I_v, P_v, and R_v be the sets obtained by Algorithm `OrderedApprox`. By Lemma 8, $I = \bigcup_{v \in V} I_v$ is an independent set in $\mathcal{I}$. Let us first consider the size of R_{v_i}. If the i-th iteration of the for-loop falls into Case 1, then we have $R_{v_i} = \emptyset$. If it falls into Cases 2 or 3, then we have $|R_{v_i}| = |U_{v_i}| - 1 \leq \gamma - 1$ and $I_{v_i} \neq \emptyset$, which implies $|R_{v_i}| \leq (\gamma - 1)|I_{v_i}|$. In Case 4, v_i is added to X and R_{v_i} is set to $R_{v_i} = \emptyset$ at the end of the i-th iteration. Note that R_{v_i} might be updated to $\{(v_i, v_k) \in U_{v_k} \mid k > j\}$ if $(v_i, v_j) \in U_{v_i}$ is chosen by I_{v_j} in the j-th iteration for some $j > i$. In either case, we have $|R_{v_i}| \leq \gamma - 1$, and there exists an edge (v_i, v_j) in $U_{v_i} \cap I_{v_j}$ if $R_{v_i} \neq \emptyset$. For $p = 1, 2, 3$, and 4, let V_p denote the set of vertices v_i such that the i-th iteration of the for-loop falls into Case p. Then we have

$$\sum_{v \in V_1} |R_v| = 0,$$
$$\sum_{v \in V_2 \cup V_3} |R_v| \leq (\gamma - 1) \sum_{v \in V_2 \cup V_3} |I_v| \leq (\gamma - 1)|I|, \text{ and}$$
$$\sum_{v \in V_4} |R_v| \leq (\gamma - 1)|I|.$$

Therefore, we obtain the following inequality

$$|R| = \sum_{v \in V} |R_v| \leq 2(\gamma - 1)|I|.$$

By Lemma 7, $\mathcal{P} = \{P_v \mid v \in V\}$ is a local subpartition of E with the residual $R = \bigcup_{v \in V} R_v$. Thus, by applying Lemma 2 to this I, we can see that I is an $(\alpha + 2\gamma - 2)$-approximate independent set in $\mathcal{I}$. □

Since a linear order $\prec$ of vertices V representing the degeneracy of $G = (V, E)$ can be computed in linear time, we have the following corollary.

Corollary 10. *Algorithm* `OrderedApprox` *computes an $(\alpha + 2k - 2)$-approximate independent set I in $\mathcal{I}$ in polynomial time, if a given graph G is k-degenerate.*

For a graph $G = (V, E)$ with the width γ, the second algorithm, called `DecomApprox`, first decomposes edge set E into forests $E_1, \ldots, E_\gamma$, for each E_i, applies `OrderedApprox` to compute an independent set I_i, and chooses a maximum independent set among them.

Algorithm `DecomApprox`$(G, \mathcal{I}, \prec)$

Input: An independence system $(E, \mathcal{I})$ defined on a graph $G = (V, E)$ and a linear order $v_1 \prec v_2 \prec \cdots \prec v_n$ of vertices V, where γ is the width of graph G with respect to $\prec$.

Output: An independent set in $\mathcal{I}$.

for $i = 1, \ldots, \gamma$ **do**
  $E_i := \emptyset$.
end for
for each v in V **do**
  for each e_i in $U_v = \{e_1, e_2, \ldots, e_{|U_v|}\}$ **do**
    $E_i := E_i \cup \{e_i\}$.
  end for
end for
for $i = 1, \ldots, \gamma$ **do**
  $I_i =$ `OrderedApprox`$(G[E_i], \mathcal{I}[E_i], \prec)$.
end for
$I \in \arg\max\{|I_i| \mid i = 1, \ldots, \gamma\}$.
Output I and halt.

Note that $G_i = (V, E_i)$ is 1-degenerate for any $i = 1, \ldots, \gamma$. Thus we have the following theorem.

Theorem 11. *Algorithm* `DecomApprox` *computes an $\alpha\gamma$-approximate independent set I in $\mathcal{I}$ in polynomial time, where γ is the width of a given graph G with respect to a given linear order of vertices V.*

Note that $\alpha < 2$ if and only if $\alpha\gamma < \alpha + 2\gamma - 2$. Therefore, an independent set I provided by Algorithm `DecomApprox` has approximation guarantee better than the one provided by Algorithm `OrderedApprox` when $\alpha < 2$. By applying Theorem 11 to graphs with degeneracy k, we have the following corollary.

