The intrinsic hardness of structure approximation

by

©Md Renesa Nizamee

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

M.Sc in Computer Science

Department of Computer Science

Memorial University of Newfoundland

April 2014

St. John's Newfoundland

Abstract

A natural question when dealing with problems that are computationally hard to solve is whether it is possible to compute a solution which is close enough to the optimal solution for practical purposes. The usual notion of "closeness" is that the value of the solution is not far from the optimal value. In this M.Sc thesis, we will focus on the generalization of this notion where closeness is defined with respect to a given distance function. This framework, named "Structure approximation", was introduced in 2007 paper by Hamilton, Müller, van Rooij and Wareham [HMvRW07], who posed the question of complexity of approximation of NP-hard optimization problems in this setting.

In this thesis, we will survey what is known about the complexity of structure approximation, in particular recasting results on Hamming approximation of NP witnesses in the Hamilton/Müller/van Rooij/Wareham framework. Specifically, the lower bounds in the NP witness setting imply that at least for some natural choices of the distance function any non-trivial structure approximation is as hard to compute as an exact solution to the original problem. In particular, results of Sheldon and Young (2013) state, in the language of structure approximation, that it is not possible to compute more than $n/2 + n^{\epsilon}$ bits of an optimal solution correctly in polynomial time without being able to solve the original problem in polynomial time. Moreover, for some problems and solution representations, even a polynomial-time algorithm for finding

$n/2 - O(\sqrt{n \ln n})$ bits of an optimal solution can be used to solve the problem exactly in polynomial time.

In addition to the lower bounds results, we will discuss algorithms and design techniques that can be used to achieve some degree of structure approximation for several well-known problems, and look at some traditional approximation algorithms to see whether the approximate solution they provide is also a structure approximation. We make a number of observations throughout the thesis, in particular extending Hamming approximation lower bounds to edit distance approximation with the same $n/2 + n^\epsilon$ parameters for several problems.

Finally, we apply these techniques to analyse the structure approximation complexity of the Phrase Alignment problem in linguistics, in particular Weighted Sentence Alignment (WSA) problem of DeNero and Klein [DK08]. We discuss several ways of defining a natural witness for this problem and their implications for its structure approximability. In a search of the most compact natural witness representation, we define a still NP-hard restriction of the WSA, a Partition Weighted Sentence Alignment, and show that the $n/2 + n^\epsilon$ lower bounds for the Hamming distance and edit distance apply in this case; this implies structure inapproximability of the WSA itself with somewhat weaker parameters.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Antonina Kolokolova for her continuous support throughout my graduate program. It would not have been possible for me to finish my research and thesis without her guidance and assistance.

I would also like to thank everyone from Department of Computer Science, Memorial University who have always been so kind and helpful to me throughout my graduate studies. A special thanks to Dr. Todd Wareham who was always been supportive to me and my research work.

Last but not the least, I would like to thank my family who is 7000 miles away from me, for providing me with their love, support and constant encouragement for all my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A computational problem is in the complexity class P when it is, colloquially speaking, computationally easy. That is it has a worst case deterministic polynomial time algorithm. A problem is in NP whenever, though it might be hard to find a solution, once found it can be verified in polynomial time. Whether every computational problem in NP can be solved in polynomial time, known as "P versus NP" problem, is a major unresolved problem in theoretical computer science. The Clay Mathematics Institute in Cambridge, MA, has named P versus NP one of its Millennium problems, and offers $1 million to anyone who provides a verified proof. The class NP contains decision versions of many important optimization problems occurring in practice. Some of them are NP-complete, meaning they are as hard as anything else in NP.

For example, suppose an advertising company on Facebook is trying to determine the biggest group of people who are friends with each other. To do that they need to solve a well known NP-hard problem called Clique (a problem is NP-hard whenever it is as hard as anything else in NP, but not necessarily in NP). Another beautiful problem

is called Knapsack: there, we are given a set of items and we have to choose the most profitable ones within a certain weight bound. This problem naturally occurs in transportation. Sending a spaceship to space needs a calculated decision of which items to take within a certain weight and size bound. It is not known how to solve these problems efficiently in the worst case, and it is believed to be hard since they are NP-hard. Sometimes the instances of problems occurring in practice are in some respect bounded, making them amenable to parameterized complexity techniques, or have enough structure for heuristics to work reasonably well. However, if we consider the non-restricted, general worst-case setting for these problems, we cannot rely on parameterized and heuristic methods for obtaining an exact solution. In this case, we may still be able to reliably compute a solution that is close to an optimal; an approximate solution.

What does it mean to find a solution close to the optimal one? There are many ways of looking at it. The standard way is to find something that would be almost as good as an optimal solution, when we are looking for a maximum clique in a graph we may be content with a clique that has a number of nodes not much smaller than the maximum. Standard notion of approximation is to get a solution that has *value* almost as good as the optimal. But sometimes these approximation algorithms are not exactly what we want.

A motivation from [HMvRW07, vRW12] was from cognitive psychology, where a natural task is to approximate belief systems. There, they do want approximated beliefs to share as many commonalities as possible with the actual beliefs in an optimal solution. In another paper [SY13], there was a beautiful example of the structure seen on a tomogram. A radiograph (x-ray) of a selected layer of the body is made by tomography. A tomogram is a two-dimensional image representing a slice or section through a three-dimensional object. We are interested in reconstructing the three-

dimensional structure of the object. Though given many x-ray images, the internal structure can be determined, it is NP hard to compute. But we are not interested in just something that resembles it superficially, we want to know what the structure of the solution is as much as possible. That is, we would like to construct an object with internal structure similar to that of the original object, as opposed to creating an object which produces similar two-dimensional images. Therefore, a different notion of approximation is needed here. This kind of approximation was introduced by Hamilton, van Rooij, Müller and Wareham [HMvRW07], who named it Structure approximation. Here, rather than focusing on the value of the solution, what we really want to know is the structure of the solution. More precisely, they define the notion of approximation in which there is an additional parameter, called the distance function. Rather than comparing the value of a candidate solution to the value of a optimal solution, they are looking at the distance between the two solutions according to that distance function. The notion of approximation then becomes finding the closest solution according the distance function. From the computer science perspective, the most natural function to consider is the Hamming distance between solutions as represented by binary strings. Most of the results we discuss concern the Hamming distance setting.

1.2 Previous work

Many results in [HMvRW07] and the follow-up paper [vRW12] are stated and proven for arbitrary distance functions or follow from the standard approximation results for distance metrics correlated with solution values. In particular, they show that there are no NP-hard problems with a structure analogue of FPTAS for an arbitrary function. Additionally, they state and prove a number of results with the distance function

chosen to be the Hamming distance. This is a very natural metric for comparing how "close" two solutions are, giving a sense of similarity of solutions. For example, in the Hamming approximation for Max-3SAT a solution close to the optimal would be considered a solution that differs from an optimal in few variable assignments, even if these variable assignments dramatically decrease the number of satisfied clauses.

Considering only the Hamming distance metric, there have been several other results, mainly providing lower bounds for such structure approximability of NP witnesses. The reconstruction of a partially specified witness, considered in the 1997 paper by Gal, Halevi, Lipton and Petrank [GHLP99], is probably the first result along these lines. There, they show that it is possible to reconstruct a satisfying assignment to a formula from $N^{1/2+\epsilon}$ bits of a witness to a related (though larger) formula. Their main proof technique relies on erasure codes, and in addition to SAT they consider Graph Isomorphism, Shortest Lattice Vector and Clique/Vertex Cover/Independent Set. In 1999, Kumar and Sivakumar [KS99] showed that for any NP problem there is a verifier with respect to which all solutions are far from each other with respect to Hamming distance: make the witnesses to be encodings of natural witnesses to the original problem by some error-correcting code, the verifier decodes the witness and then checks it using the original verifier. Then, list-decoding allows one to find a correct codeword for the witness from a string that is within $n/2 + n^{4/5+\gamma}$ Hamming distance from it. Following this, Feige, Langberg and Nissim [FLN00] show that some natural verifiers (e.g., binary strings directly encoding satisfying assignments for variants of SAT, encoding sequences of vertices for Clique/Vertex Cover, etc) are often hard to approximate to within Hamming distance $n/2-n^{\epsilon}$ for some ϵ dependent on the underlying error-correcting code. Guruswami and Rudra [GR08] improve this ϵ to $2/3 + \gamma$, but on the negative side argue that methods based on error-correcting codes can only give bounds up to $n/2 - O(\sqrt{n \log n})$.

The recent paper of Sheldon and Young [SY13] settles much of the Hamming distance approximation question, providing the lower bounds of $n/2 - n^\epsilon$ for any ϵ for many of the problems considered in [FLN00], improving on upper bounds of $n/2$ for several natural problems including Vertex Cover, and giving a surprising $n/2 + O(\sqrt{n \log n})$ lower bound for the universal NP-complete language. The latter result they extend to existence of such very hard to approximate verifiers for all paddable (in Berman-Hartmanis [BH77] sense) NP languages, improving on [KS99]. Their proof techniques avoid error-correcting codes altogether, relying instead on search-to-decision (Turing) reductions. For example, the $n/2 - n^\epsilon$ bound for SAT is achieved by showing how to determine the value of the first variable correctly assuming the existence of such approximation algorithm: add enough copies x_i of the first variable x, together with clauses $x_i = x$, to overwhelm the n^ϵ factor, and recover the correct value for x from the approximate solution by taking majority over x_i's. The proof of the better-than-$n/2$ bound for the universal language relies on downwards self-reducibility as well, though the algorithm is somewhat more involved. Most of the lower bounds results of [SY13] have a slight generalization to randomized algorithms with good parameters. We describe these results in more detail in chapter 3.

1.3 Our results

In this thesis, we summarize existing results about structure approximation, in particular recasting `NP` witness results in the structure approximation framework, as well as make a number of observations. In particular, we generalize some Hamming distance approximation results to the edit distance function, where *edit distance* $d_E(y, z)$ between strings y and z is the number of insert, replace and delete a symbol operations needed to convert y into z. We also discuss upper bounds for structure approxima-

tion, in particular noting that rounding-based Linear Programming algorithms such as the famous linear programming algorithm for Weighted MinVertexCover do provide a structurally close approximation with respect to Hamming distance. This suggests Linear Programming framework as a potential toolbox for designing structure approximation algorithms.

Additionally, we consider one problem from an applied area, linguistics, and analyse its structure approximation complexity. The problem we have studied is the phrase alignment (or, more precisely, weighted sentence alignment, WSA) problem. As approximating the structure of a witness depends on witness representation, we look at the various ways to represent a solution to the weighted sentence alignment problem. We define a restricted (albeit still NP-hard) version of WSA, which has a compact solution representation, and show that $n/2 - n^\epsilon$ lower bounds for Hamming distance and edit distance of SAT translate into the same lower bounds for this problem. For WSA itself this gives us somewhat weaker bounds of $n/3 - n^\epsilon$. A paper based on these results together with the edit distance structure approximation has been accepted for presentation at the CiE'2014 conference.

The organization of the chapters are as follows. Chapter 2 cover preliminaries and value approximation framework, lower bounds for structure approximation are presented in chapter 3, upper bounds in chapter 4, and the application to the phrase alignment problem in chapter 5. The concluding chapter contains discussions and future directions.

Chapter 2

Various approaches to approximation

There are many important combinatorial problems for which no efficient algorithm for computing the optimal solution is known. Many of these problems are NP-hard, and thus no polynomial-time algorithm can solve them optimally unless P=NP .

Definition 1. *The class* P *of polynomial-time decidable languages is* $\mathtt{P} = \bigcup_{k \geq 1} Time(n^k)$*, k is a constant.*

Here, if $f : \mathbb{N} \to \mathbb{N}$ is a function, then $Time(f(n)) = \{L|$there exists a Turing machine M which decides if $x \in L$ in at most $f(n)$ for $n = |x|$ steps for every possible input $x\}$. So $f(n)$ bounds worst-case running time of these Turing machines over all inputs of size n.

P is known to contain many natural problems, including the decision versions of linear programming, calculating the greatest common divisor, and finding a maximum matching.

Definition 2. *Let $L \subseteq \Sigma^*$. We say $L \in$* NP *if there is a two-place predicate $R \subseteq$*

$\Sigma^* \times \Sigma^*$ *such that* R *is computable in polynomial time, and such that for some* $c, d \in \mathbb{N}$ *we have for all* $x \in \Sigma^*$, $x \in L \Leftrightarrow \exists y \in \Sigma^*$, $|y| \leq c|x|^d$ *and* $R(x, y)$.

Definition 3. *[AB09] We say a language* $A \subseteq \{0,1\}^*$ *is a polynomial-time (Karp) or many-one reducible to a language* $B \subseteq \{0,1\}^*$, *denoted by* $A \leq_p B$, *if there is a polynomial-time computable function* $f : \{0,1\}^* \to \{0,1\}^*$ *such that for every* $x \in \{0,1\}^*$, $x \in A$ *if and only if* $f(x) \in B$. *We say that* B *is* NP-hard *if* $A \leq_p B$ *for every* $A \in$ NP. *We say that* B *is* NP-complete *if* B *is* NP*-hard and* $B \in$ NP.

It is easy to see that P $\subseteq$ NP, since $R(x, y)$ can ignore y and just run the polynomial-time algorithm to check if x is in the language .

Definition 4. *Decision problems are a class of computational problems with yes/no answer, or, equivalently the class of languages over a given alphabet.*

Both P and NP are classes of decision problems. For example, determining whether a graph is complete is in P (given an undirected graph, determine if every edge is present: $O(n^2)$ algorithm achieves that). However, a problem of determining if there is a subgraph of size k, which is a complete graph, -also called the Clique problem (such a subgraph is called a clique)-is NP-complete. Here, the inputs are an undirected graph G and a size parameter k. In both of these cases, the output of the algorithm is either a Yes or a No.

Definition 5. *In an* NP *search problem, given an input* $x \in \{0,1\}^*$ *the goal is to compute an answer* $y \in \{0,1\}^*$ *such that* $R(x, y)$ *from the definition of the language holds, if such a* y *exists.*

For example, the Clique search problem would be to return a set of vertices of size k that form a clique in the input graph.

Definition 6. *An* optimization problem *(either maximization or minimization) is a variant of a search problem in which there is an additional requirement that the y returned by the algorithm not only has to satisfy $R(x, y)$ (be a candidate solution), but also has to be optimal (minimal or maximal as stated in the problem definition) according to some value function $val(x, y)$.*

Maximum Clique problem (Max-Clique) could be a common example of an NP-hard optimization problem: given a group of vertices some of which have edges between them, the maximum clique is the largest subset of vertices in which each point is directly connected to every other vertex in the subset. In the decision version of the problem, Clique, given a parameter k, we are asking if there is a clique with $\geq k$ vertices. By contrast, in the optimization version of this problem, Max-Clique, we are given a graph and asked to output the actual set of vertices comprising the largest-size clique. Given such a set, we can check that it is a clique in polynomial time by the same algorithm that checks if a graph is complete; however, we don't know how to check that this is a largest such subset. Max-Clique is a maximization problem; every clique in the graph is a candidate solution, and the value function is the number of vertices in the clique.

