Approximation Algorithms for Independent Set Problems on Hypergraphs

Elena Losievskaja
Doctor of Philosophy
December 2009
School of Computer Science
Reykjavík University

Ph.D. DISSERTATION

ISSN 1670-8539

Approximation Algorithms for Independent Set Problems on Hypergraphs

by

Elena Losievskaja

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

December 2009

Thesis Committee:

Magnús Már Halldórsson, Supervisor
Professor, Reykjavík University, Iceland

Sven Þórarinn Sigurðsson
Professor, University of Iceland, Iceland

Bjarni Vilhjálmur Halldórsson
Associate Professor, Reykjavík University, Iceland

Mario Szegedy
Professor, Rutgers University, USA

Hadas Shachnai, Examiner
Associate Professor, Technion, Israel

Approximation Algorithms for Independent Set Problems on Hypergraphs

Elena Losievskaja

December 2009

Abstract

This thesis deals with approximation algorithms for the Maximum Independent Set and the Minimum Hitting Set problems on hypergraphs. As a hypergraph is a generalization of a graph, the question is whether the best known approximations on graphs can be extended to hypergraphs.

We consider greedy, local search and partitioning algorithms. We introduce a general technique, called *shrinkage reduction*, that reduces the worst case analysis of certain algorithms on hypergraphs to their analysis on ordinary graphs. This technique allows us to prove approximation ratios for greedy and local search algorithms for the Maximum Weak Independent Set problem on weighted and unweighted bounded-degree hypergraphs. For the weighted case we improve bounds using a simple partitioning algorithm. We also consider two variations of the max-greedy algorithms for the Maximum Strong Independent Set problem.

We describe an SDP-based approach for the Maximum Weak Independent Set problem on bounded-degree hypergraphs. Our approach is to use semidefinite technique to sparsify a given hypergraph and then apply combinatorial algorithms to find a large independent set in the resulting sparser instance. Using this approach we obtain the first performance ratio that is sublinear in terms of the maximum or average degree of the hypergraph. We extend this result to the weighted case and give a similar bound in terms of the average weighted degree in a hypergraph, matching the best bounds known for the special case of graphs.

We present several randomized and deterministic algorithms for the Maximum Weak Independent Set problem in a semi-streaming model. All our semi-streaming algorithms require only one pass over the stream and most of them resemble on-line algorithms in maintaining a feasible solution at all times. We introduce the *on-line minimal space* semi-streaming model and give lower and upper bounds for deterministic and randomized algorithms in this model.

Nálgunarreiknirit fyrir óháð mengi í ofurnetum

Elena Losievskaja

Desember 2009

Útdráttur

Þessi ritgerð fjallar um bestunaraðferðir fyrir ofurnet, eða nánar tiltekið um nálgunarreiknirit fyrir verkefnin *stærsta óháða mengi* (e. maximum independent set) og *minnsta skurðmengi* (e. minimum hitting set) á ofurnetum. Ofurnet eru útvíkkun á netum, og því er spurningin hvort þekktar nálgunaraðferðir fyrir net megi einnig útvíkka fyrir ofurnet.

Við skoðum fyrst reiknirit sem byggja á gráðugu vali, staðbundinni bestu, og skiptingu. Við kynnum nýja almenna smækkunaraðferð sem gerir okkur kleift að yfirfæra nálgunarniðurstöður fyrir net sem byggja á gráðugu vali og staðbundinni bestun yfir á vegin og óvegin ofurnet með takmarkaða hámarksgráðu. Fyrir vegin ofurnet bætum við nálgunina enn frekar með því að nota einfalda skiptingaraðferð. Við skoðum einnig tvær útgáfur af gráðugri aðferð til að finna stærsta *sterka* óháð mengi í ofurneti.

Við lýsum síðan aðferð sem beitir hálfákveðinni bestun til að gera ofurnetið fyrst rýrara og nýtir síðan samtektareiknirit á rýra netið. Með þessari aðferð fáum við út fyrstu nálgunarniðurstöður sem vaxa hægar en línulega sem fall af hámarks- eða meðaltalsgráðu ofurnetsins. Við útvíkkum þessar niðurstöður fyrir vegin ofurnet, og fáum sambærilegar niðurstöður sem fall af veginni meðalgráðu.

Í síðasta hluta ritgerðarinnar snúum við okkur að bunuvinnslu (e. *streaming algorithms*). Þá koma leggir netsins í runu, en reikniritið getur aðeins geymt lítinn hluta netsins. Við leiðum út ýmis slembin og ákvörðunarbundin reiknirit fyrir stærsta óháða mengi í netum og ofurnetum. Allar aðferðirnar krefjast aðeins einnar yfirferðar yfir strauminn og eru flestar raðbundnar (e. *online*) að því leyti að gildri lausn er sífellt viðhaldið. Við kynnum nýtt líkan fyrir bunuvinnslu á netum sem krefst raðbundinnar vinnslu með lágmarks plássþörf, og gefum efri og neðri mörk fyrir reiknirit í því líkani.

To my parents

Acknowledgements

I would like to thank all people who were with me through all these years of hard work for their support, encouragement and belief in me.

First of all, it is my supervisor, Magnús Már Halldórsson, without his guidance, inspiration, gentle criticism and enormous support it would be impossible for me to complete this dissertation. I'm infinitely grateful to him for sharing with me his knowledge and wisdom, for his valuable advice, comments and suggestions, for many interesting discussions related to research and beyond, for his ability of being demanding and patient at the same time, for teaching me to follow high standards in research and never give up.

I would like to thank the members of my PhD committee and the examiner, Sven Sigurðsson, Bjarni Halldórsson, Mario Szegedy and Hadas Shachnai, whose comments and suggestions helped me in improving my dissertation. I am also thankful to all my coauthors and colleagues at Reykjavík University, University of Iceland, University of Bergen and University of Rome "La Sapienza" for their collaboration and great work experience.

I would like to acknowledge the Iceland Research Fund for supporting my work with grants 50006022 and 70009021, and the Research Fund of University of Iceland.

Finally, I express my endless gratitude and love to my family and friends. To my parents, Nina and Lev, for their infinite love and belief in me, for wise thoughts and advise, for huge support and encouragement in everything I do - for all this and much more I am forever grateful to them. To my husband, Tómas Davíð, for being a very special person in my life, for always making me smile and reminding me to look on the bright side of life, for sharing all wonderful and difficult moments with me, it would be tremendously hard to accomplish this work without him. I am a very fortunate person to have truly good friends, and I am enormously grateful to all of them for their support and love, for all moments of fun we have together. Many thanks to all of you.

Publications

The following work was published during my Ph.D. studies. This dissertation is partially based on papers 1, 4, 5 and 6 below:

1. G. Agnarsson, M. M. Halldórsson and E. Losievskaja, SDP-Based Algorithms for Maximum Independent Set Problems on Hypergraphs, *In Proc. of 36th International Colloquium on Automata, Languages and Programming (ICALP)*, 12-23, 2009.

2. M. R. Fellows, F. V. Fomin, D. Lokshtanov, E. Losievskaja, F. A. Rosamond and S. Saurabh, Parameterized Low-distortion Embeddings - Graph metrics into lines and trees *CoRR abs*, 0804.3028, 2008.

3. M. R. Fellows, F. V. Fomin, D. Lokshtanov, E. Losievskaja, F. A. Rosamond and S. Saurabh, Distortion Is Fixed Parameter Tractable, *In Proc. of 36th International Colloquium on Automata, Languages and Programming (ICALP)*, 463-474, 2009.

4. M. M. Halldórsson and E. Losievskaja, Independent Sets in Bounded-Degree Hypergraphs, *In Proc. of 10th International Workshop on Algorithms and Data Structures (WADS)*, 263-274, 2007.

5. M. M. Halldórsson and E. Losievskaja, Independent sets in bounded-degree hypergraphs, *Discrete Applied Mathematics*, 157(8): 1773-1786, 2009.

6. B. V. Halldórsson, M. M. Halldórsson, E. Losievskaja and Mario Szegedy, Semi-streaming algorithms on hypergraphs, *Manuscript*, 2009.

Contents

List of Figures

List of Abbreviations

G	graph
H	hypergraph
$\mathcal{G}$	a collection of graphs
$\mathcal{H}$	a collection of hypergraphs
V	a vertex set
E	a set of edges
n	number of vertices
m	number of edges
r	rank of a hypergraph
$d(v)$	the degree of a vertex v
$D(v)$	the weighted degree of a vertex v
$d^*(v)$	the hyperdegree of a vertex v
$D^*(v)$	the weighted hyperdegree of a vertex v
$\overline{d}$	the average degree
$\overline{D}$	the weighted average degree
$\overline{d^*}$	the average hyperdegree
$\overline{D^*}$	the weighted average hyperdegree
Δ	the maximum degree
Δ^*	the maximum hyperdegree
$N(v)$	the neighborhood of a vertex v
IS	Maximum (Weak) Independent Set
SIS	Maximum Strong Independent Set
WIS	Maximum Weighted (Weak) Independent Set
VC	Minimum Vertex Cover
HS	Minimum Hitting Set
$\alpha(H)$	the size of a maximum independent set
$\alpha(H, w)$	the weight of a maximum independent set
$\ln n$	natural logarithm of n
$\log n$	binary logarithm of n

Chapter 1

Introduction

A hypergraph H consists of a set V of vertices and a set E of edges, where an edge connects an arbitrary number of vertices. As an edge in H is a subset of V, hypergraphs are sometimes referred to as *set systems* or *set collections*. In this work we consider a hypergraph as a generalization of a graph (a graph is defined by a set V of vertices and a set E of edges, where an edge connects a pair vertices).

The generalization often leads to simplification and provides an insight why some problems are hard to solve or approximate. A single statement on hypergraphs can unify several theorems on graphs [8] and gives a new tool in exploring different graph properties. Moreover, studying hypergraph properties helps to reveal the hard part of the problems on graphs, improve existing results and design new solutions to known combinatorial problems.

This thesis deals with optimization problems on hypergraphs using various algorithmic approaches and computational models. We begin by introducing the main definitions, approaches and models.

1.1 Definitions

A *hypergraph* H is a pair (V, E), where $V = \{v_1, \ldots, v_n\}$ is a discrete set of vertices and $E = \{e_1, \ldots, e_m\}$ is a collection of subsets of V, or *(hyper)edges*. If V and E are finite sets, then a hypergraph $H(V, E)$ is *finite*, otherwise H is *infinite*. Edges in E are arbitrary subsets of V, and therefore can contain arbitrary number of vertices. The *size*, or the *cardinality*, $|e|$ of an edge $e \in E$ is the number of vertices in e. An edge of size t is called a t*-edge*. Then, the *rank* r of a hypergraph H is the maximum edge size in H. A

hypergraph is r-*uniform* if all edges have the same cardinality r. A *graph* G is a special case of hypergraphs, more precisely graphs are 2-uniform hypergraphs.

There exist several types of hypergraphs depending on how the set E of edges is defined. A hypergraph H is *directed*, if every edge in E is an ordered set, otherwise H is *undirected*. A hypergraph H is a *multihypergraph*, if E is a multiset of (not necessarily distinct) vertices, i.e. E contains multiple edges and an edge in E may contain the same vertex more than once. Sometimes, a multihypergraph also allows *loops*, which are sets containing a single vertex. As opposed to a multihypergraph, a *simple* hypergraph contains only unique edges of size at least 2, every edge is a set of distinct vertices and no edge is a subset of another edge. In this work we consider *undirected simple finite* hypergraphs.

The hypergraphs can be "dense" (have many edges) and "sparse" (have few edges), the "density" of a hypergraph $H(V, E)$ is described by the *degrees of the vertices* in V. Let $d_t(v) = |\{e|v \in e, |e| = t\}|$ be t-*degree of a vertex* v, or the number of t-edges incident on v. We denote by $\Delta_t = \max_{v \in V} d_t(v)$ and $\bar{d}_t = (\sum_{v \in V} d_t(v))/|V|$ the *maximum* and the *average* t-*degree* in a hypergraph, respectively. The *degree* $d(v) = \sum_{t=2}^{r} d_t(v)$ of a vertex v is then the total number of edges incident on v. We denote by $\Delta = \max_{v \in V} d(v)$ and $\bar{d} = (\sum_{v \in V} d(v))/|V|$ the *maximum* and the *average degree* in a hypergraph, respectively. A hypergraph H is *regular*, if every vertex in E is of the same degree Δ.

Further, we can refine the degrees in hypergraphs to reflect the fact that edges in a hypergraph can be of different sizes. Let $d^*(v) = \sum_{t=2}^{r} d_t(v)^{\frac{1}{t-1}}$ be the *hyperdegree* of a vertex v. We denote by $\Delta^* = \max_{v \in V} d^*(v)$ and $\bar{d^*} = (\sum_{v \in V} d^*(v))/|V|$ the *maximum* and the *average hyperdegree* in a hypergraph, respectively. Note, in a hypergraph $d(v) \geq d^*(v)$ always holds for any vertex v, with equality holding only in graphs.

A hypergraph $H(V, E)$ is *weighted* if every vertex in V is assigned a weight, such weights might represent costs, lengths, capacities or any other quantity depending on the problem. The *weight of a hypergraph* $w(H)$ is then the total weight of the vertices. Formally, given a function $f : V \to R$ that assigns weights to the vertices of H, let $w(H) = w(V) = \sum_{v \in V} w(v)$. We define $D(v) = w(v)d(v)$ and $\bar{D} = \sum_{v \in V} w(v)d(v) / \sum_{v \in V} w(v)$ to be the *weighted degree* of a vertex v and the *average weighted degree* in H, respectively. Similarly, we define $D^*(v) = w(v)d^*(v)$ and $\overline{D^*} = (\sum_{v \in V} D^*(v))/(\sum_{v \in V} w(v))$ to be the *weighted hyperdegree* of a vertex v and the *weighted average hyperdegree* in H, respectively.

A vertex u is a *neighbor* of a vertex v, if there exist an edge $e \in E$ that includes both u and v. Given a vertex $v \in V$, we denote by $N(v)$ a set of neighbors of v. Let $N_t(v) = \{u \in V : \exists e \in E, \{u, v\} \subseteq e, |e| = t\}$ be the *set of neighbors of v in edges of size t*. Given a set $U \subseteq V$, let $N(U) = \{v \in V \backslash U : \exists u \in U, \exists e \in E, \{u, v\} \subseteq e\}$ be the *set of neighbors of vertices in U*.

A *hyperclique* is a hypergraph in which each vertex is a neighbor of all other vertices. Note, that a hyperclique need not be a uniform hypergraph. By analogy with a graph being a 2-uniform hypergraph, a clique is a 2-uniform hyperclique.

An *n-star* is a tree on $n + 1$ nodes with one node of degree n (the *root* of the star) and the others of degree 1 (the *endpoints* of the star).

Finally, we introduce some operations on vertices and edges in a hypergraph. By *deleting a vertex v from an edge e* we mean the operation of replacing e by $e \setminus \{v\}$. By *deleting a vertex v from a hypergraph H* we mean just one operation: $V = V \backslash \{v\}$, and by *deleting v with all incident edges* we mean two operations: $V = V \backslash \{v\}$ and $E = E \backslash \{e \in E | v \in e\}$.

We say that a hypergraph $H'(V', E')$ is *induced* in $H(V, E)$ on $V' \subseteq V$, if $E' = \{e \in E | e \subseteq V'\}$. A *subhypergraph* $H(V')$ is a hypergraph $H'(V', E')$ *induced* in $H(V, E)$ on the vertex set $V' \subset V$. The *dual* of a hypergraph H is a hypergraph H^* where for each edge in H there is a vertex in H^* and for each vertex v in H there is an edge e^* in H^* formed by the vertices in H^* corresponding to the edges incident on v, i.e. $e^* = \{e_i \in E | v \in e_i\}$.

There are several subsets of vertices in hypergraphs with special properties, namely: *independent sets* and *hitting sets*. An *independent* set in H is a subset of V that doesn't contain any edge of H, also referred to as a *weak independent set* [8].

WEAK INDEPENDENT SET: a set $I_w \subseteq V$ such that $\forall e \in E : |e \cap I_w| < |e|$.

If an independent set in H intersects any edge in E in at most one element, then it is said to be a *strong independent set* [8].

STRONG INDEPENDENT SET: a set $I_s \subseteq V$ such that $\forall e \in E : |e \cap I_s| \leq 1$.

It follows from these definitions that a strong independent set is a special case of a weak independent set. In the case of graphs (2-uniform hypergraphs), there is no distinction between weak and strong independent sets. In the remainder of the thesis, when we say an independent set, we mean a weak independent set, unless otherwise is stated.

The vertices not contained in a weak independent set form a *vertex cover*, or a *hitting set*. In other words, a *hitting set* is a subset of V that intersects every edge of H in at least one vertex.

HITTING SET: a set $S \subseteq V$ such that $\forall e \in E : |e \cap S| \geq 1$.

A hitting set in graphs is usually referred to as a *vertex cover*. A hitting set is closely related to a *set cover*, which is a subset of E that covers all vertices in V.

SET COVER: a set $C \subseteq E$ such that $\forall v \in V \ \exists e \in C : v \in e$.

Thus, a hitting set in H is equivalent to a set cover in the dual H^*.

In the remainder, we let $\mathcal{H}$ and $\mathcal{G}$ be the collections of all hypergraphs and graphs, respectively. We denote by H a hypergraph in $\mathcal{H}$ and by G a graph in $\mathcal{G}$, respectively.

1.2 Optimization Problems

1.2.1 Definitions

A *computational problem* $\Pi =< \mathcal{I}, \mathcal{S} >$ consists of a set $\mathcal{I}$ of *problem instances* and a set $\mathcal{S}$ of *feasible solutions*, where for each problem instance $I \in \mathcal{I}$ there is a feasible solution $S \in \mathcal{S}$. The problem instance I is a particular input string to Π and the feasible solution S is the output string corresponding to I.

A *decision problem* $\Pi_d =< \mathcal{I}, \{0, 1\} >$ is a computational problem where a feasible solution to a given instance I of Π can be either 1 ('yes') or 0 ('no'). In other words, Π_d is a problem of deciding whether I has a required property or not. For example, the decision problem k-INDEPENDENT SET asks whether a given hypergraph H has an independent set of size k, and if there exist an independent set of size k in H, then the answer is 1 ('yes'), otherwise - 0 ('no').

An *optimization problem* $\Pi_o =< \mathcal{I}, \mathcal{S}, w, goal >$ is a computational problem which also has a function $w : \mathcal{S} \rightarrow R$ that assigns a cost to every solution in $\mathcal{S}$ (usually we assume that costs are non-negative), and a $goal$ that defines whether Π_o is a minimization or a maximization problem. In other words, Π_o is a problem of finding a feasible solution to a given problem instance I of maximum or minimum cost depending on the goal. For example, the optimization problem MAXIMUM INDEPENDENT SET asks for a maximum independent set in a given hypergraph H, in this case a set of feasible solutions con-

sists of all independent sets in H and the goal is to find an independent set of maximum cardinality or weight.

In this work we consider the following computational problems:

The decision problem k-(WEAK) INDEPENDENT SET:
given a hypergraph $H(V, E)$ and an integer k,
does H have a (weak) independent set of size k?

and the optimization problems MAXIMUM (WEAK) INDEPENDENT SET (IS):
given a hypergraph $H(V, E)$,
find a maximum cardinality (weak) independent set in H.

MAXIMUM WEIGHTED (WEAK) INDEPENDENT SET (WIS):
given a weighted hypergraph $H(V, E)$,
find a (weak) independent set in H of maximum weight.

The decision problem k-STRONG INDEPENDENT SET:
given a hypergraph $H(V, E)$ and an integer k,
does H have a strong independent set of size k?

and the optimization problems MAXIMUM STRONG INDEPENDENT SET (SIS):
given a hypergraph $H(V, E)$,
find a maximum cardinality strong independent set in H.

MAXIMUM WEIGHTED STRONG INDEPENDENT SET ($WSIS$):
given a weighted hypergraph $H(V, E)$,
find a strong independent set in H of maximum weight.

The decision problem k-HITTING SET:
given a hypergraph $H(V, E)$ and an integer k,
does H have a hitting set of size k?

and the optimization problem — MINIMUM HITTING SET (HS):
given a hypergraph $H(V, E)$,
find a minimum cardinality hitting set in H.

The decision problem — k-SET COVER:
given a hypergraph $H(V, E)$ and an integer k,
does H have a set cover of size k?

and the optimization problem — MINIMUM SET COVER (HS):
given a hypergraph $H(V, E)$,
find a minimum cardinality set cover in H.

As mentioned above, in graphs there is no distinction between strong and weak independent sets, and the MAXIMUM INDEPENDENT SET problem on graphs is denoted by IS. For a unweighted hypergraph H, we denote by $\alpha(H)$ the size of a maximum independent set in H. For a weighted hypergraph H, we denote by $\alpha(H, w)$ the weight of a maximum independent set in H.

1.2.2 Applications

IS, HS, SC are all of fundamental interest, both in practical and theoretical aspects. They arise in various applications in data mining, image processing, database design, parallel computing and many others.

An example of the MAXIMUM INDEPENDENT SET problem is the problem of scheduling the poster presentation. Suppose the students are going to present the posters of their work, where each poster can have more than one student as an author and each student can have more than one poster. The goal is to assign as many posters as possible to the same time slot conditioning that each student can present at most one poster in the same time slot. We construct a hypergraph with a vertex for each poster and an edge for each student, then a maximum strong independent set represents the maximum number of posters that can be presented at the same time.

An example of the MINIMUM HITTING SET problem is the problem of hiring teachers at the school. Suppose teachers apply for positions with the lists of courses they can teach, the school administration then tries to hire the least possible number of teachers so that each course in the school program is taught by at least one teacher. We construct a

hypergraph with a vertex for each teacher and an edge for each course, then a minimum hitting set represents the minimum group of teachers that need to be hired to teach all courses at this school.

An example of the MINIMUM SET COVER problem is the virus testing problem. Suppose we have a set of viruses and a collection of strings of 20 or more consecutive bytes from viruses specific only for viruses, each such string may be common in several viruses. The goal is to find a minimum set of strings that need to be searched for to confirm the presence of viruses. We construct a hypergraph with a vertex for each virus and an edge for each string of bytes, then a minimum set cover represents the minimum set of strings such that each virus is presented by at least one string in this set.

1.2.3 Hardness

All computational problems can be divided into classes depending on how much time or/and space is needed to find an optimal solution. One of the fundamental complexity classes, P or $PTIME$, contains all decision problems that can be solved in time polynomial in the size of the problem instance using *deterministic Turing machine*. *Turing machine* is a computational model which can simulate the logic of any algorithm. In a *deterministic Turing machine* for any given configuration (described by the current state and the input symbol) there is at most one possible transition (described by the next state, output symbol and the move of the tape). It means that for any given input to a deterministic Turing machine there is at most one possible output. In a *non-deterministic Turing machine* for any given configuration there may be more than one possible transition. It means that for any given input to a non-deterministic Turing machine there may be more than one possible output. Non-deterministic Turing machine accepts a problem instance, i.e. outputs the answer 'yes', if at least one of possible outputs of its computational process results in 'yes'. All decision problems that can be solved in time polynomial in the size of the problem instance using *non-deterministic Turing machine* form the complexity class NP. Alternatively, the class NP is defined as the class of decision problems, solutions to which can be verified in polynomial time using deterministic Turing machine. It follows that, P is a subset of NP; the question whether $P = NP$ is one of the most important question in computer science and it still remains open.

Among all problems in NP there is a special class of problems, so called *NP-complete* problems. NP-complete problems have a special property, namely: if any single NP-complete problem can be solved in polynomial time by deterministic Turing machine, then so can be solved every problem in NP. By this reason, NP-complete problems are

considered to be the hardest problems in NP. All problems we consider in this work, k-INDEPENDENT SET, k-HITTING SET, k-SET COVER and k-VERTEX COVER, are known to be NP-complete [41]. A decision problem only answers the question whether a given problem instance has a solution of size k, while the corresponding optimization problem has to find an optimal solution (not only its size), it means that the above optimization problems are at least as hard as their decision versions.

