

# **Exploring the opportunities of Structure Approximation**

by

©Md Renesa Nizamee

A (Thesis, Dissertation, or Report) 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

**Month and Year Submitted**

St. John's

Newfoundland

# Abstract

Text for the abstract.

# Acknowledgements

Text for the acknowledgements.

# Table of Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iii</b>
<b>Table of Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Approximation Algorithm . . . . .	1
1.1.1 Value-approximation algorithm . . . . .	3
1.1.2 Approximation results for different problems . . . . .	4
1.2 Hardness of approximation results . . . . .	6
1.3 Structural Approximation . . . . .	7
1.3.1 Solution Structural Approximation Algorithms . . . . .	8
1.3.2 The relationship between value and structural approximation .	8
1.3.2.1 Structural approximation implies Value approximation	8
1.3.2.2 Dissociation between Structural approximation and Value approximation . . . . .	8
1.4 Motivation . . . . .	8
1.5 Our Results . . . . .	9

<b>2 Preliminaries and Previous Works</b>	<b>10</b>
2.1 Structure Approximation Framework . . . . .	10
2.2 Partial and Approximate NP-witnesses . . . . .	10
2.2.1 Kumar and Sivakumar: hard-to-approximate witnesses for all NP languages . . . . .	10
2.2.2 Sheldon and Young: Hamming Approximation of NP Witnesses	13
2.3 Main tool: Error correcting codes . . . . .	14
2.4 Feige/Langberg/Nissim technique . . . . .	15
2.5 Open problems . . . . .	17
<b>3 Case Study: KNAPSACK</b>	<b>18</b>
3.0.1 Problem 1 . . . . .	18
3.1 Problem Definition . . . . .	19
3.2 NP-hardness of KNAPSACK . . . . .	19
3.3 Different algorithms for finding the solutions for KNAPSACK . . . . .	19
3.4 Example . . . . .	20
3.5 Relation between s-FPTAS and structural approximation for NP-hard problems . . . . .	26
<b>4 Other problems and generalizations</b>	<b>27</b>
4.1 HamCycle? . . . . .	27
4.2 Other than Karp's 21 problems . . . . .	27
4.3 Edit Distance for something else . . . . .	27
4.4 Other functions? General statement about functions? . . . . .	27
<b>5 Conclusion and future work</b>	<b>28</b>
<b>Bibliography</b>	<b>29</b>

# List of Tables

1.1	Approximation results of different problems . . . . .	4
1.2	Classification of problems based on their inapproximability results .	6
3.1	Solution Table . . . . .	22
3.2	Solution Table for the newly obtained profits . . . . .	25

# List of Figures

# Chapter 1

## Introduction

### 1.1 Approximation Algorithm

There are many important computational problems that are tough to solve in a best possible way. As a matter of fact, most of those problems are NP-hard, that means no polynomial-time algorithm exists that resolves the problem in a most favourable way unless  $P = NP$ .

The class P consists of those problem which are solvable in polynomial time. That means those can be solved in time  $O(n^k)$  for some constant  $k$ , where  $n$  is the size of the input to the problem.

The class NP consists of problems those are verifiable in polynomial time. That means if a "certificate" of the solution is given; that can be verified in polynomial time in the size of the input to the problem. It is easy to see that  $P \subseteq NP$ , since any problem that can be solved in polynomial time can also have its solutions verified as correct in polynomial time. [CLRS01].

Euclidean travelling salesman problem (Euclidean TSP) could be a common example of NP-hard problem: Given a set of points on the plane, the goal is to find the shortest



tour that visits all the points. Another important NP-hard problem is independent set: given a graph  $G = (V, E)$ , a maximum-size independent set  $V^* \subset V$  needs to be found.

At this instant the question that arises is what should we do when such difficult problems come along, for which we don't expect to find polynomial-time algorithms? Algorithms that have exponential running time are not effective, if the input size isn't really small. As a result, we need to look for a solution approximately since absolutely the optimal solution is not before us. Most likely 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  $\rho$  times the minimum length of a tour, for a small value of  $\rho$ . An algorithm which produces a solution which is ensured to be within some factor of the optimum is called an approximation algorithm.

