Memorial
University of Newfoundland

Technical Report #2007-02
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, Canada

Approximating Solution Structure

(Revised: September 13, 2007)[1]

by

Matthew Hamilton[1], Moritz Muller[2], Iris van Rooij[3], and Todd Wareham[4]

[1]Department of Computer Science, University of Alberta,
Edmonton, AB, Canada

[2]Abteilung fur Mathematische Logik
Albert-Ludwigs-Universitt Freiburg, Freiburg, Germany

[3]Human-Technology, Interaction, Eindhoven University
of Technology, Eindhoven, The Netherlands

[4]Department of Computer Science, Memorial University
of Newfoundland, St. John's, NL, A1B 3X5, Canada

Department of Computer Science
Memorial University of Newfoundland
St. John's, NF, Canada A1B 3X5

2007

Technical Report #2007-02
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, Canada

Approximating Solution Structure

(Revised: September 13, 2007)[1]

Matthew Hamilton[1], Moritz Müller[2], Iris van Rooij[3], and Todd Wareham[4]

[1] Department of Computing Science, University of Alberta
Edmonton, AB, Canada
[2] Abteilung für Mathematische Logik
Albert-Ludwigs-Universitt Freiburg, Freiburg, Germany
[3] Human-Technology Interaction, Eindhoven University
of Technology, Eindhoven, The Netherlands
[4]Department of Computer Science, Memorial University
of Newfoundland, St. John's, NL, Canada

Abstract

Approximations can aim at having close to optimal value or, alternatively, they can aim at structurally resembling an optimal solution. Whereas value-approximation has been extensively studied by complexity theorists over the last three decades, structural-approximation has not yet been defined, let alone studied. However, structural-approximation is theoretically no less interesting, and has important applications in cognitive science. Building on analogies with existing value-approximation algorithms and classes, we develop a general framework for analyzing structural (in)approximability. We identify dissociations between solution value and solution structure, and generate a list of open problems that may stimulate future research.

[1]A preliminary version of this report was part of the discussion materials provided for an invited talk given by two of authors (IvR and TW) at IBFI Dagstuhl Seminar no. 07281: *Structure Theory and FPT Algorithmics for Graphs, Digraphs, and Hypergraphs*. Both the current version and our thoughts on structure-approximation in general have benefited greatly from spirited discussions with and suggestions made by various of the participants of that seminar, notably Erik Demaine, Daniel Marx, Luke Mathieson, Venkatesh Raman, and Saket Saurabh. We would therefore like to thank these participants as well as the organizers of this seminar and the IBFI Dagstuhl staff.

1 Introduction

When it is hard to compute a function exactly or optimally, one may be willing to settle for an approximation instead. But what does it mean to approximate an optimal solution? One possibility is that a candidate solution is considered an approximation to the extent that its value is close to the optimal value. We refer to this notion of approximation as **value-approximation**. This is the most common notion of approximation in computer science, and possibly the most relevant one for many applications (see [1, 20] and references). There is another possibility, however, 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 we call **structure-approximation** and is the topic of this paper.

1.1 Motivation

The need for structural-approximation algorithms naturally arises in the domain of cognitive science, where cognitive outcomes are often modeled as the outputs of intractable optimization problems. It seems implausible that human cognizers, given their limited computational resources, are capable of computing such intractable problems [7, 19]. Because the outputs of optimization problems nevertheless seem to predict and explain human cognitive outcomes rather well, it has been proposed that human cognizers may somehow *approximate* the optimal outcomes [4, 16, 17]. In this context it is structural-approximation—and *not* value-approximation—that gives the relevant notion of approximation, because an optimal output is taken be a good model of a cognitive output to the extent that it structurally resembles the cognitive output.