Given that no algorithm finds an optimal solution for NP-complete problems in polynomial time under the assumption $P \neq NP$, the question is then how well we can approximate an optimal solution to the corresponding optimization problem. Numerous results in approximation area can be split into two categories: approximation algorithms and inapproximability results. Let us first discuss approximation algorithms and then the inapproximability results.

1.3 Algorithmic Approaches

Many fundamental problems on graphs and hypergraphs are difficult to solve in polynomial time, among them are the MAXIMUM INDEPENDENT SET, the MINIMUM HITTING SET, the MINIMUM SET COVER and the MINIMUM VERTEX COVER problems. Unless $P = NP$, it is very unlikely that there exist an algorithm that solves any of these problems in time less than exponential in the size of the problem instance (i.e. graph or hypergraph size).

The computational challenge of solving hard computational problems gave rise to two main approaches in computation: *approximation* and *exact* algorithms. The difference between these two approaches is the tradeoff between the running time of the algorithm and the quality of the solution it outputs.

Approximation algorithms find a feasible solution fast (usually in polynomial time), but the solution is not guaranteed to be optimal; we are interested in bounding the approximation ratio of such algorithms, which is a relationship between the cost of the solution found by the algorithm on a given instance and the cost of an optimal solution on that instance maximized over all problem instances.

Exact algorithms always find an optimal solution, usually in exponential time; in this case we are interested in reducing the running time of the algorithm.

In this work we focus on approximation algorithms.

1.3.1 Approximation Algorithms

Approximation algorithms are algorithms that find approximate solutions to optimization problems (usually in polynomial time). As opposed to approximation algorithms, *optimal* algorithms find optimal solutions. Approximation algorithms are essential for problems in $NP \backslash P$, since these problems are unlikely to admit polynomial-time optimal algorithms, unless $P = NP$. However, approximation algorithms are also very useful for problems that do have polynomial-time optimal algorithms, but where these optimal algorithms may take too long for large inputs.

Approximation algorithms are characterized by their *approximation ratio* and *absolute error*. Suppose we have a maximization problem Π. For a given problem instance I and a given approximation algorithm A, let $OPT(I)$ and $ALG(I)$ be the costs of an optimal solution and the cost of the solution produced by A, respectively. The approximation algorithm A has a *approximation ratio* ρ for Π if $\frac{OPT(I)}{ALG(I)} \leq \rho$ for any problem instance I. In other words, the approximation ratio ρ of A is defined as $\rho = \max\limits_{\forall I} \frac{OPT(I)}{ALG(I)}$. An approximation ratio is sometimes called a *relative performance guarantee* or a *performance ratio*. The approximation algorithm A has an *absolute error* ϱ for Π if $OPT(I) - ALG(I) \leq \varrho$ for any problem instance I. In other words, the absolute error ϱ of A is defined as $\varrho = \max\limits_{\forall I}(OPT(I) - ALG(I))$. An absolute error is sometimes called an *absolute performance guarantee*.

For a minimization problem Π the approximation ratio and absolute error are defined as $\rho = \max\limits_{\forall I} \frac{ALG(I)}{OPT(I)}$ and $\varrho = \max\limits_{\forall I}(ALG(I) - OPT(I))$, respectively.

In what follows, we will consider only the approximation ratio. An algorithm A is said to be a ρ-approximation algorithm for a problem Π, if the approximation ratio of A for Π is ρ.

There are several techniques used in algorithm design. Let us give a brief outline of the techniques used in this work.

1.3.2 Algorithmic Techniques

Greedy

A *greedy algorithm* is an algorithm that obtains a solution to a problem by making a sequence of *locally optimal choices*, i.e. choices that seem to be the best at the moment. The idea is that a locally optimal choice will hopefully lead closer to a globally optimal choice. A greedy algorithm is somewhat "shortsighted" in the sense that it makes

a decision based only on the information at hand without considering the effect of these decisions in the future. The disadvantage of a greedy algorithm is that it never reconsiders its decision no matter how bad it may look later. As a result, many optimization problems can not be solved optimally with this approach. On the other hand, greedy algorithms are simple, straightforward, easy to invent and implement, and usually time efficient, which make them very useful in attacking hard problems in real-life applications. Even if in general greedy algorithms are not optimal, there are examples when greedy algorithms always find optimal solutions such as Kruskal's and Prim's algorithms for finding minimum weight spanning tree.

Local Search

A *local search* algorithm is an algorithm which starts with an (arbitrary) feasible solution and continually replaces it by a better solution while possible. The idea is that an algorithm searches for a better solution in the restricted "local" set of feasible solutions, called the *search space*, and therefore it takes polynomial time to find a better solution (if the algorithm would search in unrestricted search space, then it will essentially perform an exhaustive search, which may take exponential time). The search space depends on the current solution, and so it changes from iteration to iteration. Usually, the search space is defined to be a *neighborhood* of the current solution S, where the *neighborhood* of S is a set of solutions $\Gamma(S)$ produced by a *neighboring function* Γ. In other words, a neighboring function Γ defines which feasible solutions can be obtained from a solution S by changing some components of S. Hence, a local search algorithm iteratively improves a current solution by moving to a better solution in its neighborhood. The search for a better solution terminates either (i) when the current solution is the best possible in the current search space, and so no improvement can be made or (ii) when the improvement over several iterations is negligibly small or (iii) after predetermined number of iterations. Essentially, a local search algorithm finds a local optimal solution which might not be a global optimal solution. Nevertheless, the local search approach is usually very efficient in terms of time, and therefore is widely applied to tackle hard problems in various fields such as, for example, artificial intelligence, engineering, bioinformatics.

Partitioning

A *partitioning* algorithm is an algorithm which splits a given instance of a problem Π into disjoint partitions, finds solution in each partition and outputs the best solution found. Usually, but not necessarily, the partitions are all of the same size. Often the size of

partitions is such that Π can be solved optimally on each partition in polynomial time. Surprisingly, this simple approach gives one of the best approximation algorithms for some hard optimization problems [29, 30]. Partitioning is also often used in combination with other approximation algorithms.

Semidefinite Programming

Semidefinite programming refers to an optimization problem that can be expressed as:

$$\begin{array}{ll}\text{SEMIDEFINITE PROGRAM } SDP & \\ \text{MAXIMIZE} & W \bullet X \\ \text{SUBJECT TO} & A_j \bullet X = b_j, \quad 1 \leq j \leq m \\ & X \succeq 0\end{array}$$

where the variable $X \in \mathcal{S}^n$, and $\mathcal{S}^n$ is the space of real symmetric $n \times n$ matrices; the vector $\mathbf{b} \in \mathcal{R}^{\mathbf{m}}$ and the matrices $W, A_j \in \mathcal{S}^n$, for all $j = 1 \ldots m$, are given problem parameters [52]. The inequality $X \succeq 0$ means that X is positive semidefinite, and $W \bullet X$ denotes the inner product of matrices W and X, which is defined as $W \bullet X = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_{ij}$. SDP can be solved within an additive error of ϵ for any $\epsilon > 0$ in time polynomial in n and $\log(1/\epsilon)$ [25].

One of the properties of positive semidefinite matrices states that if X is positive semidefinite, there exists a $n \times n$ matrix U such that $X = UU^T$, where U can be found using an incomplete Cholesky factorization of X (in general this factorization is not polynomial time computable, but can be approximated to any desired degree in polynomial time). Let $\mathbf{u_1}, \ldots, \mathbf{u_n}$ be the columns of U (we might need to normalize them to obtain unit-length vectors). Then, SDP can be rewritten as a *vector program* [57]:

$$\begin{array}{lll}\text{VECTOR PROGRAM } VP & & \\ \text{MAXIMIZE} & W \bullet X & \\ \text{SUBJECT TO} & A_j \bullet X = b_j, & 1 \leq j \leq m \\ & |\mathbf{u_i}| = \mathbf{1}, & 1 \leq i \leq n \\ & \mathbf{u_i} \in \mathbf{R^n}, & 1 \leq i \leq n\end{array}$$

where $x_{ij} = \{\mathbf{u}_i \cdot \mathbf{u}_j\}$ for all $1 \leq i \leq j \leq n$. VP is equivalent to SDP [57].

We use vector programs to solve optimization problems on hypergraphs. For example, consider the MAXIMUM INDEPENDENT SET (IS). Given a hypergraph $H(V, E)$, let I be an independent set in H. We introduce a unit-length vector $\mathbf{u_i}$ for every vertex $v_i \in V$,

$1 \leq i \leq n$, and a unit-length vector $\mathbf{u_0}$ such that $\mathbf{u_i} = \mathbf{u_0}$ if $v_i \in I$, and $\mathbf{u_i} = -\mathbf{u_0}$, otherwise. Then, IS can expressed as a vector program as following:

$$
\begin{array}{lll}
VP - IS & & \\
\text{MAXIMIZE} & \sum_{v_i \in V} \frac{1+\mathbf{u}_i \cdot \mathbf{u}_0}{2} & \\
\text{SUBJECT TO} & \sum_{\substack{v_i, v_j \in V \\ 1 \leq i < j \leq |V|}} \mathbf{u}_i \cdot \mathbf{u}_j + \sum_{v_i \in V} \mathbf{u}_i \cdot \mathbf{u}_0 \leq \frac{|e|(|e|-3)}{2}, & \forall e \in E \\
& |\mathbf{u}_i| = 1, & 1 \leq i \leq |V| \\
& \mathbf{u_i} \in \mathbf{R^n}, & 1 \leq i \leq n
\end{array}
$$

An example of a solution to $VP - IS$ is given in Fig.1.1.

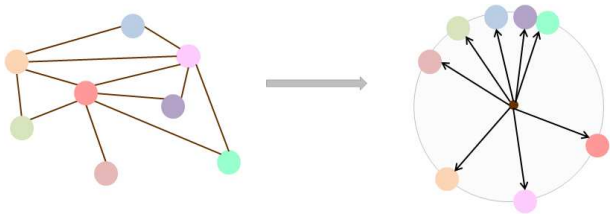

Figure 1.1: Example of a solution to $VP - IS$.

Given vectors $\mathbf{u_1}, \ldots, \mathbf{u_n}$ as a solution to $VP - IS$, we need to obtain a integer solution $y_1, \ldots, y_n$ such that $y_i = 1$, if $v_i \in I$, and $y_i = -1$, otherwise, for $1 \leq i \leq n$. This can be done using a *vector rounding* or a *hyperplane rounding* [24, 32, 42]. In *vector rounding* we take a random unit-length vector $\mathbf{r} \in \mathbf{R}^n$ and some positive constant c, and for each i we let $y_i = 1$ if $\mathbf{u}_i \cdot \mathbf{r} > c$, and $y_i = -1$, otherwise. An example of *vector rounding* is shown in Fig. 1.2.

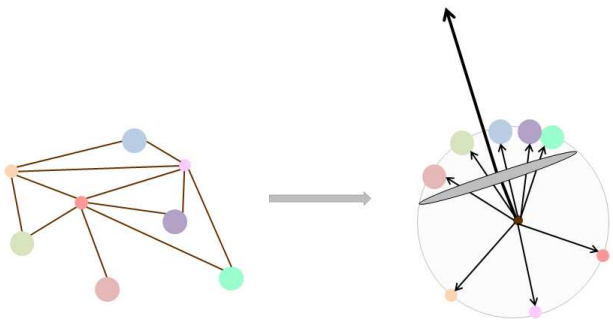

Figure 1.2: Example of *vector rounding*, all vertices above the hyperplane (marked by bigger circles) defined by the vector r are included in the independent set.

In *hyperplane rounding* we take $\log c$ random hyperplanes through the origin in $\mathbf{R^n}$, and for each i we let $y_i = 1$ if $\mathbf{u}_i \cdot \mathbf{r}_j > 0$ for all $1 \leq j \leq \log c$, where $\mathbf{r}_j$ is the normal of the hyperplane j, and $y_i = -1$, otherwise.

Semidefinite programming is a relatively new approach in optimization algorithms, but has turned out to be a very powerful technique that gives some of the best approximation algorithms for several optimization problems [32, 42].

Randomized Algorithms

A *randomized* algorithm is an algorithm whose input consists of two parts: an instance of a problem and a sequence of random bits; where the random bits determine the choices made by the algorithm and the output solution. It means that if we run the same randomized algorithm A on the same input several times, the behavior of A may differ between runs. The difference between runs is either in the found solution (if the algorithm does not guarantee to always find an optimal solution) or in the complexity of the algorithm (if the algorithm always finds an optimal solution), for example in the running time or in the working space used by the algorithm. Hence, the performance of a randomized algorithm (the output, the running time or the working space) is a *random variable*. For the optimization problems considered in this work there is no randomized algorithm that always produces an optimal solution, and so we deal with randomized algorithms whose output solutions are random variables. The aim is then to design such a randomized algorithm that produces a good approximation of an optimal solution with high probability. It is also important to note that the error probability of a randomized algorithm can be significantly reduced by running the algorithm on the same input a number of times with a new random sequence in each run.

Randomized algorithms can be more efficient in terms of time and space and simpler to design and implement than deterministic algorithms. For some problems (for example, primality testing) randomized algorithms are so efficient that they have become some of the most popular and widely used approaches in every day applications. Some randomized algorithms can be turned into deterministic algorithms using *derandomization* procedures, as for example in [47].

1.4 Computational Models

A *computational* or processing model defines how a problem instance (called *input*) is presented to an algorithm, how much time and space are available to the algorithm to make a decision on the final solution and whether the algorithm can revise its decisions. In this work we consider *off-line*, *on-line* and *streaming* models of computation.

1.4.1 Off-line Model

In an *off-line* model the algorithm is given the whole input data in the beginning. Usually an off-line model does not imply any limitations on working space and time, it is allowed to revise taken decisions at any step and it is not required to keep a feasible solution at any time during the execution. An *off-line* algorithm is an algorithm processing data according to an off-line model. An off-line algorithm A is characterized by its approximation ratio (as defined in Section 1.3.1).

1.4.2 On-line Model

In an *on-line* model the input data are presented to the algorithm piece-by-piece, the algorithm is forced to take decisions on-fly without a possibility to revise them later. On-line models represent situations when requests come in a sequence and each request needs to be handled as fast as possible, for example, in scheduling problems, booking and renting systems.

One of the important features of the on-line model is that an algorithm never knows what is going to come next, and so it has to take decisions based on the data that has arrived so far. Another important feature of on-line algorithms is that an on-line algorithm always maintains a feasible solution for the input data it has received so far. If the behavior of an on-line algorithm is predetermined, we call such an algorithm *deterministic* on-line algorithm. If an on-line algorithm uses a sequence of random bits to guide its behavior, then we call such an algorithm *randomized* on-line algorithm.

The order in which input data are presented to an on-line algorithm when analyzing its behavior is determined by an *adversary*. The goal of adversary is to present the input data in such a way that will force an on-line algorithm to take bad decisions most of the time. In this thesis we consider only *oblivious adversaries*. The *oblivious adversary* orders the input data in advance, before presenting them to the on-line algorithm.

The performance ratio of an on-line algorithm is evaluated via *competitive analysis*, the performance ratio is then called the *competitive ratio*. An on-line algorithm with the competitive ratio ρ is referred to as a ρ-competitive algorithm. The idea of competitive analysis is to compare the solution found by an on-line algorithm with a given adversary to a solution found by an optimal off-line algorithm with the same adversary. Let $E[ALG(I)]$ be the expected cost of the solution produced by an algorithm A on the input data I generated by the oblivious and $E[OPT(I)]$ be the expected cost of the solution produced by an optimal off-line algorithm on the same input data I, with the expectation taken over the random choices made by A. Then, the competitive ratio ρ of an on-line algorithm A with the oblivious adversary is defined as $\rho_A = \max_{\forall \mathcal{I}} \frac{OPT(I)}{E[ALG(I)]}$ (for a minimization problem the ratios are reversed).

1.4.3 Streaming Model

In the classical *streaming model* [36, 48, 49], the data is presented sequentially in form of a data stream, one item at a time, and the working space (the memory used by the algorithm), is significantly less than the size of the data stream. Streaming algorithms are similar to on-line algorithms in the sense that the input data is revealed one item at a time, but are restricted by space rather than the maintenance of a single feasible solution.

The main parameters of the streaming model are the size of the working space S (in bits), the number P of passes over the stream and the time T needed to process each data item in the stream. In the classical model the space S as well as the time T are required to be sublinear in N, the size of the data stream, preferably $\log^t(N)$ for some small constant t; the number P of passes is required to be a constant, preferably one or two.

The motivation for the streaming model comes from practical application related to the management of massive data sets such as, for example, real-time network traffic, on-line actions, telephone call records [23], where the input data are huge and arrive at a very high rate. It means that it is impossible to store the entire input and for each data item in the stream the decision has to be made quickly.

There are a number of results for algorithms in the classical streaming model, mostly on computing statistics for a stream, such as frequency moments [4] and L^p norms [38]. There have been attempts to study classical graph problems, where the graph (or the hypergraph) is presented as a stream. In this case, a data item in the stream often represents an edge or a vertex along with its neighbors. In the former case the data stream is called an *adjacency stream*; in the latter case the data stream is called an *incidence stream* [5]. The data items in the stream usually arrive in an arbitrary order. However, there has not been

much success with solving graph problems in the classical streaming model, an exception being the problem of counting triangles in a graph [5]. In fact, it was proven that for many graph properties it is impossible to determine whether a given graph has the property in single pass using $o(n)$ of space, where n is the number of vertices in the graph [22].

Difficulties with solving most problems on graphs (and hypergraphs) led to the extensions of the classical streaming model. One such extension is the *semi-streaming* model [48]. In the semi-streaming model the constraint on the working space is relaxed to $O(n \log^t n)$, where n is the number of vertices in the graph (hypergraph) and t is a constant. It means that the algorithm has enough memory to store the vertices with some limited information about them, but not necessarily the edges in the graph (hypergraph). Many structural graph problems can be solved or approximated within one or two passes in a semi-streaming model, for example determining whether a graph is connected [17] and finding a maximum matching [46].

1.5 Related Work

1.5.1 Inapproximability

Inapproximability results define how well a given NP-complete problem can be approximated if $P \neq NP$. These results are more on the negative side, because essentially they state the impossibility of getting a better approximation unless $P = NP$. However, inapproximability results are as important as approximation algorithms because for a given optimization problem corresponding to NP-complete decision problem Π and an approximation algorithm A for Π, they tell us if A is the best possible for Π if $P \neq NP$.

For our problems the following inapproximability results have been shown, where all the results assume that $P \neq NP$. The MAXIMUM INDEPENDENT SET problem is not approximable within $|V|^{1-\epsilon}$ for any $\epsilon > 0$ [35]. In bounded-degree hypergraphs the MAXIMUM INDEPENDENT SET problem is not approximable within $\Delta/2^{O\left(\sqrt{\log \Delta}\right)}$ [56]. As we mentioned before, the MINIMUM HITTING SET problem is equivalent to the MINIMUM SET COVER problem in the dual hypergraph. Thus, both the MINIMUM HITTING SET and the MINIMUM SET COVER problems are equivalent in terms of optimization, in particular inapproximability results hold for both problems. Namely, the MINIMUM HITTING SET and the MINIMUM SET COVER problems are not approximable within $c \log n$ for some c [50]. In hypergraphs of rank r the MINIMUM HITTING SET problem is not approximable within $r^{\frac{1}{19}}$ and the MINIMUM HITTING SET problem is not approximable

within $\ln r - O(\ln \ln r)$ [56]. As for the VERTEX SET problem, it is not approximable within 1.1666 [34], the same bound holds in case of bounded-degree hypergraphs if Δ is sufficiently large [16].

1.5.2 Approximations

Hypergraph problems tend to be more difficult to solve than the corresponding graph problems, with the MAXIMUM INDEPENDENT SET problem being a typical case.

The SIS problem in hypergraphs can be turned into an independent set problem in graphs by replacing each hyperedge with a clique (assuming that the hypergraph has no singletons which cannot belong to any independent set). The reason for considering the problem as a hypergraph problem is that the degrees in the hypergraph can be much smaller than in the corresponding clique graph. If the hypergraph is of degree Δ, then the corresponding clique graph contains no $\Delta + 1$-claw, where a k-claw is an induced star on k edges. The work of Hurkens and Schrijver [37] established that a natural local improvement method attains an approximation ratio of $k/2 + \epsilon$, for any fixed $\epsilon > 0$, on $k + 1$-claw free graphs, see also [26]. Another local search algorithm by Berman [9] approximates weighted IS in $(k+1)$-claw free graphs within a factor of $(k+1)/2$, which implies also a $\Delta/2$-approximation. A strong hardness result of $\Omega(\frac{\Delta}{\ln \Delta})$ is known for SIS, due to Hazan, Safra and Schwartz [33]. The focus of our study of SIS is to consider natural greedy methods and establish tight bounds on their approximation ratio.

For the IS problem, the best approximation ratio known in general graphs, in terms of the number n of vertices, is $O(n(\log \log n)^2/\log^3 n)$ [20], while in hypergraphs the ratio is only $O(n/\log n)$, which has a rather trivial argument [27]. In terms of the maximum degree Δ, a ratio of Δ is trivial for both graphs and hypergraphs, This ratio was slightly improved by greedy and local search algorithms, and the best improvement is achieved by SDP algorithms.

One of the most popular heuristics for IS, HS and SC is the *max-greedy* algorithm. For IS the max-greedy algorithm repeatedly deletes a vertex of maximum degree with all incident edges from a hypergraph, until the edge set of the hypergraph is empty. This algorithm approximates unweighted IS in graphs within a factor of $\frac{\Delta+2}{3}$ [31]. For SC the max-greedy algorithm repeatedly adds to the cover the set with the largest number of uncovered elements. In spite of its simplicity, it is in various ways also one of the most effective. Johnson [40] and Lovász [45] showed that it approximates the MINIMUM SET COVER problem within $H_n \leq \ln n + 1$ factor, which was shown by Feige [19] to be the best possible up to a lower order term. Generalizations to weights [15] and submodular

functions [60] also yield equivalent ratios, and under numerous variations on the objective function the max-greedy algorithm does still achieve the best known/possible approximation ratio, e.g. Sum Set Cover [21] and Entropy Set Cover [14]. Bazgan, Monnot, Paschos and Serrière [6] studied a *differential approximation ratio* of the max-greedy algorithm, which measures how many sets are *not* included in the cover. When viewed on the dual hypergraph, this is equivalent to studying the approximation ratio of the greedy set cover algorithm for IS. They proved that when modified with a post-processing phase, it has a approximation ratio of at most $\Delta/1.365$ and at least $(\Delta+1)/4$. Caro and Tuza [13] showed that the max-greedy algorithm applied to IS in r-uniform hypergraphs always finds a weak independent set of size at least $\Theta\left(n/\Delta^{\frac{1}{r-1}}\right)$. Thiele [55] extended their result to non-uniform hypergraphs and gave a lower bound on the size of an independent set found by a similar greedy algorithm as a complicated function of the number of edges of different sizes incident on each vertex in a hypergraph. Somewhat opposite to the *max-greedy* algorithm is the *min-greedy* algorithm, which for IS adds a vertex of minimum degree to the solution, deletes this vertex from the hypergraph, and if there are vertices with loops, deletes such vertices with all edges incident on them; this process is repeated until the vertex set is empty. This algorithm approximates unweighted IS in graphs within a factor of $\frac{\Delta+1}{2}$ [31], and has until now not been studied on hypergraphs.

Another popular algorithm design technique is local search. Local search gives the best approximations known of weighted and unweighted IS in bounded-degree graphs for small values of Δ, due to Berman [9] and Berman and Fujito [10]. Bazgan, Monnot, Paschos and Serrière [6] considered a simple 2-OPT local search algorithm to approximate IS in hypergraphs and proved a tight bound of $(\Delta+1)/2$. To our best knowledge there have been no results on local search algorithm in hypergraphs.

Yet another simple approach in approximation algorithm design is partitioning. This approach yields $\lceil(\Delta+1)/3\rceil$ approximation to the weighted IS in graphs, as shown in [26]. In spite of its simplicity, partitioning has not been used before to approximate IS in hypergraphs.