When we want to solve a problem in "real life" we are usually interested in finding a solution, not just knowing if one exists. Yet for the definitions of complexity classes we have always formulated problems in terms of decision, meaning we pose a question and want to know whether the answer is "yes" or "no". In fact, for every NP problem we consider, there is a corresponding search problem. However, if P = NP then any NP search problem can be solved in polynomial time, and for most problems there are polynomial-time search-to-decision reductions (that is, a search problem can be solved with polynomially many calls to the decision problem solver).

2.1 Approximation algorithms

A *heuristic* model is a computational method that uses trial and error methods to approximate a solution for computationally difficult problems. Heuristics usually do not provide a guarantee that a computed answer is close to an optimal solution, or that they will finish running in polynomial time, however they are typically simple and can be empirically useful.

This method can be useful for problems we don't care about the optimality of the solution, rather we only care about the speed at which the solution is produced.

At this instant the question that arises is; what should we do when difficult problems come along, for which we don't expect to find polynomial-time algorithms, and yet want some guarantee of the accuracy of the solution? Algorithms that have exponential running time are not effective, if the input size is not really small. As a result, we need to look for an approximate solution, since the optimal solution is not within our reach. Moreover, we would like to have a guarantee of being close to it. As an example, we can have an algorithm for Euclidean TSP that always develops a tour whose length is at most a factor ρ times the minimum length of a tour, for a small value of ρ. An algorithm that produces a solution, which is ensured to be within some factor of the optimum, is called an *approximation algorithm*. See the formal definition in the next section.

Historically, approximation algorithms were first discovered even before the notion of NP-completeness came up. Vazirani [Vaz01] credits Vizing [Viz64] (minimum edge coloring problem), Graham [Gra66] (minimum makespan problem), and Erdös (MAX-CUT problem) for being the first to come up with the concept of approximation algorithms, though the first approximation algorithm is often credited to Graham [Gra66]. The notion of an approximation algorithm was formally introduced by Garey, Graham, and Ullman [MGU72] and Johnson [Joh73].

As mentioned earlier, NP-hard problems have no polynomial time algorithms unless P = NP and, to find a solution to those problems one possible way is to use an approximation algorithm that will return a solution within some factor of the optimum or exact solution. Approximation algorithms allow finding solutions to some NP-hard industrial problems fast and with some guarantee of closeness to the optimal.

So, the performance of an approximation algorithm is an important issue. While it is assumed that every approximation algorithm is polynomial-time, there exist a lot of polynomial algorithms that can be inefficient and impractical. So, in order to get a better approximation bound one can invest more running time to the algorithm. From this trade-off between approximation result and running time the notion of approximation schemes arise[Hoc97].

PTAS (polynomial time approximation schemes) and FPTAS (fully polynomial time approximation schemes) are common types of those approximation schemes. As we may guess the better the approximation, the larger may be the running time. A problem has a PTAS if the running time is polynomial in the size of the input and a problem has a FPTAS if the running time is polynomial both in size of the input and the inverse of the performance ratio $(1/\epsilon)$ [ACG$^+$99]. So, the classes of problems that have FPTAS are contained in classes of problems that have PTAS, with the FPTAS being the best possible type of an approximation algorithm.

The classical notion of approximation is that of finding a solution with a value close to the optimal. Many problems have such algorithms, though some cannot be approximated that well in this context. More precisely, we define value-approximation as follows, following [HMvRW07].

Definition 7 (Value-approximation algorithm)**.** *If the value of a solution produced by an algorithm A on input x, $val(x, A(x))$, is always within some predefined absolute or relative factor of the optimal solution value on input x, $optval(x)$, then we refer to A*

as value-approximation algorithm. More formally, a value-approximation problem π consists of a 4-tuple: the set of valid inputs (instances) I, a set of candidate solutions cansol, a value function $val(x, y)$ and a "goal", specifying whether the value function has to be minimized or maximized.

Some variants of value-approximation are defined as follows:

Definition 8 (Additive value-approximation (v-a-approx) algorithm)**.** *Given an optimization problem π and a non-decreasing function $h : N \to N$, an algorithm A is a polynomial time $h(|x|)$ additive value-approximation (v-a-approx) algorithm for π if for every instance x of π, $|optval(x) - val(x, A(x))| \leq h(|x|)$ and A runs in time polynomial in $|x|$.*

Definition 9 (Ratio value-approximation (v-r-approx)algorithm)**.** *Given an optimization problem π and a non-decreasing function $h : N \to N$, an algorithm A is a polynomial time $h(|x|)$ ratio value-approximation (v-r-approx)algorithm for π if for every instance x, $R(x, A(x)) \leq h(|x|)$, where $R(x, A(x)) = \frac{optval(x)}{val(x,A(x))}$ (if π is a maximization problem) or $\frac{val(x,A(x))}{optval(x)}$ (if π is a minimization problem), and A runs in time polynomial in $|x|$.*

2.2 Complexity of value approximation

How good can an approximation algorithm be? Different problems have different approximation ratios (some have PTAS while others have constant-fraction lower bounds). For example, MAX-3SAT cannot have better than 7/8-approximation unless P = NP, whereas Knapsack can be approximated arbitrarily well. The types of techniques that have been successful for developing approximation algorithms include greedy algorithms, dynamic programming and linear/semidefinite programming.

2.3 Representative problems

In this section, we will list the definitions of the problems that will be used throughout the rest of this thesis. The lower and upper bounds shown here are cited from the compendium of NP optimization problems (http://www.nada.kth.se/∼viggo/problemlist/) and [ACG+99].

Problem 1 (Clique and Max-Clique)**.** *A clique in an undirected graph $G = (V, E)$ is a subset of the vertex set $C \subseteq V$, such that for every two vertices in C, there exists an edge connecting the two. The Clique decision problem $Clique(G, k)$ is, given a graph G and an integer k, to find if G contains a clique of size at least k. The optimization problem $Max - Clique(G)$ is to find a clique in G with maximum number of vertices. Max-Clique is approximable within $O(|V|/(log|V|)^2)$ but not approximable within $|V|^{1/2-\epsilon}$ for any $\epsilon > 0$.*

Problem 2 (Independent set and Max-IndependentSet)**.** *An* independent set *in a graph G is a set $I \subseteq V$ of vertices of G such that for all $u, v \in I, (u, v) \notin E$. The decision problem $IndependentSet(G, k)$ is to check whether G contains an independent set with at least k vertices; the optimization problem $Max - IndependentSet(G)$ asks to return an independent set with the maximum number of vertices.*
As Clique and IndependentSet are closely related (a clique in a graph is an independent set in its complement), the approximation ratio of Max-IndependentSet is the same as for Max-Clique.

Problem 3 (Vertex cover and Min-VertexCover)**.** *Given an undirected graph $G = (V, E)$, a subset $A \subseteq V$ is a* vertex cover *of G if every edge in G has at least one endpoint in A. Finding whether graph G has a vertex cover of size at most k is the VertexCover decision problem; finding a vertex cover of minimum size is the optimization problem Min-VertexCover. VertexCover is closely related to both Clique and*

IndependentSet problem.

Minimum vertex cover is approximable within $2-\frac{loglog|V|}{2log|V|}$ *and* $2-\frac{2lnln|V|}{ln|V|}(1-o(1))$.

Problem 4 (Ham-Path and LongestPath)**.** *A* Hamiltonian path *is a path that passes once and exactly once through every vertex of graph G. Determining whether a Hamiltonian path exists in undirected (sometimes directed) graph G is called the* Ham-Path *problem.*

The related optimization problem is the longest path problem*, that is the problem of finding a simple path of maximum length in a given graph. A path is called simple if it does not have any repeated vertices; the length of a path may either be measured by its number of edges, or (in weighted graphs) by the sum of the weights of its edges.*

Problem 5 (Max-Cut and Weighted Max-Cut)**.** *In graph theory, a* cut *is a partition of the vertices of a graph into two disjoint subsets. A size of a cut is the number of edges between vertices in different subsets; in a weighted graph, sum of weights of these edges. The Max-Cut optimization problem (respectively, Weighted Max-Cut) asks to return a cut with the maximum number of edges (respectively, with the maximum sum of weights of edges) crossing the cut. The decision version asks for the existence of a cut of size (weight) at least k for a given parameter k.*

Max-cut is approximable within 1.1383.

Problem 6 (Knapsack)**.** *Given a set of items, each with a weight and a value, determining whether or not to include each item in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible is called a* knapsack *problem. That is given* $\{(w_1, p_1)....(w_n, p_n)\}, B$*, find* $S \subset \{1, ..n\}$ *such that* $\sum_{i \in S} w_i \leq B$ *and* $\sum_{i \in S} P_i$ *is maximized; the decision version is testing for an existence of a solution with the value at least a given parameter.*

Knapsack has an FPTAS. Note, a problem has FPTAS if the running time is polynomial both in size of the input and the inverse of the performance ration$(1/\epsilon)$*.[ACG*$^+$*99].*

Problem 7 (SAT, 3-SAT, MAX 3-SAT; NAE-SAT, NAE-3SAT)**.** *A propositional formula (that is, a formula where all the variables have values true or false) is in CNF(Conjunctive normal form) if it is a conjunction (AND) of disjunctions, called clauses, of literals (variables or their negations).* SAT *is the problem of deciding if a propositional formula has a satisfying assignment (an assignment of true/false values to variables that makes the formula true).*

The version of the SAT problem in which every clause has 3 literals is called 3-SAT*. 3-SAT is called a* MAX 3-SAT *if given a 3-CNF formula ϕ (i.e. with at most 3 variables per clause), the task is to find an assignment that satisfies the largest number of clauses.*

In NAE-SAT, a satisfying assignment requires to have a special condition, namely that for every clause not only one variable is set to be true by this assignment, but also one variable per clause is set to false. NAE-3SAT is the special case of NAE-SAT where each clause has exactly 3 variables.

Problem 8 (Weighted sentence alignment (WSA) and Max-WSA)**.** *A sentence e is represented by a set $\{e_{ij}\}$ of spans from between-word positions i to j in e; f is represented by $\{f_{kl}\}$ in the same fashion. A link is an aligned pair of phrases (e_{ij}, f_{kl}). In a weighted setting, there is an additional function $\phi : \{(e_{ij}, f_{kl})\} \to R$ assigning a weight to each link. A total weight of an alignment is a product of the weights of its links. Given (e, f, ϕ), the* WSA *problem is to decide if there is an alignment a with $\phi(a) \geq 1$. The optimization* Max-WSA *problem is to find a highest weight alignment.*

2.4 Techniques for proving value inapproximability

While it is not easy to design approximation algorithms for most of the NP-hard optimization problems, it is a difficult job to show that computing approximate solutions is also hard. The PCP theorem, which was discovered in 1992 by Arora, Safra [AS92] and Arora, Lund, Motwani, Sudan and Szegedy [ALM+92], gave a new definition of NP and is useful for proving hardness of approximation results. In particular, it shows that for many NP-hard optimization problems, computing an approximate solution is as hard as computing the exact solution(unless P = NP). [AB09]

The PCP theorem has two views. The first one characterizes the NP problems by defining the PCP class in terms of prover-verifier games and the second one deals with the hardness of different NP-hard optimization problems. We will briefly talk about these two views in the following.

Recall how NP was defined in Definition 2 on page 7. Arora and Safra [AS92] used a generalization of the definition of NP to define the class PCP, consisting of languages that have witnesses that can be checked by a probabilistic verifier that has oracle access to the membership proof, is allowed to use r random bits, and allowed to query q bits from the oracle.

Definition 10. *A verifier V is an (r, q)-restricted verifier if for any input x, witness w, and random string π of length $O(r)$, the decision whether $V^w(x, \pi)$ = "yes" is based on at most $O(q)$ bits from the witness w.*

Now we will give the definition of the PCP class.

Definition 11. *A language L belongs to the class $PCP_{c,s}[r, q, d]$ if there exists an (r, q, d)-restricted verifier V with the properties that*

1. *For $x \in L$, $Pr_\rho[V^\pi$ accepts $(x, \rho)] \geq c$ for some π.*

2. *For $x \notin L$, $Pr_\rho[V^\pi$ accepts $(x, \rho)] < s$ for all π.*

where π is the random string of length r. Here, d is the alphabet size of the input. If we only consider binary string then $d = 2$. When d is omitted, assume binary alphabet.

We call c the completeness and s the soundness of the verifier. When $c = 1$, we say that the verifier has perfect completeness.

The following relation of NP with PCP is from [ALM+92]. It proves that membership for NP-languages can be probabilistically checked by a verifier which uses logarithmic amount of randomness. It always accepts a correct proof ($c = 1$) and rejects incorrect proofs with probability at least $1/2$ ($s = 1/2$).

Theorem 1. *[ALM+92][AS92]*
$\mathtt{NP} = \mathtt{PCP}[O(\log n), O(1)]$.

Another view of the PCP theorem, which deals with the hardness of approximation shows that for many NP hard problems finding a good approximate solution (where the notion of "good" is different for different problems) is as hard as finding an exact solution. Here, we will show an example for Max-3SAT (see Section 2.2 for definition), for which until 1992 it was not known whether it has a polynomial time approximation algorithm or not. The PCP theorem shows that it does not have such algorithm (unless $\mathtt{P} = \mathtt{NP}$). [AB09] state this as follows:

Theorem 2. *(PCP theorem: Hardness of Approximation view)[AB09]*
There exists $\rho < 1$ such that for every $L \in$ NP there is a polynomial-time function f mapping strings to (representations of) $3CNF$ formulae such that:

1. $x \in L \Rightarrow val(f(x)) = 1$

2. $x \notin L \Rightarrow val(f(x)) < \rho$.

The above theorem implies a corollary that says that if for some constant $\rho < 1$ there exists a polynomial time approximation algorithm for Max-3SAT, then P = NP.
This motivates the notion of gap problems.

Definition 12. *Let $\rho \in (0, 1)$. The ρ-GAP 3SAT problem is to determine, given a 3CNF formula π, whether:*

1. *π is satisfiable, in which case we say π is a YES instance of ρ-GAP 3SAT.*

2. *$val(\pi) \leq \rho$, in which case we say π is a NO instance of ρ-GAP 3SAT.*

An algorithm A is said to solve ρ-GAP 3SAT if $A(\pi) = 1$ if π is a YES instance of ρ-GAP 3SAT and $A(\pi) = 0$ if π is a NO instance.
The PCP theorem implies hardness of approximation results for many more problems, in particular Max-IndependentSet and Min-VertexCover:

Theorem 3. *There is some $\gamma < 1$ such that computing a γ-approximation to Min-VertexCover is NP-hard. For every constant $\rho < 1$, computing a ρ-approximation to Independent set is NP-hard.*

A proof of a weaker version of the PCP theorem showing that every NP statement has an exponentially-long proof that can be locally tested by only looking at a constant number of bits can be done using error correcting codes such as Walsh-Hadamard code.

2.5 Structure Approximation

Though the standard notion of approximation is the value-approximation, where a solution is defined as a good one if it is close enough to an optimal solution in its value,

this notion is not the only way to define approximation. Indeed, for some applications, as discussed in the introduction, a different notion is more appropriate. A general such notion, called “structure approximation", was introduced by Hamilton, Müller, van Rooij and Wareham in their paper "Approximating solution structure" [HMvRW07]. There, they allow an arbitrary function, given as part of the description of a problem, to denote the distance between candidate solutions. For example, a solution can be considered a good approximation if it is within a small Hamming distance of some optimal solution, or has a small Euclidean distance to an optimal solution when solutions are represented as a point in space. Note that value-approximation is a special case of structure approximation, with the distance function being the ratio between solution values. Naturally, for different distance functions the complexity of structure approximation could be quite different.