The relevance of structural approximation for cognitive modeling was also noted by Millgram [14] and it is his work that first inspired our research on structure-approximation. Millgram studied a cognitive model proposed in [18], according to which (rational) belief fixation can be modeled as the computation of the following function:

Foundational (F-) Coherence
Input: A set of propositions describing observations (or data) D and a set of propositions describing hypotheses H. A set of constraints $C = C^- \cup C^+ \subseteq H \times (D \cup H)$, with $C^- \cap C^+ = \emptyset$ and for each constraint $(p, q) \in C$ a weight $w(p, q) > 0$.
Output: A belief assignment $B : H \cup D \rightarrow \{true, false\}$, with $B(d) = true$ for all $d \in D$, such that the *coherence value* $COH(B) = \sum_{(p,q)\in C^+, B(p)=B(q)} w(p, q) + \sum_{(p,q)\in C^-, B(p)\neq B(q)} w(p, q)$ is maximized.

Because F-Coherence turns out to be NP-hard, [18] proposed several value-approximation algorithms for computing F-Coherence approximately. Millgram criticized the approach on the grounds that if one were to approximate the optimal solution to F-Coherence one would want one's beliefs to correspond approximately with the beliefs in the maximally coherent belief assignment. In other words, structure-approximation is the more natural and useful notion in this setting. Furthermore, so Millgram argued, value-approximations do not automatically yield structure-approximations, because it is possible for two belief assignments to be arbitrarily close in coherence value yet arbitrarily far from each other in terms of which beliefs are held "true" and which ones "false" (see pp. 89 in [14]).

Certain aspects of structure-approximability have been treated within complexity theory, *e.g.*, proving the existence or non-existence of efficient algorithms for (1) probabilistically checking whether or not a given graph either has a certain property P or is structurally distant from any graph

having property P [9] or (2) approximating the structural distance of a given sparse or bounded-degree graph from any graph having a certain property P [12]. However, to our knowledge, the precise type of structure-approximation envisioned by Millgram has not yet been considered by complexity theorists and no general framework for assessing degrees of structure-approximability exists so far. Because the concept of structure approximation seems to be a very natural one and is likely to have wider application in cognitive science and other disciplines we develop here a framework for quantifying and assessing structure-(in)approximability.

1.2 Aims of This Paper

Since value approximations do not generally yield structural approximations, it is currently unclear if the many results of value-(in)approximability of optimization problems that can be found in the computer science literature are of any use for purposes of approximate cognitive modeling.[2] It would benefit cognitive modeling if the relationship between value- and structure-approximation was better understood and, where the relationship is weak, if there were tools for studying structural-(in)approximability of optimization problems directly. With this paper we aim at stimulating research to this end by:

1. proposing a definition of structural-approximation, based on the notion of distance, that we believe applies to many optimization problems of practical interest;
2. defining structural-approximation classes and algorithms that are analogous to existing value-approximation classes and algorithms;
3. showing that output value and output structure can be so dissociated that it is likely that structural-approximation classes and their value-approximation analogues are distinct, while at the same time noting that we do not yet know if the value- and structural-approximation classes are all distinct (though we know that some *are*), nor whether the different structural-approximation classes are distinct from each other in the same way as the value-approximation classes are; and
4. posing many other interesting questions that naturally arise from the structural-approximation framework that we propose.

We cannot possibly hope to settle all relevant problems in a single paper, nor do we try to do so. Instead our aim is to trigger interest among complexity theorists in the concept of structural-approximation, and to invite them to contribute new relevant theoretical results on structural-(in)approximability, building in the end as rich a set of results for structural-approximation as currently exist for value-approximation.

1.3 Organization of This Paper

This paper is organized as follows. In Section 2, after a brief introduction to the existing value-approximation framework (Section 2.1), we describe a structural-approximation framework (Section 2.3), and show that value- and structural-approximability are distinct (Section 2.3). In Section 3, we summarize all known structural (in)approximability results and techniques for proving such results. Over the course of the paper, eleven open questions are proposed that we belive are promising directions for future research.

[2] This is not generally recognized and appreciated by cognitive scientists, as we see many citations in the cognitive science literature of value-approximation results when structural-approximability is actually at stake.