One of the most powerful approximation approach for hard optimization problems involves the use of semi-definite programming (SDP). It is responsible for the best ratio known for IS in graphs of $O(\Delta \log\log \Delta/\log \Delta)$ [26]. It is also involved in the complementary vertex cover problem [32], both in graphs and in hypergraphs. Yet, it has failed to yield much success for IS in hypergraphs, except for some special cases. One intuition may be that hyperedges result in significantly weaker constraints in the semi-definite relaxation than the graph edges. The special cases where it has been successful — 2-colorable r-uniform hypergraphs [26] and 3-uniform hypergraphs with a huge in-

dependence number [43] — have properties that result in strengthened constraints. The usefulness of SDP for general IS has remained open.

One of the earliest works on graph problems in streaming models considered finding a vertex with the largest number of neighbors and the maximum local degree [36], counting the number of triangles [5] and estimating common neighborhoods [12] using the classical streaming model. These results were the first ones to show that most graph problems are unlikely to be solved in the classical streaming model due to the restricted working space. The attention then turned to the semi-streaming model. There is a number of approximation algorithms for the maximum bipartite graph matching (weighted and unweighted cases), diameter and shortest paths [22] and min-cut [1]. Harder problems, such as IS, HS and SC are yet to be considered in the semi-streaming model.

1.6 Original Contribution of the Thesis

In this work we mainly focus on the MAXIMUM INDEPENDENT SET problem. Known approximation algorithms suggest several directions and research questions. A key question is to what extent approximation ratios for IS in graphs can be matched in hypergraphs. This can be asked in terms of different degree parameters, such as maximum degree, average degree, hyperdegree, etc. Given that graphs are 2-uniform hypergraphs and r-uniform hypergraphs have certain nice properties, the question is also how well we can handle uniform and non-uniform hypergraphs. Our main results are on the MAXIMUM INDEPENDENT SET problem.

Shrinkage reduction.
First, we describe a general technique that reduces the worst case analysis of certain algorithms on hypergraphs to their analysis on ordinary graphs. Given an approximation algorithm A, this technique, called *shrinkage reduction*, truncates a hypergraph H to a graph G such that an optimal solution on H is also an optimal solution in G, and A produces the same worst case approximate solution on H and G. This technique can be applied to a wide class of algorithms and problems on hypergraphs.

Greedy algorithms.
Using the shrinkage reduction technique we show that the max-greedy algorithm for IS has a approximation ratio of $(\Delta + 1)/2$ on hypergraphs, improving the bounds obtained by Bazgan et al. [6]. In addition, while their analysis required a post-processing phase, our bound applies to the greedy algorithm alone. This bound is slightly worse than the approximation ratio of the max-greedy algorithm on graphs. We also give the analysis of

the mix-greedy algorithm on hypergraphs, showing that it achieves only a ratio of $\Delta - 1$, significantly worse than on graphs. In r-uniform hypergraphs the approximation ratio of the greedy algorithms is improved to $\Theta\left(\Delta^{\frac{1}{r-1}}\right)$ (the max-degree greedy algorithm) and $1 + \frac{\Delta-1}{r}$ (the min-degree greedy algorithm).

Local search algorithms.
We extend several local search graph algorithms to the hypergraph case. In particular, using shrinkage reduction technique, we show that the t-OPT algorithm, which repeatedly tries to extend the current solution by deleting t elements while adding $t + 1$ elements, approximates IS on hypergraphs within $\Delta/2 + \epsilon(t)$, where $\lim_{t\to\infty} \epsilon(t) = 0$. We also extend the local search graph algorithm of Berman and Fürer [11] for unweighted IS and the local search graph algorithm of Berman [9] for weighted IS to the hypergraph case and obtain a $(\Delta + 1)/2$ approximation for weighted IS and $(\Delta + 3)/5 + \epsilon$ approximation for unweighted IS. All these local search algorithms achieve the same approximation ratio on hypergraphs as on graphs.

Partitioning algorithms.
We apply a simple partitioning approach [29, 30] to IS on hypergraphs and prove the bound of $\lceil(\Delta + 1)/3\rceil$ for WIS.

Semidefinite programming algorithms.
We derive the first $o(\Delta)$-approximation for IS in hypergraphs matching the $O(\Delta \log\log \Delta/\log \Delta)$-approximation for the special case of graphs. Our approach is to use an SDP formulation to sparsify the part of the instance formed by 2-edges, followed by a combinatorial algorithm on the resulting sparser instance. This is extended to obtain an identical bound in terms of the average degree $\bar{d}$ of an unweighted hypergraph. As part of the method, we also obtain a $k^{5/2-1/k}\bar{d}^{1-1/k+o(1)}$-approximation for hypergraphs with independence number at least n/k. We generalize the results to weighted hypergraphs. In that case, no non-trivial bound is possible in terms of the average degree alone, so we turn our attention to a weighted version. We give a $O(\bar{D} \log\log \bar{D}/\log \bar{D})$-approximation for IS. We apply two combinatorial algorithms to hypergraphs with few 2-edges. One is a greedy algorithm analyzed by Caro and Tuza [13] for the r-uniform case and Thiele [55] for the non-uniform case. The bound obtained in [55] is in general unwieldy, but we can show that it gives a good approximation when the number of 2-edges has been reduced. The other is a simple randomized algorithm analyzed by Shachnai and Srinivasan [54] achieving the same ratio as the greedy algorithm. This randomized algorithm can be derandomized as shown in [47].

The summary of the main results for IS in off-line model is given in Fig. 1.3.

	Previous results		Our results	
Algorithm	Graphs	Hypergraphs	Bound	Remarks
Max-degree greedy	$\frac{\Delta+2}{3}$ [31]	$\frac{\Delta}{1.365}$ [6]	$\frac{\Delta+1}{2}$	unweighted
Min-degree greedy	$\frac{\Delta+1}{2}$ [31]	–	$\Delta-1$	unweighted
Local search 2-OPT	$\frac{\Delta+1}{2}$ [26]	$\frac{\Delta+1}{2}$ [6]	$\frac{\Delta+1}{2}$	unweighted
Local search t-OPT	$\frac{\Delta}{2}+\epsilon(t)$ [26]	–	$\frac{\Delta}{2}+\epsilon(t)$	unweighted
Berman, Fujito [10]	$\frac{\Delta+3}{5}$ [10]	–	$\frac{\Delta+3}{5}$	unweighted
Berman [9]	$\frac{\Delta+1}{2}$ [9]	–	$\frac{\Delta+1}{2}$	weighted
Partitioning	$\lceil(\Delta+1)/3\rceil$ [26]	–	$\lceil(\Delta+1)/3\rceil$	weighted
SDP-based	$O\left(\frac{\bar{d}\log\log\bar{d}}{\log\bar{d}}\right)$ [26, 32]	–	$O\left(\frac{\bar{d}\log\log\bar{d}}{\log\bar{d}}\right)$	unweighted
SDP-based	$O\left(\frac{\bar{D}\log\log\bar{D}}{\log\bar{D}}\right)$ [26, 32]	–	$O\left(\frac{\bar{D}\log\log\bar{D}}{\log\bar{D}}\right)$	weighted

Figure 1.3: Summary of the results for IS in off-line model.

Streaming algorithms.

We consider algorithms in the semi-streaming model that use one pass over the stream and at most $O(r)$ time to process every edge in the stream. First, we describe two randomized algorithms: the permutation-based algorithm finds an independent set of expected weight $\Omega\left(\frac{w(V)}{\bar{D}^*}\right)$ in bounded-degree hypergraphs using $O(n\log n)$ bits of total working space; the partition-based algorithm approximates IS within $O\left(\frac{n}{c\log n}\right)$ for any constant $c>0$ in unbounded-degree hypergraphs using $O(n^c\log n)$ total working space. The working space of the first algorithm can be improved to $O(n\log r)$ bits. The second algorithm can be modified to be a variation of on-line randomized preemptive algorithm with the same approximation factor and the same size of the working memory. Next, we present two deterministic algorithms: the first one works for bounded-degree hypergraphs and finds an independent set of size $\Omega\left(\frac{pn}{\Delta^2}\right)$ using $O(n\log n)$ space, where p is the smallest edge size in a given hypergraph; the second algorithm works for unbounded-degree hypergraphs and approximates IS within $O\left(\frac{n}{c\log n}\right)$ for any constant $c>0$ using $O(n^{c+1})$ space.

Finally, we define an *on-line minimal space* streaming model, in which the working space is limited to two bits per vertex and the algorithm must maintain a feasible solution at all times. We prove lower bounds for randomized and deterministic algorithms in this model and present a randomized online minimal space streaming algorithm that finds an

independent set of expected weight $\Omega\left(\frac{w(V)}{\overline{\overline{D}}}\right)$ using two bits of memory per vertex , i.e. in total 2n bits of working memory.

The summary of the main results for IS in semi-streaming model is given in Fig. 1.4. All algorithms use one pass over the stream.

Algorithm	Bound	Space (in bits)	Time	Type	Remarks
RANDOMONFLYPERMUTE	$O\left(\overline{D^*}\right)$	$O(n \log n)$	$O(r)$	randomized	weighted
RANDOMPARTIALPERMUTE	$O\left(\overline{D^*}\right)$	$O(n \log r)$	$O(r)$	randomized	weighted
RANDOMPARTITIONAPRIORI	$O\left(\frac{n}{c \log n}\right)$	$O(n^c \log n)$	$O(r)$	randomized	unweighted
RANDOMPARTITIONONLINE	$O\left(\frac{n}{c \log n}\right)$	$O(n^c \log n)$	$O(r)$	randomized	unweighted
DETSPARSEHYPERGRAPH	$O\left(\frac{\Delta^2}{p}\right)$	$O(n \log n)$	$O(1)$	deterministic	unweighted
DETPARTITIONS	$O\left(\frac{n}{c \log n}\right)$	$O(n^{c+1})$	$O(r)$	deterministic	unweighted
RANDOMSELECT	$O\left(\overline{D}\right)$	$2n$	$O(r)$	randomized	weighted

Figure 1.4: Summary of the results for IS in semi-streaming model.

Strong independent sets.
For SIS we describe two greedy algorithms: one constructs an independent set by selecting vertices of minimum degree; the other selects vertices with the fewest neighbors. We show that both algorithms have a approximation ratio of Δ and that this bound is tight. In r-uniform hypergraphs the approximation ratio of both greedy algorithms is improved to $\Delta - \frac{\Delta - 1}{r}$.

1.7 Structure of the Thesis

The remainder of the thesis is organized as follows:

- *Chapter 2* introduces the shrinkage reduction technique and gives the analysis of max-greedy, min-degree, local search and partitioning algorithms. The greedy algorithms for SIS on uniform and non-uniform hypergraphs are presented there as well.

- *Chapter 3* describes SDP-based algorithms. First, it is explained how the number of 2-edges in hypergraphs can be reduced using SDP technique. Next, approximation ratio for the greedy and randomized algorithms are proved on hypergraphs with few 2-edges. Finally, the combination of SDP and combinatorial algorithms is outlined.
- *Chapter 4* presents semi-streaming algorithms. First, we analyze several randomized permutation and partition-based algorithms. We prove approximation ratios for bounded and unbounded hypergraphs. Next we describe deterministic algorithms for bounded and unbounded hypergraphs. Finally, we define an online streaming model and we prove lower bounds for online streaming randomized and deterministic algorithms.
- *Chapter 5* summarizes the results and outlines open problems and possible directions for further research.

Chapter 2

Combinatorial Algorithms

We describe three different approaches to weighted and unweighted IS in bounded-degree hypergraphs: local search, greedy and partitioning. First, we introduce a general reduction technique for the worst case analysis of approximation algorithms on hypergraphs, and then apply it to local search and greedy algorithms. Next, we present a partitioning algorithm for IS on bounded-degree hypergraphs. Finally, we discuss greedy algorithms for SIS on uniform and non-uniform hypergraphs.

2.1 Shrinkage Reduction

Shrinkage reduction is a general technique that reduces the worst case analysis of algorithms on hypergraphs to their analysis on graphs. It is based on a *shrinkage* hypergraph, or *shrinkage* for short.

Definition 2.1.1 *A hypergraph H' is a* shrinkage *of H if $V(H') = V(H)$, $|E(H')| = |E(H)|$ and for any edge $e \in E(H)$ there exist an edge $e' \in E(H')$ such that $e' \subseteq e$. In other words, the edges of H might be truncated in H' into sets of smaller size (and at least 2).*

Shrinkage reduction works for *hereditary* optimization problems. Given a hypergraph H and any optimization problem Π, let S_H be a feasible solution to Π in H.

Definition 2.1.2 *An optimization problem on hypergraphs is* hereditary*, if for any shrinkage H' of a hypergraph H it satisfies $\mathcal{S}_{H'} \subseteq \mathcal{S}_H$.*

Many problems on hypergraphs are hereditary, including the MINIMUM HITTING SET, the MAXIMUM INDEPENDENT SET, the Minimum Coloring and the Shortest HyperPath[1]. An example of non-hereditary problem is the Longest HyperPath. Given a hereditary problem, the essence of shrinkage reduction is the following.

Proposition 2.1.3 *Let A be an approximation algorithm for a hereditary problem. Suppose we can construct a shrinkage graph G of a hypergraph H such that an optimal solution in H is also an optimal solution in G and A produces the same worst approximate solution on H and G, then the approximation ratio of A on hypergraphs is no worse than on graphs.*

Note, that Proposition 2.1.3 applies also to non-deterministic (and randomized) approximation algorithms.

It is not easy to give a general rule on how to construct a shrinkage for an arbitrary approximation algorithm. In the following sections we describe reductions for the greedy set cover and local search algorithms for weighted and unweighted IS in bounded-degree hypergraphs. The comparison of the MAXDEGREEGREEDY and the MINDEGREEGREEDY algorithms, described in Sections 2.3.1, suggests the following conjecture.

Conjecture 2.1.4 *The shrinkage reduction technique is applicable only to algorithms that do not alter edge sizes during the execution.*

2.2 Local Search

The idea of the local search approach is to start with a (arbitrary) solution and continually replace it by a better solution found in its neighborhood while possible. We need formal definitions to determine what a "better solution" and a "neighborhood" mean.

A *neighborhood function* Γ maps a solution $S \in \mathcal{S}_I$ into a set of solutions $\Gamma_I(S) \subseteq \mathcal{S}_I$, called the *neighborhood* of S. A feasible solution $\tilde{S}$ is *locally optimal w.r.t.* Γ, or Γ-*optimal* for short, if it satisfies $w(\tilde{S}) \leq w(S)$ ($w(\tilde{S}) \geq w(S)$) for all $S \in \Gamma_I(S)$ for a minimization (maximization) problem. A feasible solution S^* is *globally optimal*, or *optimal* for short, if it satisfies $w(S^*) \leq w(S)$ ($w(S^*) \geq w(S)$) for all $S \in \mathcal{S}_I$ for a minimization (maximization) problem. To specify more precisely the neighborhood functions used in our local search algorithms, we need the following definition.

[1] A *(hyper)path* in a hypergraph is a sequence of edges $e_1, e_2, \ldots, e_p$ such that $e_i \cap e_{i+1} \neq \emptyset$ for any $1 \leq i \leq p-1$ and $e_i \cap e_j = \emptyset$ for any i, j such that $|i-j| > 1$.

Definition 2.2.1 *A neighborhood function Γ is said to be* edge-monotone *for a hereditary problem on hypergraphs if for any shrinkage H' of a given hypergraph H and any solution $S \in \mathcal{S}_{H'}$ the neighborhood of S satisfies $\Gamma_{H'}(S) \subseteq \Gamma_H(S)$.*

In other words, edge-monotonicity means that edge reduction can only decrease the neighborhood size.

A Γ-optimal algorithm is a local search algorithm that given an instance I, starts with a (arbitrary) solution S and repeatedly replaces it by a better solution found in $\Gamma_I(S)$ until S is Γ-optimal. The approximation ratio $\varrho_{\Gamma,I}$ of a Γ-optimal algorithm on a instance I is the maximum ratio between the weights of Γ-optimal and optimal solutions over all Γ-optimal solutions on I, i.e. $\varrho_{\Gamma,I} = \max\limits_{\forall \tilde{S} \in \mathcal{S}_I} \frac{w(\tilde{S})}{w(S^*)}$ $\left(\varrho_{\Gamma,I} = \max\limits_{\forall \tilde{S} \in \mathcal{S}_I} \frac{w(S^*)}{w(\tilde{S})} \right)$ for a minimization (maximization) problem. The approximation ratio $\rho_{\Gamma,\mathcal{I}}$ of a Γ-optimal algorithm is the worst approximation ratio over all instances I in the class of instances $\mathcal{I}$.

In the following theorem we show that if a neighborhood function Γ is edge-monotone, then for the Minimum Cover problem the analysis of a Γ-optimal algorithm on hypergraph reduces to the analysis of this algorithm on graphs. The reduction is based on the construction of a shrinkage graph with special properties. Note, that a shrinkage graph is needed only for the analysis, but not for the Γ-optimal algorithm itself.

Theorem 2.2.2 *Given an edge-monotone neighborhood function Γ and a hypergraph H with an optimal cover S^* and a Γ-optimal cover $\tilde{S}$, there exists a shrinkage graph G of H on which S^* and $\tilde{S}$ are also optimal and Γ-optimal covers, respectively.*

Proof: Given H, S^* and $\tilde{S}$, we construct a shrinkage G as follows. From each edge e in $E(H)$, we arbitrarily pick vertices u and v such that $\{u, v\} \cap \tilde{S} \neq \emptyset$ and $\{u, v\} \cap S^* \neq \emptyset$, and add (u, v) to $E(G)$.

Any edge in $E(G)$ contains at least one vertex from $\tilde{S}$ and at least one vertex from S^*, and so $\tilde{S}$ and S^* are covers in G, i.e. $\tilde{S}, S^* \in \mathcal{S}_G$. Since G is a shrinkage of H and the Minimum Cover problem is hereditary, $\mathcal{S}_G \subseteq \mathcal{S}_H$ by definition. For all $S \in \mathcal{S}_H$ we have $w(S^*) \leq w(S)$, and so $w(S^*) \leq w(S)$ for all $S \in \mathcal{S}_G$. Thus, S^* is an optimal cover in G. The local optimality of $\tilde{S}$ in G follows by the same argument and the fact that Γ is edge-monotone. ■

Corollary 2.2.3 *If a neighborhood function Γ is edge-monotone for IS, then $\rho_{\Gamma,\mathcal{H}} \leq \rho_{\Gamma,\mathcal{G}}$.*

Proof: Given a hypergraph $H(V, E)$, the vertices not contained in a weak independent set I form a vertex cover S in H, i.e. $I = V \backslash S$. Given an edge-monotone neighborhood

function Γ for IS, we define a new neighborhood function $\Gamma'(S) = \{S' : V \backslash S' \in \Gamma(V \backslash S)\}$. Note, that $\Gamma'(S)$ is edge-monotone for the MINIMUM HITTING SET problem. Moreover, if I^* and $\tilde{I}$ are optimal and Γ-optimal weak independent sets in H, then $S^* = V \backslash \tilde{I}^*$ and $\tilde{S} = V \backslash \tilde{I}$ are optimal and Γ-optimal covers in H, respectively. The claim then follows from Theorem 2.2.2. ■

The simplest local search algorithm for IS is t-Opt, which repeatedly tries to extend the current solution by deleting t elements while adding $t + 1$ elements. It is easy to verify that the corresponding neighborhood function $\Gamma(S) = \{S' \in \mathcal{S}_H : |S \oplus S'| \leq t\}$ defined on $\mathcal{S}_H$ is edge-monotone (where $\oplus$ is the symmetric difference). Then, the following two theorems are straightforward from Corollary 2.2.3 and the results of Hurkens and Schrijver on graphs [37].

Theorem 2.2.4 *t-Opt approximates IS within $\Delta/2 + \epsilon$, where $\lim_{t \to \infty} \epsilon(t) = 0$.*

Theorem 2.2.5 *2-Opt approximates IS within $(\Delta + 1)/2$.*

Theorem 2.2.6 *For every $\epsilon > 0$, IS can be approximated within $(\Delta + 3)/5 + \epsilon$ for even Δ and within $(\Delta + 3.25)/5 + \epsilon$ for odd Δ.*

Proof: We extend the algorithm SIC(G, Δ, k) of Berman and Fürer [11] for IS in bounded degree graphs to the hypergraph case. Let us call this algorithm HSIC. Given a hypergraph $H(V, E)$ and a weak independent set I in H, let B_I equal $V - I$ if the maximum degree of H is three, and otherwise equal the set of vertices that have at least two incident edges with vertices in I. Let $Comp(I)$ be the subhypergraph induced by B_I. The formal description of the algorithm is given in Fig. 2.1.

```
ALGORITHM HSIC(H, Δ, k)
  INPUT: a hypergraph H(V,E) with maximum degree Δ and k > 0

  If Δ ≤ 2 then compute a maximum independent set I exactly
  Else
     Let I be any maximal weak independent set
     Repeat
        Do all possible local improvements of size O(k log n)
        If Δ = 3 then l = 1 else l = 2
        Recursively apply HSIC(Comp(I), Δ − l, k)
        and select the resulting weak independent set if it is bigger
     Until I has no improvements
  Output I
```

Figure 2.1: The algorithm HSIC

There are two neighborhood functions in HSIC. The first function which maps a solution I to a set of all possible local improvements of size $O(k \log n)$, is t-optimal with $t = O(k \log n)$, and therefore edge-monotone. The second function, which maps a solution I to a set of weak independent sets in $Comp_H(I)$, is edge-monotone, because shrinking H to H' reduces the degree of some vertices, implying $B_I(H') \subseteq B_I(H)$. Consequently, a weak independent set in $Comp_{H'}(I)$ is also a weak independent set in $Comp_H(I)$. Thus, both neighborhood functions are edge-monotone and the approximation ratio of HSIC is no worse than the approximation ratio of SIC(G, Δ, k) by Corollary 2.2.3. ■

Theorem 2.2.7 *Weighted IS is approximable within $(\Delta+1)/2$ on hypergraphs of a constant rank r.*

Proof: We extend the algorithm SQUAREIMP of Berman [9] for weighted IS in bounded degree graphs to the hypergraph case. Let us call this algorithm HSQUAREIMP. Let I be a weak independent set in H. We say that (B, C) is an *improvement* of I, if there is a vertex $v \in I$ such that $B \subseteq N(v) \cap (V \backslash I)$, $C \subseteq N(B) \cap I$, $(I \backslash C) \cup B$ is a weak independent set and $w^2((I \backslash C) \cup B) > w^2(I)$. The formal description of the algorithm is given in Fig. 2.2.

ALGORITHM HSQUAREIMP (H)
INPUT: a hypergraph $H(V, E)$

$I \leftarrow \emptyset$
While there exist an improvement (B, C) of I
  $I \leftarrow (I \setminus C) \cup B$
Output I

Figure 2.2: The algorithm HSQUAREIMP

The neighborhood function in HSQUAREIMP is edge-monotone. Shrinking H to H' reduces the degree of some vertices and so every improvement B, C of I in H' is also an improvement of I in H. Hence, the approximation ratio of HSQUAREIMP is no worse than the approximation ratio of SQUAREIMP by Corollary 2.2.3.

Note, that finding an improvement (B, C) takes $O(n2^{\Delta^2(r-2)(r-1)})$ steps. Namely, in the worst case we check every vertex $v \in I$, every possible subset $B \subseteq N(v) \cap (V \backslash I)$ and every possible subset $C \subseteq N(B) \cap I$ to see whether $(I \backslash C) \cup B$ is a weak independent set and $w^2((I \backslash C) \cup B) > w^2(I)$. Since $|N(v) \cap (V \backslash I)| \leq \Delta(r-2)$, there are at most $2^{\Delta(r-2)}$ possible B-sets. Similarly, since $|N(B) \cap I| \leq \Delta(r-2)(\Delta(r-1)-1)$, there

are at most $2^{\Delta(r-2)(\Delta(r-1)-1)}$ possible C-sets. In total, we consider at most $2^{\Delta^2(r-2)(r-1)}$ possible pairs (B, C) for every vertex $v \in I$ until an improvement is found. ■

2.3 Greedy

2.3.1 Weak Independent Set

The idea of the greedy approach is to construct a solution by repeatedly selecting the best candidate on each iteration. There are two variations, called MAXDEGREEGREEDY and MINDEGREEGREEDY, depending on whether we greedily reject or add vertices.