The following definitions are due to [HMvRW07]. Note that in this definition the distance function is required to be integer-valued.

Definition 13 (Solution distance(sd) function)**.** *A* solution distance function *(sd-function) is a function* $d : \Sigma^* \times \Sigma^* \to \mathbb{N}$ *associated with an optimization problem* Π *such that for an instance* x *of* Π *and candidate solutions* $y, y' \in cansol(x)$*,* $d(y, y')$ *is the distance between these solutions.*

We assume, following [HMvRW07], that the sd-function d is a always a metric. That is, it has the following four properties:

1. For all $y, d(y, y) = 0$
2. For all distinct y and y', $d(y, y') > 0$
3. For all y and y', $d(y, y') = d(y', y)$ (symmetry)
4. For all y, y' and y'', $d(y, y') \leq d(y, y'') + d(y'', y')$ (triangle inequality)

Definition 14 (Structure-approximation algorithm)**.** *Given an optimization problem* Π*, a sd-function* d*, and a non-decreasing function* $h\colon \mathbb{N} \to \mathbb{N}$*, an algorithm* A *is a polynomial-time* $h(|x|)/d$ structure-approximation(s-approx) algorithm *if for every instance* x *of* Π*,* $d(A(x), optsol(x)) \leq h(|x|)$ *and* A *runs in time polynomial in* $|x|$*.*

Most of the results presented in [HMvRW07, vRW12] concern Hamming distance as a distance function in the structure approximation setting. To our knowledge, this is the only sd-function that has been studied in other literature, albeit in a different context and in the following section we will survey the known results about the complexity of Hamming distance approximation. However, [HMvRW07] also considered several other functions (in particular, edit distance, characteristic function with 1 on the optimal solution and 0 on all others, and several constructed examples), and proved a number of results about the general setting. We will discuss their results in the corresponding sections of the next two chapters.

Definition 15 (Hamming distance)**.** *The* Hamming distance *between two strings of equal length is the number of positions at which the corresponding symbols are different.*

[HMvRW07, vRW12] asks for actual solutions, as opposed to strings close to the optimal. So inapproximability results for strings or codes still hold, but positive results might not. In [HMvRW07, vRW12] settings, an algorithm must return a valid (though not necessarily optimal) solution, and most NP-witness papers allow arbitrary strings. Although this does not matter for some problems (e.g., SAT), it makes a difference for problems like Max-Clique, where to be a candidate solution a set of vertices must form a clique, rather than just be a set.

[HMvRW07] also define analogues of value-approximation and a notion of structure approximability preserving reduction that can be used to compare languages by hardness of structure approximability.

In the next section we will show an example of a problem with a good value-approximation algorithm, which however fails to achieve any significant structure approximation.

2.6 Example: Knapsack

Let us look at a specific example of the Knapsack problem, defined on on page 14 . We will show that existing value approximation algorithms for this problem do not give a good structure approximation.

Example 1. *Let us consider a scenario where a burglar is trying to steal something from a jewelery shop. The burglar has a Knapsack and that can carry a certain weight, B=12.*

The jewelery shop has some expensive items for the burglar to choose from. Each item is associated with a certain weight and profit.

Weight and profit breakdown for each item

	Weight	*Profit*
Item A	*5*	*2*
Item B	*2*	*3*
Item C	*3*	*4*
Item D	*6*	*5*

The following dynamic programming algorithm determines the value of the best solution to the Knapsack problem:

- Let $MaxP$ = profit of most valuable item; in the example above 5 (item D).
- Set $n*MaxP$ = upper bound on the profit that can be achieved by any solution.
- For each $i \in 1, ..., n$ and $p \in 1, ..., nP$, let $S_{i,p}$ denote a subset of $1, ..., i$ whose

total profit is exactly p and whose total weight (that is, sum of weights of elements of $S_{i,p}$) is minimized.

- Let $A(i,p)$ denote the weight of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

$$A(i+1,p) = \begin{cases} \min\{A(i,p), w_{i+1} + A(i,p-p_{i+1})\}, & if p_{i+1} \leq p \\ A(i+1,p) = A(i,p) & otherwise \end{cases}$$

See the resulting solution table for the problem in the example 1 below.

Item#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	∞	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	5	2	∞	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	5	2	3	7	8	5	∞	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	5	2	3	6	8	5	8	9	13	14	11	∞	16	∞	∞	∞	∞	∞	∞

Table 2.1: Knapsack: solution table

As the burglar could carry items with weight less or equal to 12, the solution table shows that the burglar can take the items weighing up to 11, will be most profitable for him. To find this number, scan the last line of the table backwards from $n \times maxP$ until a cell with the value smaller than the capacity is found. To find the actual set, use the algorithm below:

Let $A[i,P]$ be the maximum value of items that can be placed in the Knapsack.

Let $n = i$ and $m = P$

$S = \emptyset$

for $n = i$ **downto** 1

  if $A[n,m] \neq A[n-1,m]$ **then**

    $S = S \cup A_n$

    $m = m - w_i$

return S

Coming back to our example, it is easy to see that the items that can be placed in the Knapsack with maximum profit achieved are items B, C and D.

It can easily be seen that if the profits of objects were small numbers that is polynomial in n, the algorithm above is a polynomial algorithm. However, if the profits are exponentially large, in that case the running time of the algorithm will be exponentially big making it infeasible. However, this algorithm can be turned into an FPTAS for Knapsack; we will follow [Vaz01] in the description of the algorithm below.

The idea of the FPTAS is to use rounding to make the profits of the objects smaller than the original so that the modified profits can be viewed as numbers bounded by a polynomial in n and $1/\epsilon$. This will also ensure the profit is at least within (1-ϵ) of the optimum solution. The FPTAS algorithm for KNAPSACK is given below:

- Given $\epsilon > 0$, let $K = \dfrac{\epsilon MaxP}{n}$
- For each object i, define $p'_i = \left\lfloor \dfrac{p_i}{K} \right\rfloor$
- Using these as profits of objects, apply dynamic programming algorithm above to find the most profitable set, S'.
- Output S'.

Example 2. *Suppose 4 items are given each with a profit and weight given below, and $B = 12$. Suppose also that we want to find a solution within 0.5 of the optimum solution. So, $\epsilon = 0.5$.*

Weight and profit breakdown for each item

	Weight	*Profit*
Item A	*3*	*500*
Item B	*6*	*750*
Item C	*4*	*450*
Item D	*2*	*1000*

Here,

$MaxP$=*1000* $K = \frac{0.5 \times 1000}{4} = 125$

With this K *we will modify the actual profit.*

Weight and Modified profit

	Weight	*Profit*
Item A	*3*	*4*
Item B	*6*	*6*
Item C	*4*	*3*
Item D	*2*	*8*

Now using the above breakdown for each item, using the dynamic programming will give us the approximate solution for this KNAPSACK problem.

Item#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	..	32
1	∞	∞	∞	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	∞	∞	3	∞	6	∞	∞	∞	9	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	4	3	∞	6	7	∞	10	9	∞	∞	13	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	4	3	∞	6	7	2	10	9	6	5	13	8	∞	∞	12	11	∞	∞	15	∞	∞

Table 2.2: Knapsack: solution table for the newly obtained profits

From this solution table it can easily be seen that the maximum modified profit one can get while staying within weight bound B=*12 is 18. So, by using the algorithm shown in the previous example, we get that the optimal solution consists of items* A*,* B *and* D*.*

The algorithm described above gives a very good approximation of the value of the optimal solution. However, would it translate into the structure of the solution being similar, that is, would the set computed by the FPTAS algorithm have a large overlap with the optimal solution set? Below we present a counterexample showing that such approximate solution can be completely disjoint from the optimal solution.

Example 3. *Consider an instance of Knapsack consisting of* $2n$ *elements* $\{a_1, a_2, ..., a_n\}$ *and* $\{b_1, b_2, ..., b_n\}$ *with weights* $w(a_i) = w(b_i) = 2^i$*; let the capacity be* $B = 2^{n+1} - 1$*.*

Thus, an optimal solution would contain exactly one of a_i *or* b_i *for any* i*. Now, set* $a_i s$ *to have profit* $p_{a_i} = 2^{2n-i}$ *and* $b_i s$ *to have profit* $p_{b_i} = 2^{2n-i} + 1$*.*

So, the optimal solution is $\{b_1, b_2, ..., b_n\}$*, but since* $K = \frac{\epsilon MaxP}{n}$*, an approximation algorithm will round according to* $p'_i = \left\lfloor \frac{p_i}{K} \right\rfloor$ *formula which for* $K \geq 2$ *makes* $p'_{a_i} = \left\lfloor \frac{p_{b_i}}{K} \right\rfloor = \left\lfloor \frac{p_{a_i}}{K} \right\rfloor = p_{b_i}$*. But in this case the algorithm can not differentiate between returning* $\{b_1, b_2, ..., b_n\}$ *and returning* $\{a_1, a_2, ..., a_n\}$ *after rounding, and may therefore return a solution* $\{a_1, ...a_n\}$*, which has no elements in common with the optimal solution.*

That is, the Hamming distance between a binary string encoding the optimal solution and a binary string encoding the approximate solution for this input is the maximal possible, $2n$*.*

Chapter 3

Lower Bounds/ Inapproximibility results

There are several papers that analyze the complexity of structure approximation with respect to Hamming distance, in particular in the context of NP witnesses [GHLP99, KS99, FLN00, SY13]. The majority of them present inapproximability results. Structure approximation with respect to an arbitrary distance function, to our knowledge, has been considered only by Hamilton, van Rooij, Müller and Wareham [HMvRW07] and in the follow-up paper by van Rooij and Wareham [vRW12]. In addition to the results in the general setting, [HMvRW07, vRW12] present several results on value-related distance functions, as well as Hamming and edit distances.

The following two sections cover some of the general inapproximability results from [HMvRW07, vRW12]. After that we proceed surveying the body of work on Hamming distance approximation. We will finish the chapter with some of our results concerning edit distance function.

3.1 Basic distance functions

Depending on a specific problem and a chosen distance function, a solution which is close to an optimal solution in structure may or may not be as close to an optimal solution in value. For example, in MaxClique with Hamming distance function between vectors indicating whether a vertex is in a clique, a solution structurally close to an optimal one agrees with the optimal on a large number of vertices. As a subset of a clique also forms a clique, and since the value of a solution is the number of vertices in a clique, a large subset of a clique is a large clique, giving closeness in value. But consider an instance of SAT where every clause contains the same variable y together with another variable x_i (so $\phi = \bigwedge_{i=1}^{m}(x_i \vee y)$. Now, an assignment that sets $y = true$ and all $x_i = false$ satisfies this formula. However, there is an assignment with Hamming distance 1 from the optimal, setting $y = false$, which would falsify every clause. Also, already for MaxClique closeness in value does not imply closeness in structure. [HMvRW07, vRW12] present other examples where two notions are uncorrelated for the Max-Cut and TSP problem.

3.1.1 "Characteristic" distance functions

Structure approximation framework generalizes notions of solving decision and optimization problems, as well as traditional value approximation. Indeed a decision problem can be viewed as a structure approximation problem with distance function $d(y, optsol) = 1$ if y is not a correct witness for x and $d(y, optsol) = 0$ otherwise. For example, in the Knapsack problem, $d_x(y, z)$ can be designed as follows: Let $x = \langle (w_1, p_1), ..(w_n, p_n), B, P\rangle$ be an instance of decision version of Knapsack. Let $d_x(y, z) = 0$ if $y, z \in cansol$ and either both y and z have profits greater than the threshold P or both have profit less than P. Let $d_x(y, z) = 1$ otherwise. Now, any

approximation within $d(y, optsol) < 1$ has to have y as a witness. When there are no assumptions about $d(y, z)$ being efficiently computable, optimizations problems can be similarly recast as structure approximation with the distance between two optimal or two non-optimal solutions being 0, and 1 otherwise.

To define an optimization problem in this context, use a (not efficiently computable unless $P = NP$) distance function such that $d_x(y, z) = 0$ iff y and z are both optimal solutions or $y = z$, and $d_x(y, z) = 1$ otherwise. Note that here we allow the distance between distinct solutions to be 0; if this presents a problem with the definition of a distance function as a metric, choose it to be a very small ϵ instead. Alternatively, [HMvRW07] use $d_x(y, z) = 0$ iff $z = y$ and $d_x(y, z) = 1$ otherwise.

Note that for all these functions NP-hardness of structure approximation follows trivially from NP-hardness of the original problem for any parameter smaller than (arbitrary defined) the distance between optimal and non-optimal solutions, and is trivially approximable for a value larger than this parameter.

3.1.2 Value-related distance functions

In general, we can not have the notion of structure approximation where closeness in value implies closeness in structure or vice versa. However in some cases the distance function is closely enough correlated with the value difference, so that approximability and inapproximability in these contexts become related. The following lemma (Lemma 14 for the lower bound and Lemma 17 for the upper bound in [HMvRW07]) outlines this correspondence.

Call a function $f\colon \Sigma^* \times \mathbb{N} \to \mathbb{N}$ non-decreasing if it becomes a non-decreasing function $\mathbb{N} \to \mathbb{N}$ when its first argument is fixed.

Lemma 3.1.1. *[HMvRW07] Let $f\colon \Sigma^* \times \mathbb{N} \to \mathbb{N}$ be non-decreasing and Π an optimization problem; let d be an arbitary distance function. Then*

- *If* $\forall x$ *and* $\forall y \in cansol(x)$*, the value difference between* y *and the nearest optimal solution is bounded by a (non-decreasing function of) the distance, that is,* $|optval(x) - val(x,y)| \leq f(x, d(y, optsol(x)))$*, then for any function* $h(|x|)$*, a* $h(|x|)/d$*-s-approximation algorithm for* Π *produces a value within* $f(x, h(|x|))$ *additive factor and within* $1 + f(x, h(|x|))$ *ratio of the optimal. Thus, a lower bound on value approximation for* Π *implies a lower bound for the structure approximation with respect to such* d*.*

- *If* $\forall x$ *and* $\forall y \in cansol(x)$*, the distance between* y *and an optimal solution is bounded by a (non-decreasing function of) the value, that is,* $d(y, optsol(x)) \leq f(x, |optval(x) - val(x,y)|)$*, then value-approximating* Π *within an additive (respectively, multiplicative) factor of* $h(|x|)$ *gives an* $f(x, h(|x|))/d$ *s-approximation (respectively,* $f(x, optval(x) \cdot (h(|x|) - 1))/d$ *s-approx) algorithm for* Π*. That is, an upper bound on the value approximation gives an upper bound on the structure approximation of* Π*.*

The proofs follow by direct calculations.

The following corollary gives an immediate structure inapproximability for Max-Clique with respect to the Hamming distance function d_H from the $|V|^{1/2-\epsilon}$ lower bound on the value-approximation of Max-Clique, based on the PCP theorem.

Corollary 1. *Max-Clique is not* $|x|^{1/c}/d_H$ *s-approximable for any* $c \geq 6$ *unless* $\mathtt{P} = \mathtt{NP}$*.*

Note that even in case when the value is defined in a non-standard way, Lemma 3.1.1 gives inapproximability.