As mentioned earlier, NP-hard problems has no polynomial time algorithms unless  $P = NP$  and to find out the solution of those problems the best possible way is to use Approximation algorithm which will return the solution within some factor of the optimum or exact solution. Approximation algorithms also bring down the running time of the problem which could have exponentially huge running time if Approximation Algorithms were not used.

So, the performance of Approximation algorithm is an important issue. While it is assumed that every approximation algorithm is polynomial but there exist a lot of polynomial algorithm which can be inefficient and impractical. So, in order to get 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 and FPTAS are those approximation schemes. As we may guess that the better

the approximation, the larger may be the running time. A problem has PTAS if the running time is polynomial in the size of the input and a problem has FPTAS if the running time is polynomial both in size of the input and the inverse of the performance ratio ( $1/\epsilon$ ). [ACG<sup>+</sup>99]. So, the classes of problems that has FPTAS are contained in classes of problems that has PTAS. For structure approximation we have sPTAS and sFPTAS as structure approximation scheme.

[PTAS] Let  $P$  be an NP optimization problem. An algorithm  $A$  is said to be a polynomial time approximation scheme (PTAS) for  $P$  if, for any instance  $x$  of  $P$  and any rational value  $r > 1$ ,  $A$  when applied to input  $(x, r)$  returns an  $r$ -approximate solution of  $x$  in time polynomial in  $|x|$ . [ACG<sup>+</sup>99]

[FPTAS]

An algorithm  $A$  is said to be a fully polynomial time approximation scheme (FPTAS) for  $P$  if, for any instance  $x$  of  $P$  and any rational value  $r > 1$ ,  $A$  when applied to input  $(x, r)$  returns an  $r$ -approximate solution of  $x$  in time polynomial both in  $|x|$  and in  $1/(r - 1)$ . [ACG<sup>+</sup>99]

### 1.1.1 Value-approximation algorithm

Many types of approximation algorithms have already been defined. [Value-approximation algorithm] If the  $val(x, A(x))$  is within some absolute or relative factor of  $optval(x)$ ; we refer to it as value-approximation algorithm.

Some classes of value-approximation are defined as following:

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

[HMvRW07]

[Ratio value-approximation (v-r-approx)algorithm] Given an optimization problem  $\pi = (I, cansol, val, goal)$  and a non decreasing function  $h : N \rightarrow N$ , an algorithm  $A$  is a polynomial time  $h(|x|)$  ratio value-approximation (v-r-approx)algorithm for  $\pi$  if for every instance  $x \in I$ ,  $R(x, A(x)) \leq h(|x|)$ , where  $R(x, A(x)) = \frac{optval(x)}{val(x, A(x))}$  (if goal=MAX) or  $\frac{val(x, A(x))}{optval(x)}$  (if goal=MIN), and  $A$  runs in polynomial time in  $|x|$ . [HMvRW07]

### 1.1.2 Approximation results for different problems

A question may arise that how good can be an approximation algorithm? A nice sorting of these approximation results can be found in [Hoc97] where the author sorted the results according to "good", "better", "best", "better than best", "wonderful" categories.

Table 1.1: Approximation results of different problems

Category	Category definition	Example
Good	Has a $\delta$ -approximation for some constant $\delta$	Vertex cover problem, MAX CUT
Better	Has an Approximation Scheme(PTAS, FPTAS) for the problem	KNAPSACK, Minimum makespan
Best	Has a polynomial c-approximation algorithm	k-Center problem
Better than best	Has a certain asymptotic performance ratios	Bin packing, Edge coloring
Wonderful	Delivers solution within 1 of the optimum	Edge coloring in simple graphs, Coloring of planar graphs, Min max degree spanning tree

[c-approximation algorithm] For an optimization problem if the problem of approximation within  $c - \delta$  is NP-complete for any  $\delta > 0$ , then it is called a polynomial

c-approximation algorithm. [Hoc97]

Problem definitions[Hoc97] :

**[VC] Vertex Cover**

*Instance:* A graph  $G = (V, E)$ , vertex weight  $w_v$  for all  $v \in V$

*Objective:* Find a subset of vertices  $C \subseteq V$  that forms a cover so that each edge  $e \in E$  has at least one of its endpoints in  $C$  such that  $\sum_{v \in C} w_v$  is minimized.