The MAXDEGREEGREEDY Algorithm

The MAXDEGREEGREEDY algorithm constructs a cover S by adding a vertex of maximum degree, deleting it with all incident edges from the hypergraph, and iterating until the edge set is empty. It then outputs the remaining vertices as a weak independent set I. The formal description of the algorithm is given in Fig. 2.3.

ALGORITHM MAXDEGREEGREEDY (H)
INPUT: a hypergraph $H(V, E)$

$S = \emptyset$
While the edge set of H is not empty
  Add a vertex v of maximum degree to S
  Delete v with all incident edges on v from H
Output $I = V \setminus S$

Figure 2.3: The algorithm MAXDEGREEGREEDY

Given a hypergraph $H(V, E)$, let S^* be a minimum cover. Then, the approximation ratio of MAXDEGREEGREEDY is:

$$\rho = \max_{\forall H} \frac{n - |S^*|}{n - |S|} \,. \tag{2.1}$$

The analysis has two parts. First we prove that the worst case for MAXDEGREEGREEDY occurs on graphs. Namely, we describe how to reduce any hypergraph to a graph (actually, a multigraph) G for which MAXDEGREEGREEDY has no better approximation ratio. We then show that the bound actually holds for (multi)graphs.

Lemma 2.3.1 *Given a hypergraph H with a minimum cover S^*, there exists a shrinkage G of H on which S^* is still a cover and where* MAXDEGREEGREEDY *constructs the same cover for G as for H.*

Proof: The proof is by induction on s, the number of iterations of MAXDEGREEGREEDY. For the base case, $s = 0$, the claim clearly holds for the unchanged empty graph.

Suppose now that the claim holds for all hypergraphs for which MAXDEGREEGREEDY selects $s - 1 \geq 0$ vertices. Let u_1 be the first vertex chosen by MAXDEGREEGREEDY, $E(u_1)$ be the set of incident edges, and H_1 be the remaining hypergraph after deleting u_1 with all incident edges. Based on $E(u_1)$, we form a set $E'(u_1)$ of ordinary edges as follows. If u_1 is contained in both S and S^*, then for each edge e in $E(u_1)$ we pick an arbitrary vertex v from e and add (u_1, v) to $E'(u_1)$. If u_1 is only in S and not in S^*, then for each edge e in $E(u_1)$ we pick an arbitrary vertex u from e that is contained in S^* and add (u_1, u) to $E'(u_1)$; such a vertex u must exist, since e is covered by S^*. This completes the construction of $E'(u_1)$.

By the induction hypothesis, there is a shrinkage G_1 of H_1 with a greedy cover of $S \setminus \{u_1\}$ and G_1 is still covered by S^*. We now form the multigraph G on the same vertex set as H with the edge set $E'(u_1) \cup E(G_1)$, and claim that it satisfies the statement of the lemma. Since G_1 is covered by S^* and all edges of $E'(u_1)$ are also covered by vertices of S^*, S^* covers all edges of G. The edge shrinkage only decreases the degrees of vertices, but does not affect the degree of u_1. Therefore, u_1 remains the first vertex chosen by MAXDEGREEGREEDY and, by induction, the vertices chosen from G_1 are the same as those chosen from H_1. Hence, MAXDEGREEGREEDY outputs the same solution on G as on H, completing the lemma. ■

From Lemma 2.3.1 it follows immediately that the approximation ratio of MAXDEGREEGREEDY on hypergraphs is no worse than on graphs. Sakai, Togasaki, and Yamazaki [53] obtained a lower bound on the size of weighted independent set I produced by a weighted generalization of MAXDEGREEGREEDY on graphs. In unweighted case this bound reduces to a Caro-Wei improvement of the Turan bound on graphs $|I| \geq \sum\limits_{v \in V} \frac{1}{d(v)+1}$. For completeness we give below the proof from [53] adapted for unweighted multigraphs.

Lemma 2.3.2 *Given a (multi)graph $G = (V, E)$,* MAXDEGREEGREEDY *finds an independent set of size at least $\sum\limits_{v \in V} \frac{1}{d(v)+1}$.*

Proof: Let s be the number of iterations of MAXDEGREEGREEDY on G. For $0 \leq i \leq s$, let G_i be the remaining (multi)graph after i iterations. We denote by $d_{G_i}(v)$ and $N_{G_i}(v)$ the degree and the neighborhood of a vertex $v \in V(G_i)$. Note, that since G_i is a multigraph,

$N_{G_i}(v)$ is a multiset and $d_{G_i}(v) = |N_{G_i}(v)|$. For a vertex $u \in N_{G_i}(v)$ let $e_{G_i}(v,u)$ be the number of multiple edges (v,u) in G_i. Let $f(G_i) = \sum_{u \in V(G_i)} \frac{1}{d_{G_i}(u)+1}$ be a potential function on a graph G_i. We show that $f(G_{i+1}) \geq f(G_i)$ for $0 \leq i \leq s$. Consequently, $f(G_s) \geq f(G_0)$, where G_0 is the original graph G and G_s is a collection of isolated vertices. Then, MAXDEGREEGREEDY outputs a weak independent set of size at least:

$$|I| = f(G_s) \geq f(G) = \sum_{u \in V(G)} \frac{1}{d_G(u)+1} \,. \tag{2.2}$$

Let v_i be the vertex chosen by MAXDEGREEGREEDY on the iteration i. Then,

$$\begin{aligned} f(G_{i+1}) &= \sum_{u \in V(G_{i+1})} \frac{1}{d_{G_{i+1}}(u)+1} \\ &= \sum_{u \in V(G_i)} \frac{1}{d_{G_i}(u)+1} - \frac{1}{d_{G_i}(v_i)+1} + \sum_{u \in V(G_i) \cap N_{G_i}(v_i)} \left(\frac{1}{d_{G_{i+1}}(u)+1} - \frac{1}{d_{G_i}(u)+1} \right) \\ &= f(G_i) - \frac{1}{d_{G_i}(v_i)+1} + Y \,, \end{aligned} \tag{2.3}$$

where

$$\begin{aligned} Y &= \sum_{u \in V(G_i) \cap N_{G_i}(v_i)} \left(\frac{1}{d_{G_{i+1}}(u)+1} - \frac{1}{d_{G_i}(u)+1} \right) \\ &= \sum_{u \in N_{G_i}(v_i)} \frac{1}{e_{G_i}(v,u)} \left(\frac{1}{d_{G_i}(u) - e_{G_i}(v,u) + 1} - \frac{1}{d_{G_i}(u)+1} \right) && (2.4) \\ &\geq |N_{G_i}(v_i)| \min_{u \in N_{G_i}(v_i)} \frac{1}{(d_{G_i}(u) - e_{G_i}(v,u) + 1)\,(d_{G_i}(u)+1)} \\ &\geq |N_{G_i}(v_i)| \min_{u \in N_{G_i}(v_i)} \frac{1}{d_{G_i}(u)\,(d_{G_i}(u)+1)} \\ &\geq \frac{|N_{G_i}(v_i)|}{d_{G_i}(v_i)\,(d_{G_i}(v_i)+1)} && (2.5) \\ &= \frac{1}{d_{G_i}(v_i)+1} && (2.6) \end{aligned}$$

and (2.4) follows from $d_{G_{i+1}}(u) = d_{G_i}(u) - e_{G_i}(v,u)$, which is minimized when $e_{G_i}(v,u) = 1$; (2.5) holds by the greedy rule $d_{G_i}(v_i) \geq \max_{u \in G_i} d_{G_i}(u)$. It follows from (2.3) and (2.6) that $f(G_{i+1}) \geq f(G_i)$ completing the proof. ■

Lemma 2.3.3 *The approximation ratio of* MAXDEGREEGREEDY *on (multi)graphs is at most* $\frac{\Delta+1}{2}$.

Proof: We show that MAXDEGREEGREEDY attains its worst approximation ratio on regular graphs. First we refine $\overline{d}$ as follows: let $k \in [0,1]$ be the value so that kn vertices are of degree Δ and the remaining $(1-k)n$ vertices have average degree $d^{'} \leq \Delta - 1$. Then,

$$\overline{d} = k\Delta + (1-k)d^{'} \ . \tag{2.7}$$

Since each vertex can cover at most Δ of the m edges of the graph, any optimal cover S^* is of size at least

$$|S^*| \geq \frac{m}{\Delta} = \frac{\overline{d}n}{2\Delta} = \frac{n\left(k\Delta + (1-k)d^{'}\right)}{2\Delta} \ . \tag{2.8}$$

We also rewrite (2.2) as

$$|I| \geq \sum_{v \in V} \frac{1}{d(v)+1} \geq \frac{kn}{\Delta+1} + \sum_{v \in V \ : \ d(v)<\Delta} \frac{1}{d(v)+1} \ . \tag{2.9}$$

Since $f(d) = \frac{1}{d+1}$ is a convex function, we can apply Jensen's inequality[2] to (2.9):

$$|I| \geq \frac{kn}{\Delta+1} + \frac{(1-k)n}{d^{'}+1} \ . \tag{2.10}$$

Note, that the same result follows from the harmonic-arithmetic mean inequality applied to (2.9). Combining (2.1), (2.8) and (2.10) we obtain an upper bound on the approximation ratio of MAXDEGREEGREEDY:

$$\begin{aligned} \rho &= \max_{\forall H} \frac{n-|S^*|}{n-|S|} = \max_{\forall H} \frac{n-|S^*|}{|I|} \leq \frac{2\Delta - k\Delta - (1-k)d^{'}}{2\Delta\left(\frac{k}{\Delta+1} + \frac{1-k}{d^{'}+1}\right)} \\ &= \frac{(\Delta+1)(d^{'}+1)}{2\Delta}\left(1 + \frac{\Delta - d^{'} - 1}{\Delta + 1 - k(\Delta - d^{'})}\right), \end{aligned} \tag{2.11}$$

where (2.11) is clearly maximized when $k = 1$, yielding a bound of $\frac{\Delta+1}{2}$. ■

Theorem 2.3.4 *The approximation ratio of* MAXDEGREEGREEDY *on hypergraphs is* $\frac{\Delta+1}{2}$.

Proof:

The upper bound is straightforward from Lemmas 2.3.1 and 2.3.3, because G and H have the same number of edges and the same maximum degree. The edge reduction in $E(H)$ might create multiple edges in $E(G)$, but they do not affect the approximation ratio of MAXDEGREEGREEDY.

[2] Jensen's inequality for a convex function f: $\sum_{i=1}^{n} f(x_i) \geq nf\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)$

For the lower bound, consider the graph $G_{\Delta+1,\Delta+1}$, formed by the complete bipartite graph $K_{\Delta+1,\Delta+1}$ missing a single perfect matching. MAXDEGREEGREEDY may remove vertices alternately from each side, until two vertices remain as a maximal weak independent set. The optimal solution consists of one of the bipartitions, of size $\Delta+1$. By taking independent copies, this can be extended to hold for arbitrarily large instances. ■

Theorem 2.3.5 *The approximation ratio of* MAXDEGREEGREEDY *in r-uniform hypergraphs is at most* $\left(\frac{r-1}{r}\right)\prod_{i=1}^{\Delta}(1+\frac{1}{i(r-1)}) = \Theta\left(\Delta^{\frac{1}{r-1}}\right)$.

Proof: We assume that $r \geq 3$ since 2-uniform hypergraphs are ordinary graphs and the analysis of the greedy algorithm on graphs is given in Lemma 2.3.3.

Caro and Tuza [13] showed that MAXDEGREEGREEDY finds an independent set I of size at least:

$$|I| \geq \sum_{v\in V}\prod_{i=1}^{d(v)}\left(1-\frac{1}{i(r-1)+1}\right) = \sum_{v\in V}\prod_{i=1}^{d(v)}\frac{i}{i+\frac{1}{r-1}} = \sum_{v\in V}\frac{d(v)!}{\left(d(v)+\frac{1}{r-1}\right)^{\underline{d(v)}}}, \quad (2.12)$$

where $x^{\underline{y}} = x(x-1)\ldots(x-y+1)$. The function $f(d) = \frac{d!}{\left(d+\frac{1}{r-1}\right)^{\underline{d}}} = \binom{d+\frac{1}{r-1}}{d}^{-1}$ is convex because its first derivative is monotonically increasing on the interval $[1,\Delta]$. Therefore, we can apply Jensen's inequality to (2.12):

$$|I| \geq n\binom{\overline{d}+\frac{1}{r-1}}{\overline{d}}^{-1}.$$

A maximum independent set in a hypergraph on n vertices contains $n-|S|$ vertices, where S is a minimum hitting set. Since there are at most $\overline{d}n/r$ edges in a r-uniform hypergraph and each vertex from S covers at most Δ edges, there are at least $\frac{\overline{d}n}{r\Delta}$ vertices in S. The approximation ratio of MAXDEGREEGREEDY is then at most

$$\rho \leq \frac{n-\frac{\overline{d}n}{r\Delta}}{n\binom{\overline{d}+\frac{1}{r-1}}{\overline{d}}^{-1}} = \left(1-\frac{\overline{d}}{r\Delta}\right)\binom{\overline{d}+\frac{1}{r-1}}{\overline{d}} \leq \left(1-\frac{1}{r}\right)\binom{\Delta+\frac{1}{r-1}}{\Delta}$$

since $f\left(\overline{d}\right) = \left(1-\frac{\overline{d}}{r\Delta}\right)\binom{\overline{d}+\frac{1}{r-1}}{\overline{d}}$ is maximized when $\overline{d}=\Delta$. ■

Theorem 2.3.6 *The approximation ratio of* MAXDEGREEGREEDY *in r-uniform hypergraphs is at least* $\left(\frac{r-1}{r}\right)\prod_{i=1}^{\Delta}\left(1+\frac{1}{i(r-1)}\right) = \Theta\left(\Delta^{\frac{1}{r-1}}\right)$.

Proof: Let n be a multiple of $\prod_{j=1}^{\Delta}(j(r-1)+1)$ and for any $1 \leq i \leq \Delta$ let $x_i = \frac{n}{i(r-1)} \prod_{j=i}^{\Delta} \frac{j(r-1)}{j(r-1)+1}$. We define a chain of regular r-uniform hypergraphs $H^{(1)} \subset H^{(2)} \ldots \subset H^{(\Delta-1)} \subset H^{(\Delta)}$, where our hypergraph $H(V,E) = H^{(\Delta)}$.

The first hypergraph $H^{(1)}$ is defined on rx_i vertices and consists of x_1 disjoint edges, i.e $V^{(1)} = \{v_1^{(1)}, \cdots, v_{rx_1}^{(1)}\}$ and $E^{(1)} = \{e_1^{(1)}, \cdots e_{x_1}^{(1)}\}$, where $e_j^{(1)} = \{v_{(j-1)r+1}^{(1)} \cdots v_{jr}^{(1)}\}$ for any $j \in [1, x_1]$. Let $T^{(1)} = E^{(1)}$ and $U^1 = \{v_r^{(1)}, v_{2r}^{(1)}, \cdots, v_{rx_1}^{(1)}\}$. It is easy to see that $H^{(1)}$ is a 1-regular and r-uniform.

For $2 \leq i \leq \Delta$, let $y_i = ix_i$. The hypergraph $H^{(i)}$ consists of $H^{(i-1)}$, an additional set of vertices $U^{(i)} = \{u_1^{(i)}, \ldots, u_{x_i}^{(i)}\}$ and an additional set of edges $T^{(i)} = \{t_1^{(i)}, \ldots, t_{y_i}^{(i)}\}$, connecting $U^{(i)}$ to $H^{(i-1)}$, i.e $V^{(i)} = V^{(i-1)} \cup U^{(i)}$ and $E^{(i)} = E^{(i-1)} \cup T^{(i)}$. The first y_{i-1} edges in $T^{(i)}$ are the copies of the edges in $T^{(i-1)}$ with the last vertex in each copy replaced by a vertex from $U^{(i)}$, i.e $t_j^{(i)} = t_j^{(i-1)} \backslash \{v_{jr}^{(i-1)}\} \cup \{u_{\lceil j/i \rceil}^{(i)}\}$, for each $j \in [1, y_{i-1}]$. Let the replaced vertices form the set $W^{(i)} = \{v_r^{(i-1)}, v_{2r}^{(i-1)}, \ldots, v_{y_{i-1}r}^{(i-1)}\} = w_1, w_2, \ldots, w_{y_{i-1}r}$. The last $y_i - y_{i-1}$ edges in $T^{(i)}$ are formed by the vertices in $U^{(i)}$ and $W^{(i)}$: $t_j^{(i)} = u_{\lceil j/i \rceil}^{(i)} \cup \{w_j^{(i)}, w_{j+(y_i - y_{i-1})}^{(i)}, \ldots, w_{j+(r-2)(y_i - y_{i-1})}^{(i)}\}$, for each $j \in [y_{i-1}+1; y_i]$. The hypergraph $H^{(i)}$ is i-regular by induction: each vertex in $U^{(i)}$ is a root of a hyperstar with i edges, while each vertex in $H^{(i)} \backslash U^{(i)}$ has $i-1$ incident edges in $E^{(i-1)}$ and one incident edge in $T^{(i)}$. Then, the hypergraph $H(V,E) = H^{(\Delta)}$ is Δ-regular and r-uniform.

We show now that MAXDEGREEGREEDY finds a cover S of size $\sum_{i=1}^{\Delta} x_i$ in H, while an optimal cover S^* in H is of size $|E|/\Delta$. Thus, the ratio between the sizes of the optimal independent set $I^* = V \backslash S^*$ and the greedy independent set $I = V \backslash S$ is the one defined in (2.12). Since the hypergraph $H = H^{(\Delta)}$ is Δ-regular, MAXDEGREEGREEDY might start by selecting all vertices in $U^{(\Delta)}$ and deleting all edges in $T^{(\Delta)}$. The remaining hypergraph is $H^{(\Delta-1)}$ and MAXDEGREEGREEDY might continue by selecting all vertices in $U^{(\Delta-1)}$ and deleting $T^{(\Delta-1)}$. Inductively, MAXDEGREEGREEDY might select all vertices in $U^{(\Delta)} \cup \ldots \cup U^{(1)}$ as a minimal cover S of size $\sum_{i=1}^{\Delta} ix_i$ and output the remaining $(r-1)x_1$ vertices as a maximal independent set I.

Let $z_1 = x_1$ and $z_i = x_i - z_{i-1}/i$, for any $2 \leq i \leq \Delta$. An optimal cover S^* includes all vertices from $U^{(1)}$ and the last z_i vertices from each $U^{(i)}$ for $2 \leq i \leq \Delta$ (note that by definition x_i is multiple of any $j \in [i+1, \Delta]$, so z_i is also a multiple of any $j \in [i+1, \Delta]$). The vertices in $U^{(1)}$ cover all edges in $T^{(1)}$, and the first x_1 edges in every $T^{(i)}$ for $2 \leq i \leq \Delta$. By induction, the last z_i vertices in $U^{(i)}$ cover the remaining edges in $T^{(i)}$ and z_i edges in every $T^{(j)}$, where $j \in [i+1, \Delta]$. Consequently, all edges in H are covered by the vertices from S^*. Since H is Δ-regular and no two vertices from S^*

appear in the same edge (by construction of H), S^* is an optimal cover of size $|E|/\Delta$. Then, an optimal independent set is of size $|I^*| = n - |E|/\Delta = n(r-1)/r$, because $|E| = n\Delta/r$ in a Δ-regular r-uniform hypergraph. Finally, the ratio in (2.12) can be simplified to $\frac{n}{rx_1} = \frac{n(r-1)}{r}\frac{1}{(r-1)x_1}$, which is exactly $|I^*|/|I|$. ■

The MINDEGREEGREEDY Algorithm

The MINDEGREEGREEDY algorithm iteratively adds a vertex of minimum degree into the weak independent set and deletes it from the hypergraph. If the vertex deletion results in loops (edges containing only one vertex), then the algorithm also deletes the vertices with loops along with all edges incident on such vertices. The algorithm terminates when the vertex set is empty. In Fig. 2.4 is the formal description of the algorithm.

ALGORITHM MINDEGREEGREEDY(H)
INPUT: a hypergraph $H(V, E)$

$I = \emptyset$
While the vertex set V is not empty
  Add a vertex v of minimum degree to I
  Delete v from H
  Delete all vertices with loops along with all edges incident on them from H
Output I

Figure 2.4: The algorithm MINDEGREEGREEDY

Theorem 2.3.7 *The approximation ratio of* MINDEGREEGREEDY *is at most* $\Delta - 1$.

Proof: Let I and I^* be the greedy and the optimal solutions. We split the sequence of iterations of the algorithm into epochs, where a new epoch starts when the algorithm selects a vertex of degree Δ. Clearly, if the algorithm always selects a vertex of degree less than Δ, the whole sequence of iterations is just one epoch. Let I_t and I_t^* be the set of vertices from the greedy and the optimal solutions, respectively, deleted during epoch t. Then, $|I| = \sum_t |I_t|$ and $|I^*| = \sum_t |I_t^*|$. We show that $|I_t^*|/|I_t| \leq \Delta - 1$ for every epoch t.

Consider an iteration i in epoch t. The algorithm selects a vertex v_i, whose set of neighbors in 2-edges we denote by $N(v_i)$. The vertices of $N(v_i)$ are deleted in the iteration along with all incident edges. The maximum number of nodes removed in the iteration i that can belong to I_t^* is at most the degree of v_i. If i is the first iteration in t, then

$d(v_i) = \Delta$; for any other iteration in the same epoch $d(v_i) < \Delta$ (by the definition of an epoch).

Suppose one of the deleted edges is incident on a vertex u outside of $N(v_i)$. Then, in iteration $i+1$, the vertex u will have degree at most $\Delta - 1$, and therefore, the degree of v_{i+1} is at most $\Delta - 1$. Thus, the iteration $i+1$ will be in the same epoch as i, and the maximum number of nodes removed in any such iteration that can belong to I_t^* is at most $\Delta - 1$.

The last iteration of an epoch occurs when a vertex v_j is chosen whose neighborhood is contained in $N(v_j) \cup \{v_j\}$. This neighborhood then forms a hyperclique, because any vertex in $N(v_j)$ has at least the degree of v_j and all its neighbors are contained in $N(v_j) \cup \{v_j\}$. Notice that we may assume without loss of generality that the hypergraph is *simple*, namely that no edge is a proper subset of any other edge. Therefore, since the degree of v_j is at most Δ, any edge of the hyperclique contains at most $\Delta - 1$ vertices, and the maximum number of nodes removed in this iteration that can belong to an optimal solution I_t^* is at most $\Delta - 2$.

We see that in any epoch t the maximum number of deleted vertices that belong to I_t^* is at most Δ in the first iteration, at most $\Delta - 2$ in the last iteration and at most $\Delta - 1$ in any intermediate iteration. Amortized, the maximum number of deleted vertices that belong to I_t^* in any iteration of epoch t is at most $\Delta - 1$, while exactly one deleted vertex belongs to I_t. Therefore, $|I_t^*|/|I_t| \leq \Delta - 1$ for every epoch t. ■

Theorem 2.3.8 *The approximation ratio of* MINDEGREEGREEDY *is at least* $\Delta - 1$ *for* $\Delta = 3$ *and at least* $\Delta - 2 + \frac{2}{\Delta+1}$ *for any* $\Delta \geq 4$.

Proof: We consider two cases: $\Delta = 3$ and $\Delta \geq 4$, and describe hard hypergraphs for both cases. Let an *n-star* refer to a star with $n+1$ vertices.

Case I: $\Delta = 3$. For any $l \geq 2$ we construct a 3-regular hypergraph, composed of l 2-stars (see Fig. 2.5).

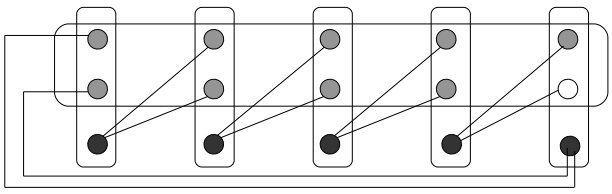

Figure 2.5: Example of a hard 3-regular hypergraph for MINDEGREEGREEDY, where the grey vertices represent an optimal solution, the black vertices represent the greedy solution.