Corollary 2. *Consider a version of the Max-Clique problem where sd-function is standard, but value function* $val(x, y)$ *is 5 if* y *is a maximum clique, 4 if* y *is an empty set and 0 otherwise. This version of Max-Clique is not* $|x|^{1/c}/d_H$ *s-approximable for any* $c \geq 10$ *unless* $\mathtt{P} = \mathtt{NP}$*.*

3.2 General distance functions

The following results apply to any positive integer-valued metric distance function.

3.2.1 Self paddablity

One technique that was used by [HMvRW07, vRW12] to prove inapproximibility for general functions is self-paddability. Here the original instance x of Π is augmented with a polynomial amount of padding, so that the result is another instance x' of the same optimization problem Π. A candidate solution to this new instance x' can be interpreted as a tuple $\langle y_1, \ldots, y_{h(|x|)} \rangle$ of $h(|x|)$ candidate solutions to the original instance. The solution is optimal if each candidate solution in the tuple is optimal. Otherwise the distance of the solution to the optimal is the sum of distances of each candidate solution y_i to an optimal solution of x. To avoid trivial cases, computation of a padded instance x' from x and $h(|x|)$, as well as decoding of the solution to the original instance must be in polynomial time.

The following theorem uses this idea to give a "list-decoding" algorithm for computing an optimal solution to x given an algorithm that computes a close enough solution to the padded instance. For the chosen parameters below, at least one solution in the tuple of the answers to the padded instance will be optimal for the original instance. Just like in the list decoding setting, it is enough to go through the (sufficiently short, i.e. polynomially long) list of possibilities to find a solution that works.

Theorem 4. *[HMvRW07, vRW12] Let Π be an* NP*-optimization problem that is $h(|x|)$-self-paddable (described in previous paragraph), where h is polynomially bounded. Let d be an integer valued sd-function, and let $c\colon \mathbb{N} \to \mathbb{N}$ be such that $c(|x'|) < h(|x|)$, where x' is the padded instance. If Π is $c(|x|)/d$-s-approximable and the decision version Π_D of Π is* NP*-hard, then* P = NP*.*

We interpret the decision problem as an optimization problem; all solutions of the decision problem will be in $optsol(x)$. For example, if Π_D is the Knapsack problem, then all candidate solutions to x of value less than the profit bound are in $optsol(x)$, when x is interpreted as an instance of the corresponding optimization version of the Knapsack problem.

Suppose there exists an algorithm A that gives $c(|x|)$/d-s-approximation for Π. Let x' be the padded instance. Compute $y = A(x')$ and decompose y into $\{y_1, y_2, ..., y_{h(|x|)}\}$. Now since both c and d are positive integer valued and $c(|x'|) < h(|x|)$, at least one y_i has distance 0 from an optimal solution, and thus is optimal. Now to find this y_i compute the value of every solution in $\{y_1, y_2, ..., y_{h(|x|)}\}$ and take the best one. Since $|x'|$ is polynomial in $|x|$, and A runs in time polynomial in $|x'|$, A finds y in time polynomial in $|x|$. As decoding y and evaluating $val(y_i)$ takes polynomial time, and there are $h(|x|)$ (polynomially) many of y_is, finding the optimal y_i also takes polynomial time. Thus, using A one can decide Π in polynomial time, contradicting the assumption that Π_D is NP-hard (unless P = NP).

Lemma 3.2.1. *[vRW12] Let Π be an NP-hard problem, and d an arbitrary integer-valued distance function. Suppose that for an $\alpha \in \mathbb{N}, \alpha \geq 1$, the problem Π is n^α-self-paddable, where n is the length of the instance. Suppose also that the padding is compact, that is, the size of the padded instance $padsize(x, n^\alpha) \in O(n^{\alpha+r})$ for some constant r. Then for any ϵ, $0 < \epsilon \leq 1$, an $n^{1-\epsilon}/d$ s-approximation for Π can be used to recover an optimal solution to Π for some α.*

Proof. The idea of the proof is to use theorem 4 with $g(x) = n^{1-\epsilon}$ and $h(x) = n^\alpha$; if we can show that the padding is small enough that is $g(padsize(x, n^\alpha) < n^\alpha$, then by Theorem 4, a $g(x)/d$ s-approximation algorithm can be used to recover an optimal solution to Π. The calculation below is used to define α as a function of ϵ, r and the constant hidden in the O-notation. For the latter, let us state the limit on padding

size as $padsize(x, n^\alpha) \leq cn^{\alpha+r}$ for some $c \geq 1$ and $r \geq 0$. Now, applying g to the padsize and bounding by h, we obtain the following:

$$
\begin{aligned}
(cn^{\alpha+r})^{1-\epsilon} &< n^\alpha \\
(1-\epsilon)\log(cn^{\alpha+r}) &< \alpha \log n \\
(1-\epsilon)\log c + (1-\epsilon)(\alpha+r)\log n &< \alpha \log n \\
\log c + (1-\epsilon)(\alpha+r)\log n &< \alpha \log n \qquad \text{(since } c \geq 1\text{)} \\
\log c/\log n + (\alpha + r - \epsilon\alpha - \epsilon r) &< \alpha \\
\log c/\log n + r - \epsilon(\alpha + r) &< 0 \\
\log c/\log n + r &< \epsilon(\alpha + r)
\end{aligned}
$$

Setting $\alpha = (c+r)/\epsilon$, the right side of the inequality becomes $\epsilon(\alpha+r) = \epsilon((c+r)/\epsilon + r) = c + r(1+\epsilon)$. Now, for the left side, $\log c/\log n + r < \log c + r < c + r$, which in turn is less than $c + r(1+\epsilon)$, completing the proof.

□

3.2.2 Relation between s-FPTAS and structure approximation for NP-hard problems

In [HMvRW07] it was shown that no NP-hard problem has s-FPTAS. That means any NP-hard problem is hard to structure-approximate if structure-approximation implies having a s-FPTAS. In particular, Knapsack is an NP-hard problem that it easy to value-approximate, but hard to structure-approximate.

3.2.3 Structure inapproximabilty in parameterized complexity setting

In their original paper [HMvRW07], parameterized setting is already considered. They define a $\langle p \rangle$-fixed-parameter s-approximation algorithm as an analog of s-approximation algorithm, where the running time of the algorithm is $f(p)|x|^{\alpha}$ for some function f and constant $\alpha > 0$. With that definition they prove the following results.

Suppose that the distance function is such that, for any string y and constant c, there are $O(|y|^c)$ strings within distance c from y. For example, Hamming distance satisfies this property for any integer constant c, but in characteristic distance functions described in Section 3.1.1, any solution is distance 1 from the optimal. For such a function, existence of $\langle p \rangle$-fixed-parameter s-approximation algorithm implies decision problem is an FPT. Thus, if $\langle p \rangle$-Π_D is $W[1]$-hard, Π is not c/d s-$\langle p \rangle$-fp-approximable unless FPT $= W[1]$, for any integer constant c and distance function d satisfying the above property.

In particular, edit distance satisfies the property: number of strings within edit distance c of a given one is bounded by $2\binom{n+c+1}{c} \in O(n^{c+1})$. Here, $n+c+1$ is the largest number of possible insertion locations and a replacement can be modeled by a pair of one insert and one delete. Now since, $\langle m \rangle$-LCS_D is $W[t]$-hard, LCS is not c/d_E s-$\langle m \rangle$- or s-$\langle m, |\sum| \rangle$-fp-approximable unless $FPT = W[t], t \geq 1$.

3.3 Hamming distance inapproximability via NP-witnesses

Lemma 3.2.1 already gives Hamming distance s-approximation lower bounds for a number of NP-hard problems. For example, a padded instance for diverse problems

such as Max-IndependentSet, Min-VertexCover, Max-Cut and Chromatic number consist of $|x|^{\alpha+1}$ disconnected copies of the original graph. One can verify the optimal Independent set in the new graph G' is the union of the optimal Independent sets in each copy of G. Similarly, the optimal cut is the union of cuts in the copies and optimal colouring of G' induces an optimal colouring in each copy of G. By Lemma 3.2.1 above, these problems are not $|x|^{\alpha}/d_H$-s-approximable. Similarly, a padded instance for SAT and related problems can be constructed by taking a conjunction of $|x|^{\alpha+1}$ copies of formula with variables having distinct names in each copy. This gives $|x|^{\alpha}/d_H$-s-inapproximability results for SAT, 3SAT, etc. For Hamiltonian path and indirect Hamiltonian path add dedicated start and end vertices and connect them to form a chain. Clique can be reduced to Independent set and Set cover by disjoint copies of the input sets.

Note that in the "NP witness" setting such as [KS99, FLN00, SY13] that we will discuss below an approximation algorithm is allowed to return a string which is not necessarily a feasible solution, however in the [HMvRW07] setting it must return a solution that is feasible. So in the Max-IndependentSet example above, for the [HMvRW07] setting the approximation algorithm has to return an independent set, albeit of a suboptimal size, whereas in the NP witness setting it can return any string encoding a set of vertices, not necessarily independent. This makes upper bounds for the [HMvRW07] setting harder, but lower bounds, easier respectively. Most of the papers that give lower bounds use the concept of error correcting codes, however the recent results of [SY13] supersede the previous results using only search to decision reductions.

3.3.1 Error correcting codes

Error correcting codes have been an important tool in complexity theory. While these were used to prove PCP theorem, papers like [FLN00, GHLP99, KS99] use

error correcting codes as a tool to show structure (in)approximability with respect to Hamming distace for some of the NP-hard problems. We give some definitions of error correcting codes, following [FLN00] presentation for the first two definitions.

Definition 16. *(Error correcting code)*
Let Σ be an alphabet of size q. An $[n, k, d]_q$ error correcting code is a function $C : \Sigma^k \to \Sigma^n$ with the property that for every $a, b \in \Sigma^k$ we have a $dist(C(a), C(b))$ is greater than or equal to d.
Here, $dist(x, y)$ is the normalized Hamming distance between the two vectors x and y of equal length, i.e., the fraction of characters that differ in x and y. Also $0 \leq dist(x, y) \leq 1$.

List decoding is an alternative decoding method of error-correcting codes for large error rates. It allows the decoder to output a list of all codewords that differs from the received word in a certain number of positions. So, the possibility of handling larger number of errors increases with the use of this method.

Definition 17. *(List decodable codes)*
Let Σ be an alphabet of size q. An $[n, k, d]_q$ error correcting code C is δ list decodable if there exists a Turing Machine D(the list decoder) that on input $c \in \Sigma^k$ outputs in poly(n) time a list containing all words $a \in \Sigma^k$ that satisfy $dist(C(a), C(b)) \leq \delta$.

Another type of error correcting codes is called erasure code, where we know the positions of the corrupted or affected bits without knowing the values stored in those. An important question for structure approximation is the number of candidate solutions within a given distance d from a fixed solution, in particular from the optimal solution. If there are no solutions within distance d of the optimal, then a d-s-approx algorithm must return the exact solution. If there are polynomially many such solutions, then it still might be possible to compute the optimal from a d-approximation,

if one can enumerate all solutions with distance d from the one returned by the algorithm (list-decoding regime), and check each of them.

In the error correcting codes setting, the distance d is a property of a code such that no two codewords are within d from each other. Then the question becomes, given this code distance d, how many codewords are within distance $e \geq d$ from a given codeword. The answer to that is given by Johnson bound. Here, we will state it only for binary codes.

Lemma 3.3.1 (Johnson bound)**.** *Let $J(n, d, e)$ be the maximum number of codewords within a Hamming distance of e of any codeword. Then, if $e/n < 1/2(1 - \sqrt{2d/n})$, then $J(n, d, e) \leq 2nd$.*

3.3.2 Gal, Halevi, Lipton and Petrank: partial and approximate NP-witnesses

In 1999, Gal, Halevi, Lipton and Petrank [GHLP99] considered a problem of recovering a solution of an NP-hard problem where only a part of the solution is known (with omissions chosen adversarially). In that context they analyzed 3SAT, Graph Isomorphism, the Shortest Lattice Vector problem, Graph 3-Colorability, Vertex Cover and Clique.

The main conclusion is that the solution to the original instance can be recovered from a partial solution to a suitably constructed larger instance. Suppose we look at the problem Graph Isomorphism. Their result [GHLP99] implies that finding a small part of the isomorphism is enough to recover the whole isomorphism if it is encoded in a specific way. So, if we can find a way to construct a small part of the isomorphism, with their technique we can easily build the whole isomorphism.

One more thing that should be considered with their technique is the issue of fault

tolerance. That means if a solution is sent to us through a communication channel which omits a part of it, can we still recover the full solution? Again the answer is "yes" if solution is encoded in a specific way.

Their work is also motivated by cryptographic applications where a question is the relation between retrieving partial information about a secret and computing the whole secret.

The main problems [GHLP99] studied are SAT and Graph Isomorphism.

Theorem 5. *[GHLP99] For any $\epsilon > 0$, given a CNF formula over n variables it is possible to construct in probabilistic polynomial time another formula Φ' over $N = n^{O(1)}$ variables (with $|\Phi'| = |\Phi| + n^{O(1)}$), such that with probability almost 1, given any $N^{1/2+\epsilon}$ bits from a satisfying assignment to Φ', one can efficiently construct a satisfying assignment to Φ.*

Let Graph Isomorphism be the following problem:

Definition 18 (Graph Isomorphism)**.** *Let $G(V, E)$ and $G'(V', E')$ be two graphs. They are isomorphic if there exist bijections $f\colon V \to V'$ and $g\colon E \to E'$ that preserve the endpoint relations of G and G' (that is if $u \in V$ is an endpoint of $e \in E$ then $f(u) \in V'$ is an endpoint to $g(e) \in E'$). Given two graphs G and G',* Graph Isomorphism problem *is to determine whether they are isomorphic.*

Theorem 6. *There is a (deterministic) polynomial time procedure that on a given pair of graphs (G, H) constructs adaptively $l = poly(n)$ oracle queries $(G_1, H_1), ..., (G_l, H_l)$ such that if on each query the oracle answers with a map on only $O(\log n)$ of the variables (which is a part of isomorphism between the two graphs in this query), then procedure finds an isomorphism between G and H. [GHLP99]*

In this paper they also show results for some more NP-complete problems such as the Shortest Lattice Vector problem and Graph 3-Colorability. They mention that they

have obtained the results for Vertex Cover and Clique, with construction for Clique using the standard reduction without padding.

3.3.3 Kumar and Sivakumar: hard-to-approximate witnesses for all NP languages

Motivated by [GHLP99] as well as the then-recent PCP theorem, Kumar and Sivakumar [KS99] set out to generalize the results of [GHLP99] to all of NP problems. They presented several constructions based on list-decodable codes (in fact, now their paper is most cited for their code constructions). There, [KS99] considered both the erasure setting of [GHLP99] (where only a subset of bits is known, but known bits are all correct), and a "noisy proof system" setting much more akin to the structure approximation with respect to Hamming distance: a witness was guaranteed to agree with some correct witness on at least a given fraction of bits. Their main idea can be described as follows (following the notation in [FLN00]).