2 A Framework for Solution Structure Approximation

In this section, we will lay out the basic definitions for a theory of structural-approximability. Many of these definitions are variants of those underlying the theory of value-approximability of optimization problems; to highlight these relationships as well as the differences, we have where possible re-used much of the notation and language of this theory as given in Ausiello *et al.* [1].

2.1 Background: Optimization Problems and Solution Value Approximation Algorithms

In this section we review the basics of value-approximability, starting with the type of problem being approximated.

Definition 2.1 *(Adapted from [1, Definition 1.1.6] and [13, Section 2]) An optimization problem* Π *is characterized by the quadruple of objects* $(I, cansol, val, goal)$ *where:*

1. I *is the set of instances of* Π*;*
2. $cansol$ *is a function that associates to any input instance* $x \in I$ *the set of* **candidate solutions** *of* x*;*
3. val *is the* **value function***, defined for pairs* (x, y) *such that* $x \in I$ *and* $y \in cansol(x)$*. For each such pair* (x, y)*,* $val(x, y)$ *provides a positive integer which is the value of the candidate solution; and*
4. $goal \in \{MIN, MAX\}$ *specifies whether* Π *is a maximization or a minimization problem, respectively.*

Given an input instance x of an optimization problem, let $optval(x) = goal\{val(x, y) \mid y \in cansol(x)\}$ and $optsol(x) = \{y \mid y \in cansol(x) \text{ and } val(x, y) = optval(x)\}$. In this paper, we will focus on NP **optimization problems**, in which the length of each candidate solution y is polynomially bounded in the length of x, it can be decided if $x \in I$ and $y \in cansol(x)$ in polynomial time, and $val(x, y)$ can be computed in polynomial time. These are the problems in problem class NPO defined in [1, Section 1.4.2].

Example 2.2 *In this paper, we will use the following optimization problems as running examples:*

Maximum Clique (Max Clique) *[8, Problem GT19]*
Input*: Graph* $G = (V, E)$*.*
Candidate Solutions*: All subsets* $V' \subseteq V$ *that are cliques,* i.e*, for all* $v, v' \in V'$*,* $(v, v') \in E$*.*
Value to be Maximized*: The number of vertices in the solution vertex-subset,* i.e.*,* $|V'|$*.*

Weighted Max Cut *[8, Problem ND16]*
Input*: Edge-weighted graph* $G = (V, E, w)$*.*
Candidate Solutions*: All bipartitions of* V*.*
Value to be Maximized*: The sum of the weights of the edges in* E *with endpoints on different sides of the solution vertex-partition.*

Comment: *Note that* Weighted Max Cut *is a special case of* F-Coherence *(*i.e.*, the problems are equivalent when $D = \emptyset$ and $C = C^-$ in* F-Coherence*)*

Maximum k-Satisfiability (Max k-Sat, $k \geq 2$) *[8, Problem L05]*
Input*: A formula F from propositional logic consisting of a conjunction of $|C|$ disjunctive clauses over n Boolean variables, where each clause contains exactly k negated or unnegated variables.*
Candidate Solutions*: All truth assignments over n Boolean variables.*
Value to be Maximized*: The maximum number of clauses in F satisfied by the solution truth-assignment.*

Longest Common Subsequence (LCS) *[8, Problem SR10]*
Input*: An alphabet Σ and a set S of m strings over Σ such that the length of the longest string in S is n.*
Candidate Solutions*: All strings that are subsequences of all strings in S.*
Value to be Maximized*: The number of symbols in the solution-string.*

Unless otherwise specified, any reference to one of these problems will be to the problem and solution value function as defined above. ∎

We distinguish between exact and approximation algorithms for optimization problems:

Definition 2.3 *[1, Adapted from Definition 3.1] Given an optimization problem Π, an algorithm A is an* **exact algorithm** *for Π if, for any given instance $x \in I$, it returns a solution $A(x) \in optsol(x)$.*

Definition 2.4 *[1, Definition 3.1] Given an optimization problem $\Pi = (I, cansol, val, goal)$, an algorithm A is an* **approximation algorithm** *for Π if, for any given instance $x \in I$, it returns a solution $A(x) \in cansol(x)$.*

Many types of approximation algorithms have been defined such that $val(x, A(x))$ is within some absolute or relative factor of $optval(x)$ (see Chapter 3 of Ausiello *et al.* [1] for details); we refer to such algorithms as **value-approximation (v-approx) algorithms**. Several important classes of v-approx algorithms are defined below.