For $1 \leq i \leq l$, each 2-star H_i has a root t_i and two endpoints v_i and u_i, connected to the root by the edges (t_i, v_i) and (t_i, u_i). The root t_i of each star H_i is connected to the endpoints of the preceding star by one edge (t_i, u_{i-1}, v_{i-1}) (the root of the last star is connected to the endpoints of the first star by an edge (t_l, u_1, v_1)). The endpoints of all stars are connected into one edge $(u_1, u_2, \ldots, u_l, v_1, v_2, \ldots, v_l)$.

Since the hypergraph is regular, the algorithm might start by selecting the root of the first star, adding it to the independent set and deleting it from the hypergraph. After this deletion, the endpoints of the second star have loops, and so the algorithm deletes the endpoints of the second star with all incident edges, reducing by one the degree of the endpoints of all other stars and the root of the second star. The algorithm proceeds this way, choosing all the roots of the stars for a solution of size l. On the other hand, an optimal solution is of size $l(\Delta - 1) - 1$ and includes the endpoints of all but one stars. Therefore, the approximation ratio is $\rho = \Delta - 1 - \frac{1}{l}$, approaching $\Delta - 1$, when l is large.

Case II: $\Delta \geq 4$. We construct a Δ-regular hypergraph, composed of Δ blocks and a vertex s. For $1 \leq i \leq l$, each block is a Δ-star H_i with a root t_i and Δ endpoints $\{v_i^1, \ldots, v_i^\Delta\}$ connected to the root by Δ edges $\{(t_i, v_i^1), (t_i, v_i^2), \ldots, (t_i, v_i^\Delta)\}$. In each block the vertices $\{v_i^1, \ldots, v_i^{\Delta-1}\}$ are connected to the vertex s by a single edge $(s, v_i^1, \ldots, v_i^{\Delta-1})$; the vertex v_i^Δ is connected to the vertices $\{v_i^1, \ldots, v_i^{\Delta-1}\}$ by $\Delta - 1$ edges of cardinality $\Delta - 1$ each (see Fig. 2.6).

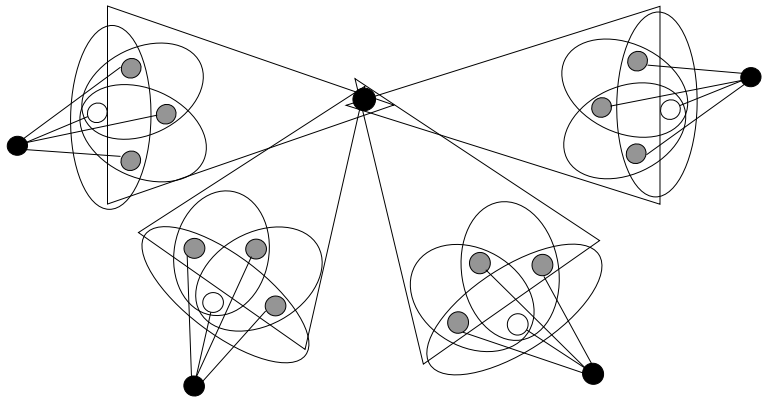

Figure 2.6: Example of a hard 4-regular hypergraph for MINDEGREEGREEDY, where the grey vertices represent an optimal solution, the black vertices represent the greedy solution.

The hypergraph is regular, and so the algorithm might start by selecting the vertex s. The deletion of s doesn't change the degree of the remaining vertices, because s has no

incident 2-edges and the algorithm doesn't delete any edges. This leaves disjoint regular Δ-stars, where the greedy algorithm chooses only the roots of the stars for a solution of size $\Delta+1$. On the other hand, an optimal solution is of size $\Delta(\Delta-1)$ and includes $\Delta-1$ endpoints from each star. Therefore, the approximation ratio is $\rho = \frac{\Delta-1}{1+1/\Delta} = \Delta-2+\frac{2}{\Delta+1}$. ■

Theorem 2.3.9 MINDEGREEGREEDY *attains the approximation ratio of* $1 + \frac{\Delta-1}{r}$ *in* r*-uniform* Δ*-regular hypergraphs.*

Proof: We assume that $r \geq 3$, because 2-uniform hypergraphs are ordinary graphs and the analysis of the greedy algorithm on graphs can be found in [31].

Let I_{max} and I_{min} be the largest and the smallest maximal weak independent sets in H, respectively. The approximation ratio of any non-trivial approximation algorithm for IS is bounded by the maximum ratio between I_{max} and I_{min} taken over all hypergraphs:

$$\rho \leq \max_{\forall H} \frac{|I_{max}|}{|I_{min}|} \,. \tag{2.13}$$

Any minimal cover S in H is of size at least

$$|S| \geq \frac{|E|}{\Delta} = \frac{\Delta|V|}{r\Delta} = \frac{|V|}{r} \,, \tag{2.14}$$

where in the last equality we use the fact that the number of edges in r-uniform Δ-regular hypergraph is exactly $|E| = \frac{\Delta|V|}{r}$. It is also easy to prove that any minimal cover is of size at most:

$$|S| \leq \frac{\Delta|V|}{\Delta + r - 1} \,. \tag{2.15}$$

For the reader's convenience we include here the proof of (2.15) from [8]. Since S is a minimal cover, for any vertex $v \in S$ there is at least one edge in E covered only by v. Consequently, each such edge includes $r-1$ vertices from $V \backslash S$ and the total degree of vertices in $V \backslash S$ is at least $|S|(r-1)$. On the other hand, the total degree of vertices in $V \backslash S$ is at most $\Delta(|V| - |S|)$. From $|S|(r-1) \leq \Delta(|V| - |S|)$ the inequality (2.15) follows immediately.

Any vertex in V belongs either to a minimal cover or to a maximal weak independent set. Consequently, any maximal weak independent set is of size at least:

$$|I_{min}| \geq |V| - \frac{\Delta|V|}{\Delta + r - 1} \tag{2.16}$$

and at most

$$|I_{max}| \leq |V| - \frac{|V|}{r}\,, \tag{2.17}$$

where the first inequality involves the upper bound on the size of a minimal cover in H from (2.15), and the second inequality uses the lower bound from (2.14).

Finally, combining together (2.13), (2.16) and (2.17) we obtain the upper bound on the approximation ratio of any non-trivial approximation algorithm for IS

$$\rho \leq \frac{1-\frac{1}{r}}{1-\frac{\Delta}{\Delta+r-1}} = \frac{\Delta+r-1}{r} = 1 + \frac{\Delta-1}{r}\,. \tag{2.18}$$

For the lower bound, we construct a Δ-regular r-uniform hypergraph H composed of a hyperclique B on $\Delta+1$ vertices and a set A of $r-1$ vertices. The edges of the hyperclique are all possible Δ-combinations of $\Delta+1$ vertices. Each vertex of the hyperclique except one is connected to the set A by one edge.

Since the hypergraph is regular, the MINDEGREEGREEDY algorithm might start by selecting vertices in the set A. The deletion of the first $r-2$ vertices reduces the size of the incident edges from r to 2 and doesn't produce loops. The deletion of the last vertex in A creates loops on all vertices in B, and the algorithm deletes the set B and all incident edges. Thus, the greedy weak independent set includes $r-1$ vertices from A and one vertex from B, while an optimal weak independent set includes Δ vertices from B and $r-2$ vertices from A. The approximation ratio is then $\frac{r-2+\Delta}{r}$. ■

Remarks. We conjecture that it should be possible to prove that MINDEGREEGREEDY have the worst approximation ratio in Δ-regular r-uniform hypergraphs, and so the result of Theorem 2.3.9 applies to arbitrary r-uniform hypergraphs. In any case, the approximation ratio of MINDEGREEGREEDY in Δ-regular r-uniform hypergraphs is worse than the approximation ratio MAXDEGREEGREEDY in r-uniform hypergraphs.

2.3.2 Strong Independent Set

There are two greedy algorithms for the SIS problem in hypergraphs. Both algorithms iteratively construct a maximal strong independent set by selecting vertices either of minimum degree (the MAXDEGREEGREEDYSIS algorithm) or with fewest neighbors (the MAXNEIGHBORGREEDYSIS algorithm).

Lemma 2.3.10 *Any maximal strong independent set is a Δ-approximation.*

Proof: Each node in the optimal solution is dominated by a node in the maximal solution, i.e. either by itself or by its neighbor. However, each node in the maximal solution can dominate at most Δ optimal vertices, as its neighborhood is covered by at most Δ edges, each containing at most one optimal vertex. ■

Lemma 2.3.11 *There exist Δ-regular hypergraphs where the approximation ratio of* MAXDEGREEGREEDYSIS *is* Δ.

Proof: For any $l \geq 2$ we construct the hypergraph $H_l(V, E)$, composed of a vertex s and l cliques on l vertices each. The vertex s is connected to the cliques by l edges, so that the i-th edge includes the vertex s and the i-th vertex from each clique (see Fig.2.7).

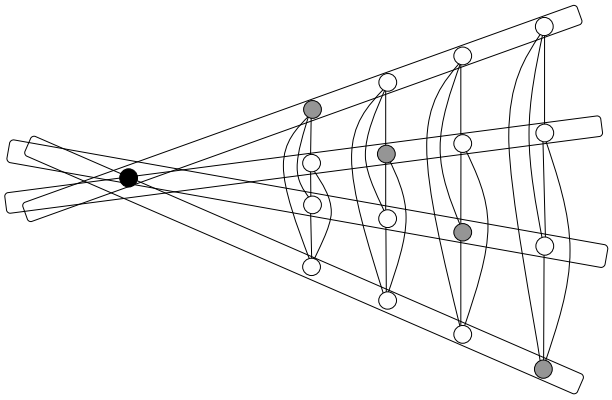

Figure 2.7: Example of a hard 4-regular hypergraph for MAXDEGREEGREEDYSIS, where the grey vertices represent an optimal solution, the black vertex represents the greedy solution.

Each vertex in the hypergraph has degree l, and so the hypergraph is regular with $\Delta = l$. The maximum strong independent set is of size l and includes the i-th vertex from the i-th clique. MAXDEGREEGREEDYSIS is a non-deterministic algorithm: in the worst case the vertex s is selected first and no more vertices can be added to the solution. Thus, the approximation ratio is Δ. ■

Lemma 2.3.12 *There exist Δ-regular hypergraphs where the approximation ratio of* MAXNEIGHBORGREEDYSIS *approaches* Δ.

Proof: For any $m \geq 2$ and $l \geq 2$, we construct the hypergraph $H_{m,l}(V, E)$, composed of m subgraphs on $3l$ vertices each. For $1 \leq i \leq m$, each subgraph H_i consists of sets U_i, W_i and T_i of l vertices each. Vertices in W_i and T_i form a complete bipartite graph (W_i, T_i) without one matching. For each vertex in W_i there is an edge containing this

vertex and the set U_i. All subgraphs are connected by one edge, containing all T sets (see Fig. 2.8).

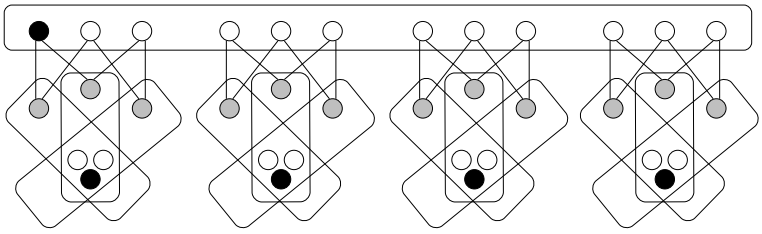

Figure 2.8: Example of a hard 3-regular hypergraph $H_{4,3}$ for MAXNEIGHBORGREEDYSIS, where the grey vertices represent an optimal solution and the black vertices represent the greedy solution.

Each vertex in the hypergraph has degree l, and so the hypergraph is regular with $\Delta = l$. We can easily verify that every vertex in U and W has the same number of neighbors, namely $2l - 1$, and every vertex in T has $l(m + 1) - 2$ neighbors. In each subgraph H_i every vertex in U_i is a neighbor of $l - 1$ vertices in U_i and l vertices in W_i; every vertex in W_i is a neighbor of l vertices in U_i and $l - 1$ vertices in T_i; every vertex in T_i is a neighbor of $l - 1$ vertices in W_i, $l - 1$ vertices in T_i and $(m - 1)l$ vertices in T-sets from the other $m - 1$ subgraphs.

A maximum strong independent set is of size ml and includes all W sets. MAXNEIGHBORGREEDYSIS is a non-deterministic algorithm, and so it might start by selecting a vertex from U_1, delete U_1 and W_1 from the subgraph and reduce the number of neighbors of any vertex in T_1 to $l(m-1)$. Since $m \geq 2$, the vertices in T_1 have at least the same number of neighbors as the vertices in any of the U and W sets of the remaining subgraphs. Thus, the algorithm might proceed by selecting a vertex from U_2 and so on until all U and W sets are deleted. From the remaining edge composed of all T sets, the algorithm adds only one vertex to the solution. Therefore, the greedy solution is of size $m + 1$ and the approximation ratio is approximately $l = \Delta$ provided m is large. ■

Theorem 2.3.13 *In r-uniform hypergraphs the approximation ratio of* MAXDEGREEGREEDYSIS *and* MAXNEIGHBORGREEDYSIS *is at most* $\Delta - \frac{\Delta-1}{r}$.

Proof: Let v_i be the vertex chosen by the algorithm (MAXDEGREEGREEDYSIS or MAXNEIGHBORGREEDYSIS) on the i-th iteration; let d_i and n_i denote the degree and the number of neighbors of v_i, respectively. The greedy algorithm terminates when the vertex set is

empty, say after t iterations:

$$\sum_{i=1}^{t}(n_i+1) = n \ . \tag{2.19}$$

Since the vertex v_i has n_i neighbors, its degree is at least:

$$d_i \geq \frac{n_i}{r-1} \ . \tag{2.20}$$

Any neighbor v_j of v_i has degree $\frac{n_i}{r-1}$ at least. The reason is simple: in MAXDEGREEGREEDYSIS the vertex v_i has the smallest degree, and so the degree of v_j is at least the degree of v_i; in MAXNEIGHBORGREEDYSIS the vertex v_j has at least the same number of neighbors as v_i and consequently, it is degree is $d_j \geq \frac{n_j}{r-1} \geq \frac{n_i}{r-1}$. Then, the total sum of degrees of all vertices in the hypergraph equals to $\overline{d}n$:

$$\begin{aligned}
\overline{d}n &\geq \sum_{i=1}^{t}\left(d_i + \sum_{j=1}^{n_i} d_j\right) \geq \sum_{i=1}^{t}\left(\frac{n_i}{r-1} + \sum_{j=1}^{n_i}\frac{n_i}{r-1}\right) \\
&= \frac{1}{r-1}\sum_{i=1}^{t} n_i(n_i+1) = \frac{1}{r-1}\left(\sum_{i=1}^{t}(n_i+1)^2 - n\right) \qquad (2.21) \\
&\geq \frac{1}{r-1}\left(\frac{n^2}{t} - n\right) \qquad (2.22)
\end{aligned}$$

where in (2.21) we use Cauchy-Schwarz inequality[3]. From (2.22) we can derive the lower bound on the size of the greedy solution:

$$t \geq \frac{n}{\overline{d}(r-1)+1} \ .$$

Let δ be the minimum degree in a given hypergraph. Since the number of edges in r-uniform hypergraphs is $\overline{d}n/r$ and each edge includes at most one vertex from a maximum strong independent set, the size of any maximum strong independent set is at most:

$$\alpha \leq \frac{\overline{d}n}{\delta r} \ .$$

Then, the approximation ratio of the greedy algorithm (MAXDEGREEGREEDYSIS or MAXNEIGHBORGREEDYSIS) is at most:

$$\rho = \max_{\forall \alpha, t}\frac{\alpha}{t} \leq \frac{\overline{d}}{\delta r}(\overline{d}(r-1)+1) \ .$$

[3] Cauchy-Schwarz inequality for one dimensional space: $\sum_{i=1}^{n} x_i^2 \geq \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$

Let k be such that $\overline{d} = k\Delta+(1-k)\delta$. Then, it is easy to verify that $f(k) = \frac{k\Delta+(1-k)\delta}{\delta r}((k\Delta+(1-k)\delta)(r-1)+1)$ is maximized when $\Delta = \delta$ or $k = 1$, i.e. in regular hypergraphs. ■

Theorem 2.3.14 *In r-uniform hypergraphs the approximation ratio of* MAXDEGREEGREEDYSIS *and* MAXNEIGHBORGREEDYSIS *is at least* $\Delta - \frac{\Delta-1}{r}$.

Proof: We describe the construction for the MAXDEGREEGREEDYSIS algorithm; for MAXNEIGHBORGREEDYSIS it is similar. The hypergraph H is composed of r subgraphs on $\Delta r - \Delta + 1$ vertices each. The first $r-1$ subgraphs H_i are disjoint, each of them consists of a vertex s, a set A of Δ independent vertices and a set B of $\Delta(r-2)$ vertices. The r-th subgraph is connected to the first $r-1$ subgraphs and contains a vertex s and a set C of $\Delta(r-1)$ vertices. In each subgraph the vertex s is connected to all other vertices by Δ-edges: in the first $r-1$ subgraphs each such edge includes one vertex from A and $r-2$ vertices from B, while in the last subgraph each such edge includes $r-1$ C-vertices. In each of the first $r-1$ subgraphs there are also $\Delta-1$ edges incident on each vertex in A: half of these edges includes $(r-1)$ vertices from B, the other half of the edges includes $(r-3)$ vertices from B and two vertices from C. We can specify the edges such that all edges have the cardinality r and all vertices in the hypergraph have the same degree Δ (see Fig.2.9).

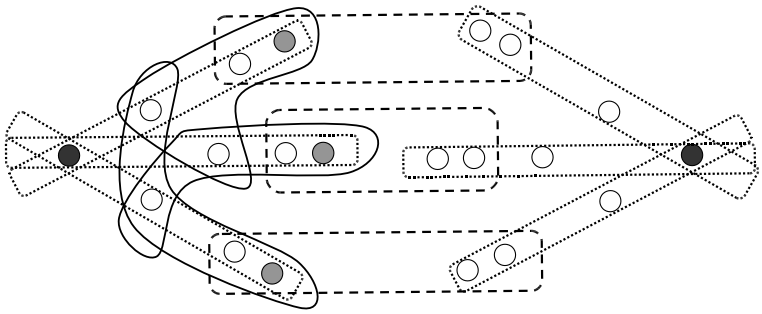

Figure 2.9: Part of a hard 3-regular 4-uniform hypergraph for MAXNEIGHBORGREEDYSIS, where one of the first $r-1$ subgraphs is connected to the last subgraph. The black vertices represent the s-vertices, the grey vertices represent the set A and the white vertices represent the sets B and C.

A maximum strong independent set is of size $(r-1)\Delta+1$ and consists of all A-sets and the vertex s from the last subgraph. The greedy algorithm might start by selecting the vertex s from the first subgraph and deleting the sets A and B in the first subgraph. This deletion reduces the size of one edge in the last subgraph by $r-2$ vertices, but doesn't

reduce the degree of any of the remaining vertices. Thus, on the next iteration the greedy algorithm might repeatedly select vertices s from each subgraphs, and form a maximal strong independent set of size r. Therefore, the approximation ratio is $\Delta - \frac{\Delta-1}{r}$. ■

2.4 Partitioning

The idea of the partitioning approach is to split a given hypergraph into k induced subhypergraphs so that IS can be solved optimally on each subhypergraph in polynomial time. This is based on the strategy of [29] for ordinary graphs. Note, that the largest of the solutions on the subhypergraphs is a k-approximation of IS, since the size of any optimal solution is at most the sum of the sizes of the largest weak independent sets on each subhypergraph.

We extend a partitioning lemma of Lovász [44] to the hypergraph case.

Lemma 2.4.1 *The vertices of a given hypergraph can be partitioned into $\lceil(\Delta + 1)/3\rceil$ sets, each inducing a subhypergraph of maximum degree at most two.*

Proof: Start with an arbitrary vertex partitioning into $\lceil(\Delta + 1)/3\rceil$ sets. While a set contains a vertex v with degree more than two, move v to another set that properly contains at most two edges incident on v. A set with at most two edges incident on v exists, because otherwise the total number of edges incident on v would be at least $3\lceil(\Delta+1)/3\rceil \geq \Delta+1$. Any such move increases the number of edges between different sets, and so the process terminates with a partition where every vertex has at most two incident edges in its set. ■

The method can be implemented in time $O(\sum_{e\in E}|e|)$ by using an initial greedy assignment as argued in [29].

Lemma 2.4.2 *Weighted IS in hypergraphs of maximum degree two can be solved optimally in polynomial time.*

Proof: Given a hypergraph $H(V, E)$ we consider the dual hypergraph $H'(E, V)$, whose vertices $e_1, \ldots, e_m$ correspond to the edges of H and the edges $v_1, \ldots, v_n$ correspond to the vertices of H, i.e. $v_i = \{e_j : v_i \in e_j \text{ in } H\}$. The maximum edge size in H' equals to the maximum degree of H, thus H' is a graph, possibly with loops. A vertex cover in H is an edge cover in H' (where an edge cover in H' is defined as a subset of edges that touches every vertex in H'), and a minimum weighted edge cover in graphs can be found in polynomial time via maximum weighted matching [18]. All edges not in a minimum

cover in H' correspond to the vertices in H that form a maximum weak independent set in H. ■

The following result is straightforward from Lemmas 2.4.1 and 2.4.2.

Theorem 2.4.3 *Weighted IS is approximable within $\lceil(\Delta+1)/3\rceil$ in polynomial time.*

2.5 Conclusions

In this section we analyze several approaches to IS on bounded-degree hypergraphs. We propose a general technique, called *shrinkage reduction*, that reduces the worst case analysis of certain algorithms on hypergraphs to their analysis on ordinary graphs. This technique allows us to show that the max-degree greedy algorithm for IS has a approximation ratio of $(\Delta+1)/2$. It also allows us to apply results on local search algorithms on graphs to obtain a $(\Delta+1)/2$ approximation for WIS and $(\Delta+3)/5-\epsilon$ approximation for IS. We improve the bound in the weighted case to $\lceil(\Delta+1)/3\rceil$ using a simple partitioning algorithm. We also consider another natural greedy algorithm for IS that adds vertices of minimum degree and achieves only a ratio of $\Delta-1$, significantly worse than on ordinary graphs. For SIS, we give two variations of the basic greedy algorithm and describe a family of hypergraphs where both algorithms approach the bound of Δ.

Chapter 3

SDP Algorithms

This section deals with approximations of IS in non-uniform hypergraphs of low degree. Our approach is to use a semi-definite technique to sparsify a given hypergraph and then apply combinatorial algorithms to find a large independent set in the resulting sparser instance. We start by introducing the semi-definite sparsifying algorithm, and then show how the sparsifying algorithm can be combined with greedy and randomized algorithms to achieve an $O(\overline{D} \log\log \overline{D}/\log \overline{D})$ approximation factor.

3.1 Semidefinite Programming

We use semidefinite programming to find large subgraphs with few 2-edges. More generally, we find subgraphs of large weight and small weighted average degree. This is obtained by rounding the vector representation of a suitable subgraph; such a subgraph is found by a result of Alon and Kahale [3]. Along the way, we twice eliminate vertices of high-degree to ensure degree properties.

3.1.1 Preliminaries

Let us recall the definition of a vector coloring of a graph [42].

Definition 3.1.1 ([42]) *Given a graph G and a real number $h \geq 1$, a* vector h-coloring *of G is an assignment of a $|V(G)|$-dimensional unit vector $\vec{v_i}$ to each vertex v_i of G so that for any pair v_i, v_j of adjacent vertices the inner product of their vectors satisfies*

$$\vec{v_i} \cdot \vec{v_j} \leq -\frac{1}{h-1} \ . \tag{3.1}$$

The *vector chromatic number* $\overline{\chi}(G)$ is the smallest positive number h, such that there exists a feasible vector h-coloring of G.