Definition 19. *Let L be a language in* NP*. Then by definition there is a polynomial-time computable relation $R_L(x, y)$ such that $x \in L$ iff $\exists y, |y| \leq |x|^c \wedge R_L(x, y)$, where c is a constant. Now, consider a different relation $\hat{R}_L(x, w)$, with $|w| \leq |x|^d$, defined as $\hat{R}_L(x, w) \equiv \exists y(w = Code(y) \wedge R_L(x, y))$. Here, $Code(y)$ is an error-correcting code (more on it later).*

Thus, if there is a polynomial time algorithm that decodes $w = Code(y)$ to obtain y, then the relation $\hat{R}_L$ is a polynomial-time relation.

Now, assume that for some specific code such a decoding algorithm exists. Also assume that this code can recover a string with $1/2 - \epsilon$ fraction of errors (possibly in the list-decoding regime). Then to decide L it is enough to find a $1/2+\epsilon$ approximation of w. That is, approximating w to within $1/2 + \epsilon$ is as hard as solving the original

problem; in particular, if L is NP-complete, then such approximation is NP-hard.
Note that information-theoretically this is not an optimal witness: w can be significantly larger than y, so half the bits of w could be longer than all of y. This is one of the issues addressed by [FLN00].

Theorem 7 ([KS99]). *Given an approximation w_1 to a binary string w, it is possible to recover w if w_1 agrees with w on at least $1/2+\epsilon$ fraction of bits, where $\epsilon \geq 1/|w|^{1/5}$.*

Another way to state it is that there is a way to recover a witness given a string that agrees with it on at least $|w|/2 + |w|^{4/5+\epsilon'}$ bits, for $\epsilon' > 0$. The proof of this claim follows from the constructions for erasure codes in the main body of the paper, together with the Johnson's bound.
More specifically, the paper provides three constructions of erasure codes. All three constructions consist of the Reed-Solomon code as an outer code, with three different inner codes. That is, a string y is first viewed as a sequence of evaluations of a polynomial $p_y(u) = \Sigma_j y_j u^j$ over $u \in F_q$ for a chosen field F_q with q elements. Treating this as a binary string, each of the q terms is $\log q$ bits in length; these $\log q$ bit blocks are each encoded by an "inner code". The choice of the inner code determines the final parameters.

Theorem 8 ([KS99]). *For every language $L \in$ NP, every witness predicate $R_L(x, y)$, and every $\epsilon > 0$ there exists a witness predicate $\hat{R}_L(x, w)$ as defined above that given N^δ bits of w, recovers y satisfying $R_L(x, y)$ in polynomial time. The three constructions give the following parameters:*

1. *Using Hadamard inner code: $\delta = 3/4 + \epsilon$ and $|w| = q^2$, where q, the field size, is a power of two $\geq (4n)^{1/3\epsilon}$.*

2. *Using a probabilistic construction for the inner code: $\delta = 1/2+\epsilon$, and $|w| = q^{1+\alpha}$,*

for $\frac{1/2-\epsilon}{1/2+\epsilon} < \alpha < \frac{1/2+\epsilon}{1/2-\epsilon}$. *However, to construct such an* $\hat{R}_L$ *an algorithm with a slightly superpolynomial running time (*$O(2^{(\log q)^3})$ *is needed.*

3. *Using l-wise* δ*-biased sample space* $\delta = 2/3 + \epsilon$, *and* $|w| = q^{1+\alpha}$, *for* $\frac{1/3-\epsilon}{2/3+\epsilon} < \alpha < \frac{1/6+\epsilon+\gamma}{1/3-\epsilon}$, *where* $0 < \gamma < \epsilon$ *is a very small constant.*

Those are the parameters on which [FLN00] is based. We omit the technical code-theoretic details of the constructions, since we are using the codes in the black-box fashion.

Relatively recently a better construction of such codes appeared due to Guruswami and Rudra [GR08], (journal version [GR11]). They produce explicit codes list-decodable up to $1/2 + \epsilon$ fraction of errors. Moreover, their codes produce $|w| = O(n^3/\epsilon^{3+\gamma})$ for a small constant γ (or $\tilde{O}(n/\epsilon^{5+\gamma})$; note that the optimal non-constructive bounds are $O(n/\epsilon^2)$). Thus, where the codes of Kumar and Sivakumar can recover a witness string w from $|w|/2 + |w|^{4/5+\epsilon'}$ bits, for $\epsilon' > 0$, codes of Guruswami and Rudra can recover w given $|w|/2 + |w|^{2/3+\epsilon'}$ bits.

So what is the relevance of Kumar and Sivakumar work to the structure approximation setting? Intuitively, their result (as improved by Guruswami and Rudra) states that for every language in NP, there exists a witness predicate, with respect to which it is as hard to structure-approximate a witness w to within Hamming distance $|w|/2 + |w|^{2/3+\epsilon'}$ as it is so solve the original problem. Thus, for an NP-complete L, such witness is NP-hard to approximate much better than by taking a random string (note that a random string with high probability agrees with a satisfying assignment on $|w|/2 + |w|^{1/2}$ bits.) And therefore for such a predicate, with distance function being the Hamming distance, the "structure approximation by dumb luck" of [HMvRW07] is the best possible.

Problem	Hamming distance
3SAT	$1/2 - n^{\epsilon}$
Max-Clique	$1/2 - n^{\epsilon}$
Chromatic Number	$2/3 - n^{\epsilon}$
Independent Set	$1/2 - n^{\epsilon}$
Vertex Cover	$1/2 - n^{\epsilon}$
Feedback Edge Set	$1/2 - n^{\epsilon}$ (not tight)
Directed Hamiltonian Cycle	$1/2 - n^{\epsilon}$
Hamiltonian Cycle	$2/5 - n^{\epsilon}$(not tight)

Table 3.1: [FLN00] results

3.3.4 Feige, Langberg and Nissim: Inapproximability for natural witnesses

Whereas [KS99] construct a witness in a special form, [FLN00] consider natural witnesses. The witness for a problem is not universal, but different problems have different natural witnesses. For the SAT problem a satisfying assignment is a witness. Similarly a characteristic vector of a set of elements in a Knapsack is the witness for a Knapsack problem. If we consider the Clique problem, the set of the vertices in a clique is a witness. Alternatively, the edges can be marked as 0 and 1 where the edges marked with 1 would form a clique. This can be another witness for the clique problem.

In [FLN00], Feige, Langberg and Nissim take some of the problems from Karp's list of 21 NP-complete decision problems [Kar72] and analyze their witness approximation (Hamming distance) complexity. Table 3.1 shows their results for the problems they mention in the paper.

Below, we will present an outline of their proof of structure inapproximability of SAT. Based on [KS99], they show that given a satisfiable Boolean formula ϕ, it is NP-hard to find an assignment to the n variables of ϕ which is of Hamming distance significantly less than $n/2$ to some satisfying assignment of ϕ.

Claim 1. *[FLN00] Unless* $\mathtt{P} = \mathtt{NP}$*, there is no polynomial time* $(n/2-n^{\epsilon})/d_H$ *s-approx algorithm for 3SAT for some* $\epsilon > 0$ *(which depends on code construction).*

Proof. The main idea of the proof is finding the 'core' of the problem and 'amplifying' it. The 'core' of the problem consists of those variables which are hard to approximate. Amplification involves copying the core multiple times.

Let $R = \{(w, \phi_0) | \exists a \text{ s.t } w = C(a) \text{ and } \phi_0(a) \text{ is true}\}$, where ϕ_0 is a SAT formula and the size of the witness w is n.

At first, using any standard reduction such as Cook's theorem, R is transformed into a 3SAT formula ϕ_1. A part of that formula encodes a satisfying assignment to ϕ_0; call these variables $w_1, w_2, ..., w_n$, and the rest $z_1, z_2, ..., z_\ell$. The variables $w_1, w_2, ..., w_n$ form the core.

Then comes the amplification step where a 3CSP formula ϕ_2 is constructed by duplicating the variables $w_1, w_2, ..., w_n$ a $poly(n) = m$ times so that the number of auxiliary variables $z_1, z_2, ..., z_\ell$ is small compared to the total number of variables. So, ϕ_2 becomes $\phi_2(w_1, w_2, ..., w_n; w_1^1, w_2^2, ..., w_n^m; z_1, z_2, ..., z_\ell)$ and it can be defined as $\phi_2 = \phi_1(w_1, w_2, ..., w_n; z_1, z_2, ..., z_\ell) \wedge \bigwedge_{i=1}^{n}(w_i \leftrightarrow w_i^1) \wedge \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m-1}(w_i^j \leftrightarrow w_i^{j+1})$. The number of variables (and thus the length of the witness) of ϕ_2 is $N = mn + \ell$.

Now, suppose there is a polynomial-time algorithm that can find an assignment x which approximates a satisfying assignment for ϕ_2 within Hamming distance $N/2-N^{\epsilon}$. For $j = 1, \ldots, m$ let x^j be the restriction of x over the variables $w_1^j, ..., w_n^j$. As x is within Hamming distance $N/2 - N^{\epsilon}$ of a satisfying assignment to ϕ_2, we can say by the definition of ϕ_2 that there exists some j such that x^j is within Hamming distance $n/2 - n^{\epsilon}$ from $C(a)$ where a is an satisfying assignment of ϕ_0. This contradicts the witness approximability of the relation R from theorem 7 and concludes the proof.

□

Given that result, and using a s-approximation preserving reduction, a similar inap-

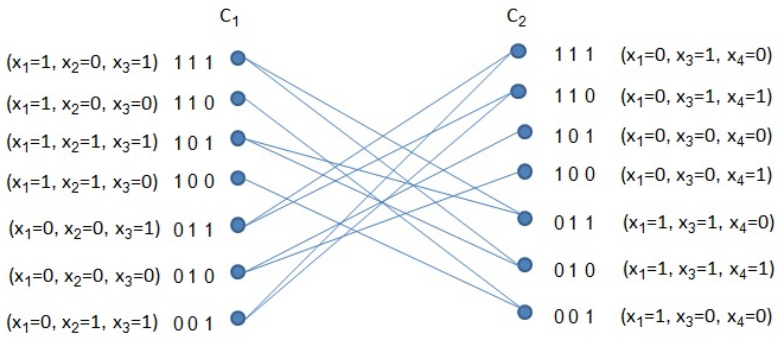

Figure 3.1: (a) Reduction for $\phi = (x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_3 \vee \bar{x_4})$

proximability result for the Clique problem follows.

Claim 2. *[FLN00] Unless* $\mathtt{P} = \mathtt{NP}$*, there is no polynomial time* $(n/2 - n^{\epsilon})/d_H$ *s-approx algorithm for Clique for some* $\epsilon > 0$*. Here, the natural witness is the indicator vector for the vertices in clique.*

Proof. Similar to Claim 1, the basic idea here is also to find the core of instance H of the problem and then do amplification. Let ϕ be one of a class of 3SAT formulas from claim 1 which are NP-hard to $(n/2 - n^{\epsilon})/d_H$ s-approximate.
Now, consider a reduction $3SAT \leq {}_pClique$ that works as follows (see Figure 3.1):

1. For each clause $(l_{1i} \vee l_{2i} \vee l_{3i})$ introduce vertices $V_{1i}, ...V_{7i}$, corresponding to their assignments satisfying the formula.

2. Add an edge for every pair of consistent vertices. Now a clique in this graph encodes a satisfying assignment if its size is at least the number of clauses in the original formula.

Now, for amplification add $2h$ vertices for each variable x_j of ϕ. Call them $t_{j,1}.....t_{j,h}$ and $f_{j,1}.....f_{j,h}$. Now, connect each $t_{j,i}$ to all $t_{j,i'}$ and to all vertices V consistent with $x_j = T$ (that is, vertices corresponding to either assignments where $x_j = T$ or clauses

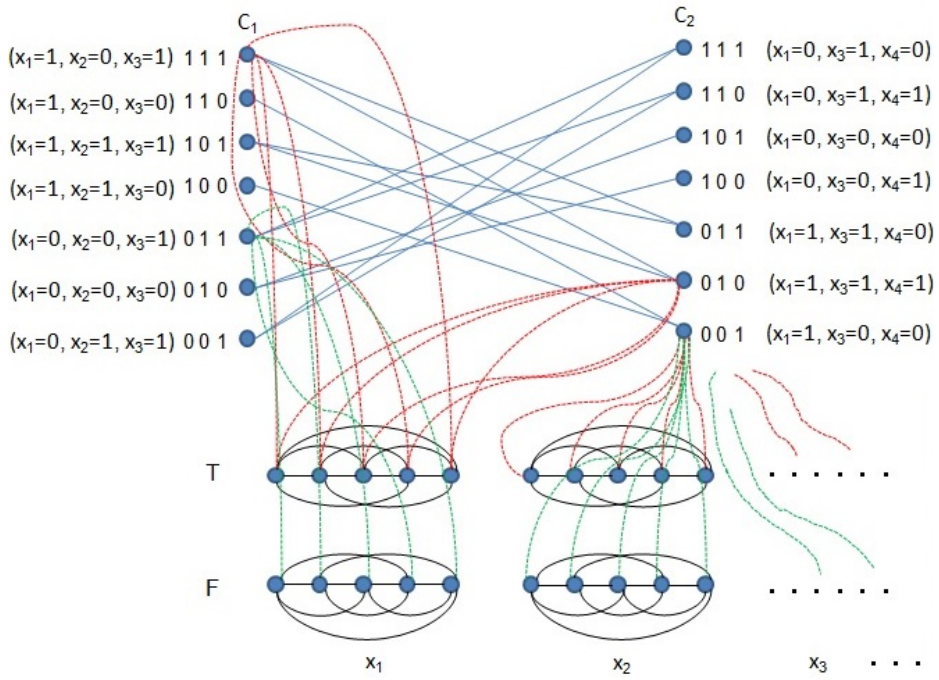

Figure 3.2: (b) Reduction for $\phi = (x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_3 \vee \bar{x_4})$. Let $h = 5$ here. Not all edges from $t_{i,j}$ and $f_{k,l}$ to V's are shown

without x_j). Similarly, connect each $f_{j,i}$ to all $f_{j,i'}$ and to all vertices V consistent with $x_j = F$. The total number of vertices for ϕ with n vertices and m clauses will become $N = 7m + 2hn$. Figure 3.2 illustrates the resulting graph, only showing some of the edges between the new vertices and vertices $V_{i,j}$.

Now, every clique of the original graph receives $3h + 2h \times (n - 3)$ additional vertices. Now, suppose there is an $(N/2 - N^{\epsilon})/d_H$ s-approx algorithm for the Clique problem. Note that for a correct witness clique in the instances obtained from the reduction above, either all t_j variables or all f_j variables are present in a clique for any given j, but not both. When $h(\epsilon)$ is large enough, it will get majority of each $t_{j,1}.....t_{j,h}$, $f_{j,1}, \ldots, f_{j,h}$ correct. Then, taking majorities of variables corresponding to each x_j with an appropriate sign, a truth assignment to the original ϕ can be recovered.

□

They also presented results for problems listed in the Table 3.1. For the rest of Karp's problems they state that the reductions can be adapted to achieve similar results, and state that these proofs will appear in the future version of the paper. However, it seems that these proofs never appeared and possibly were not written (Feige, personal communication).

3.3.5 Sheldon and Young: Hamming Approximation via search-to-decision reductions

Guruswami and Rudra briefly mention an unpublished manuscript of Sheldon and Young [SY03] that achieves hardness of structure approximation for SAT which is $|w|/2 + |w|^{\epsilon}$ for any ϵ. We have contacted Prof. Young to ask about the manuscript; he has sent us an expanded version that appeared on arXiv a few days later as [SY12]. Then, in 2013, a version with several more generalizations was published [SY13]. Here, we present their results following their journal version [SY13].