Definition 2.5 *Given an optimization problem Π and a non-decreasing function $h : \mathcal{N} \to \mathcal{N}$, 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 2.6 *Given an optimization problem $\Pi = (I, cansol, val, goal)$ and a non-decreasing function $h : \mathcal{N} \to \mathcal{N}$, an algorithm A is a* **polynomial-time $h(|x|)$ ratio value-approximation (v-r-approx) algorithm** *for Π 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 time polynomial in $|x|$.*

Definition 2.7 *Given an optimization problem Π, Π has a* **polynomial-time value-approximation scheme (v-PTAS)** *if there is an algorithm A that takes as inputs pairs $(x, k) \in \Sigma^* \times \mathcal{N}$, $k \geq 1$, such that for every fixed k, A is a $\frac{1}{k}$ v-r-approx algorithm that runs in time polynomial in $|x|$.*

Definition 2.8 *Given an optimization problem* Π, Π *has a* **fully polynomial-time value-approximation scheme (v-FPTAS)** *if there is an algorithm* A *that takes as inputs pairs* $(x, k) \in \Sigma^* \times \mathcal{N}$, $k \geq 1$, *such that* A *is a* $\frac{1}{k}$ *v-r-approx algorithm that runs in time polynomial in* $|x|$ *and* k.

2.2 Solution Structure Approximation Algorithms

Although value-approximation algorithms find natural application in many situations, some situations require a type of approximability that is based on the degree of similarity of the *structures* of candidate solutions to optimal solutions (see Section 1.1). We can capture this notion of solution similarity using the concept of a distance function. Let $d : \Sigma^* \times \Sigma^* \rightarrow \mathcal{N}$ be a **solution distance (sd) function** associated with an optimization problem Π such that for an instance x of Π and $y, y' \in cansol(x)$, $d(y, y')$ is the distance between these solutions. As it will occasionally be convenient to define the minimum distance of y to a set X of solutions, let $d(y, X) = \min_{y' \in X} d(y, y')$. Note that each sd-function assumes a particular representation for the candidate solutions of its associated optimization problem. We will assume that each sd-function d is a metric, and hence satisfies the following four properties:

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

Note that for a problem Π, there may be many such sd-functions. In certain situations, we may wish to use versions of sd-functions that have been normalized by their maximum possible values. We denote the normalized version of sd-function $d(y, y')$ by $d^N(y, y') = \frac{d(y,y')}{d_{max}(x)}$ where $d_{max}(x) = \max_{y,y' \in cansol(x)} d(y, y')$.

Example 2.9 *Consider the following candidate solution representations for the optimization problems defined in Example 2.2:*

- Max Clique*: A* $|V|$*-length binary vector such that a 1 (0) at position* i *means that vertex* v_i *is (is not) in the solution vertex-subset.*
- Weighted Max Cut*: A* $|V|$*-length binary vector such that a 1 (0) at position* i *means that vertex* v_i *is on side 1 (2) of the solution vertex-bipartition.*
- Max k-Sat*: An* n*-length binary vector such that a 1 (0) at position* i *means that Boolean variable* v_i *is set to True (False) in the solution truth-assignment.*
- LCS*: A string over* Σ *of length less than or equal to the length of the shortest string in* S.