A vector representation given by a vector coloring is used to find a sparse subgraph by the means of *vector rounding* [42]: choose a random vector $\vec{b}$, and retain all vertex vectors whose inner product with $\vec{b}$ is above a certain threshold. The quality (i.e. sparsity) of the rounded subgraph depends on the vector chromatic number of the graph. In order to approximate independent sets we need to use this on graphs that do not necessarily have a small vector chromatic number but have a large independent set.

A graph with a large independent set contains a large subgraph with a small vector chromatic number, and there is a polynomial time algorithm to find it. This comes from the following variation of a result due to Alon and Kahale [3]:

Theorem 3.1.2 ([27]) *Let $G = (V, E, w)$ be a weighted graph and ℓ, p be numbers such that $\alpha(G, w) \geq w(G)/\ell + p$. Then, there is a polynomial time algorithm that gives an induced subgraph G_1 in G with $w(G_1) \geq p$ and $\overline{\chi}(G_1) \leq l$.*

3.1.2 Sparsifying Algorithm

Let us now present the algorithm for finding a large-weight low 2-degree hypergraph. It assumes that it is given the size α of the maximum independent set in the graph. We can sidestep that by trying all possible values for α, up to a sufficient precision (say, factor 2).

The algorithm SPARSEHYPERGRAPH (see Fig. 3.1) can be implemented to run in polynomial time. The subgraph G_1 in G_0 with small vector chromatic number and large independent set can be found in polynomial time [3]. A vector representation can be found within an additive error of ϵ in time polynomial in $\ln 1/\epsilon$ and n using the ellipsoid method [25] and Choleski decomposition.

3.1.3 Analysis

Lemma 3.1.3 *The graph G_0 has weight at least $w(H)(1-1/2k)$ and independence number at least $\alpha(H)/2$.*

Proof: The graph G has the same weight as H, or $w(V)$. The independence number of G is also at least that of H, since it contains only a subset of the edges. Let $X =$

ALGORITHM SPARSEHYPERGRAPH (H, α)
INPUT: Hypergraph $H(V, E)$, and its independence number α
OUTPUT: Induced hypergraph $\hat{H}$ in H of maximum degree $2k\overline{D}$ and maximum 2-degree $\sqrt{2k\overline{D}}$

Let $k = w(H)/\alpha$ and $a = 1 + \frac{1}{\ln \ln \overline{D} - 1}$.
Let G be the graph induced by the 2-edges of H.
Let G_0 be the subgraph of G induced by nodes of degree at most $2k\overline{D}$ in H.
Find an induced subgraph G_1 in G_0 with $w(G_1) \geq \frac{(a-1)w(G)}{2ak}$
with a vector $2ak$-coloring.
Choose a random $|V(G_1)|$-dimensional vector $\vec{b}$.
Let G_2 be the subgraph of G_1 induced by vertices $\{v \in V(G_1) : \vec{v} \cdot \vec{b} \geq c\}$,
where $c = \sqrt{\frac{ak-2}{ak} \ln (2k\overline{D})}$.
Let $\hat{V}$ be the set of vertices of degree at most $\sqrt{2k\overline{D}}$ in G_2.
Output $\hat{H}$, the subhypergraph in H induced by $\hat{V}$.

Figure 3.1: The sparsifying algorithm SPARSEHYPERGRAPH

$V(G) - V(G_0)$ be the high-degree vertices deleted to obtain G_0. Then,

$$\sum_{v \in X} w(v)d(v) \geq \sum_{v \in X} w(v) \cdot 2k\overline{D} = 2k\overline{D}w(X) \ . \tag{3.2}$$

Since

$$\overline{D} \cdot w(V) = \sum_{v \in V} w(v)d(v) \geq \sum_{v \in X} w(v)d(v) \ , \tag{3.3}$$

we get from combining (3.3) with (3.2) that the weight $w(X)$ of the deleted vertices is at most $w(V)/2k$. Thus, $w(G) \geq (1 - 1/2k)w(H)$. Also, G_0 has a maximum independent set of weight at least $\alpha(G_0, w) \geq \alpha(G, w) - w(X) \geq \alpha(H, w) - w(X) \geq w(H)/2k$.
■

Observe that $\alpha(G_0, w) \geq w(H)/2k \geq w(G_0)/2k = w(G_0)/2ak + w(G_0)(a-1)/2ak$. Then, Theorem 3.1.2 ensures that a subgraph G_1 can be found with $w(G_1) \geq w(G_0)(a-1)/2ak$ and $\overline{\chi}(G_1) \leq 2ak$. From that, a vector $2ak$-coloring of G_1 can be found.

The main task is to bound the properties of the rounded subgraph G_2. Karger et al. [42] estimated the probability that G_2 contains a given vertex or an edge. Let $N(x)$ denote the tail of the standard normal distribution: $N(x) = \int_x^\infty \phi(z)dz$, where $\phi(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$ is the density function. Let $\tau = \sqrt{\frac{2(ak-1)}{ak-2}}$.

Lemma 3.1.4 ([42]) *A graph G_2 induced in $G_1(V_1, E_1)$ after vector-rounding contains a given vertex in V_1 with probability $N(c)$ and a given edge in E_1 with probability $N(c\tau)$.*

The following lemma states well-known bounds on the tail of the normal distribution.

Lemma 3.1.5 ([51]) *For every* $x > 0$, $\phi(x)\left(\frac{1}{x} - \frac{1}{x^3}\right) < N(x) < \phi(x)\frac{1}{x}$.

We can now bound the weight of the subgraph found.

Lemma 3.1.6 *$\hat{V}$ has expected weight* $\Omega\left(\frac{w(G_1)}{(k\overline{D})^{\frac{1}{2}-\frac{1}{k}}\sqrt{\ln(k\overline{D})}}\right)$. *This can be derandomized to obtain an induced subgraph $\hat{V}$ with this much weight and maximum 2-degree at most* $\sqrt{2k\overline{D}}$.

Proof: First, for any edge (u, v) in G_1 and G_2 we define a weight function $w(u, v) = w(u) + w(v)$. Let $w(V_1) = \sum_{v \in V(G_1)} w(v)$ and $w(E_1) = \sum_{(v,u) \in E(G_1)} (w(v) + w(u))$ be the weight of vertices and edges in G_1. Similarly, let $w(V_2)$ and $w(E_2)$ be the weight of vertices and edges in G_2. Let X_i be an indicator random variable with $X_i = 1$, if V_2 contains $v_i \in V_1$ and $X_i = 0$ otherwise. Then, $w(V_2) = \sum_{v_i \in V_1} w(v_i)X_i$. Using Lemma 3.1.4 and linearity of expectation we bound $E[w(V_2)]$ by

$$E[w(V_2)] = w(V_1)N(c) \ . \tag{3.4}$$

Similarly, we bound $E[w(E_2)]$ by

$$E[w(E_2)] = w(E_1)N(c\tau) \leq 2k\overline{D}w(V_1)N(c\tau) \ , \tag{3.5}$$

where in the last inequality we use the fact that maximum degree in G_1 is bounded by $2k\overline{D}$ (since we deleted the high-degree vertices from G and G_1 is an induced subgraph in G). Combining (3.4) and (3.5), we get that

$$E\left[w(V_2) - \frac{w(E_2)}{\sqrt{2k\overline{D}}}\right] = w(V_1)N(c) - \sqrt{2k\overline{D}}w(V_1)N(c\tau) \ . \tag{3.6}$$

Observe that

$$c\tau = \sqrt{\frac{2(ak-1)}{ak}\ln(2k\overline{D})} = \sqrt{2\left(1 - \frac{1}{ak}\right)\ln(2k\overline{D})}$$

and

$$exp(-(c\tau)^2/2) = (2k\overline{D})^{-1+1/ak} \ .$$

Then, by Lemma 3.1.5

$$N(c\tau) < \phi(c\tau)\frac{1}{c\tau} = \frac{(2k\overline{D})^{-1+1/ak}}{\sqrt{2\pi} \cdot \sqrt{\frac{2(ak-1)}{ak}\ln(2k\overline{D})}} \tag{3.7}$$

and

$$N(c) > \phi(c)\frac{1}{c}\left(1-\frac{1}{c^2}\right) = \frac{(2k\overline{D})^{-1/2+1/ak}}{\sqrt{2\pi}\cdot\sqrt{\frac{ak-2}{ak}\ln(2k\overline{D})}}\left(1-\frac{ak}{(ak-2)\ln(2k\overline{D})}\right) \quad . \quad (3.8)$$

Combining (3.6), (3.7) and (3.8), we deduce that

$$\begin{aligned} E\left[w(V_2)-\frac{w(E_2)}{\sqrt{2k\overline{D}}}\right] &> w(V_1)\frac{(2k\overline{D})^{-1/2+1/ak}}{\sqrt{2\pi}\cdot\sqrt{\frac{ak-2}{ak}\ln(2k\overline{D})}}\left(1-\frac{ak}{(ak-2)\ln(2k\overline{D})}-\sqrt{\frac{ak-2}{2(ak-1)}}\right) \\ &= \Omega\left(\frac{w(V_1)}{(k\overline{D})^{1/2-1/k}\sqrt{\ln(k\overline{D})}}\right) \quad , \end{aligned} \quad (3.9)$$

where in the last line we use $a = 1+\frac{1}{\ln\ln\overline{D}-1}$.

The weight of vertices with degree greater than $\sqrt{2k\overline{D}}$ is at most $\sum_{v_i\in V_2} w(v_i)d(v_i)/\sqrt{2k\overline{D}} = w(E_2)/\sqrt{2k\overline{D}}$. After deleting all such vertices from G_2, the expected weight of $\hat{V}$ is $E\left[w(V_2)-\frac{w(E_2)}{\sqrt{2k\overline{D}}}\right]$ which is bounded by (3.9).

Finally, we can apply derandomization technique from [47] to derandomize the vector rounding in polynomial time. In our algorithm an elementary event corresponds to an edge in G_2 and involves only two vectors corresponding to the endpoints of the edge. This completes the proof. ■

We can bound the weight of the resulting hypergraph $\hat{H}$ in terms of the original hypergraph. We have that $w(\hat{V}) = \Omega\left(\frac{w(G_1)}{(k\overline{D})^{\frac{1}{2}-\frac{1}{k}}\sqrt{\ln k\overline{D}}}\right)$, while using $a = 1+\frac{1}{\ln\ln\overline{D}-1}$, we have that

$$w(G_1) = \frac{(a-1)w(G)}{2ak} = \frac{w(G)}{2k\ln\ln\overline{D}} = \frac{w(H)\left(1-\frac{1}{2k}\right)}{2k\ln\ln\overline{D}} = \Omega\left(\frac{w(H)}{k\ln\ln\overline{D}}\right) \quad .$$

Theorem 3.1.7 *Let H be a hypergraph with average weighted degree $\overline{D}$. The* SPARSEHYPERGRAPH *algorithm finds an induced hypergraph in H of weight $\Omega\left(\frac{w(H)}{k^{3/2-1/k}\overline{D}^{1/2-1/k}\ln\ln\overline{D}\sqrt{\ln(k\overline{D})}}\right)$, maximum 2-degree at most $\sqrt{2k\overline{D}}$, and maximum degree at most $2k\overline{D}$.*

3.2 Greedy Algorithm

Given a hypergraph H on n vertices with average degree $\overline{d}$, our GREEDYSDP algorithm first finds a sparse induced hypergraph H' in H using the SPARSEHYPERGRAPH algorithm and then uses the GREEDY algorithm to find an independent set in H'.

The GREEDY algorithm is a natural extension of the max-degree greedy algorithm on graphs and uniform hypergraphs and was analyzed by Thiele [55]. Given a hypergraph $H(V,E)$ with rank r, for any vertex $v \in V$ let $\overline{d}(v) = (d_1(v), \ldots, d_r(v))$ be the *degree vector* of v, where $d_i(v)$ is the number of edges of size i incident on v. Then, for any vertex $v \in V$ let

$$f(\overline{d}(v)) = \sum_{i_1}^{d_1(v)} \sum_{i_2}^{d_2(v)} \cdots \sum_{i_r}^{d_r(v)} \prod \binom{d_1}{i_1} \prod \binom{d_2}{i_2} \cdots \prod \binom{d_r}{i_r} \frac{(-1)^{\sum_{j=1}^{r} i_j}}{\sum_{j=1}^{r} (j-1)i_j + 1}$$

and let $F(H) = \sum_{v \in V} f(\overline{d}(v))$. The GREEDY algorithm iteratively chooses a vertex $v \in V$ with $F(H \backslash v) \geq F(H)$ and deletes v with all incident edges from H until the edge set is empty. The remaining vertices form an independent set in H.

Caro and Tuza [13] showed that in r-uniform hypergraphs the GREEDY algorithm always finds a weak independent set of size at least $\Theta\left(n/\Delta^{\frac{1}{r-1}}\right)$. Thiele [55] extended their result to non-uniform hypergraphs and gave a lower bound on the independence number as a complicated function of the degree vectors of the vertices in a hypergraph. Using these two bounds, we prove the following lemma.

Lemma 3.2.1 *Given a hypergraph H on n vertices with maximum 2-degree $\sqrt{d}$ and maximum degree d, the* GREEDY *algorithm finds an independent set of size $\Omega(n/\sqrt{d})$.*

Proof: First, we truncate edges in H to a maximum size three by arbitrarily deleting excess vertices. The resulting hypergraph H' still has maximum 3-degree d and maximum 2-degree $\sqrt{d}$, and is now of rank 3. Moreover, an independent set in H' is also independent in H. Thus, to prove the claim it is sufficient to bound from below the size of an independent set found by the greedy algorithm in H'.

As shown in [55], GREEDY finds an independent set in a rank-3 hypergraph of size at least

$$\alpha(H') \geq n \sum_{j=0}^{d} \sum_{i=0}^{\sqrt{d}} \binom{d}{j} \binom{\sqrt{d}}{i} \frac{(-1)^{(j+i)}}{i+2j+1} \quad . \tag{3.10}$$

By using the equality $\sum_k \binom{n}{k} \frac{(-1)^k}{x+k} = x^{-1} \binom{x+n}{n}^{-1}$ we can simplify (3.10) as:

$$\begin{aligned} \alpha(H') &\geq n \sum_{i=0}^{\sqrt{d}} (-1)^i \binom{\sqrt{d}}{i} \frac{1}{2} \left(\sum_{j=0}^{d} \binom{d}{j} \frac{(-1)^j}{j + (i+1)/2} \right) \\ &= \frac{n}{2(d+1)} \sum_{i=0}^{\sqrt{d}} (-1)^i \binom{\sqrt{d}}{i} \binom{(i+1)/2 + d}{d+1}^{-1} \quad . \end{aligned} \tag{3.11}$$

We show that for any value of d

$$F_d = \sum_{i=0}^{\sqrt{d}} (-1)^i \binom{\sqrt{d}}{i} \binom{(i+1)/2+d}{d+1}^{-1} \tag{3.12}$$

is lower bounded by $x\sqrt{d}$ for some $x > 0$. Then, from (3.11) the GREEDY algorithm finds an independent set of size at least $\Omega(n/\sqrt{d})$.

Let $f_d(i) = \binom{\sqrt{d}}{i} \binom{(i+1)/2+d}{d+1}^{-1}$. Abusing binomial notation, we assume that $\binom{\sqrt{d}}{i} = 0$, for any $i > \sqrt{d}$ and $\sqrt{d}$ integral. Then,

$$F_d = \sum_{i=0}^{\sqrt{d}} (-1)^i f_d(i) \ . \tag{3.13}$$

We define

$$q_d(i) = \frac{(i+2)(i+4)\cdots(i+2d+2)}{(i+3)(i+5)\cdots(i+2d+1)} \tag{3.14}$$

for any $i \geq 0$. Using Stirling's approximation for the factorial function[1] we obtain

$$q_d(0) = \frac{2^{2d+1}(d+1)d}{(2d+1)} = \sqrt{\pi d}\left(1 + O\left(\frac{1}{d}\right)\right)$$

and

$$q_d(1) = \frac{(2d+3)}{2^{2d+1}(d+1)(d+1)} = \sqrt{\frac{d}{\pi}}\left(1 + O\left(\frac{1}{d}\right)\right) \ .$$

Note that $q_d(i+2) = \frac{(i+3)(i+2d+4)}{(i+2)(i+2d+3)} q_d(i) > q_d(i)$, and so $q_d(i) > \sqrt{d}$ for any $i \geq 0$. Then, from the definition of $f_d(i)$ and (3.14) we have that $\frac{f_d(i+1)}{f_d(i)} = \frac{\sqrt{d}-i}{q_d(i)} < 1$. From (3.12) and (3.13) it follows that $F_d > f_d(0) - f_d(1)$ and $f_d(0) = q_d(0)$, then

$$\begin{aligned} F_d &> f_d(0) - f_d(1) \\ &= f_d(0)\left(1 - \frac{\sqrt{d}}{q_d(0)}\right) \\ &= q_d(0) - \sqrt{d} \\ &= (\sqrt{\pi} - 1)\sqrt{d}\left(1 + O\left(\frac{1}{d}\right)\right) \ . \end{aligned} \tag{3.15}$$

Thus, from (3.11), (3.12) and (3.15) the GREEDY algorithm finds an independent set of size at least $\Omega(n/\sqrt{d})$. ■

[1] Stirling's approximation: $N! = \sqrt{2\pi N}\left(\frac{N}{e}\right)^N \left(1 + O\left(\frac{1}{N}\right)\right)$

The bound on the approximation ratio of GREEDYSDP then follows from Lemma 3.2.1, Theorem 3.1.7 and the fact that truncating edges in SPARSEHYPERGRAPH doesn't increase the weight of a maximal independent set in a hypergraph.

Theorem 3.2.2 *Given a hypergraph H on n vertices with average degree $\overline{d}$ and the independence number $\alpha(H) = n/k$, the* GREEDYSDP *algorithm finds an independent set of size at least $\Omega\left(\frac{n}{k^{5/2-1/k}\overline{d}^{1-1/k}\ln\ln\overline{d}\sqrt{\ln(k\overline{d})}}\right)$.*

From Theorem 3.2.2 it is easy to see that if the maximum independent set in H is relatively big, say $\Omega\left(\frac{n\ln\ln\overline{d}}{\ln\overline{d}}\right)$, i.e. $k = O\left(\frac{\ln\overline{d}}{\ln\ln\overline{d}}\right)$, then GREEDYSDP obtains an approximation ratio of $O\left(\frac{\overline{d}}{\ln\overline{d}}\right)$. However, if the maximum independent set is at most $\Omega\left(\frac{n\ln\ln\overline{d}}{\ln\overline{d}}\right)$, then GREEDY alone is within a factor of $O\left(\frac{\overline{d}\ln\ln\overline{d}}{\ln\overline{d}}\right)$. Therefore, we run both GREEDY and GREEDYSDP and output the larger independent set found. We call this algorithm GREEDYSDP-IS.

Theorem 3.2.3 *Given a hypergraph H with average degree $\overline{d}$, the* GREEDYSDP-IS *approximates the maximum independent set within a factor of $O\left(\frac{\overline{d}\ln\ln\overline{d}}{\ln\overline{d}}\right)$.*

Corollary 3.2.4 *For small k the approximation factor of* GREEDYSDP-IS *is $O\left(\overline{d}^{1-\frac{1}{k}+o(1)}\right)$.*

The implication is the first approximation factor for the independent set problem in hypergraphs that is sublinear in the average degree.

Corollary 3.2.5 *The independent set problem in hypergraphs is $o(\Delta)$-approximable.*

3.3 Randomized Algorithm

The RANDOMIS algorithm extends the randomized version of Turán bound on graphs and was analyzed by Shachnai and Srinivasan in [54]. Given a hypergraph $H(V, E)$, the algorithm creates a random permutation π of V and adds a vertex v to the independent set I, if there is no edge e such that e contains v and v appears last in π among the vertices of e. Clearly, RANDOMIS outputs a feasible independent set I, since it never contains the last vertex in any edge under the permutation π.

Shachnai and Srinivasan [54] analyzed RANDOMIS on weighted hypergraphs. They gave a lower bound on the probability that a vertex $v \in H$ is added by the algorithm to the independent set, using conditional probabilities and the FKG inequality. In uniform hypergraphs the lower bound on the size of a independent set found by RANDOMIS follows by summing the probabilities over the vertices and applying linearity of expectation, giving a bound identical to that of Caro and Tuza [13].

Theorem 3.3.1 ([54], Theorem 2) *For any $r \geq 2$ and any r-uniform hypergraph H,* RANDOMIS *finds an independent set of size at least* $\sum_{v \in V} \binom{d(v)+1/(r-1)}{d(v)}^{-1} = \Omega\left(\sum_{v \in V} \frac{w(v)}{(d(v))^{\frac{1}{r-1}}}\right)$.

To extend the bound to non-uniform weighted hypergraphs, Shachnai and Srinivasan introduced the following potential function on a vertex v:

$$f(v) = \min_{j=1,2,\cdots,a(v)} (d_j(v))^{-\frac{1}{r_j(v)-1}} \ ,$$

where a vertex v lies in edges of $a(v)$ different sizes: $r_j(v)$, for $j = 1, 2, \cdots, a(v)$, and $d_j(v)$ is the number of edges of size $r_j(v)$. Using similar analysis as in Theorem 3.3.1, they proved the following bound:

Theorem 3.3.2 ([54], Theorem 3) *Given a weighted hypergraph $H(V, E)$, the expected weight of the independent set produced by* RANDOMIS *is at least* $\Omega\left(\sum_{v \in V} \frac{w(v)}{a(v)^{1/b(v)}} f(v)\right)$, *where* $b(v) = (\min_j(r_j(v) - 1))$.

Shachnai and Srinivasan also show in [54] how to derandomize RANDOMIS for hypergraphs with bounded maximum degree, or logarithmic degree and sparse neighborhoods.

Our algorithm RANDOMSDP first uses SPARSEHYPERGRAPH to find an induced hypergraph H' in H with maximum 2-degree at most $\sqrt{2k\overline{D}}$ and maximum degree at most $2k\overline{D}$; and then uses RANDOMIS to find an independent set in H'.

The bound on the approximation ratio of RANDOMSDP follows from Theorem 3.1.7 and Theorem 3.3.2, using that $\Omega\left(\sum_{v \in V} \frac{w(v)}{a(v)^{1/b(v)}} f(v)\right) = \Omega\left(\sum_{v \in V} \frac{w(v)}{\sum_{i=2}^{r} d_i^{\frac{1}{i-1}}}\right)$ by the definitions of $a(v)$, $b(v)$ and $f(v)$.

Theorem 3.3.3 *Given a weighted hypergraph H with average weighted degree $\overline{D}$, the* RANDOMSDP *algorithm finds an independent set of weight at least* $\Omega\left(\frac{w(H)}{k^{2-1/k}\overline{D}^{1/2-1/k} \ln\ln\overline{D}\sqrt{\ln(k\overline{D})}}\right)$.

From Theorem 3.3.3 it follows that the RANDOMSDP algorithm approximates IS within a factor of $O\left(\frac{\overline{D}}{\ln\overline{D}}\right)$ if $\alpha(H, w) = \Omega\left(\frac{w(V)\ln\ln\overline{D}}{\ln\overline{D}}\right)$, whereas RANDOMIS alone finds an approximation within a factor of $O\left(\frac{\overline{D}\ln\ln\overline{D}}{\ln\overline{D}}\right)$ if $\alpha(H, w) = O\left(\frac{w(V)\ln\ln\overline{D}}{\ln\overline{D}}\right)$. Therefore, given a hypergraph H, we run both RANDOMIS and RANDOMSDP on H and output the larger of the independent sets. We call this algorithm RANDOMSDP-IS.

Theorem 3.3.4 *Given a hypergraph $H(V, E)$ with average weighted degree $\overline{D}$, the* RANDOMSDP-IS *approximates the weight of a maximum independent set in H within a factor of* $O\left(\frac{\overline{D}\ln\ln\overline{D}}{\ln\overline{D}}\right)$.

3.4 Conclusions

In this section we introduce an SDP-based approximation technique, which first uses an SDP formulation to sparsify the hypergraph and then applies a combinatorial algorithm on the resulting sparser instance. Using this technique we derive the first $o(\Delta)$-approximation for IS in hypergraphs matching the $O(\Delta \log\log \Delta / \log \Delta)$-approximation for the special case of graphs. We then extend our results to obtain an identical bound in terms of the average degree $\overline{d}$ of an unweighted hypergraph. As part of the method, we also obtain a $k^{5/2-1/k}\overline{d}^{1-1/k+o(1)}$-approximation for hypergraphs with independence number at least n/k. We generalize the results to the weighted case and give a $O(\overline{D} \log\log \overline{D} / \log \overline{D})$-approximation for WIS.