**[MAX CUT] Maximum Cut**

*Instance:* A graph  $G = (V, E)$ .

*Objective:* Find a subset  $C$  of the vertices that maximizes the number of edges that have one vertex in  $C$  and one not in  $C$ .

**[MM] Minimum makespan**

*Instance:* A set of jobs  $J=1, \dots, n$  with associated processing times  $p_1, \dots, p_n$ ; a positive integer  $m < n$ .

*Objective:* Partition  $J$  into subsets  $J_1, \dots, J_m$  such that  $\max_{1 \leq i \leq m} \sum_{j \in J_i} p_j$  is minimized.

**[k-CEN] k-Center**

*Instance:* A graph  $G = (V, E)$  with edge weights  $w_{(i,j)}$  for  $(i, j) \in E$  and a parameter  $k \leq |V|$

*Objective:* Find a set  $S, |S| = k$  so that  $\max_{v \in V} \min_{s \in S} w_{(v,s)}$  is minimized.

**[PA] Packing**

*Instance:* Let  $Z_+$  denote the non-negative integers,  $A$  be an  $m * n$  matrix over  $Z_+$ ,  $b$

Table 1.2: Classification of problems based on their inapproximability results

Class	Factor of Approximation that is hard	Examples
I	$1 + \epsilon$	MAX-3SAT
II	$O(\log n)$	SETCOVER
III	$2^{\log^{1-\lambda} n}$	LABELCOVER
IV	$n^\epsilon$	CLIQUE

be a vector over  $Z_+^m$ , and  $x$  and  $w$  be vectors over  $Z_+^n$ .

*Objective:* Maximize  $w^T \cdot x$  subject to  $Ax \leq b$ .

### [COL-EDGE] Edge coloring

*Instance:* A graph  $G = (V, E)$

*Objective:* Assign a color (namely, an integer) to each edge that no two adjacent edges have the same color and such that the number of different color is minimized.

## 1.2 Hardness of approximation results

The main motivation behind designing an approximation algorithm for a NP-hard problem is to find an efficient approximate solution for that problem. But not all NP-hard problems has approximate solutions. Some of these problems don't have any efficient algorithms that can achieve an approximation beyond a certain threshold. These results are known as "inapproximability" or "hardness of approximation results", which are proved under the general hypothesis  $P \neq NP$ .

The problems having inapproximability results are divided into four classes based on the hardness of their approximation ratio. The following table shows the different classes and their approximation ration which is hard to achieve.[HOCHBAUM+see dr. Arora's papers]

The main idea to prove the inapproximability result for a NP-hard problem is to start with a known inapproximable problem first and then perform a gap-preserving reduction from that problem to the given problem. As MAX-3SAT is the most basic inapproximable problem so all other problems from different classes are proved to be inapproximable by the help of MAX-3SAT.

[Gap preserving reduction] Let  $\Pi$  and  $\Pi'$  are two maximization problems. A gap-preserving reduction from  $\Pi$  to  $\Pi'$  with parameters  $(c, p), (c', p')$  is a polynomial-time algorithm  $f$ . For each instance  $I$  of  $\Pi$ , algorithm  $f$  produces an instance  $I' = f(I)$  of  $\Pi'$ . The optimal of  $I$  and  $I'$ , say  $OPT(I)$  and  $OPT(I')$  respectively, satisfy the following property:

$$OPT(I) \geq c \implies OPT(I') \geq c'$$

$$OPT(I) < \frac{c}{p} \implies OPT(I') < \frac{c'}{p'}$$

Add PCP!

## 1.3 Structural Approximation

As we have seen in the last section, what is Approximation Algorithm and how it plays an important part to find a solution for some computation problems. The next question that can arise in one's mind is that how can we approximate an optimum solution. One possibility is that a candidate solution is considered an approximation to the extent that its value is closer to the optimal value. This type of approximation is called value-approximation. There is another possibility, according to which a candidate solution is considered an approximation of the optimal solution to the extent that it resembles the optimal solution itself. This kind of approximation is called structure-approximation [HMvRW07].