Corollary 12. *Algorithm* `DecomApprox` *computes an αk-approximate independent set I in $\mathcal{I}$ in polynomial time if a given graph G is k-degenerate.*

4. Approximation for bipartite graph

We note that Algorithms `OrderedApprox` and `DecomApprox` do not provide an independent set with *small* approximation ratio if a given graph has no *small* degeneracy. Such examples include complete graphs and complete bipartite graphs.

In this section, we consider an approximation algorithm for problem (4) where the input graph is bipartite and its degeneracy might not be bounded, and analyze the approximation ratio of the algorithm if all the local independence systems in the one-side of vertices are k-systems with independence oracles. Here an independence oracle of an independence system $(E, \mathcal{I})$ answers either "$I \in \mathcal{I}$" or "$I \notin \mathcal{I}$" for a given $I \subseteq E$. Namely, let $(E, \mathcal{I})$ be an independence system defined on a bipartite graph $G = (V_1 \cup V_2, E)$. We consider the case, where every $v \in V_2$ satisfies that $(E_v, \mathcal{I}_v)$ is a k-system.

Definition 13. *For a positive $k \in \mathbb{R}$, an independence system $(E, \mathcal{I})$ is called a* k-system *if any subset $F \subseteq E$ satisfies*

$$k|I| \geq |J| \text{ for any two maximal independent sets } I \text{ and } J \text{ in } \mathcal{I}[F]. \tag{8}$$

Note that any independence system $(E, \mathcal{I})$ is a $|E|$-system and that an independence system is a 1-system if and only if it is a matroid. By definition, matchoids are independence systems such that local independence systems are all 1-systems, and hence the families of b-matchings are also the ones satisfying that local independence systems are all 1-systems. Moreover, the families of timed matchings are independence systems such that local independence systems $(E_v, \mathcal{I}_v)$ are all k-systems if any time label L_e with $e \in E_v$ is disjoint from L_f with $f \in E_v$ except for at most k edges in E_v. Our algorithm called `BipartiteApprox` can be regarded as variant of Algorithm `OrderedApprox` with a linear order $\prec$ such that $w \prec v$ hold for any $v \in V_1$ and $w \in V_2$. Note that in this order all the vertices $w \in V_2$ fall into Case 4 at the for-loop in Line 3 in `OrderedApprox`, and hence $X = V_2$ holds after the iteration of the last vertex in V_2. Different from `OrderedApprox`, Algorithm `BipartiteApprox` updates R_w for $w \in V_2$ $(= X)$ more carefully.

Algorithm `BipartiteApprox`$(G, \mathcal{I})$

Input: An independence system $(E, \mathcal{I})$ defined on a bipartite graph $G = (V_1 \cup V_2, E)$, where $V_1 = \{v_1, \ldots, v_{n_1}\}$ and $(E_v, \mathcal{I}_v)$ is a k-system with an independence oracle for every $v \in V_2$.

Output: An independent set in $\mathcal{I}$.

1: $P_v := E_v$ for $v \in V_1$.
2: $R_v := \emptyset$ and $J_v := \emptyset$ for $v \in V_2$.
3: **for** $i = 1, \ldots, n_1 (= |V_1|)$ **do**
4: $\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i})$.
5: $\quad$ **for** $(w, v_i) \in I_{v_i}$ **do**
6: $\quad\quad J_w := J_w \cup \{(w, v_i)\}$
7: $\quad\quad R_w := R_w \cup \{(w, v_j) \in E_w \mid j > i \text{ and } J_w \cup \{(w, v_j)\} \notin \mathcal{I}_w\}$.
8: $\quad$ **end for**
9: $\quad$ **for** $j = i + 1, \ldots, n_1$ **do**
10: $\quad\quad P_{v_j} = P_{v_j} \setminus (\bigcup_{v \in V_2} R_v)$.
11: $\quad$ **end for**
12: **end for**
13: $R = \bigcup_{v \in V_2} R_v$.
14: Output $I = \bigcup_{v \in V_1} I_v$ and halt.

Algorithm `BipartiteApprox` calls local oracles $\mathcal{A}_v$ in an arbitrary order of $v \in V_1$. More precisely, for each $v \in V_1$ and $w \in V_2$, we initialize $P_v = E_v$ and $R_w = \emptyset$, update P_v and R_w accordingly, and compute $I_v = \mathcal{A}_v(P_v)$. Here P_v and $R = \bigcup_{w \in V_2} R_v$ correspond to those in Lemma 2.