Chapter 4

Streaming Algorithms

In this section we present algorithms for WIS on bounded and unbounded-degree hypergraphs in a semi-streaming model. We start by introducing randomized permutation-based algorithms for bounded-degree hypergraphs. We continue by analyzing several randomized and deterministic partitioning algorithms for bounded and unbounded-degree hypergraphs. Finally, we define an online streaming model, prove lower bounds for randomized and deterministic algorithms in this model and give a randomized online streaming algorithm for bounded-degree hypergraphs.

4.1 Permutation-based Algorithms

The idea of permutation-based algorithms is to randomly permute vertices and use this random permutation to decide which vertices to include in the independent set. Our streaming algorithms are based on the results for the off-line randomized algorithm RANDOMOFFLINE from [54]. Given a set of vertices V, RANDOMOFFLINE creates a permutation π on V. In every edge e the last vertex v according to the permutation π is added to the set S. When the algorithm terminates, S forms a hitting set and $V \backslash S$ is an independent set. For convenience, in Fig. 4.1 we give the description of the RANDOMOFFLINE algorithm.

Shachnai and Srinivasan [54] proved the following approximation ratio for the RANDOMOFFLINE algorithm for *IS* on hypergraphs.

Theorem 4.1.1 ([54]) *Given a hypergraph H, the off-line algorithm* RANDOMOFFLINE *finds an independent set of expected weight $\Omega\left(\frac{w(H)}{\overline{D^*}}\right)$.*

ALGORITHM RANDOMOFFLINE (H)
INPUT: a hypergraph $H(V, E)$

$S \leftarrow \emptyset$
Let π be a random permutation of the vertices in V
For every edge $e \in E(H)$
  Let v be the last vertex in e with respect to π
  $S \leftarrow S \cup \{v\}$
Output $I = V \backslash S$

Figure 4.1: The algorithm RANDOMOFFLINE

Observation 4.1.2 *The bound of* $\Omega\left(\frac{w(H)}{\overline{D^*}}\right)$ *on hypergraphs corresponds to Turán bound on graphs, which is the best possible bound in terms of* $\overline{D^*}$.

RANDOMOFFLINE leads easily to a semi-streaming algorithm, which creates a permutation of V in advance and then process edges one by one as they come in the stream. In this case we need $O(n \log n)$ working space to store the whole permutation. It means that this semi-streaming algorithm finds an independent set of expected weight $\Omega\left(\frac{w(H)}{\overline{D^*}}\right)$ using $O(r)$ time to process each eadge and $O(n \log n)$ total working space.

In fact, we do not need to know the set of vertices in advance. We can construct a random permutation π on-the-fly, starting with an empty set. For each edge in the stream, if the edge contains a new vertex v, we randomly place v among the vertices in π, and then process the edge the same way as in RANDOMOFFLINE. Let us call this algorithm RANDOMONFLYPERMUTE (Fig. 4.2).

ALGORITHM RANDOMONFLYPERMUTE (E)
INPUT: a stream E of edges

$S \leftarrow \emptyset$
Let $\pi = \emptyset$ be an ordered set
For every edge e in the stream E such that $e \cap S = \emptyset$
  For every vertex $u \in e$ such that $u \notin \pi$
    Randomly place u among the vertices in π
  Let v be the last vertex in e with respect to π
  $S \leftarrow S \cup \{v\}$
Output $I = V \backslash S$

Figure 4.2: The off-line algorithm RANDOMONFLYPERMUTE

The permutation π is random, because inserting a new vertex v in the ordered set π is equivalent to sampling $x_v \in [0, 1)$ using the uniform distribution on $[0, 1)$ and placing v in

π so that the numbers x_u corresponding to the vertices in π form an increasing sequence. Then, RANDOMONFLYPERMUTE finds an independent set within the same approximation factor as RANDOMAPRIORIPERMUTE.

Theorem 4.1.3 *The* RANDOMONFLYPERMUTE *algorithm finds an independent set of expected weight* $\Omega\left(\frac{w(V)}{D^*}\right)$ *using* $O(r)$ *time to process each edge and* $O(n \log n)$ *total working space.*

We can significantly reduce the space in RANDOMONFLYPERMUTE by constructing only a partial random permutation. Each vertex v is associated with a finite sequence b_v of random bits, where each bit in b_v is independently set to 0 or 1 with equal probability. A partial random permutation of the vertex set V is then a lexicographic order of the corresponding bit sequences $\{b_v\}_{v \in V}$. Consider an edge $e \in E$. Let $\{b_v | v \in e\}$ be the set of bit sequences associated with the vertices in e. For vertices $u, v \in e$, we say that $u \succ v$ if b_u follows b_v in the lexicographical order. We define the vertex $u \in e$ to be the *last* in e, if b_u is lexicographically last in the set $\{b_v | v \in e\}$. Let us call this algorithm RANDOMPARTIALPERMUTE.

The idea of RANDOMPARTIALPERMUTE is that for each vertex $v \in e$ we use the minimal number of bits to determine if b_v is the last in $\{b_v | v \in e\}$. In other words, we just need to determine which vertex in e is the last one, and the relative order of other vertices in e is not important.

The formal description of this algorithm is given in Fig. 4.3. The algorithm maintains a hitting set S. At the termination of the algorithm the difference $V \backslash S$ forms an independent set. Let $b_v[j]$ be the j-th bit in b_v.

Theorem 4.1.4 *The* RANDOMPARTIALPERMUTE *algorithm finds an independent set of expected weight* $\Omega\left(\frac{w(V)}{D^*}\right)$ *using expected* $O(n \log r)$ *space and* $O(r)$ *time to process each edge.*

Proof: First, we show that the partially ordered set $B = \{b_v | v \in V\}$ forms a partial permutation of V. Let $x_v \in [0, 1)$ be a random number from the uniform distribution on $[0, 1)$. A random permutation π of V is then an ordering of V such that if u follows v in π, then $x_u > x_v$. Each x_v can be viewed as an infinite sequence of random bits, where each bit in x_v is independently set to 0 or 1 with equal probability. To create a bit sequence b_v the RANDOMPARTIALPERMUTE algorithm sets each bit in b_v to 0 or 1 independently with equal probability and uses as many bits of x_v in b_v as needed to determine if b_v is lexicographically last in $\{b_v | v \in e\}$ for every edge e such that $v \in e$. Hence, b_v is a prefix of x_v. It means that the set $\{b_v | v \in V\}$ forms a partial permutation of V and we

ALGORITHM RANDOMPARTIALPERMUTE (E)
INPUT: a stream E of edges

$V \leftarrow S \leftarrow \emptyset$
For each edge e in the stream E do
  For each vertex $v \in e$ such that $v \notin V$
    $V \leftarrow V \cup \{v\}$
    $b_v \leftarrow \emptyset$
  If the edge e is not covered by a vertex from S
    $U \leftarrow \{e\}$
    $j \leftarrow 1$
    While $|U| \neq 1$ do
      For every $v \in U$ such that $b_v[j]$ is not defined
        $b_v[j] = 1$ with probability $\frac{1}{2}$, otherwise $b_v[j] = 0$
      If $\exists v \in U$ such that $b_v[j] = 1$
        Delete from U all vertices with $b_v[j] = 0$
      $j \leftarrow j + 1$
  $S \leftarrow S \cup U$
Output $I = V \backslash S$

Figure 4.3: The algorithm RANDOMPARTIALPERMUTE

can apply the same argument as in the proof of Theorem 4.1.1 to show that the bound of RANDOMPARTIALPERMUTE corresponds to T'uran bound on graphs.

Now we show that the algorithm uses an expected $O(n \log r)$ space to store the sets V, S and the set of bit sequences $\{b_v\}$. Given a vertex v, we say that we *open* the j-th bit in b_v, if the algorithm assign $b_v[j]$ its value, i.e. sets $b_v[j]$ to 0 or 1 with equal probability. To simplify the analysis we slightly change the process of opening the bits for a vertex v: instead of gradually opening the bits in b_v as needed, we open bits in blocks of $3 \log r$ bits, i.e. when the vertex first appears in the stream we open the first block of $3 \log r$ bits, and then every time we need to open more bits, we open next block of $3 \log r$ bits. This modification might result in opening more bits than we actually need for some vertices, but it allows us to show that with high probability $6 \log r$ bits for any vertex v would be sufficient to decide if v belongs to the independent set or to the hitting set.

Consider a vertex v. We open the first $3 \log r$ bits in b_v, when v first appears in the stream. Let us count how many more bits we need to open in b_v until v is put in the independent set or in the hitting set. Consider an edge e incident on v such that e is not covered by a vertex from S at the time e appears in the stream. Let $b_{v,e}$ be the bit sequence b_v at the time e appears in the stream. Let $U_{\succ v}(e) = \{u \in e | u \succ v\}$, $U_{\prec v}(e) = \{u \in e | v \succ u\}$ and $U_{=v}(e) = \{e\} \setminus (U_{\succ v}(e) \cup U_{\prec v}(e))$. We need to open more bits in $b_{v,e}$ only if $U_{\succ v}(e) = \emptyset$ and $U_{=v}(e) \neq \emptyset$, because in this case the vertices in $U_{=v}(e)$ have the highest bit sequences

and these bit sequences are exactly the same as $b_{v,e}$. In any other case the opened bits in $b_{v,e}$ are sufficient to decide which vertex covers e, namely e is covered either by a vertex $u \in U_{\succ v}(e)$ if $U_{\succ v}(e) \neq \emptyset$ or by the vertex v if $U_{\succ v}(e) = \emptyset$ and $U_{=v}(e) = \emptyset$. We say that an edge e is *problematic for* v, if $U_{\succ v}(e) = \emptyset$ and $U_{=v}(e) \neq \emptyset$. In other words, e is problematic for v, if v has a chance to cover e, but we need to open more bits in $b_{v,e}$ to decide if v covers e. Hence, we open more bits in b_v only when we consider an edge problematic for v.

We compute the expected number of vertices in e that have the same bit sequence as v. To simplify the analysis we only consider those bits that have not been considered for previous problematic edges and further at the beginning of the algorithm, and each time we have finished considering a problematic edge we open $3 \log r$ new bits. This guarantees that each time we consider an edge to be problematic we have $3 \log r$ bits that we can consider to be random. This simplification does not decrease chances of v to be in S, because if b_v has low previous bits, considering new random bits increases the chance of v being in S; and if b_v has high previous bits, then v is likely to be in S already.

Consider an edge e problematic for v. We compute the expected number of vertices that have the same bit sequence as v and condition the computation of this expectation over all the values that v can take. The bit sequence of v is random, but the bit sequences of other vertices in e are conditioned on being less than or equal to the bit sequence for v. Then, the expected number of vertices that have the same bit sequence as v in e at the $3 \log r$ bits under consideration can be determined by: summing over all the r^3 possible values that the bit sequence for v can take, and multiplying the probability that v has a given bit sequence $b_v^{(i)}$ with the expected number of vertices in e that have the same bit sequence as v. The probability that v has a given bit sequence i is $\frac{1}{r^3}$. The expected number of other vertices in e with the same bit sequence as v is bounded by $\frac{(r-1)}{i+1}$ if v has the bit sequence $b_v^{(i)}$. This can be seen by noting that $(r-1)$ is the maximum number of nodes in $e - \{v\}$ and $i+1$ is the number of bit sequences preceding $b_v^{(i)}$. Then, we get an expression

$$\sum_{i=0}^{r^3} \frac{1}{r^3}(r-1)\frac{1}{i+1} \leq \frac{3 \log r}{r^2} \leq \frac{1}{r}\,.$$

Each time we encounter a problematic edge e we open $3 \log r$ bits and then as many bits as needed to determine which of the vertices in $U_{=v}(e)$ has the highest bit sequence. The expected number of bits we open at a problematic edge in addition to $3 \log r$ is less than $\log\left(1 + \frac{1}{r}\right)$, as in expectation less than $1 + \frac{1}{r}$ nodes belong to $U_{=v}(e)$, the node v and the

nodes equal to v bounded in 4.1. In total we open in expectation

$$\sum_{i=0}^{\infty}\left(3\log r+\log\left(1+\frac{1}{r}\right)\right)\left(\frac{1}{r}\right)^{i}\leq\sum_{i=0}^{\infty}\left(3\log r+\frac{1}{r}\right)\left(\frac{1}{r}\right)^{i}\leq 6\log r+o(1)$$

bits, where we used the fact that $\log\left(1+\frac{1}{r}\right)\leq\frac{1}{r}$ and $\sum_{i=0}^{\infty}\left(\frac{1}{r}\right)^{i}=\frac{r}{r-1}\leq 2$ for any $r\geq 2$. This concludes the proof. ■

4.2 Partitioning Algorithms

We now consider unbounded-degree graphs and hypergraphs. The algorithms of preceding sections do not offer non-trivial guarantees when degrees can be arbitrarily large. Even off-line approximation is known to be extremely difficult in this case.

The idea of partitioning algorithms is to partition a given hypergraph into disjoint subhypergraphs, find a solution in one (some, or all) partition and output the best solution found. First, we describe the RANDOMPARTITIONAPRIORI algorithm (in Fig. 4.4) for unbounded-degree hypergraphs. In this algorithm we use partitioning approach from the off-line algorithm [26] which approximates *IS* within a factor of $O\left(n/\log n\right)$. The number of vertices is known in advance and the edges arrive in a random order. Let c be a positive number.

ALGORITHM RANDOMPARTITIONAPRIORI (V, c, E)
INPUT: a vertex set V, a constant c and a stream E of edges

Randomly select a set $V'\subseteq V$ of $c\log n$ nodes
$E'\leftarrow\emptyset$
For each edge e in the stream E do:
  If $e\subseteq V'$ then $E'\leftarrow E'\cup\{e\}$
Let I be a maximum independent set in $H(V', E')$
Output I

Figure 4.4: The algorithm RANDOMPARTITIONAPRIORI

Theorem 4.2.1 *The* RANDOMPARTITIONAPRIORI *algorithm approximates* IS *within* $O\left(\frac{n}{c\log n}\right)$ *using* $O(r)$ *time to process each edge and* $O(n^c\log n)$ *total working space for any constant* $c>0$.

Proof: Let OPT be a maximum independent set in a given stream with n vertices and let OPT' be a maximum independent set in the subhypergraph H' induced by the vertex

set V'. Obviously, $|OPT'| \geq |OPT \cap V'|$. Then, the expected size of the independent set output by the algorithm is $E[OPT'] \geq E[OPT \cap V'] = \sum\limits_{v \in V} Pr[v \in V' \cap OPT] = \sum\limits_{v \in OPT} Pr[v \in V'] = \sum\limits_{v \in OPT} \frac{c \log n}{n} = \frac{OPT}{n/(c \log n)}$. It means that RANDOMPARTITIONAPRIORI approximates *IS* within $O\left(\frac{n}{c \log n}\right)$. The algorithm requires $O(r)$ time to process each edge in the stream.

Now we show that the total working space of RANDOMPARTITIONAPRIORI is $O(n^c \log n)$ bits. Every edge in $H(V', E')$ can be represented as a bit sequence of length $c \log n$, where j-th bit equals 1, if the vertex v_j belongs to the edge, and 0, otherwise. The number of edges in E is at most n^c, which is the number of all possible subsets over $c \log n$ vertices. It means that the algorithm needs at most $n^c c \log n$ bits to store all edges in $H(V', E')$. ■

Further, we modify the RANDOMPARTITIONAPRIORI algorithm to be a variation of an online randomized preemptive algorithm (see Fig. 4.2). The algorithm works as follows. Let c be a positive constant. We define N to be un upper bound on the number of vertices in a hypergraph. We start with $N = 2$ and every time the number of vertices in the stream exceeds N, we double N. The algorithm selects a subset V' of vertices of expected size $c \log N$ and stores the set of edges E' formed by the vertices in V'. Everytime N is doubled, we update V' to insure that the expected size of V' is $c \log N$. In order to update V', we associate a number y_v with each vertex v in the stream. If $y_v \geq c \log N$, then $v \in V'$, and $v \notin V'$ otherwise. Then, every time N is doubled, we exclude those vertices from V' which have $y_v < c \log N$. The numbers y_v are calculated in the following way. When a new vertex v appears in the stream, we sample $x_v \in [0, 1)$ using the uniform distribution on $[0, 1)$ and define $y_v = \max\limits_{x_v \leq \frac{y}{2^y}} y$. After all edges in the stream are seen, the algorithm finds an optimal independent set in the graph formed by the set of vertices V' and the set of edges E'.

Theorem 4.2.2 *The algorithm* RANDOMPARTITIONONLINE *approximates* IS *within* $O\left(\frac{n}{c \log n}\right)$ *using* $O(r)$ *time to process each edge and the total working space of* $O(n^c \log n)$ *bits.*

Proof: Let OPT be a maximum independent set in a given stream with n vertices. The approximation guarantee follows by the same argument as in 4.2.1.

Now we show that the algorithm uses $O(n^c \log n)$ bits to space to store the sets V and V' and the set of edges E'. Every vertex in V' has $y_v \geq c \log N$, hence $x_v \leq \frac{c \log N}{N}$. Let z_v be an indicator random variable such that $z_v = 1$, if $x_v \leq \frac{c \log N}{N}$ and 0, otherwise. Since the total number of vertices appeared in the stream is at most N, the expected size of V'

ALGORITHM RANDOMPARTITIONONLINE ($c,\ E$)
INPUT: a constant c and a stream E of edges

$N \leftarrow 2$
$V \leftarrow V' \leftarrow E' \leftarrow \emptyset$
For each edge e in the stream E do:
  For each vertex $v \in e$ such that $v \notin V$
    $V \leftarrow V \cup \{v\}$
  If $|V| > N$ then
    $N \leftarrow 2 * N$
    For each vertex $v \in V'$
      If $y_v < c \log N$ then $V' \leftarrow V' \backslash \{v\}$
    $E' \leftarrow E' \cap V'$
  For each vertex $v \in e$ such that $v \notin V$
    Sample $x_v \in [0, 1)$ using the uniform distribution on $[0, 1)$
    Let $y_v = \max\limits_{x_v \leq \frac{y}{2^y}} y$
    If $y_v \geq c \log N$ then $V' \leftarrow V' \cup \{v\}$
  If $e \subseteq V'$ then $E' \leftarrow E' \cup \{e\}$
Let I be a maximum independent set in $H(V', E')$
Output I

Figure 4.5: The algorithm RANDOMPARTITIONONLINE

is $E[V'] = \sum\limits_{N} z_v = \sum\limits_{N} \frac{c \log N}{N} = c \log N$. Thus, we need $O(\log N)$ bits to store the sets V and V' on expectation. To store the edges in E' it is sufficient to use $O(N^c \log N)$ bits by the same argument as in the proof of 4.2.1. Since $N \leq 2n$, the total expected working space is $O(n^c \log n)$. ■

Now we turn our attention to deterministic algorithms. The first algorithm, DETSPARSEHYPERGRAPH, works for bounded-degree hypergraphs and uses $O(n \log n)$ space. In the DETSPARSEHYPERGRAPH algorithm the idea is to store a subhypergraph $H(V', E')$ such that E' contains at most n edges, where n is the number of vertices. The algorithm assumes that the vertex set V is not known in advance and vertices are coming with edges in the stream. The algorithm starts by storing all coming vertices along with incident edges until the number of edges in E' exceeds n. Once the number of edges in E' exceeds n, the algorithm deletes the vertex with the largest index from V' and deletes all edges incident on it from E'. After going through the stream, DETSPARSEHYPERGRAPH uses the MAXDEGREEGREEDY to find an independent set in $H(V', E')$. The formal description of DETSPARSEHYPERGRAPH is given in Fig. 4.6.

The MAXDEGREEGREEDY algorithm is given in Fig.2.3.

ALGORITHM DETSPARSEHYPERGRAPH ($n,\ E$)
INPUT: a number n of vertices and a stream E of edges

$E' \leftarrow S \leftarrow V' \leftarrow \emptyset$
For each edge e in the stream E do:
  If $e \cap S = \emptyset$ then
    $V' \leftarrow V' \cup \{e\}$
    $E' \leftarrow E' \cup e$
  If $|E'| > n$ then
    Let $v_j \in V$ be the vertex with the maximum index in V'
    $V' \leftarrow V' \backslash \{v_j\}$
    $S \leftarrow S \cup \{v_j\}$
    $E' \leftarrow E' \backslash \{e | v_j \in e\}$
Let $I = MaxDegreeGreedy(H(V', E'))$
Output I

Figure 4.6: The algorithm DETSPARSEHYPERGRAPH

Theorem 4.2.3 *The* DETSPARSEHYPERGRAPH *algorithm finds an independent set of size* $\Omega\left(\frac{pn}{\Delta^2}\right)$ *using* $O(1)$ *time to process each edge and* $O(n \log n)$ *space, where* p *is the smallest edge size in a given hypergraph.*

Proof: First, we estimate how much of working memory DETSPARSEHYPERGRAPH uses to store $H(V', E')$. For each of at most n edges in $H(V', E')$ we need $O(\log n)$ bits by the same argument as in 4.2.1. Thus, the total working space is $O(n \log n)$.

Next, we show that DETSPARSEHYPERGRAPH finds an independent set of size $\Omega\left(\frac{n}{\Delta^2}\right)$. Throughout the algorithm the hypergraph H always contains at least $n - \Delta$ edges, because we delete a vertex with all incident edges from H whenever the number of edges exceeds n and this deletion reduces the number of edges in E' by at most Δ. Since each vertex is of degree at most Δ, the hypergraph H always contains at least $\frac{p(n-\Delta)}{\Delta} = \frac{pn}{\Delta} - p$ vertices. The MAXGREEDY finds an independent set in hypergraphs of size at least $\frac{2n}{\Delta+1}$ as proven in Theorem 2.3.4. It means that DETSPARSEHYPERGRAPH finds an independent set of size $\frac{2p(n-\Delta)}{\Delta(\Delta+1)} = \Omega\left(\frac{pn}{\Delta^2}\right)$, which completes the proof. ■

Next, we consider a deterministic algorithm for unbounded-degree hypergraphs. The DETPARTITIONS algorithm works for unbounded-degree hypergraphs and uses $O(n^{c+1} c \log n)$ space. It knows the vertex set V apriori, and so it starts by splitting V into disjoint subsets, each of $c \log n$ nodes, where $c > 0$ is some constant. Then the algorithm stores only the edge within each partition. After seeing the whole stream, the algorithm finds a maximum independent set in each partition and outputs the largest independent set found.

ALGORITHM DETPARTITIONS ($V,\ c,\ E$)
INPUT: a set of vertices V, a constant c and a stream E of edges

Split V into sets $\{V_1, V_2, \ldots, V_{\frac{n}{c\log n}}\}$ each of $c\log n$ nodes
For each $j = 1$ to $\frac{n}{c\log n}$ do:
  $E_j \leftarrow \emptyset$
For each edge e in the stream E do:
  If $\exists\ V_j$ such that $e \subseteq V_j$:
    $E_j \leftarrow E_j \cup e$
For each $j = 1$ to $\frac{n}{c\log n}$ do:
  Let I_j be a maximum independent set in $H(V_j, E_j)$
Output $max_j\ I_j$

Figure 4.7: The algorithm DETPARTITIONS

Theorem 4.2.4 *The* DETPARTITIONS *algorithm approximates* IS *within* $O\left(\frac{n}{c\log n}\right)$ *factor using* $O(r)$ *time to process each edge and* $O(n^{c+1})$ *space, for any constant* $c > 0$.

Proof: Each partition $H(V_j, E_j)$ contains $c\log n$ and at most n^c edges (which is the number of all possible subsets over $c\log n$ vertices). By the same argument as in 4.2.1, we need $O(n^c \log n)$ bits to store each partition $H(V_j, E_j)$, and in total $O(n^{c+1})$ bits to store all partitions.