For Max Clique, Weighted Max Cut, *and* Max k-Sat, d *will be the* **Hamming distance** (d_H) *between the given solution-vectors,* i.e., *the number of positions at which these vectors differ. For* LCS, d *will be the* **Edit distance** (d_E) *between the two given solution strings,* i.e., *the minimum number of single-symbol insertion, deletion, or substitution operations required to transform one string into the other. Both Hamming and Edit distance are metrics; moreover, Hamming distance can be computed in linear time and Edit distance can be computed in quadratic time [10, Section 11.3].* ∎

By analogy with the v-approx algorithms described in Section 2.1, we distinguish several classes of structural-approximation algorithms. As we do not compute ratios in structural-approximation, the first algorithm-type below corresponds to both v-a- and v-r-approx algorithms when (when d is unnormalized and $h(|x|)$ is arbitrary and when d is normalized and $h(|x|) = \epsilon$ for some ϵ, $0 < \epsilon < 1$, respectively).

Definition 2.10 *Given an optimization problem Π, a sd-function d, and a non-decreasing function $h : \mathcal{N} \to \mathcal{N}$, 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 $|x|$.*

Definition 2.11 *Given an optimization problem Π and a normalized sd-function d^N, Π has a* **polynomial-time d^N-structure-approximation scheme (d^N-s-PTAS)** *if there is an algorithm A that takes as inputs pairs $(x, k) \in \Sigma^* \times \mathcal{N}$, $k \geq 1$, such that for every fixed k, A is a $\frac{1}{k}/d^N$ s-approx algorithm that runs in time polynomial in $|x|$.*

Definition 2.12 *Given an optimization problem Π and a normalized sd-function d^N, Π has a* **fully polynomial-time d^N-structure-approximation scheme (d^N-s-FPTAS)** *if there is an algorithm A that takes as inputs pairs $(x, k) \in \Sigma^* \times \mathcal{N}$, $k \geq 1$, such that A is a $\frac{1}{k}/d^N$ s-approx algorithm that runs in time polynomial in $|x|$ and k.*

Two useful extensions of this framework are as follows:

1. By slightly reformulating basic definitions, we can also investigate the approximability of **search problems**, in which candidate solutions either are or are not solutions.

   This can be done either by reformulating structure-approximability to handle both and optimization problems or by reformulating search problems as optimization problems and reuse the framework as stated above. In the interests of simplicity, we do the latter as follows: Transform a given search problem Π with instance-set I, candidate-solution function $cansol(x)$, solution-function, and solution-function $sol(x)$ to an optimization problem $\Pi' = (I, cansol, val, goal)$ in which $val(x, y) = 1$ if $y \in sol(x)$ and 0 otherwise and $goal = MAX$. This reformulation has the interesting side effect that if the given instance x of Π has no solution, all candidate solutions are considered optimal in Π' and hence may be returned by an exact algorithm. However, as we are focusing on NP search problems (whose solutions can be verified in polynomial time), observe that Π and Π' reduce to each other, and the existence of a polynomial-time exact algorithm for Π' is hence equivalent to the existence of a polynomial-time exact algorithm for Π.

2. By choice of an appropriate candidate-solution set and associated sd-function, we can investigate the approximability of optimization and search problems relative to partial solutions. For example, an approximate solution for k-SAT (see below) might be a ternary vector that has the option of specifying a particular variable as true (1), false (0), or unspecified (*).

Example 2.13 *In this paper, we will use the following search problem as a running example:*

k-SATISFIABILITY (k-SAT) *[8, Problem L02]*
Input*: A formula F from propositional logic consisting of a conjunction of $|C|$ disjunctive clauses over n Boolean variables, where each clause contains exactly k negated or unnegated variables.*

Candidate Solutions*: All truth assignments over n Boolean variables.*
Output*: A truth-assignment over the n Boolean variables that satisfies all clauses in F, and the symbol $\perp$ if no such truth-assignment exists.*

The transformation described in extension (1) above creates the following optimization problem:

k-SATISFIABILITY-I ($k \geq 2$)
Input*: A formula F from propositional logic consisting of a conjunction of $|C|$ disjunctive clauses over n Boolean variables, where each clause contains exactly k negated or unnegated variables.*
Candidate Solutions*: All truth assignments over n Boolean variables.*
Value to be maximized*: The value assigned to the solution truth-assignment (1 if the truth-assignment satisfies F and 0 otherwise).*