Lemma 14. *Algorithm* `BipartiteApprox` *satisfies the following five conditions at the end of the i-th iteration of the for-loop in Line 3:*

(i) $P_{v_j} \subseteq E_{v_j}$ *with* $I_{v_j} = \mathcal{A}_{v_j}(P_{v_j})$ *for all* $v_j \in V_1$ *with* $j \leq i$,
(ii) $R_v \subseteq E_v$ *for all* $v \in V_2$,
(iii) $\{J_v \subseteq E_v \mid v \in V_2\}$ *is a partition of* $\bigcup_{j \leq i} I_{v_j}$,
(iv) J_v *is a maximal independent set in* $\mathcal{I}[J_v \cup R_v]$ *for all* $v \in V_2$, *and*
(v) $\mathcal{P} = \{P_v \mid v \in V_1\} \cup \{P_w = \emptyset \mid w \in V_2\}$ *is a local subpartition of E with residual* $R = \bigcup_{v \in V_2} R_v$.

Lemma 15. *Algorithm* `BipartiteApprox` *satisfies that $\bigcup_{j<i} I_{v_j} \in \mathcal{I}$ in the beginning of the i-th iteration of the for-loop in Line 3.*

Theorem 16. *Let $(E, \mathcal{I})$ be an independence system defined on a bipartite graph $G = (V_1 \cup V_2, E)$ such that $(E_v, \mathcal{I}_v)$ is a k-system with an independence oracle for every $v \in V_2$. Then Algorithm* `BipartiteApprox` *computes an $(\alpha + k)$-approximate independent set I in $\mathcal{I}$ in polynomial time.*

Before concluding this section, we remark that independence systems on graphs are $2k$-systems if all local independence systems are k-systems, which is shown in the following lemma. Since the maximization for k-systems $(E, \mathcal{I})$ are approximable with ratio k by computing a maximal independent set in $\mathcal{I}$, Algorithm `BipartiteApprox` improves the ratio $2k$ to $(\alpha + k)$ for independence systems on graph bipartite graphs $G = (V_1 \cup V_2, E)$ if the all local independence systems are k-systems and α-approximation local oracles in V_1 are given for α with $\alpha < k$.

Lemma 17. *An independence system $(E, \mathcal{I})$ on graph $G = (V, E)$ is a $2k$-system if all local independence systems $(E_v, \mathcal{I}_v)$ are k-systems.*

5. Hypergraph generalization of the probslem

In this section we consider independence systems defined on hypergraphs and present two algorithms. The first algorithm corresponds to Algorithm `OrderedApprox` for problem (4) whose input graph is a forest, and the second algorithm corresponds to Algorithm `DecomApprox` for problem (4).

Definition 18. *Let $G = (V, E)$ be a hypergraph with a vertex set V and a hyperedge set $E \subseteq 2^V$. For each vertex v in V, let $(E_v, \mathcal{I}_v)$ be an independence system on the set E_v of hyperedges incident to v, i.e., $E_v = \{e \in E \mid v \in e\}$, and let $\mathcal{I} = \{I \subseteq E \mid I \cap E_v \in \mathcal{I}_v \text{ for all } v \in V\}$. We say that $(E, \mathcal{I})$*

is an independence system defined on a hypergraph G.

The generalization contains the maximum matching problem for hypergraphs. Similarly to the graph case, we make use of local oracles $\mathcal{A}_v$ for each v in V, which satisfy (5) and (6). In order to generalize the idea of Algorithm `OrderedApprox` to the hypergraph case, we define the upward and downward edge sets with respect to a linear order of vertices V.

Definition 19. *For a hypergraph $G = (V, E)$, let $\prec$ be a linear order of vertices V. For a vertex v, let $U_v = \{e \in E_v \mid v \prec w \text{ for some } w \in e\}$ and $D_v = \{e \in E_v \mid w \prec v \text{ for all } w \in e\}$ be the sets of* upward *and* downward *edges incident to v, respectively. We define the* width *of G (with respect to a linear order $\prec$) as* $\max_{v \in V} |U_v|$.

For a hypergraph $G = (V, E)$ and a vertex set $W \subseteq V$, let $G[W] = (W, E[W])$, where $E[W] = \{e \cap W \mid e \in E,\ |e \cap W| \geq 2\}$. Note that for $W \subseteq V$, $G[W]$ is the subgraph of G induced by W if G is a graph. Similarly to the graph case, a hypergraph G of width k satisfies that $G[W]$ has a vertex of degree at most k for all $W \subseteq V$. Therefore, a linear order certifying that a hypergraph G has width k can be computed in linear time.