If we consider Approximation algorithm in the domain of cognitive science, we can

see the importance of structural-approximation right away. The cognitive outcomes of problems are usually modelled as the outcome of intractable optimization problems and it may not be realistic to compute such outputs by the cognitive scientists with their limited resources. But as the outcome of optimization problem seems to predict and explain human cognitive outcomes well, it has been proposed that the cognitive scientists may somehow approximate the optimal outputs[add the reference page 2]. In this context it is called structural-approximation. Millgram[] has done some work which is relevant to structural-approximation and [HMvRW07] was inspired by that research.

Add F Coherence

### **1.3.1 Solution Structural Approximation Algorithms**

### **1.3.2 The relationship between value and structural approximation**

#### **1.3.2.1 Structural approximation implies Value approximation**

Consider the MAX CLIQUE problem (see figure ). The graph at left side has a MAX CLIQUE of value 4 and the graph at right side has a MAX CLIQUE of value 4. Though they differ by  $(4-3)=1$  in their value but they are as far apart in structure as possible.

Figure [max clique]

#### **1.3.2.2 Dissociation between Structural approximation and Value approximation**

MAX CUT

## 1.4 Motivation

---

## 1.5 Our Results



# Chapter 2

## Preliminaries and Previous Works

---

### 2.1 Structure Approximation Framework

---

### 2.2 Partial and Approximate NP-witnesses

---

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

Motivated by [?] as well as then-recent PCP theorem, Kumar and Sivakumar [?] set out to generalize the results of [?] 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, [?] considered both the erasure setting of [?]

(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 [?]).

**Definition 1.** *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  $\hat{R}_L(x, w) \equiv \exists y, (w = \text{Code}(y) \wedge R_L(x, y))$ . Here,  $\text{Code}(y)$  is an error-correcting code (more on it later).*

Thus, if there is a polynomial time algorithm that decodes  $w = \text{Code}(y)$  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 [?].

**Theorem 1** ([?]). *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, and three different inner codes. That is, a string  $y$  is first viewed as a sequence of evaluations of a polynomial  $p_y(u) = \sum_j y_j u^j$  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 2** ([?]). *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  $N^\delta$  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  $\delta$ -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 [?] 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 [?], (journal version [?]). 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  $\gamma$  (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 witness 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.

### 2.2.2 Sheldon and Young: Hamming Approximation of NP Witnesses

Guruswami and Rudra briefly mention an unpublished manuscript of Sheldon and Young [?] that achieves hardness of structure approximation for SAT  $|w|/2 + |w|^\eta$  for any  $\eta$ . They were motivated by Feige, Langberg and Nissim’s work [?] and set to strengthen their hardness result. They show, for the universal NP-complete language, no deterministic polynomial-time algorithm can achieve Hamming distance  $n/2 + O(\sqrt{n \log n})$  (unless  $P=NP$ ). Also, no randomized polynomial-time algorithm can achieve  $n/2 + o(\sqrt{n \log n})$  with probability  $1 - 1/n^O(1)$  (unless  $RP=NP$ )[?].

Another contribution of Sheldon and Young’s paper was to find out the lower bounds

for randomized algorithms. This result is shown in the following theorem.

**Theorem 3.** *Fix any  $\epsilon > 0$ . For the universal NP-complete language  $\nu$  and it's witness relation  $R_\nu$ , no polynomial-time algorithm achieves Hamming distance  $n/2 + \sqrt{\epsilon n \ln n}$  with probability  $1 - O((n^{2\epsilon} + 1)\sqrt{\epsilon \ln n})^{-1}$ .*

Here we will discuss their result for SAT.

**Lemma 2.2.1.** *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  $P=NP$ .*

*Proof.* They prove the lemma by describing a polynomial time algorithm  $A'$  that solves a SAT in polynomial time. Given a SAT instance  $I$  it create new instance  $I'$  from  $I$  by duplicating a variable  $X$ ;  $m$  times where  $m = \lceil 1 + (n/2)^{1/\epsilon} - n \rceil$ . That means  $A'$  adds to  $I$  new variables  $X^1, X^2, \dots, X^m$  and adds new clauses  $(X^1 = X^2) \wedge (X^2 = X^3) \wedge \dots \wedge (X^{m-1} = X^m)$ .