Sheldon and Young were motivated by Feige, Langberg and Nissim's work [FLN00] and set to strengthen their hardness result. Indeed, they improve [FLN00] results for SAT, HamPath and Clique family of problems from $\exists\epsilon$ to $\forall\epsilon$. Additionally, they show that for the universal NP-complete language, no deterministic polynomial-time algorithm can achieve Hamming distance $n/2 + O(\sqrt{n \log n})$ (unless P = NP). Note that in particular it cannot even achieve $n/2$ s-approximation. Also, no randomized polynomial-time algorithm can achieve $n/2 + O(\sqrt{n \log n})$ with probability $1 - 1/n^{O(1)}$(unless RP=NP). They later extend it to all paddable (in Berman-Hartmanis's sense) NP languages. However, this is in the settings more similar to [KS99], rather than natural witnesses. This result is shown in the following theorem.

Theorem 9. *[SY13] Let universal* NP*-complete language be* $\mathcal{U} = \langle V, x, 1^t \rangle$*, where* V

is the encoding of any verifier, x is a string and 1^t is a "padding" string of t ones, such that, for some witness w of length at most $t, V(x, w)$ accepts within t steps. Let $V_u(\langle V, x, 1^t\rangle, u)$ be a verifier for $\mathcal{U}$ which runs $V(x, u)$ and accepts if $V(x, u)$ accepted in t steps. Fix any $\alpha > 0$. Then, there is no $n/2 + O(\sqrt{\alpha n \log n})/d_H$ s-approx algorithm for $\mathcal{U}$ unless $\mathtt{P} = \mathtt{NP}$.

They also consider 'natural' verifiers for various well known NP complete problems such as 3SAT, Vertex Cover and Hamiltonian cycle, and prove both upper and lower bounds for them. Though apart from [HMvRW07, vRW12], other papers summarized in this chapter rely on error correcting codes, [SY13] uses search-to-decision reductions and amplification to achieve their lower bounds.
Main results from [SY13] are:

- The NP language $\mathcal{U}$ is not $n/2 + O(\sqrt{n \log n})/d_H$ s-approximable.
- For some NP languages, the natural witness can be approximated to within $n/2$.
- For some NP languages, it is hard to $(n/2 - n^\epsilon)/d_H$ s-approximate the natural witness, $\forall \epsilon > 0$.

Here we will describe some of the results from [SY13]

Lemma 3.3.2. *[SY03, SY12, SY13] Suppose that, for SAT (with the natural witness relation), there exists $\epsilon > 0$ such that some polynomial-time algorithm A achieves Hamming distance $n/2 - n^\epsilon$. Then* $\mathtt{P} = \mathtt{NP}$.

Proof. They prove the lemma by describing a polynomial time algorithm A' that solves SAT in polynomial time using a search-to-decision reduction to A. Given a SAT instance I it creates new instance I' from I by duplicating an arbitrarily chosen variable x, k times where $k = \left\lceil 1 + (n/2)^{1/\epsilon} - n \right\rceil$. That means A' adds to I new variables $x^1, x^2, ...x^k$ and adds new clauses $(x = x^1) \wedge (x^1 = x^2) \wedge ... \wedge (x^{k-1} = x^k)$.

Algorithm A' runs on the instance I' of A to achieve Hamming distance $(k+n)/2 - (k+n)^{\epsilon}$. We assume b to be a true or false value which is assigned to at least $k/2+1$ copies of the variable x in I'. We set $x = b$ and substitute b for x in I' and simplify the resulting formula to get a new formula I'' and use recursion on I'' thus assigning values to all the variables remaining. □

Example 4. *Suppose we have a SAT formula,* $(x \vee y) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{y} \vee \bar{u})$ *and we assume* $\epsilon = 1/3$. *Here, n will be 4. As we know from the lemma* $k = \left\lceil 1 + (n/2)^{1/\epsilon} - n \right\rceil$ *so,* $k = \lceil 1 + (4/2)^3 - 4 \rceil = 5$ *here.*

Amplifying y, obtain $I' = (x \vee y) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{y} \vee \bar{u}) \wedge (y \vee \bar{y_1}) \wedge (y_1 \vee \bar{y}) \wedge (y_1 \vee \bar{y_2}) \wedge (y_2 \vee \bar{y_1}) \wedge (y_2 \vee \bar{y_3}) \wedge (y_3 \vee \bar{y_2}) \wedge (y_4 \vee \bar{y_3}) \wedge (y_3 \vee \bar{y_4}) \wedge (y_4 \vee \bar{y_5}) \wedge (y_5 \vee \bar{y_4})$.

Now, suppose A computes an assignment within $n/2 - n^{\epsilon} = 9/2 - 9^{1/3} < 4.5 - 2 < 2.5$. *So, it will make at most 2 errors. Even if both errors are on* y_i*s, most of* y_i *are correct.*

Corollary 3. *If that verifier mentioned in theorem 3.3.2 has a randomized* $(n/2 - n^{\epsilon})$*-Hamming-approximation algorithm A working with probability* $1/2 + 1/n^c$*, then* RP=NP.

We omit the proof.

A proof of Theorem 9, on structure inapproximability of the universal language to within $(n/2 + \sqrt{\alpha n \ln n})/d_H$, is more involved.

Suppose there is an algorithm $A_{\mathcal{U}}$ that gives $(n/2 + \sqrt{\alpha n \ln n})/d_H$ s-approximation of $\mathcal{U}$. Now, to find a witness for $\mathcal{U}$, do the following steps repeatedly:

1. Run $A_{\mathcal{U}}$ to find a string u approximating a witness S.

2. Throw out all strings that are far from w, that is, distance more than $(n/2 + \sqrt{\alpha n \ln n})$. If there are polynomially many strings left, brute-force check them to find a witness S.

3. Map all remaining string to a set of strings of smaller $S(\mathcal{U})$; hence the verifier to "undo" this mapping before verifying; this will be the verifier for the next iteration.

3.4 Edit distance

3.4.1 Lower bound

Consider $d_E(y, z)$ to be the *edit distance* between strings y and z, that is, the number of insert, replace and delete a symbol operations needed to convert y into z. This function, even though in some respect related to Hamming distance, nevertheless has a very different behavior. For example, a string 01010101 and a string 10101010 have the maximal Hamming distance of $n = 8$, however their edit distance is just 2, corresponding to deleting a 0 in front and inserting it in the back of the string. For Hamming distance, a random string is expected to be within $n/2$ from any string, but it is not clear what expected edit distance between two random strings is. If two strings are far in the edit distance though, then in particular they are far in the Hamming distance, since replacement is an edit distance operation. So, lower bounds on edit distance approximability imply lower bounds for the Hamming distance, but the reverse is not immediate.

However, in case when one of the strings is a string consisting of all 0s or all 1s then the two notions coincide, as long as the length of the approximating string is the same. Indeed, even edit distance with transpositions to a string of all 1s from any given string is equivalent to Hamming distance.

Lemma 3.4.1. *For any string x of length n, its Hamming distance to a string of n 1s is equal to the edit distance.*

Proof. The Hamming distance between x and the string consisting of n 1s is the number of $0s$ in x, call it k. Now, since replacements are one of the operations counted in edit distance, the edit distance is no greater than the Hamming distance k. Suppose that edit distance is $k' < k$. Consider the corresponding sequence of k' operations converting the string of all 1s into x. Since $k' < k$, one of the operations is not replacement. Suppose that the first such operation in the sequence is a deletion. Then, as we assumed that $x = n$, there will be a corresponding insertion later in the sequence. It can be seen that the deletion removed a 1, and insertion introduced a 0. Now, modifying indices of the symbols after the deletion point, it is possible to simulate this pair of operations with one replacement. A similar argument holds for the first operation being an insertion. (Alternatively, it is sufficient to say that there are only two operations that increase the number of 0s in a string of all 1s: a replacement and an insertion. As insertion has to have a corresponding deletion, the most efficient way to obtain a string with k 0s out of a string of all 1s is k replacements). □

Now, consider Sheldon-Young proof that a natural witness for SAT cannot be Hamming-distance-approximated to within $n/2 - n^\epsilon$ unless $\mathtt{P} = \mathtt{NP}$. Their proof [SY13] proceeds as follows. First, note that it is enough to have an algorithm determining the value of one variable; the rest can be computed by applying the same algorithm on the reduced formula. Now, the proof proceeds by amplifying an (arbitrary, say the first) variable z_i $n^{1/\epsilon}$ times, that is introducing $n^{1/\epsilon}$ new variables and adding clauses stating that they are equivalent to z_i. Now, if there is an algorithm that is guaranteed to return a witness within $n/2 - n^\epsilon$ Hamming distance of a satisfying assignment, then such a string will be correct on majority of copies of z_i. Taking the majority thus gives the correct value of this variable, and repeating the process n times each time reducing the formula with computed variables results in a correct satisfying assignment.

Theorem 10. *Unless* P = NP*, no algorithm can approximate the natural witness to SAT to within edit distance* $n/2 - n^{\epsilon}$.

Proof. Note that a natural witness for this problem consists of either $n^{1/\epsilon}$ 0s or $n^{1/\epsilon}$ ones, together with $n-1$ symbols of arbitrary values for the rest of the variables; moreover, we can assume that all values of the copies of z_i are together, for example forming the first $n^{1/\epsilon}$ positions of the string. Now, suppose there is an algorithm that approximates the satisfying assignment above, with $n^{1/\epsilon}$ copies of z_i, to within edit distance $N/2 - N^{\epsilon}$ rather than Hamming distance, where $N = n + n^{1/\epsilon}$. Let y' be a string returned by the approximation algorithm and y the corresponding optimal solution. Consider only the first $n^{1/\epsilon}$ positions in y', ones corresponding to the copies ofz_i. Without loss of generality, assume that $z_i = 1$ in y. These positions can be changed to 0 (to obtain y') by either a replacement or an insertion/deletion pair moving values of the remaining $n-1$ variables into the first $n^{1/\epsilon}$ positions. But as discussed above, in this case the number of insert/delete pairs is at least as large as the number of replacements. Therefore, the same argument as for the Hamming distance applies, and bounding the edit distance between y and y' by $N - N^{\epsilon}$ means that majority of the copies of z_i in y' have a correct value.

□

Note that to adapt the [SY13] proof of inapproximability of VertexCover to the edit distance setting an extra trick is required. A natural witness to VertexCover is a binary string of length n, where a bit corresponding to a vertex is 1 iff that vertex is in the cover. In the [SY13] construction of a new graph, a copy of a vertex is made and a long even-length path (on $n^{1/\epsilon}$ vertices) is added between a vertex and its copy. Then the argument proceeds by showing a majority of the vertices on the path will have a correct value. Then, having more even vertices from that path in

the approximate cover corresponds to the original copied vertex being in the minimal cover, and otherwise the original vertex is not in the minimum cover.

Theorem 11. *Unless* $\mathtt{P} = \mathtt{NP}$*, no algorithm can approximate the natural witness to VertexCover to within edit distance* $n/2 - n^{\epsilon}$*.*

Proof. Consider the [SY13] construction described above, but with a different naming convention for the variables in the witness. Let variables $v_1 \dots v_n$ be the original variables, v' a copy of a selected variable e.g. of v_1, $u_1 \dots u_{n^{1/\epsilon}}$ be even variables on the path from v to v' and $w_1 \dots w_{n^{1/\epsilon}}$ be the odd variables on that pass. Now, in the witness the first $n^{1/\epsilon}$ positions will correspond to the u_i variables, followed by v_is, in turn followed by the w_is.

Now, the same kind of argument as before applies. A minimal cover will be encoded by either a string of $n^{1/\epsilon}$ 0s followed by some string of length $n+1$ followed by $n^{1/\epsilon}$ 1s, or a similar string with 0s at the beginning and 1s at the end. Now, similarly to the SAT construction, we would like to argue that a sequence of $N/2 - N^{\epsilon}$ of arbitrary edit operations (inserts, deletes, replacements) would not result in any string that is further than the Hamming distance from the original, if the inner n variables are ignored.

Consider a pair of insert/delete operations needed to convert a string encoding an optimal cover to an approximate string. Suppose, without loss of generality, that the optimal string starts with 1s and ends with 0s. Consider deleting a value from the u part of the string and inserting it into the w part. Now, the middle part of the string, corresponding to the v variables, could become maximally far from the minimal vertex cover at that point (i.e., if it was of the form 01010101), however we are only concerned with the u and w parts as we are trying to determine the value of the copied variable. The pair of insert-delete operations then introduces at most one 0 into the u part (by shifting the v part into it), and at most one 1 into the w part by

insertion. Therefore, the "damage done" to these parts of the string is no more than from doing two replacements.

Therefore, if there exists a structure approximation algorithm for vertex cover that can consistently return a string within edit distance $n/2 - n^{\epsilon}$ from an optimal cover, then this algorithm can be used to determine exactly whether any given variable is in the optimal cover. By Turing/search-to-decision reduction, from there the actual cover can be computed. In this reduction, if a vertex was determined to be in the cover, then recurse on a graph without this vertex, and otherwise recurse on a graph without this vertex and all of its neighbours. □

So far, we have discussed the complexity of approximating an NP witness, however for the majority of practical problems it is approximating an optimal solution which is of interest. But since lower bounds on decision problems imply lower bounds on optimization problems, the results above give inapproximability of the optimization version of this problem, in particular MaxSAT and MinVertexCover.

Chapter 4

Algorithms design for Structure approximation

One of the main open question posed in [HMvRW07] was to develop an algorithm design toolkit for structure approximation algorithms. This question remains valid even despite strong lower bounds presented in the previous chapter: are there algorithms giving matching upper bounds to these lower bounds? In this chapter we will look at some examples of algorithm design techniques applicable in structure approximation settings.

4.1 Symmetric Problems

A number of NP complete problems have the following property: for any solution its complement is a solution of the same value. Thus a complement of an optimal solution is an optimal solution. Moreover, for some such problems a complement of a binary string encoding in an optimal solution itself encodes an optimal solution. As any binary string is within Hamming distance $n/2$ to any given binary string or

it's complement, any binary string gives $n/2$ approximation to an optimal solution of such problems.

In particular, for any instance of not-all-equal 3SAT (NAE-3SAT), for any satisfying assignment, complementing all the variables also gives a satisfying assignment. Thus any string is Hamming distance $n/2$ from a satisfying assignment if there exists one.

Observation 1. *The $n/2 - n^{\epsilon}$ lower boung proof for SAT also applies to NAE-SAT.*

Proof. The new clauses $z_i \leftrightarrow z_{i+1}$ become $(\bar{z}_i \vee z_{i+1}) \wedge (z_i \vee \bar{z}_{i+1})$. An assignment that gets all z variables to the same value satisfies one and falsifies another variable in these clauses. □

Therefore, the lower bound for NAE-SAT is almost tight with only a n^{ϵ} difference between upper and lower bound.

Weighted max cut is another example where [HMvRW07] were able to give $n/2$ approximation algorithm. The minimum cut can be solved by network flows, but the maximum cut is NP-hard. Now, if we want to encode this solution into binary string we will have a string of 0s and 1s where for example 0 corresponds to the vertices being on the right side of the partition and 1 on the left side. Notice if we switch 0s with 1s we will still get an solution with the same value as we have only flipped the sides by switching 0s and 1s.