We will assume the same solution representation and d as for MAX k-SAT*. Unless otherwise specified, any reference to* k-SAT *will be to the problem, value function, and sd-function as described above.* ∎

Despite all these possibilities and potentials, it can be argued that our proposed framework is ultimately unnecessary, as all types of structural approximability described in this section (including both extensions described above) can be simulated within the value-approximability framework by using appropriately-defined (albeit significantly more complex) solution-value functions and/or candidate-solution sets. We believe that for now, given the difficulties likely to arise in dealing with such solution-value functions in the value-approximability framework, it may be more productive to do the initial investigations into structure-approximability within a specifically-tailored framework such as that given above. Even if this work is eventually folded into and continued within the value-approximability framework, it should eventually lead to a broader interpretation of this framework to encompass and encourage exploration of the different ways in which optimal solutions for combinatorial optimization problems can be approximated, as well as the relationships between these different notions of approximation. A first step in this direction is given in the next section.

2.3 The Relationship Between Solution Value and Structure

For some classic optimization problems, structural approximations directly imply value approximations. Consider, for example, MAX CLIQUE, where all solutions that are close to optimal in structure are also at least that close in value, *i.e.*, $optval(x) - val(x, y) \leq d_H(y, optsol(x))$. This fact will be of use several times in Section 3. However, as illustrated in Example 2.14 the reverse is not necessarily the case: solutions which are close in value can yet be distant in structure.

Example 2.14 *Consider the* MAX CLIQUE *problem. For $n \geq 1$, let G_n be a graph on $2n + 1$ vertices consisting of two cliques C_1 and C_2 on n and $n + 1$ vertices, respectively (see part (a) of Figure 1 for G_3). Observe that the optimal solution s^* for G_n is the subset of vertices corresponding to C_2. Consider the solution s consisting of the vertices in C_1. Though s and s^* differ in value by 1, they are as far apart in structure as possible,* i.e.*, $d(s, s^*) = |V|$.* ∎

For other classic problems the dissociation between value- and structure-approximation is even more complete, with neither closeness in structure implying closeness in value, nor closeness in value implying closeness in structure.

Example 2.15 *Consider the* WEIGHTED MAX CUT *problem:*

- *For $n > 2$, let X_n be a graph with central vertex a that is connected by an edge of weight $c > 0$ to each of the peripheral vertices $b_1, b_2, \ldots, b_{(n-1)}$,* e.g., *$G = (V, E, w)$ such that $V = \{a, b_1, b_2, \ldots, b_{(n-1)}\}$, $E = \{(a, b_i) \mid 1 \leq i \leq n-1\}$, and $w(e) = c$ for all $e \in E$ (see part (b) of Figure 1 for X_5). Observe that s^* is such that a is on one side of the bipartition and all other vertices are on the other side; without loss of generality, let a be on side 1 and all other vertices be on side 2. Let s be the solution in which all vertices (including a) are on side 2. Note that though $d(s, s^*) = 1$, $val(x, s) = 0$ and $val(x, s^*) = (n-1)c$,* i.e., *though s is structurally close to s^*, it is far (indeed, as far as possible) from s^* in terms of value.*

- *For $n > 2$, let Y_n be as above with the addition of an edge between each pair of peripheral vertices; moreover, let c_+ be the weight of each central-peripheral edge and c_- be the weight of every peripheral-peripheral edge (see part (c) of Figure 1 for Y_5). Without loss of generality, assume that a is on side 1 of the bipartition. If we let n', $0 \leq n' \leq n-1$, be the number of peripheral vertices that are on side 2 of the bipartition in some solution s, then $val(x, s) = n'c_+ + n'((n-1) - n'))c_-$. If $c_+ < c_-$, $val(x, s)$ is maximized (and hence s is optimal) when $n' = \frac{(n-1)}{2}$. To see this, note that $val(x, s)$ increases as n' increases from 0. When $n' = \frac{(n-1)}{2}$,*

$$\begin{aligned} val(x,s) &= \frac{(n-1)}{2}c_+ + \frac{(n-1)}{2}((n-1) - \frac{(n-1)}{2})c_- \\ &= \frac{(n-1)}{2}c_+ + \frac{(n-1)}{2}(\frac{(n-1)}{2})c_- \\ &= \frac{(n-1)}{2}c_+ + (\frac{(n-1)}{2})^2 c_- \end{aligned}$$