Algorithm  $A'$  runs on the instance  $I'$  of  $A$  to achieve hamming distance  $(m+n)/2 - (m+n)^\epsilon$ . We assume  $b$  to be a true or false value which is assigned to atleast  $m/2 + 1$  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. [?]  $\square$

Suppose we have a SAT formula,  $(x \vee y) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{y} \vee \bar{u})$  and we assume  $\epsilon = 1/3$ . It has 4 variables in the form of  $x, y, z$  and  $u$  so,  $n$  will be 4 here. As we know from the lemma  $m = \lceil 1 + (n/2)^{1/\epsilon} - n \rceil$  so,  $m = \lceil 1 + (4/2)^3 - 4 \rceil = 5$  here.

UNFINISHED!

## 2.3 Main tool: Error correcting codes

---


## 2.4 Feige/Langberg/Nissim technique

This paper plays an important part in structure approximation research though the results the authors get out of this paper are of a negative nature. They show that for many well known NP-complete problems, it is NP-hard to produce a solution whose Hamming distance from an optimal solution is considerably closer than getting a solution by taking only a random string [?].

Here they deal with the natural witnesses of the problems. The solution for an optimal value for a NP-complete problem can be called a witness. The witness of a problem is not universal but different problems has different natural witnesses. Suppose, if we consider a SAT problem then the satisfying assignment can be called a witness for that problem. Similarly the bound on the KNAPSACK can be the witness for a KNAPSACK problem and if we consider a Clique problem it can have two witnesses. The set of the vertices by which the clique is created can be a witness and similarly 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.

They use many to one reduction in their approach. Such as, from SAT we can get the witness approximation for Clique and so on where Sheldon and Young's [?] approach is the turing reduction.

In this paper they takes some of the problems from Karp's list of 21 NP-complete decision problems [citation] and find out their witness approximation (Hamming distance). The following table shows the result of the problems they mention in the paper.



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

Now we will see the witness inapproximability of SAT which was showed in this paper. They show that given a satisfiable Boolean formula  $\Phi$ , it is NP-hard to find an assignment  $x$  to the variables of  $\Phi$  which is of Hamming distance significantly less than  $1/2$  to some satisfying assignment of  $\Phi$ .

**Claim 1.** *Approximating satisfying assignment for 3SAT (and 3CSP) formula within Hamming distance  $1/2 - \epsilon$  is NP-hard.*

*Proof.* Let  $R = (w, \phi_0) | \exists a \text{ s.t. } w = C(a) \text{ and } \phi_0(a) \text{ is true}$  where  $\phi_0$  is a SAT formula and the size of the witness  $w$  is  $n$ . They use a technique of finding out the 'core' of the problem and later 'amplify' it. The 'core' of the problem is those variables which are hard to approximate which is  $\phi_1$  here. And by 'amplify' they mean the amplification of the 'core' variables to obtain the final formula.

At first using any standard reduction like Cook's theorem,  $R$  is transformed into a 3SAT formula. Then the variables of  $\phi_1$  is denoted as  $w_1, w_2, \dots, w_n; z_1, z_2, \dots, z_l$ , where the  $w_1, w_2, \dots, w_n$  in a satisfying assignment to  $\phi_1$  represent a witness  $w$  to  $\phi_0$  in  $R$  and  $z_1, z_2, \dots, z_l$  are the auxiliary variables which are added by the reduction.