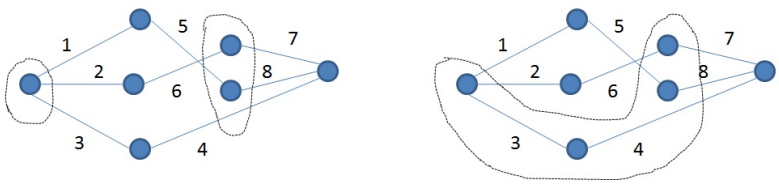

Figure 4.1: Structure approximability by dumb luck: Max Cut

Now, any string is within $n/2$ from either an optimal solution or its complement, in particular string 00...0 or a random string. In [HMvRW07], this is referred to as "structure approximation by dumb luck". For example, in the graph on figure 4.1 the optimal cut is illustrated by the first picture, and a cut which is distance one from the optimal by the second picture.

4.2 Subset encoding problems

A number of NP decision problems have solutions encoding subsets of elements of the same weights. For example, in Clique or Vertex cover (unweighted) the solutions encode the sets of vertices where all the vertices have the same weight. As they are decision problems, solution value is the size of the subsets and a solution is accepted if that size is above or below a given threshold k. In this case an empty set solution gets at least half of the elements right whenever $k < n/2$ and solution containing all elements gets at least half of elements right if $k \geq n/2$. Thus, an all-zero string gives a Hamming approximation of $n/2$ when $k < n/2$ and string of all-ones gives a Hamming approximation of $n/2$ when $k \geq n/2$. Therefore, a simple algorithm that compares k to $n/2$ and outputs either string of all-zeros or string of all-ones respectively gives $n/2$ approximation for such subset encoding problems [SY13]. In particular this gives an almost tight upper bound for cligue, independent set and vertex cover.

4.3 Structure approximation from value approximation algorithms

Sometimes an existing value approximation algorithm gives us a structure approximation of particular problem. The following Vertex cover 2-approximation algorithm

does so. We follow [HP13] for the proof below. Let $ApproxVertexCover$ be an algorithm that chooses an edge from G, adds both endpoints to the vertex cover, and removes the two vertices and all the edges adjacent to these two vertices from G. This process is repeated till G has no edges. Clearly, the resulting set of vertices is a vertex cover, since the algorithm removes an edge only if it is being covered by the generated cover.

Theorem 12. *$ApproxVertexCover$ is a 2-approximation algorithm for $Min-VertexCover$ that runs in $O(n^2)$ time.*

Proof. Every edge picked by the algorithm contains at least one vertex of the optimal solution. As such, the cover generated is at most twice larger than the optimal. □

Observation 2. *This algorithm gives k-approximation with respect to Hamming distance.*

Proof. Consider an encoding of the solution by a string of length n with 1 for vertices in the cover. Then, every vertex which is 1 in the optimal solution is also an 1 in the approximate solution. Now, the approximate solution has at most k extra 1s, giving the Hamming distance k. □

4.4 Structure approximation algorithms from linear programming

Linear programs are a common framework for representation of NP optimization problems. There, constraints are represented as linear inequalities; there is a linear goal function to be minimized or maximized. Candidate solutions are all sets of values that satisfy the constraints; the value of the solution is the value of the goal function.

More precisely, a linear program in the canonical form is (following presentation in [Vaz01]):

$$\begin{array}{lll} \text{Minimize} & f(\bar{x}) = \Sigma_{j=1}^{n} c_j x_j & \\ \text{Subject to} & \Sigma_{j=1}^{n} a_{ij} x_j \geq b_i, & i = 1, \ldots, m \\ & x_j \geq 0, & j = 1, \ldots, n \end{array}$$

There are polynomial-time algorithms to find an optimal solution to a linear program such as interior point methods. However, when an additional type of constraint is allowed, specifying that all variables have to assume integer (or 0-1) values, the resulting class of *integer* linear programs allows to represent NP-hard programs. To see that ILP is NP-hard [Kar72], consider encoding a SAT formula by a system of linear inequalities stating that sum of the literals in every clause is at least 1 (here, representing negated literals $\overline{x_i}$ as $1 - x_i$), and empty goal function.

4.4.1 Linear programming algorithms providing structure approximation

Consider a classic linear programming approximation algorithm for the Vertex Cover problem. To construct a corresponding integer linear program, use variables $x_1 \ldots x_n$ corresponding to the vertices. Now, a condition $x_i + x_j \geq 1$ for every edge (i, j) in the graph, together with a requirement that x_i are integers, states that at least one vertex is in the cover. And the goal function to minimize is $\Sigma_{i=1}^{n} w_i x_i$ (that is, the total weight of vertices in the cover is minimized; for the unweighed case, set each $w_i = 1$).

$$\begin{aligned} \text{Minimize} \quad & f(\bar{x}) = \Sigma_{i=1}^{n} w_i x_i \\ \text{Subject to} \quad & x_i + x_j \geq 1, \qquad \forall i, j \;\; E(v_i, v_j) \end{aligned}$$

A famous 2-approximation algorithm for the (weighted) vertex cover program solves the linear problem above without the requirement that x_i have to be integers, then rounds the resulting values of x_i so that if $x_i \geq 1$ then it becomes a 1 (and thus the corresponding vertex is placed into the cover), otherwise it is set to 0 (so the vertex is not in the cover). This algorithm gives a vertex cover of cost at most twice the optimal even for the weighted vertex cover case.

Note (and [SY13] also make this observation) that the resulting cover will not only have the cost at most twice the optimal, but also contain the optimal cover. Thus, a binary string encoding this approximate cover will differ from a string encoding an optimal cover in at most $2k$ places, where k is the size of the cover. Therefore, these approximate solution is within Hamming distance k from the optimal. As edit distance is always bounded by the Hamming distance, it is also at most k edit distance from the optimal solution.

Chapter 5

Structure approximating Phrase Alignment problem with Hamming distance and edit distance metrics

The phrase alignment problem arises in the context of machine translation and natural language inference [MGM08]. It is a common task in these areas to determine whether one sentence can be converted into another by replacing blocks of text with semantically equivalent blocks, and possibly changing the order of the blocks. For example, a sentence "The president of the USA spoke on New Year's day" and the sentence "On January 1st, Obama gave a talk" convey the same information; we can convert the former into the latter by replacing "the president of the USA" with "Obama", "on New Year's day" with "on January 1st" and "spoke" with "gave a talk". In a more general statement of the problem, in particular in the natural language inference setting, the original sentence can contain much more information than the resulting sentence; however, here we consider the setting when the alignment must be bijective, that is, each word of the first and each word of the second sentence occur

in exactly one linked pair. Following the setting of DeNero and Klein [DK08], we call the two sequences of words to be aligned "sentences", a consecutive block of words a "phrase", and an aligned pair a "link". A set of links such that each word (in either sentence) occurs in exactly one link is called an alignment of the sentences. So in the example above, an alignment will be {(the president of the USA, Obama), (spoke, gave a talk), (on New Year's day, on January 1st)}. In practice though, there can be various degrees of how good a certain link is: there is a better correspondence between "Obama" and "the president of the USA", than with "Obama" and "the president", for example; "spoke" and "gave a talk" might not be as close semantically as the other two links. But either of them would be better than aligning "the president of the USA" with the "New Year's day". Thus, an additional parameter is needed to fully specify the problem, that is a scoring function assigning a weight to each potential link. The weighted phrase alignment problem is defined then as finding a phrase alignment with the best weight.

Now, following DeNero and Klein [DK08], we formally define a *weighted sentence alignment (WSA)* problem as follows. Let e and f be sentences. The phrases in e are represented by a set $\{e_{ij}\}$, where e_{ij} is a sequence of words from in-between-word position i to j in e; f is represented by $\{f_{kl}\}$ in the same fashion. A link is an aligned pair of phrases (e_{ij}, f_{kl}). An alignment is a set of links such that every word (token), in either sentence, occurs in exactly one link (here, we treat each occurrence of a word as a separate word). A weight function $\phi : \{(e_{ij}, f_{kl})\} \to \mathbb{R}$ assigns a weight to each link. A total weight of an alignment a, denoted $\phi(a)$, is a product of weights of its links. Now, an optimization version of the weighted sentence alignment problem asks, given (e, f, ϕ), to find the alignment with the maximum weight. A decision version of this problem can be stated as finding an alignment a of weight $\phi(a) \geq 1$.

Theorem 13. *[DK08] The decision version of the WSA problem is NP-complete.*

Proof. DeNero and Klein in [DK08] show NP-hardness of WSA by the following reduction from 3SAT. Let F be a formula with n variables and m clauses. The construction will produce an instance I of WSA consisting of sentences e and f, and a function ϕ such that there is an alignment of weight (at least) 1 in I if and only if F is satisfiable. For that, let sentence e consist of blocks of words as follows, with one word for each occurrence of a literal: $x_i^1 \dots x_1^{p_i} \bar{x}_i^1 \dots \bar{x}_i^{q_i}$, where p_i and q_i are the number of positive and negative occurrences of x_i in$F$, respectively. Thus, the length of e will be $\leq 3m$, with equality if every clause in F contains exactly 3 literals. Now, sentence f will contain two types of words. The first m words, $c_1 \dots c_m$, will correspond to the clauses of F. They will be followed by "slack words" $s_1 \dots s_n$, one for each variable in F. Finally, the function ϕ will only have values 0 and 1, and it will have the value 1 in two cases. First, if the link is of the form (c_i, l_k), where literal l_k occurs positively in clause c_i (for all occurrences of l_k). This will be used to align each clause with a literal that makes it true. Second, each slack variable s_i corresponding to a variable i will be aligned with all possible substrings of $x_i^1 \dots x_1^{p_i} \bar{x}_i^1 \dots \bar{x}_i^{q_i}$ in which either all positive or all negative copies of the variable (or both) are present. For example, if there is one positive occurrence of x_i and two negative occurrences of x_i, then the links with $\phi(e_{i,j}, f_{k,l}) = 1$ will be for $f_{k,l} = s_i$ and $e_{i,j}$ either $x_i\bar{x}_i\bar{x}_i$, or $\bar{x}_i\bar{x}_i$, or $x_i\bar{x}_i$, or x_i. The first one covers both positive and negative, the second covers all negative, and the last two all positive occurrences of the literal. These slack variables are needed to ensure that either only positive or only negative literals are left unmatched to be aligned with clause words.

To see that this reduction works, note that a satisfying assignment becomes an alignment in which every clause word is matched with one literal that makes it true (starting from the front of the block for positive and end of the block for negative), and slack variables cover the literals that remain unmatched to clauses. For the other

direction, note that there is exactly one link for each slack variable: if it is matched with a block that contains all positive occurrences of the corresponding variable in F, the corresponding variable can be set to false, otherwise it can be set to true (if it is matched with the block containing all occurrences, then either assignment works).

Assuming that F has exactly 3 variables per clause, $|e| = 3m$, $|f| = m + n$, and $|\phi| \leq (3m)^2(m+n)^2$, therefore the resulting instance is polynomial size, and the reduction runs in polynomial time.

Therefore, WSA is NP-hard. As an alignment can be checked for validity (by asserting that each word appears exactly once) and the weight of the alignment can be computed in polynomial time, the decision version of WSA is NP-complete.

□

Alternatively, NP-hardness of WSA can be shown by a reduction from the VertexCover problem. There, we are given an undirected graph $G = (V, E)$ with n vertices and m edges, and asked whether there exists a subset of k vertices called a cover such that every edge has as its endpoint at least one vertex in the cover. In an optimization version, a minimum-size such cover is sought. To show $VertexCover \leq_p WSA$, construct the instance as follows. The words of e will be blocks of copies of each vertex v_i, where the length of each such block is the degree of v_i, denoted $deg(v_i)$, plus 1, so $|e| = 2m + n$. The words of f will be of three types. The first m words $c_1 \ldots c_m$ will correspond to edges of G; the next n words are the "slack variables" $s_1 \ldots s_n$ covering leftover copies of vertices, with one extra copy always covered by s_i, and the final $n - k$ words $t_1 \ldots t_{n-k}$ in f will ensure that the size of the cover is at most k. Thus, $|f| = m + n + (n - k) = m + 2n - k$. With this intuition, define ϕ so that $\phi(v_{i,j}, c_l) = 1$ if edge c_l has v_i as its endpoint (for each copy $v_{i,j}$ of v_i), then $\phi(v_{i,j} \ldots v_{i,v_i)+1}, s_i) = 1$ for each i and all j, $1 \leq j \leq deg(v_i)$. Finally, each t_l can cover a block of the same vertices for every vertex (except for the last copy), so

$\phi(v_{i,1} \ldots v_{i,deg(v_i)}, t_l) = 1$ for every t_l and every v_i.

If there is a vertex cover of size k in G, then an alignment in the constructed instance will link all vertices other than the k vertices in the cover with t-variables, will link each edge with a copy of a vertex in the cover (in order starting from $v_{i,1}$), and variables s_i will be linked with a block of remaining copies of the corresponding vertices (consisting of at least one special copy, more if some edges have both endpoints in the cover). For the other direction, variables t_l denote vertices not in the cover, so the cover consists of the remaining vertices. If there is a cover of size smaller than k, then some s_i variables link to the whole block corresponding to such extra v_i, but this is allowed by our definition of ϕ.

5.1 Defining a natural witness for WSA

Before we can talk about structure approximation of WSA, we need to define what is meant by the witness (or feasible solution) to the WSA problem. Here, we will consider an alignment of any weight to be a feasible solution; the question remains how to represent an alignment. In DeNiro and Klein [DK08], an alignment is visualized as a matrix with words of e as columns, words of f as rows and a cell (i, k) highlighted (say, set to 1) if the block with the i^{th} word of e is linked to the block with the k^{th} word of f. Each link thus becomes a rectangular all-ones block in the matrix. This representation is not the most efficient in terms of space, although it is convenient for visualization of the solution. In particular, for the instances coming from the $3SAT \leq_p WSA$ reduction above, any feasible solution will only have $3m$ cells out of $3m \times (m + n) = N$ possible cells highlighted. In this case, it is trivial to approximate the witness to an instance of WSA produced from this 3SAT reduction: an all-zero matrix already gives a $N - (m + n)$ Hamming distance approximation.

Now, notice that the reduction above proves NP-hardness for a special case of the problem: that where all phrases in f are single words. For this restricted problem, a Hamming distance (and therefore an edit distance) approximation by an all-zero matrix is $|e|*|f|-|f|$ close to any solution. One may object that an all-zero matrix is not a valid alignment: here, we can construct an alignment by matching first $|f|-1$ words of e with words of f, and all the remaining words of e as one phrase to the last word of f. This gives us a $|e|*|f|-2|f|$ Hamming approximation for the alignment represented as $|e| \times |f|$ matrix.

As we are looking for natural (and compact) witnesses, we will use a different representation of the solution. For that, notice that finding a solution to WSA involves solving two problems: first, we need to determine how to break each sentence into phrases, and second, to determine an optimal alignment using only links involving these phrases. So a feasible solution can consist of two components: the first component with two binary strings of lengh $|e|-1$ and $|f|-1$, with 1 in between-phrase positions and 0 otherwise. The second component can list the order of phrases in f mapping to phrases in e; if there are n phrases in each, then the length of that component is $n \log n$.