Algorithm `OrderedApprox`* for hypergraphs works similarly to Algorithm `OrderedApprox`. Different from Algorithm `OrderedApprox`, the algorithm updates P_v and R_v based on the fact that G is a hypergraph. Note that such differences appear in Lines 4, 14, and 28 in the description of the algorithm. For each $v_i \in V$, define a set $B_{v_i} \subseteq U_{v_i}$ as follows

$$B_{v_i} = \left\{ e \in U_{v_i} \setminus \left(\bigcup_{l<i} R_{v_l} \right) \;\middle|\; \begin{array}{l} e \in \mathcal{A}_{v_i}(P_{v_i} \cup \{e\}) \text{ and} \\ |\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})| > |\mathcal{A}_{v_i}(P_{v_i})| \end{array} \right\}.$$

It separately treats the following four cases as in the description of the algorithm.

Case 1: $U_{v_i} = \emptyset$

Case 2: $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} = \emptyset$

Case 3: $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} \neq \emptyset$

Case 4: $U_{v_i} \neq \emptyset$ and $P_{v_i} = \emptyset$

We show the following two lemmas, which respectively correspond to Lemmas 7 and 8 for the graph case.

Lemma 20. *Algorithm* `OrderedApprox`* *satisfies the following three conditions*

(i) $I_{v_j} \subseteq P_{v_j} \subseteq E_{v_j}$ *with* $|I_{v_j}| = |\mathcal{A}_{v_j}(P_{v_j})|$ *for all* $j \leq i$,

(ii) $R_{v_j} \subseteq U_{v_j}$ *for all* $j \leq i$, *and*

(iii) $\mathcal{P} = \{P_v \mid v \in V\}$ *is a local subpartition of E with residual* $R = \bigcup_{j \leq i} R_{v_j}$

at the end of the i-th iteration of the for-loop.

Different from the graph case, the following lemma requires that $v \in V$ $e \cap f = \{v\}$ holds for $e, f \in P_v$, i.e, $\mathcal{A}_v(P_v) \in \mathcal{I}$. Note that if $\max_{e \in E} |e| \leq 2$, the required condition is satisfied.

Lemma 21. *Algorithm* `OrderedApprox`* *satisfies that in the beginning of the i-th iteration of the for-loop,*

$$I \cup \bigcup_{j<i} I_{v_j} \in \mathcal{I} \text{ for any independent set } I \in \mathcal{I}[P_{v_i}] \qquad (9)$$

Algorithm `OrderedApprox`$^*(G, \mathcal{I}, \prec)$

Input: An independence system $(E, \mathcal{I})$ defined on a hypergraph $G = (V, E)$ and a linear order $v_1 \prec v_2 \prec \ldots \prec v_n$ of vertices V.

Output: An independent set in $\mathcal{I}$.