Then comes the amplification step where a 3CSP formula  $\phi_2$  is constructed by duplicating the variables  $w_1, w_2, \dots, w_n$  so that the number of auxiliary variables  $z_1, z_2, \dots, z_n$  is small compare to the number of total variable. the construction of  $\phi_2$  looks like  $\phi_2(w_1, w_2, \dots, w_n; w_1^1, w_2^2, \dots, w_n^m; z_1, z_2, \dots, z_n)$  and it can be defined as

$$\phi_2 = \phi_1(w_1, w_2, \dots, w_n; z_1, z_2, \dots, z_n) \wedge \bigwedge_{i=1}^n (w_i = w_i^1) \wedge \bigwedge_{i=1}^n \bigwedge_{j=1}^{m-1} (w_i^j = w_i^{j+1})$$

In some point it contradicts the witness inapproximability of relation  $R$  above thus proving the claim.

□

The rest of the problems e.g., Clique, Chromatic number etc are done based on many to one reduction. The remaining problems in the Karp's list which are not discussed here can be done by slight adjustment to the standard reductions. The authors of this paper state that those proofs will appear in a future version of their paper though it never appeared. So, I made a contact with one of the authors, Dr. Uriel Feige and he told me that partly because of the lack of space and partly because of the technique they used in this paper to prove the witness inapproximability of some of the problem, would produce a proof for remaining of the problems.

## 2.5 Open problems



# Chapter 3

## Case Study: KNAPSACK

KNAPSACK is a very interesting NP-hard problem in the field of theoretical Computer Science. These problems have been intensively studied in the last few decades due to their simple structure and the similarity with some more complex problems. And the most amazing thing about KNAPSACK is it has tons of real life applications. Suppose there is a fire evacuation going on in a town. The inhabitants of that town will try to take those things which are valuable to them within a certain limit. This is a real life KNAPSACK example. Or one can think of a humanitarian mission going on in a place where a Helicopter is trying to bring the people food, clothes etc. But as the helicopter can carry a certain amount of weight, it will carry those items that are most important to the people within that weight limit.

[picture]

### 3.0.1 Problem 1

**A natural problem with easy value-approximation (a FPTAS), but hard to structure approximate.**

### 3.1 Problem Definition

[KNAPSACK]

Given a set  $S = (a_1, a_2, \dots, a_n)$  of objects, with specified size and profits,  $\text{size}(a_i) \in \mathbb{Z}^+$  and profit  $(a_i) \in \mathbb{Z}^+$ , and a "knapsack capacity"  $B \in \mathbb{Z}^+$ , find a subset of objects whose total size is bounded by  $B$  and the total profit is maximized. [Vaz01]

### 3.2 NP-hardness of KNAPSACK

---

### 3.3 Different algorithms for finding the solutions for KNAPSACK

Different algorithms have been studied in the recent past to find out the optimal/approximate solution for KNAPSACK [MT90–powerpoint].

- The Greedy Algorithm : The above definition of KNAPSACK is for 0-1 KNAPSACK. There is also another kind of variation which we call "Fractional KNAPSACK". The difference between 0-1 and fractional KNAPSACK is mainly in choosing the object to obtain maximum profit within a certain weight bound. In 0-1 KNAPSACK, an item has to be taken fully in KNAPSACK or it is left behind. But with Fractional KNAPSACK, any fraction of the item can be taken to obtain maximum profit keeping the weight bound in mind.

The greedy algorithm is very efficient for finding the solution for Fractional KNAPSACK. The algorithm is discussed below in brief:

- Compute the profit per weight  $\frac{p_i}{w_i}$  for each item.

- Obeying a greedy strategy, take as much as possible of the item with the greatest profit per weight .
  - If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next profit per weight , and so forth until there is no more room.
- Branch and Bound Algorithms
  - Dynamic Programming Algorithms : Discussed in the next section
  - Reduction Algorithms
  - Approximate Algorithms: Discussed in the next section
  - Exact Algorithms for large size problems

### 3.4 Example

We will see a real life example of KNAPSACK problem here and we would like to find out the solution of that problem using dynamic programming.

Let us consider a scenario where a Burgler is trying to steal something from a Jewellery shop. The Burgler has a KNAPSACK and that can carry a certain weight,  $B=12$ .