What part of computing this witness, and thus of solving the WSA problem, is the hardest? Consider again the set of instances of WSA resulting from the reduction. We would like to define a special case of WSA for which we could use as small a witness as possible, and still have the NP-hardness reduction above work. As noted above, one special property of this reduction is that it always produces a partition of f where every phrase is exactly one word. The information encoded in the second part of the witness described in the previous paragraph, the string of $|f|-1$ bits denoting the phrase boundaries in f, is therefore redundant.

Secondly, ϕ involved in the reduction has a special property that it can only take values

0 and 1. In that case, after solving the first part of the problem (finding splitting points between phrases in e and f), the second part can be computed in polynomial time by the standard network flow algorithm for bipartite perfect matching, with phrases of e and f forming the vertices of the bipartite graph, and an edge connecting two vertices v and u iff $\phi(v, u) = 1$. Thus, in this case it is enough to compute a witness that contains only the binary strings denoting splitting points between phrases, as described above.

Now, combining the two restrictions we will define a problem PWSA, which is a special case of WSA satisfying the properties above.

Definition 20 (PWSA)**.** *The PWSA (for "partition" WSA) problem is defined as follows. Given as input* (e, f, ϕ) *where* $\phi : \{(e_{ij}, f_{kl})\} \rightarrow \{0, 1\}$*, find a partition of* e *into phrases such that there is an alignment of weight 1 of phrases in this partition with words of* f*.*

A natural witness w *for PWSA is a binary string* $w_1 \dots w_{|e|-1}$ *such that if* e_{ij} *is a phrase in the optimal alignment, then* $w_i = w_j = 1$*, or* $w_j = 1$ *and* $i = 0$*, or* $w_i = 1$ *and* $j = |e|$*; and* $\forall k, i < k < j, w_k = 0$*. Note that* w *has to have* $|f| - 1$ *1s for any valid alignment.*

Here, the NP-hardness follows by the same $SAT \leq_p WSA$ reduction as in theorem 13, where the satisfying assignment is recovered from w by running the network flow algorithm and determining, as before, the values of the variables of F from the links with slack variables s_i. Moreover, for variables with more than two positive and two negative occurrences the value can be determined directly from w. Suppose a slack variable covers all positive occurrences of a variable v, and leaves out some negative occurrences. Then, there will be no splitting points within the block denoting the positive literals, but there will be as many splitting points for the negative literals as there are clauses which use them. From that, already, it can be inferred that the

negative occurrences were used to satisfy the clauses, thus the variable needs to be set to false. So if a substring w_{ij} of w corresponding to a block of encoding a literal v (without the endpoints) is of the form 1111....0000, then we can immediately infer that $v = true$, otherwise if it is of the form 000....1111, $v = false$. It would not work if there is exactly one positive or negative occurrence of a variable; but this can be resolved by modifying the reduction so that there is always an extra "$v_i\bar{v}_i$" (or a single dummy variable) in the middle of each block, and $\phi(x \ldots x) = \phi(\bar{x} \ldots \bar{x}) = 0$.

5.2 Hamming and edit distance inapproximability of PWSA

In this section we will show that PWSA cannot be Hamming or edit distance structure approximated to within $n/2 - n^{\epsilon}$, with respect to the witness defined above. From there, structure inapproximability of WSA can be derived, albeit with weaker parameters.

Theorem 14 (Hamming inapproximability of PWSA)**.** *Let (e, f, ϕ) be a valid input to PWSA. If there is a polynomial-time algorithm $A(e, f, \phi)$ computing a string w which is within Hamming distance $n/2 - n^{\epsilon}$ of a witness, then P=NP.*

Proof. We will show how to use such a structure approximation algorithm A for PWSA to compute the exact value of the first variable in F, in a manner similar to the proof of Hamming inapproximability of SAT.

Let F be a formula on n variables and m clauses. Let (e, f, ϕ) be an instance of PWSA constructed from F by the reduction in theorem 13. Now, we will amplify the part of (e, f, ϕ) with respect to the first variable v in F as follows. Choose k such that $n^k > 1.5m$. Imagine that F is augmented with $n^{k/\epsilon}$ copies of the dummy clause

$(v \vee \bar{v})$. If the reduction is applied to this extended formula, it will have an effect of introducing $n^{k/\epsilon}$ copies of the literal v and $n^{k/\epsilon}$ copies of the literal $\bar{v}$ as words of e (that is, the first $n^{k/\epsilon} + p$ words of e will be copies of v, and the following $n^{k/\epsilon} + q$ words of e will be copies of $\bar{v}$, where p and q are the numbers of positive and negative occurrences of v in the original F.) The clauses $(v \vee \bar{v})$ will become $n^{k/\epsilon}$ new words in f (say first $n^{k/\epsilon}$ words of f). Finally, $\phi(e_{ij}, f_{kl})$ can be defined as before. This amplification preserves the correctness of the reduction, as the link (e_{ij}, s_1) forces only copies of v or only copies of $\bar{v}$ to be used to satisfy the dummy clauses. Now, if w is a correct witness (of length $N = 3m + 2n^{k/\epsilon} - 1$) to this instance, the value of v can be determined immediately: if w starts with a string of at least $n^{k/\epsilon}$ 1s, then $v = true$, and if w starts with at least $n^{k/\epsilon}$ 0s, then $v = false$.

Suppose that there is an algorithm A that returns a "corrupted" string w' that agrees with w on at least $N/2 + N^{\epsilon}$ bits. Here, we are not even concerned whether w' is a valid alignment (i.e., has $|f| - 1$ ones); any such w' will work. That is, w' agrees with w on $(3m + 2n^{k/\epsilon} - 1)/2 + (3m + 2n^{k/\epsilon} - 1)^{\epsilon} \geq (3m + 2n^{k/\epsilon} - 1)/2 + n^k$ locations. Now, suppose that all the errors lie within the $2n^{k/\epsilon}$ positions corresponding to extra copies of v and $\bar{v}$. Since we chose k such that $n^k > 1.5m$, and ignoring $-1/2$, there are at $n^{k/\epsilon} + n^k - 1.5m > n^{k/\epsilon}$ correct bits in that block, that is more than half of copies of v and $\bar{v}$ are computed correctly. Taking majority now gives us the correct value of v. □

Now, this result can be extended to show edit distance inapproximability in much the same way as we have done for VertexCover.

Corollary 4. *PWSA cannot be approximated in polynomial time to within edit distance $n - n^{\epsilon}$ unless $P = NP$.*

Proof. We will use the same class of instances as in theorem 14. Note that the

substring of w that we are interested in is $w_1 \dots w_r$, where $r = 2n^{k/\epsilon} + p + q$, which is the block corresponding to the first variable v in F. In a correct witness, this substring is either of the form 1111....000000 or 000....11111, with the number of 0s and 1s at least $n^{k/\epsilon}$ each. Now, suppose an approximation algorithm A produces a string w' which is edit distance $N/2 - N^{\epsilon}$ of w; that is, w' can be converted to w with at most $N/2 + N^{\epsilon}$ insertion, deletion and replacement operations. Consider a substring $w'_1 \dots w'_r$ in w'. As for the case of VertexCover, we can argue that the Hamming distance between $w_1 \dots w_r$ and $w'_1 \dots w'_r$ is at most $N/2 - N^{\epsilon}$. Indeed, suppose for the sake of contradiction that the Hamming distance between $w_1 \dots w_r$ and $w'_1 \dots w'_r$ is greater than the edit distance between these two substrings. As they have the same size, the number of insertions is the same as the number of deletions. Now, it is sufficient to say that the pair insertion/deletion can introduce at most one 0 in the "1111...1" part, and at most one 1 in the "0000..000", by the same argument as in theorem 11. Therefore, the Hamming distance inapproximability implies edit distance inapproximability with the same parameters. □

In the proofs above, we have shown inapproximability results for the problem PWSA, in which the second sentence is assumed to be partitioned as one word per phrase. A more realistic scenario would be to assume that the witness consists of the partition strings for both e and f (here, we are still assuming that ϕ takes values in $\{0, 1\}$). The corollary below shows that for a weaker bound, there is still an inapproximability. The weakening here comes from the fact that our block becomes a smaller fraction of the total length of the witness, since f contains $n^{k/\epsilon}$ words corresponding to the dummy clauses.

Corollary 5. *WSA with $\phi \in \{0, 1\}$ cannot be approximated to within Hamming distance or edit distance $2n/3 + n^{\epsilon}$.*

Proof. Consider the same reduction as before, but now our witness consists of both the string encoding the splitting points of e and a string encoding the splitting points of f. Thus, the total length N of the witness becomes, ignoring "-1"s, $N = (3m + 2n^{k/\epsilon}) + (n^{k/\epsilon} + m + n) = 4m + 3n^{k/\epsilon} + n$. If the calculation above is done with this value of N, then we end up with only $0.5n^{k/\epsilon}$ guaranteed correct positions in our $2n^{k/\epsilon}$ block of interest. We need c, $0 < c < 1$, such that $N * c + N^{\epsilon} - (N - 2n^{k/\epsilon}) > n^{k/\epsilon}$. Such c can be determined from the following calculation:

$$
\begin{aligned}
N * c + N^{\epsilon} - (N - 2n^{k/\epsilon}) &= (4m + 3n^{k/\epsilon} + n) * c + (4m + 3n^{k/\epsilon} + n)^{\epsilon} - (4m + n^{k/\epsilon} + n) \\
&> (4m + 3n^{k/\epsilon} + n) * c + n^{k} - (4m + n^{k/\epsilon} + n) \\
&= (3c - 1)n^{k/\epsilon} + n^{k} - 4(1 - c)m - (1 - c)n \\
&> (3c - 1)n^{k/\epsilon} \qquad \text{(when } k \text{ chosen so that } n^{k} > 4m + n\text{)} \\
&= n^{k/\epsilon} \qquad \text{(for } c = 2/3\text{)}
\end{aligned}
$$

□

Chapter 6

Conclusion and future work

In this thesis, we considered the complexity of approximating solution structure of decision and optimization NP problems in the [HMvRW07, vRW12] sense. Most of known results in this setting apply to Hamming distance function, with Sheldon and Young [SY13] giving the strongest such results. In particular, any paddable NP-hard problem has witnesses not approximable to within $n/2 + \sqrt{n \log n}$ and the same bound holds for the universal NP problem. Furthermore, for natural witnesses to many NP-hard problems such as SAT and Vertex cover, it is possible to achieve $n/2$ approximation, yet NP-hard to do better than $n/2 - n^{\epsilon}$ for any ϵ.

To the best of our knowledge, distance functions other than Hamming distance were not considered by anybody other than [HMvRW07, vRW12]. In particular, they give a number of inapproximation results for self-paddable functions. In this thesis, we also extend some Hamming distance inapproximability results to edit distance (note that, large Hamming distance does not necessarily imply large edit distance). We also make a number of observations, in particular concerning usability of existing value approximation algorithms for structure approximation.

The lower bounds are pessimistic though, and imply quite strong inapproximabil-

ity for these problems. Is there any way to get around these bounds in practice? In fact, many practical problems have strong (constant) restrictions on some of the parameters, leading to efficient parameterized algorithms. Already [HMvRW07] introduced the notion of parameterized structure approximation setting, and proved, in particular, a lower bound on parameterized complexity of Longest Common Subsequence with respect to edit distance metric. This would be an interesting direction to investigate with respect to problems and metrics occurring in practical applications. Another question concerns upper bounds. Even though, as we note here, some of the classical linear programming relaxation algorithms can be recast in the structure approximation framework, it is not clear how to make a general statement of this form. Here, as well, a parameterized framework can be useful.

For the more theoretical direction, it would be interesting to see if there is a generic way to build a lattice of hardness implications for various metrics. We conjecture , in particular, that any metric with a certain "locality property" (that is, one "unit of change" only affects a small, though not necessarily constant) number of positions should be inapproximable by generalizing Hamming distance results. Alternatively, one wonders if there is a non-trivial, practically interesting metric for which there is, indeed, a fast approximation algorithm for any NP-hard problem. In that respect, considering various metrics and their interrelation w.r.t. computational problems is a promising area with a possibility for new approaches to problems from a wide variety of computational fields.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[ACG+99] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization: Problems and Their Approximability Properties*. Springer, 1999.

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *In Proc. 33rd Ann. IEEE Symp. on Found. of Comp. Sci*, pages 14–23, 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs; a new characterization of np. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, SFCS '92, pages 2–13. IEEE Computer Society, 1992.

[BH77] Leonard Berman and Juris Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[DK08] John DeNero and Dan Klein. The complexity of phrase alignment problems. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 25–28. Association for Computational Linguistics, 2008.

[FLN00] Uriel Feige, Michael Langberg, and Kobbi Nissim. On the hardness of approximating NP witnesses. In *APPROX*, pages 120–131, 2000.

[GHLP99] Anna Gal, Shai Halevi, Richard J. Lipton, and Erez Petrank. Computing the partial solutions. In *14th Annual IEEE Conference on Computational Complexity (CCC'99)*, pages 34 – 45, 1999.

[GR08] Venkatesan Guruswami and Atri Rudra. Soft Decoding, Dual BCH Codes, and Better List-Decodable ε-Biased Codes. In *IEEE Conference on Computational Complexity*, pages 163–174, 2008.

[GR11] Venkatesan Guruswami and Atri Rudra. Soft Decoding, Dual BCH Codes, and Better List-Decodable ε-Biased Codes. *IEEE Transactions on Information Theory*, 57(2):705–717, 2011.

[Gra66] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[HMvRW07] Matthew Hamilton, Moritz Müller, Iris van Rooij, and Todd Wareham. Approximating solution structure. In Erik Demaine, Gregory Z. Gutin, Daniel Marx, and Ulrike Stege, editors, *Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs*, number 07281 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, 2007.

[Hoc97] Dorit S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[HP13] Sariel Har-Peled. Approximation algorithms. Course Lecture, 2013.

[Joh73] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 38–49. ACM, 1973.

[Kar72] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KS99] Ravi Kumar and D. Sivakumar. Proofs, Codes, and Polynomial-Time Reducibilities. In *IEEE Conference on Computational Complexity*, pages 46–53, 1999.

[MGM08] Bill MacCartney, Michel Galley, and Christopher D Manning. A phrase-based alignment model for natural language inference. In *Proceedings of the conference on empirical methods in natural language processing*, pages 802–811. Association for Computational Linguistics, 2008.

[MGU72] R.L. Graham M.R. Garey and J.D. Ullman. An analysis of some packing algorithms. *Combinatorial Algorithms (Courant Computer Science Symposium,No. 9)*, page 39–47, 1972.

[SY03] Daniel Sheldon and Neal E. Young. Hamming approximation of NP witnesses. (Unpublished manuscript referenced in Guruswami/Rudra 2008), 2003.

[SY12] Daniel Sheldon and Neal E. Young. Hamming approximation of np witnesses. *CoRR*, abs/1208.0257, 2012. http://arxiv.org/abs/1208.0257.

[SY13] Daniel Sheldon and Neal E. Young. Hamming approximation of np witnesses. *Theory of Computing*, 9(22):685–702, 2013. First draft circulated in 2003.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[Viz64] V.G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 1964.

[vRW12] Iris van Rooij and Todd Wareham. Intractability and approximation of optimization theories of cognition. *Journal of Mathematical Psychology*, 56(4):232 – 247, 2012.