However, when $n' = \frac{(n-1)}{2} + 1$,

$$\begin{aligned} val(x,s) &= (\frac{(n-1)}{2} + 1)c_+ + (\frac{(n-1)}{2} + 1)((n-1) - (\frac{(n-1)}{2} + 1))c_- \\ &= (\frac{(n-1)}{2} + 1)c_+ + (\frac{(n-1)}{2} + 1)(\frac{(n-1)}{2} - 1)c_- \\ &= (\frac{(n-1)}{2} + 1)c_+ + ((\frac{(n-1)}{2})^2 - 1)c_- \end{aligned}$$

which is less when $c_+ < c_-$ and will continue to drop in value as n' increases to $n-1$.

Given the above, let s be the solution in which all vertices are on side 1 and s^* be any optimal solution in which a and exactly half of the b_i are on side 1 and the remaining half of the b_i are one side 2. As $val(x, s) = (n-1)c_+$, $val(x, s^*) = \frac{(n-1)}{2}c_+ + \frac{(n-1)(n-1)}{4}c_-$, and

$$\begin{aligned} (n-1)c_+ &= \frac{(n-1)}{2}c_+ + \frac{(n-1)(n-1)}{4}c_- \\ \frac{(n-1)}{2}c_+ &= \frac{(n-1)(n-1)}{4}c_- \\ c_+ &= \frac{(n-1)}{2}c_-, \end{aligned}$$

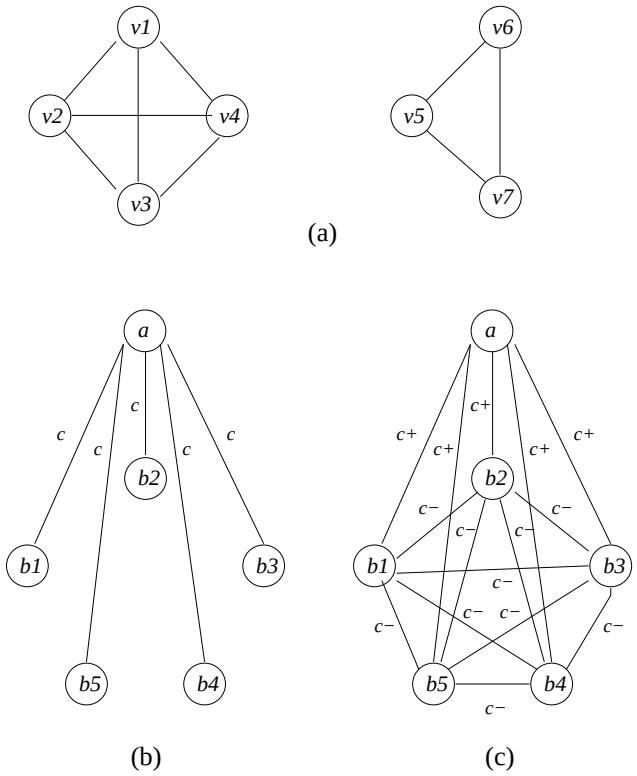

Figure 1: Near-Optimal Solution Value Does Not Imply Near-Optimal Solution Structure. (a) Graph G_3. (b) Graph X_5. (c) Graph Y_5. See main text for details.

we know that by setting $c_+ = \frac{(n-1)}{2}c_- - \epsilon$ *for an appropriate* $\epsilon > 0$, $val(x, s)$ *can be made to be arbitrarily close to* $val(x, s^*)$*; however,* $d_H(s, s^*) = n/2$, i.e., *though* s' *is close to* s^* *in terms of value, it is structurally far away.* ∎

Moreover, v-approximability is not necessarily correlated with s-approximability.