1: $X := \emptyset$.
2: $P_v := D_v$ for $v \in V$.
3: **for** $i = 1, \ldots, n$ **do**
4: $\quad B_{v_i} = \left\{ e \in U_{v_i} \setminus \left(\bigcup_{l<i} R_{v_l} \right) \;\middle|\; \begin{array}{l} e \in \mathcal{A}_{v_i}(P_{v_i} \cup \{e\}) \text{ and} \\ |\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})| > |\mathcal{A}_{v_i}(P_{v_i})| \end{array} \right\}$.
5: $\quad$ **if** $U_{v_i} = \emptyset$ **then**
6: $\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i})$.
7: $\quad\quad R_{v_i} := \emptyset$.
8: $\quad$ **else if** $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} = \emptyset$ **then**
9: $\quad\quad$ Choose an edge $e \in U_{v_i}$ arbitrarily.
10: $\quad\quad$ **if** $e \notin \mathcal{A}_{v_i}(P_{v_i} \cup \{e\})$ **then**
11: $\quad\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i} \cup \{e\})$.
12: $\quad\quad$ **else** (i.e., $|\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})| = |\mathcal{A}_{v_i}(P_{v_i})|$)
13: $\quad\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i})$. /* $|I_{v_i}| = |\mathcal{A}_{v_i}(P_{v_i} \cup \{e\})|$ */
14: $\quad\quad$ **end if**
15: $\quad\quad P_{v_i} := P_{v_i} \cup \{e\}$.
16: $\quad\quad R_{v_i} := U_{v_i} \setminus \{e\}$.
17: $\quad\quad$ **for** $v_j \in e \setminus \{v_i\}$ **do**
18: $\quad\quad\quad P_{v_j} := P_{v_j} \setminus \{e\}$.
19: $\quad\quad$ **end for**
20: $\quad$ **else if** $U_{v_i} \neq \emptyset$, $P_{v_i} \neq \emptyset$, and $B_{v_i} \neq \emptyset$ **then**
21: $\quad\quad$ Choose an edge $b \in B_{v_i}$ arbitrarily.
22: $\quad\quad I_{v_i} := \mathcal{A}_{v_i}(P_{v_i} \cup \{b\}) \setminus \{b\}$. /* $I_{v_i} \subseteq P_{v_i}$ and $|I_{v_i}| = |\mathcal{A}_{v_i}(P_{v_i})|$ */
23: $\quad\quad R_{v_i} := U_{v_i} \setminus \{b\}$.
24: $\quad$ **else** (i.e., $U_{v_i} \neq \emptyset$ and $P_{v_i} = \emptyset$)
25: $\quad\quad I_{v_i} := \emptyset$.
26: $\quad\quad R_{v_i} := \emptyset$. /* R_{v_i} might be updated in line 32 */
27: $\quad\quad X := X \cup \{v_i\}$.
28: $\quad$ **end if**
29: $\quad$ **for** $e \in I_{v_i}$ **do**
30: $\quad\quad$ **for** $v_j \in e \setminus \{v_i\}$ **do**
31: $\quad\quad\quad$ **if** $v_j \in X$ **then**
32: $\quad\quad\quad\quad R_{v_j} := \{e \in U_{v_j} \setminus (\bigcup_{l<i} R_{v_l}) \mid v_i \prec u \text{ for some } u \in e\}$.
33: $\quad\quad\quad\quad X := X \setminus \{v_j\}$.
34: $\quad\quad\quad$ **end if**
35: $\quad\quad$ **end for**
36: $\quad$ **end for**
37: $\quad$ **for** $j = i + 1, \ldots, n$ **do**
38: $\quad\quad P_{v_j} := P_{v_j} \setminus (\bigcup_{l \leq i} R_{v_l})$.
39: $\quad$ **end for**
40: **end for**
41: $R = \bigcup_{v \in V} R_v$.
42: Output $I = \bigcup_{v \in V} I_v$ and halt.

if $e \cap f = \{v_j\}$ holds for $j \leq i$ and for $e, f \in P_{v_j}$.

Lemma 22. *Algorithm* `OrderedApprox`* *computes an $(\alpha + \delta(\gamma - 1))$-approximate independent set I in $\mathcal{I}$ in polynomial time if a given hypergraph G satisfies that for $v \in V$, $e \cap f = \{v\}$ for $e, f \in D_v$ with respect to a given linear order $\prec$ of vertices V, where γ is the width of G with respect to $\prec$, and $\delta = \max_{e \in E} |e|$.*

We have the following corollary.

Corollary 23. *Algorithm* `OrderedApprox`* *computes an α-approximate independent set I in $\mathcal{I}$ in polynomial time if G is a hypergraph of width* 1.

We can also show that Algorithm `DecomApprox` can be generalized for the hypergraph case. Namely, for a hypergraph

$G = (V, E)$ of width γ, Algorithm `DecomApprox`* first decomposes edge set E into $E_1, \ldots, E_\gamma$, where $G_i = (V, E_i)$ is a hypergraph of width 1, for each E_i, applies `OrderedApprox`* to compute an independent set I_i, and chooses a maximum independent set among them.

Algorithm `DecomApprox`$^*(G, \mathcal{I}, \prec)$

Input: An independence system $(E, \mathcal{I})$ defined on a hypergraph $G = (V, E)$ and a linear order $v_1 \prec v_2 \prec \cdots \prec v_n$ of vertices V, where γ is the width of hypergraph G with respect to $\prec$.