Let OPT be a maximum independent set in the stream $H(V, E)$. Let OPT' be the largest independent set over all partitions $\{H(V_j, E_j)\}$. By pigeonhole principle, $OPT' = \max_j |OPT \cap V_j| \geq \frac{\sum_j |OPT \cap V_j|}{n/c\log n} = \frac{|OPT|}{n/c\log n}$. It means that DETPARTITIONS approximates *IS* within $O\left(\frac{n}{c\log n}\right)$, which completes the proof. ■

4.3 Minimal Space Algorithms

We consider here two related questions. One is how well we can compute independent sets from a stream if the space allowed is the absolute minimum possible? The other relates to the situation where the algorithm is at all times constrained to maintain a valid independent set.

Any algorithm that finds a non-trivial feasible independent set must be able to output a solution. The simplest setup for a minimal space consumption is to allow the algorithm only one bit per vertex, recording whether the vertex is *in* the current solution or not. The amount of *additional* space can be restricted to a constant. Since there is no space to store

information about the edges previously seen, the assignment of nodes as being outside the solution must be irrevocable: no vertex can be reintroduced into the solution.

We build on these ideas to introduce the following model.

Definition 4.3.1 *In the* online streaming model*, the algorithm must maintain a specific feasible solution at all times. Initially, the solution consists of all vertices. Vertices can be removed from the solution, but cannot be added back.*

This model captures an irrevocability property that is similar to *online* algorithms. In the *online independent set* problem, vertices arrive one by one, along with all incident edges to previous vertices. The online algorithm must irrevocably determine whether the node is to be included in the constructed feasible independent set solution. In the above online streaming problem, nodes can change their state, but only in one direction: a selected vertex can be deselected, but once omitted from the current independent set, it can never be readmitted. The online problem is therefore incomparable, since edges are presented in a special order determined by the vertex ordering.

The online independent set problem is known to be very hard to approximate [28, 2]; e.g., a competitive factor of $n-1$ is the best possible for a deterministic algorithm, even when restricted to trees. However, bounded-degree graphs are comparatively easy, since a factor of Δ is trivial for a deterministic greedy algorithm.

We consider deterministic and randomized algorithms for the independent set problem in the online streaming model, and obtain matching upper and lower bounds in terms of degree parameters.

First, we give a lower bound for deterministic streaming algorithms.

Theorem 4.3.2 *The performance ratio of any deterministic algorithm in the online streaming model is $\Omega(n)$, even for trees of maximum degree $\log n$. This holds even if the algorithm is allowed to use arbitrary extra space.*

Proof: Assume that $n = 2^k$. Let A be any deterministic algorithm.

We maintain the invariant that the independent set selected by A contains at most one node in each connected component. We join the n vertices together into a single tree in k rounds. In round i, for $i = 1, 2, \ldots, k$, $n/2^i$ edges are presented. Each edge connects together two components; in either component, we choose as endpoint the node that is currently in A's solution, if there is one, and otherwise use any node in the component. This ensures that the algorithm cannot keep both vertices in its solution, maintaining the invariant.

In the end, the resulting graph is a tree of maximum degree at most k, and A's solution contains at most one node. ■

This result shows that no deterministic algorithm can attain a performance ratio in terms of $\overline{d}$ alone, nor a ratio of $2^{o(\Delta)}$.

We remark that a deterministic algorithm that is allowed no extra space has no means of distinguishing between vertices in a meaningful way. This implies that no non-trivial bound is then possible. We call the model *memoryless* if no additional space, beyond the n bits to denote the solution, are allowed.

Observation 4.3.3 *There is no deterministic memoryless online streaming algorithm that attains a performance ratio that is a function of Δ alone.*

When allowing additional space, we can match the previous lower bound in terms of Δ.

Theorem 4.3.4 *There is a deterministic algorithm in the online streaming model with a performance ratio of $O(2^{\Delta})$.*

Proof: We consider an algorithm that maintains additional information in the form of a counter c_v for each node v, initialized as zero.

When an edge arrives between two nodes in the current solution I, we compare the counters of the nodes. The node whose counter is smaller, breaking symmetry arbitrarily, is then removed from the current solution I. The counter of the other node, e.g. u, is then increased. We then say that u *eliminated* v. We say that a node u is *responsible* for a vertex x if u eliminated x, or, inductively, if u eliminated a node that was responsible for x.

Let $R(k)$ denote the maximum, for any node v with $c_v = k$, of the number of nodes for which v is responsible. We claim that $R(k) \leq 2^k - 1$. It then follows that the size of I is at least $n/2^{\Delta}$, since c_v is at most the degree of v. For the base case $R(0) = 0$, since the node never eliminated another vertex. Assume now that $R(t) \leq 2^t - 1$, for all $t < k$. Consider a node with $c_v = k$, and let $u_1, u_2, \ldots, u_k$ denote the vertices eliminated by k in order. On the i-th elimination, the value of c_v was $i - 1$, hence the value of c_{u_i} was at most $i - 1$. Once eliminated, the counter c_{u_i} for node u_i stays unchanged. Hence, by the inductive hypothesis, u_i was responsible for at most $R(i - 1)$ other nodes. We then have that

$$R(k) \leq \sum_{t=1}^{k} (R(t-1) + 1) = \sum_{t=0}^{k-1} 2^t = 2^k - 1,$$

establishing the claim. ■

We now turn our attention to randomized algorithms in the online model. The algorithm RANDOMDELETE given in Fig. 4.8 selects an endpoint at random from each edge in the stream and removes it from the current solution.

ALGORITHM RANDOMDELETE (E)
INPUT: a stream E of edges

$V \leftarrow S \leftarrow \emptyset$
For each edge e in the stream E do:
  For each $u \in e$ such that $u \notin V$
    $V \leftarrow V \cup \{u\}$
  Randomly select a vertex $v \in e$
  $S \leftarrow S \cup \{v\}$
Output $V \setminus S$

Figure 4.8: The algorithm RANDOMDELETE

Note that RANDOMDELETE is memoryless.

Theorem 4.3.5 *The performance ratio of* RANDOMDELETE *on graphs is* $2^{O(\overline{d})}$. *Any randomized memoryless algorithm in the online streaming model has performance ratio* $2^{\Omega(\overline{d})}$.

Proof: *Upper bound.* Each vertex v belongs to the final solution $V \setminus S$ with probability $2^{-d(v)}$. Therefore, the expected size of the $V \setminus S$ is $\sum_V 2^{-d(v)} \geq n/2^{\overline{d}}$, using the linearity of expectation and Jensen's inequality.

Lower bound. A memoryless algorithm can do no better than randomly select the vertex to be eliminated. It can improve on *RandomDelete* by not eliminating a vertex if its neighbors are already in the hitting set, but when both endpoints are in the independent set solution, it has no means for distinguishing the two.

Consider the graph with vertex set $V = \{v_1, v_2, \cdots, v_n\}$ and edges $\{v_i, v_j\}$ for any $|i - j| \leq k$. Edges arrive in the stream in lexicographic order: $(v_1, v_2), (v_1, v_3), \ldots, (v_1, v_k)$, $(v_2, v_3), \ldots, (v_{k+1}), (v_3, v_4)$, etc. Note, that all but the first and the last k vertices have degree $2k$. Thus, the average degree $\overline{d} \leq \Delta = 2k$.

Let I be the independent set constructed by the algorithm. Consider the first vertex v_1. There are two cases, depending on whether v_1 ends up in I.

Case 1: $v_1 \in I$. It means that all the neighbors of v_1 are deleted. The probability of this event is $P[v_1 \in I] = 2^{-k}$. The remaining stream is identical to the original one with $V = V \setminus \{v_1, v_2, \cdots, v_k\}$.

Case 2: $v_1 \notin I$. Suppose v_1 was selected to cover the t-th edge incident on v_1 for some $t \in [1,k]$. Then, the first $t-1$ neighbors of v_1 were selected to cover the first $t-1$ edges incident on v_1 and were deleted as well. The remaining stream is identical to the original one with $V = V \backslash \{v_1, v_2, \cdots, v_t\}$.

Thus, a vertex $v_i \in V$ is inserted in I only in Case 1 and the probability of this event is 2^{-k}, for any vertex v_i with $i \in [1, n-k]$. Note, that the last k vertices form a clique, and so only one vertex from this clique contributes to I. Thus, the expected size of the independent set found by the algorithm is $\frac{n-k}{2^k} + 1 < \frac{n}{2^k} + 1 < \frac{n}{2^{\overline{d}/2}}$. On the other hand, an optimal solution is of size at most $\frac{n}{\Delta} \geq \frac{n}{2k}$, which implies the performance ratio. ■

The algorithm has the special property of being *oblivious* in that the solution maintained, or any other part of memory, is never consulted in the operation of the algorithm until it is output.

Corollary 4.3.6 *The* RANDOMDELETE *algorithm attains a* $2^{\theta(\overline{d})}$ *performance ratio on graphs even in the* oblivious *model.*

The permutation-based algorithms of Section 3 are all of the online streaming type. Thus, we do know that reasonable approximation can be obtained by randomized online streaming algorithms if we have modest amount of extra space. On the other hand, if we have no extra space, Theorem 4.3.2 shows that the approximation is much worse. How much extra space is then really needed to be successful?

Our next result suggests that one extra bit per vertex is sufficient. For this to work, we need to know in advance the degree parameter ($\overline{D}$ or Δ).

Without loss of generality, we can assume that all vertices have positive weight and $w(v) \geq 1$ for any vertex v. Suppose we know $\overline{D}$ in advance (a constant factor approximation suffices), then we can use the following streaming algorithm (Fig. 4.9). The algorithm maintains a hitting set S. When we receive an edge e, for every new vertex $v \in e$ we sample $y \in [0,1)$ using the uniform random distribution on $[0,1)$, and if $y > 1/2\overline{D}$, we include v in S. If S does not contain any vertex from e, then we randomly select a vertex u in e and include it in S.

Theorem 4.3.7 *The* RANDOMSELECT *algorithm finds an independent set of expected weight* $\Omega\left(\frac{w(V)}{\overline{D}}\right)$*, using* $O(r)$ *time to process each edge and 2 bits per vertex total working space, i.e* $2n$ *bits.*

ALGORITHM RANDOMSELECT ($\overline{D}$, E)
INPUT: Average degree $\overline{D}$, a stream E of edges

$V \leftarrow S \leftarrow \emptyset$
For each edge e in the stream E do
  For each vertex $v \in e$ such that $v \notin V$
    $V \leftarrow V \cup \{v\}$
    sample $y_v \in [0, 1)$ using the uniform random distribution
    if $y_v > \frac{1}{2\overline{D}}$ then
    $S \leftarrow S \cup \{v\}$ [*Phase 1*]
  If $e \cap S = \emptyset$ then [*Phase 2*]
    Randomly select a vertex $v \in e$
    $S \leftarrow S \cup \{v\}$
Output $I = V \backslash S$

Figure 4.9: The algorithm RANDOMSELECT

Proof: For each vertex in the stream the algorithm uses only two bits of working memory: one to mark if the vertex has been seen in the stream, and one to mark if the vertex is in S or not. Thus, the algorithm needs $O(n)$ space to store the sets V, S.

Now, we show that the expected weight of $V \backslash S$ is $\Omega\left(\frac{w(V)}{\overline{D}}\right)$. A vertex v is in $V \backslash S$ after the termination of the algorithm if v was never included in S during the execution of the algorithm. The vertex v is included in S either when it first appears in the stream (Phase 1) or when v is selected to cover some edge containing it (Phase 2). We say that an edge e is *covered in Phase 1* if there is a vertex $v \in e$, which was included in S in Phase 1.

To simplify the analysis we change Phase 2 of the RANDOMSELECT algorithm as follows: instead of considering only edges e such that $e \cap S = \emptyset$, we consider all edges e not covered in Phase 1. Let S_{org} and S_{mod} be the covers found by the original and modified algorithms, respectively. Consider a vertex $v \in S_{org}$. If v is included in S_{org} in Phase 1, then v is included in S_{mod} in Phase 1 as well. If v is included in S_{org} in Phase 2, then v covers some edge e not covered in Phase 1, and so is included in S_{mod} in Phase 2. Thus, $S_{org} \subseteq S_{mod}$.

The probability of v not being included in S during the Phase 1 is:

$$P[v \notin S \text{ after Phase 1 }] = \frac{1}{2\overline{D}} . \tag{4.1}$$

Let X_v be an indicator random variable with $X_v = 1$, if v survives Phase 1, i.e. v is not included in S during Phase 1, and $X_v = 0$ otherwise. Then, the total expected weight of

vertices passed through the Phase 1

$$\sum_{v \in V} w(v) X_v = \frac{w(V)}{2\overline{D}} \, . \tag{4.2}$$

An edge is not covered in Phase 1 iff none of its vertices are in S during Phase 1, the probability of such an event is

$$P[e \text{ is not covered in Phase 1 }] = \frac{1}{\left(2\overline{D}\right)^{|e|}} \leq \frac{1}{\left(2\overline{D}\right)^{2}} \, . \tag{4.3}$$

For any edge e in the stream we define a weight function $w(e) = \sum\limits_{v \in e} w(v)$. The weight of all edges in the stream is then

$$w(E) = \sum_{e \in E} w(e) = \sum_{e \in E} \sum_{v \in e} w(v) = \sum_{v \in V} w(v) d(v) = w(V)\overline{D} \, . \tag{4.4}$$

Let Y_e be an indicator random variable with $Y_e = 1$, if e is not covered in Phase 1, and $Y_e = 0$ otherwise. Then, the total expected weight of edges not covered in Phase 1 is

$$\sum_{e \in E} w(e) Y_e \leq \frac{w(E)}{4\overline{D}^2} = \frac{w(V)}{4\overline{D}} \, . \tag{4.5}$$

In each edge that is not covered in Phase 1, the algorithm deletes one vertex. It means that on expectation the weight of deleted vertices in modified Phase 2 is at most $\frac{w(V)}{4\overline{D}}$. Thus, the total expected weight of vertices survived both Phase 1 and modified Phase 2 is at least $\frac{w(V)}{2\overline{D}} - \frac{w(V)}{4\overline{D}} = \frac{w(V)}{4\overline{D}}$, which concludes the proof. ■

4.4 Conclusions

In this section we present semi-streaming algorithms that approximate WIS in one pass using at most $O(r)$ time to process every edge in the stream. We describe several randomized and deterministic algorithms for bounded-degree and unbounded-degree hypergraphs, some of them are based on permutations and use at least $O(n \log r)$ and at most $O(n^2 \log n)$ space, the others are based on the partitions and use $O(n^{c+1})$ space, where $c > 0$ is an arbitrary constant. We also define an *on-line minimal space* streaming model and prove lower bounds for randomized and deterministic algorithms in this model.

Chapter 5

Conclusions

In this work we present approximation algorithms for the MAXIMUM INDEPENDENT SET problem, for both weak and strong variations of the problem, for weighted and unweighted cases. We introduce a shrinkage reduction technique, which allows us to extend several greedy, local search and partitioning graph algorithms to hypergraph case. We also describe SDP-based approach which combines semidefinite programming with greedy and randomized algorithms. This approach allows us to obtain the first $o(\Delta)$-approximation for IS in hypergraphs, matching the bound on graphs. Finally, we consider a semi-streaming model and give several deterministic and randomized algorithms for bounded and unbounded-degree hypergraphs. We introduce online an semi-streaming model and give lower and upper bounds on randomized and deterministic algorithms in this model.

To conclude we outline possible directions for further work:

- Show the capabilities and limitations of the shrinkage reduction technique, in particular classify optimization problems and algorithms to which this technique can be applied.
- Extend the SDP-based approach to coloring and covering problems on hypergraphs.
- Prove the results on greedy, local search, partitioning and SDP-based algorithms in terms of average or maximum hyperdegree.
- Improve the approximation ratios and the size of working space of semi-streaming algorithms.

Bibliography

[1] K.J. Ahn and S.Guha, Graph sparsification in the semi-streaming model, *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP)*, 5556: 328–338, 2009.

[2] N. Alon, U. Arad and Y. Azar, Independent sets in hypergraphs with applications to routing via fixed paths, Approximation, Randomization and Combinatorial Optimization (APPROX-RANDOM), 16–27, 1999.

[3] N. Alon and N. Kahale, Approximating the independence number via the θ-function, *Mathematical Programming* 80(3): 253–264, 1998.

[4] N. Alon, Y. Matias and M.Szegedy, The space complexity of approximating the frequency moments, *Journal of Computer and System Sciences* 58(1): 1167–1181, 1999.

[5] Z. Bar-Yossed, R. Kumar and D.Sivakumar, Reductions in streaming algorithms, with an application to counting triangles in graphs, *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 623–632, 2002.

[6] C. Bazgan, J. Monnot, V. Paschos and F. Serrière, On the differential approximation of MIN SET COVER, *Theoretical Computer Science*, 332:497–513, 2005.

[7] S. Ben-David, A. Borodin, R. M. Karp, G.Tardos and A. Wigderson, On the power of randomization in on-line algorithms, *Algorithmica*, 11:2–14, 1994.

[8] C. Berge, Hypergraphs, *North-Holland*, 1989.

[9] P. Berman, A $d/2$ approximation for Maximum Weight Independent Set in d-claw free graphs, *Nordic Journal of Computing*, 7:178–184, 2000.

[10] P. Berman and T. Fujito, On approximation properties of the Independent Set Problem for low degree graphs, *Theory of Computing Systems*, 32(2):115–132, 1999.

[11] P. Berman and M. Fürer, Approximating maximum independent set in bounded degree graphs, *Proc. 5th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 365–371, 1994.

[12] A. L. Buchsbaum, R. Giancarlo and J.Westbrook, On finding common neighborhoods in massive graphs, *Theoretical Computer Science*, 1-3(299):707–718, 2003.

[13] Y. Caro and Z. Tuza, Improved lower bounds on k-independence, *Journal of Graph Theory*, 15(1):99–107, 1991.

[14] J. Cardinal, S. Fiorini and G. Joret, Tight results on Minimum Entropy Set Cover, *Algorithmica*, 50(1):49–60, 2008.

[15] V. Chvátal, A greedy heuristic for the set-covering problem, *Mathematics of Operations Research*, 4(3):233–235, 1979.

[16] A. Clementi and L. Trevisan, Improved non-approximability results for vertex cover problems with density constraints, *Proc. 2nd Ann. International Conference on Computing and Combinatorics*, 332–342, 1996.

[17] C. Demetrescu, I. Finocchi and A. Ribichini, Trading off space for passes in graph streaming problems, *Proc. of 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 714–723, 2006.

[18] J. Edmonds, Paths, trees and flowers, *Canadian Journal of Mathematics*, 17:449–467, 1965.

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

[20] U. Feige, Approximating maximum clique by removing subgraphs, *SIAM Journal of Discrete Mathematics*, 18(2): 219–225, 2005.

[21] U. Feige, L. Lovász and P. Tetali, Approximating min-sum set cover, *Algorithmica*, 40(4):219–234, 2004.

[22] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri and J. Zhang, On graph problems in a semi-streaming model, *Theoretical Computer Science*, 348(2): 207–216, 2005.

[23] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M.Strauss, Quicksand: quick summary and analysis of network data, *DIMACS Technical Report*, 2001-43, 2001.

[24] M. X. Goemans and D. P. Williamson, Improved approximation algorithm for maximum cut and satisfiability problems using semidefinite programming, *Journal of the ACM*, 42: 1115–1145, 1995.

[25] M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, 1(2): 169–197, 1981.

[26] M. M. Halldórsson, Approximations of independent sets in graphs, *Approximation, Randomization and Combinatorial Optimization (APPROX-RANDOM)*, LNCS 1441:1–13, 1998.

[27] M. M. Halldórsson, Approximations of weighted independent set and hereditary subset problems, *Journal Graph Algorithms and Applications*, 4(1), 1–16, 2000.

[28] M. M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi, Online independent sets, *Theoretical Computer Science*, 289(2): 953–962, 2002.

[29] M. M. Halldórsson and H.-C. Lau, Low-degree graph partitioning via local search with applications to Constraint Satisfaction, Max Cut, and 3-Coloring, *Journal of Graph Algorithms and Applications*, 1:1–13, 1997.

[30] M.M. Halldórsson and E. Losievskaja, Independent sets in bounded-degree hypergraphs, *Discrete Applied Mathematics*, 157: 1773–1786, 2009.

[31] M. M. Halldórsson and J. Radhakrishnan, Greed is good: approximating independent sets in sparse and bounded-degree graphs, *Algorithmica*, 18(1):143–163, 1997.

[32] E. Halperin, Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs, *SIAM Journal on Computing*, 31(5):1608–1623, 2002.

[33] E. Hazan, S. Safra and O. Schwartz, On the complexity of approximating k-dimensional matching, *Approximation, Randomization and Combinatorial Optimization (APPROX-RANDOM)*, 59–70, 2003.

[34] J. Håstad, Some optimal inapproximability results, *Proc. 29th Ann. ACM Symposium on Theory of Computing (STOC)*, 1–10, 1997.

[35] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Mathematica*, 182: 105–142, 1999.

[36] M. Henzinger, P.Raghavan and S.Rajagopalan, Computing on data streams, *In: External Memory Algorithms, DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 50:107-118, 1999.

[37] C. A. J. Hurkens and A. Schrijver, On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems, *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[38] P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation, *Proc. 41th IEEE Symposium on Foundations of Computer Science (FOCS)*, 189–197, 2000.

[39] S. Jukna, Extremal combinatorics with applications in computer science, *Springer-Verlag*, 2001.

[40] D. S. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, 9:256–278, 1974.

[41] R. M. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations*, 85–103, 1972.

[42] D. Karger, R. Motwani and M. Sudan, Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2): 246–265, 1998.

[43] M. Krivelevich, R. Nathaniel and B. Sudakov, Approximating coloring and maximum independent set in 3-uniform hypergraphs, *Journal of Algorithms*, 41(1):99–113, 2001.

[44] L. Lovász, On decomposition of graphs, *Acta Mathematica Hungarica*, 18(3-4):359–377, 1967.

[45] L. Lovász, On the ratio of optimal integral and fractional covers, *Discrete Mathematics*, 13:383–390, 1975.

[46] A. McGregor, Finding graph matchings in data streams, *Approximation, Randomization and Combinatorial Optimization (APPROX-RANDOM)* 170–181, 2005.

[47] S. Mahajan and H. Ramesh, Derandomizing semidefinite programming based approximation algorithms. *SIAM Journal of Computing* 28(5): 1641–1663, 1999.

[48] S. Muthukrishnan, Data streams: algorithms and applications, *http://athos.rutgers.edu/~muthu/stream-1-1.ps*, 2003.

[49] I. Munro and M. Paterson, Selection and sorting with limited storage, *Theoretical Computer Science*, 12:315-323, 1980.

[50] R. Raz and S. Safra A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP, *Proc. 29th Ann. ACM Symposium on Theory of Computing (STOC)*, 475–484, 1997.

[51] A. Rényi, Probability theory, *Elsevier, New York*, 1970.

[52] R. Saigal, L. Vandenberghe and H. Wolkowicz, Handbook of semidefinite programming: theory, algorithms and applications, *Springer*, 2000.

[53] S. Sakai, M. Togasaki and K. Yamazaki, A note on greedy algorithms for the maximum weighted independent set problem, *Discrete Applied Mathematics*, 126(2-3):313–322, 2003.

[54] H. Shachnai and A. Srinivasan, Finding large independent sets of hypergraphs in parallel, *SIAM Journal on Discrete Mathematics*, 18(3):488–500, 2005.

[55] T. Thiele, A lower bound on the independence number of arbitrary hypergraphs, *Journal of Graph Theory*, 32:241–249, 1999.

[56] L. Trevisan, Non-approximability results for optimization problems on bounded degree instances, *Proc. 33rd Ann. ACM Symposium on Theory of computing (STOC)*, 453–461, 2001.

[57] V. V. Vazirani, Approximation algorithms, *Springer*, 2001.

[58] S. Vishwanathan, Private communications, mentioned in [27], 1998.

[59] V. Voloshin, Coloring mixed hypergraphs: theory, algorithms and applications, *AMS*, 2002.

[60] L. A. Wolsey, An analysis of the greedy algorithm for the submodular set covering problem, *Combinatorica*, 2(4):385–393, 1982.

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539