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

#### Weight and profit breakdown for each item

	Weight	Profit
<b>Item A</b>	5	2
<b>Item B</b>	2	3
<b>Item C</b>	3	4
<b>Item D</b>	6	5

The dynamic programming algorithm is as follows:

- Let  $P$  = Profit of most valuable item; in this example it is 5 for Item D.
- Let  $nP$  = upper bound on the profit that can be achieved by any solution.
- For each  $i \in 1, \dots, n$  and  $p \in 1, \dots, nP$ , let  $S_{i,p}$  denote a subset of  $a_1, \dots, a_i$  whose total profit is exactly  $p$  and whose total size is minimized.
- Let  $A(1, p)$  denote the size of the set  $S_{1,p}$  ( $A(i, p) = \infty$  if no such set exists).
- $$A(i+1, p) = \begin{cases} \min\{A(i, p), \text{size}(a_i + 1) + A(i, p - \text{profit}(a_i + 1))\}, & \text{if } \text{profit}(a_i + 1) \geq p \\ A(i + 1, p) = A(i, p), & \text{otherwise} \end{cases}$$

Using this algorithm we will now produce the solution table for this KNAPSACK problem.

Table 3.1: Solution Table

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

As the burgler could carry items less or equal to 12, the solution table shows that the burgler can take the items weighing upto 11 which will be most profitable for him. From the solution table and the Weight/profit table it can easily be seen that those items are Item B, C and D.

(Note: this section is mainly from Approximation algorithm–V. Vazirani) It can easily be seen that if the profits of objects were small number, then a regular polynomial algorithm can be used to solve this problem. But what if the profits are huge for objects which to be considered. In that case the running time of the algorithm will be exponentially big and sometimes it may run for years depending on the input and the number of objects of the problem.

In this situation FPTAS can be used to find out the approximate solution for the problem.

- $A$  is a FPTAS (Fully Polynomial Time Approximation Scheme) is for each fixed  $\epsilon > 0$ , the running time of  $A$  is bounded by a polynomial in the size of  $I$  and  $1/\epsilon$ . [V01]

So, using this FPTAS we will 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 atleast within  $(1-\epsilon)$  of the optimum solution. The FPTAS algorithm for KNAPSACK is given below:

- Given  $\epsilon > 0$ , let  $K = \frac{\epsilon P}{n}$
- For each object  $a_i$ , define  $profit'(a_i) = \left\lfloor \frac{profit(a_i)}{K} \right\rfloor$
- Using these as profits of objects, apply dynamic programming algorithm to find the most profitable set,  $S'$ .
- Output  $S'$

Suppose 4 items are given each with a profit and weight associated with those. Say the bound for the KNAPSACK,  $B=12$  here. And we want to find out the solution within 0.5 of the optimum solution. So,  $\epsilon=0.5$  here.

**Weight and profit breakdown for each item**

	Weight	Profit
<b>Item A</b>	3	500
<b>Item B</b>	6	800
<b>Item C</b>	4	450
<b>Item D</b>	2	1000

Here,

$$P=1000 \quad K = \frac{0.5 \times 1000}{4} = 125$$

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

**Weight and Modified profit**

	Weight	Profit
<b>Item A</b>	3	4
<b>Item B</b>	6	6
<b>Item C</b>	4	3
<b>Item D</b>	2	8

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

Table 3.2: Solution Table for the newly obtained profits

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



From this solution table it can easily be seen that the maximum modified profit one can get is 18 keeping the KNAPSACK bound,  $B=12$  in mind. So, the KNAPSACK can carry Item A, B and D for the profit to get maximized.

### **3.5 Relation between s-FPTAS and structural 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. Going back to the original problem at the start of this Chapter, it can easily be conclude that KNAPSACK is a NP-hard problem that it easy to value-approximate but hard to structure-approximate.

# Chapter 4

## Other problems and generalizations

4.1 HamCycle?

4.2 Other than Karp's 21 problems

4.3 Edit Distance for something else

---

4.4 Other functions? General statement about functions?

## Chapter 5

### Conclusion and future work

# Bibliography

# Bibliography

- [ACG<sup>+</sup>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.
- [CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [HMvRW07] Matthew Hamilton, Moritz Müller, Iris van Rooij, and Todd Wareham. Approximating solution structure. In *Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs*. 2007.
- [Hoc97] Dorit S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.
- [Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.