Output: An independent set in $\mathcal{I}$

$\mathcal{Q} := \emptyset$
for $i = 1, \ldots, n$ **do**
  $U'_{v_i} := \{e \in U_{v_i} \mid v_i \prec v \text{ for all } v \in e \setminus \{v_i\}\}$.
  for each $e \in U'_{v_i}$ **do**
    $u := \max_{v \in e} v$.
    $\mathcal{Q}_e := \{Q' \in \mathcal{Q} \mid Q' \cap U_v = \emptyset \text{ for all } v \in e \setminus \{u\}\}$.
    if $\mathcal{Q}_e \neq \emptyset$ **then**
      $Q \in \mathcal{Q}_e$
    else
      $Q := \emptyset$.
      $\mathcal{Q} := \mathcal{Q} \cup \{Q\}$.
    end if
    $Q := Q \cup \{e\}$.
  end for
end for
for $Q \in \mathcal{Q}$ **do**
  $I_Q := d$`OrderedApprox`$^*(G[Q], \mathcal{I}[Q], \prec)$.
end for
$I \in \arg\max\{|I_Q| \mid Q \in \mathcal{Q}\}$.
Output I and halt.

Thus we have the following theorem.

Theorem 24. *Algorithm* `DecomApprox`* *computes an* $(\alpha + \alpha(\delta - 1)(k - 1))$*-approximate independent set* I *in* $\mathcal{I}$ *in polynomial time if* G *is a hypergraph of width* k*, where* $\delta = \max_{e \in E} |e|$.

Note that Theorem 24 is regarded as a generalization of Corollary 12.

Acknowledgments

I give special gratitude to Kazuhisa Makino for his valuable discussions and carefully proofreading of the manuscript. I am also grateful to Yusuke Kobayashi, Akitoshi Kawamura, and Satoru Fujishige for constructive suggestions. This work was supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society".

References

[1] Amano, Y.: Locally defined independence systems on graphs, *Discrete Applied Mathematics*, Vol. 326, pp. 1–16 (2023).
[2] Avidor, A., Berkovitch, I. and Zwick, U.: Improved Approximation Algorithms for MAX NAE-SAT and MAX SAT, *Approximation and Online Algorithms*, pp. 27–40 (2005).
[3] Chekuri, C. and Kumar, A.: Maximum coverage problem with group budget constraints and applications, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, Springer, pp. 72–83 (2004).
[4] Cook, W., Cook, W., Cunningham, W., Pulleyblank, W. and Schrijver, A.: *Combinatorial Optimization*, A Wiley-Interscience publication, Wiley (1997).
[5] Freuder, E. C.: A sufficient condition for backtrack-free search, *Journal of the ACM (JACM)*, Vol. 29, No. 1, pp. 24–32 (1982).
[6] Fujito, T.: A 2/3-Approximation of the Matroid Matching, *Algorithms and Computation: 4th International Symposium, ISAAC'93, Hong Kong, December 15-17, 1993. Proceedings*, Vol. 762, Springer Science & Business Media, p. 185 (1993).
[7] Jenkyns, T.: Matchoids: a generalization of matchings and matroids., PhD Thesis, University of Waterloo (1975).
[8] Kostakos, V.: Temporal graphs, *Physica A: Statistical Mechanics and its Applications*, Vol. 388, No. 6, pp. 1007–1023 (2009).
[9] Lee, J., Sviridenko, M. and Vondrák, J.: Matroid matching: the power of local search, *SIAM Journal on Computing*, Vol. 42, No. 1, pp. 357–379 (2013).
[10] Lick, D. R. and White, A. T.: k-Degenerate Graphs, *Canadian Journal of Mathematics*, Vol. 22, No. 5, p. 1082–1096 (online), DOI: 10.4153/CJM-1970-125-1 (1970).
[11] Lovász, L.: The matroid matching problem, *Algebraic methods in graph theory*, Vol. 2, pp. 495–517 (1978).
[12] Mandal, S. and Gupta, A.: 0-1 Timed Matching in Bipartite Temporal Graphs, *Conference on Algorithms and Discrete Applied Mathematics*, Springer, pp. 331–346 (2020).
[13] Matula, D. W. and Beck, L. L.: Smallest-last ordering and clustering and graph coloring algorithms, *Journal of the ACM (JACM)*, Vol. 30, No. 3, pp. 417–427 (1983).
[14] Oxley, J. G.: *Matroid theory*, Vol. 3, Oxford University Press, USA (2006).
[15] Plummer, M. D. and Lovász, L.: *Matching theory*, Elsevier Science Ltd. (1986).
[16] Schrijver, A. et al.: *Combinatorial optimization: polyhedra and efficiency*, Vol. 24, Springer (2003).
[17] Welsh, D. and Society, L. M.: *Matroid Theory*, L.M.S. monographs, Academic Press (1976).