Example 2.16 *Consider the version of the* Max Clique *problem in which the solution-representation and value function are standard but the sd-function* d *is such that* $d(y, y') = 0$ *if* $y = y'$ *and 1 otherwise. Observe that this problem is not v-r-approximable within a factor of* $|V|^{\frac{1}{2}-\epsilon}$ *for any epsilon* > 0 *unless* $P = NP$ *[11, Theorem 5.3], but it is 1-s-approximable (by the trivial algorithm that returns an arbitrary solution* $y \in cansol(x)$*).* ∎

Example 2.17 *Consider the version of the* Max Clique *problem in which the solution-representation and sd-function are standard but the the value function* $val(x, y)$ *is 5 if* y *is a maximum clique, 4 if* y *is the empty set, and 0 otherwise. Observe that this problem is 1-v-a-approximable (by the trivial algorithm that returns the empty-set solution) but is not* $|x|^{\frac{1}{c}}$ *s-approximable for* $c \geq 10$ *unless* $P = NP$ *(see Corollary 3.6 below).* ∎

These examples are admittedly artificial but they do highlight the non-equivalence of value- and structure-approximability in general. It is tempting to attribute this non-equivalence to the wildly-varying complexity of computing $d(y, optsol(x))$ for different d, which is linear time for d as defined

in Example 2.16 but NP-hard for Hamming distance relative to MAX CLIQUE (this follows from the fact that for the solution y that is the empty set, $|optval(x) - val(x,y)| = d_H(y, optsol(x))$). However, such arguments are invalidated by the fact that value-functions themselves can show similar complexity variations, from polynomial time in Examples 2.15 and 2.16 to NP-hard in Example 2.17. The roots of this non-equivalence are left for now as an interesting research direction within the approximation research program sketched at the end of Section 2.2. In any case, as we will see in the remainder of this paper, there are nonetheless some relationships between solution value and structure that can be exploited in certain cases.

3 Known Techniques and Results

In this section, we list all techniques known at this time for proving both polynomial-time and fixed-parameter s-approximability and s-inapproximability, as well as sample results derived using these techniques. The first set of techniques (Section 3.1.1) exploits relationships between solution value and distance, and is hence only applicable to certain optimization problems; however, the remaining techniques, discussed Sections 3.1.2–5, are applicable to all optimization and search problems.

Many of the s-inapproximability techniques described in this section operate by showing that if a particular type of s-approx algorithm exists for a given optimization problem Π then the standard decision problem associated with Π can be solved in polynomial time. This decision problem is defined as follows.

Definition 3.1 *Given an optimization problem Π, the* **standard decision problem Π_D associated with Π** *is the decision problem that, given an instance x of Π, asks if asks if $optval(x) \leq k$ (if $goal = MIN$) or $optval(x) \geq k$ (if $goal = MAX$).*

3.1 Polynomial-Time Structure (In)approximability

3.1.1 Structure Approximability by Dumb Luck

In rare cases, the nature of the candidate-solution set of a problem allows the derivation of very loose s-approximability results.

Theorem 3.2 WEIGHTED MAX CUT *is $0.5|x|/d_H$ s-approximable.*

Proof: Given a solution s of WEIGHTED MAX CUT, let $\overline{s}$ be the solution corresponding to the bitwise complement of s. Note that if s is optimal then so is $\overline{s}$. The required algorithm is then trivial – simply generate and return an arbitrary candidate solution s to the given input x. We prove that $d_H(s, s^*) \leq 0.5|V| \leq 0.5|x|$ for some optimal solution s^* for x by contradiction: Assume there is a candidate solution s for x such that $d_H(s, s^*) > 0.5|V|$ for *every* optimal solution s^* for x. Let s_c^* be the closest such optimal solution to s. As $d_H(s, s_c^*) > 0.5|V|$, this means that at least half of the bits of s and s_c^* are different. This means that at least $0.5|V|$ bits of s and $\overline{s_c^*}$ are the same, such that $d_H(s, \overline{s_c^*}) \leq 0.5|V|$. However, as s_c^* is optimal, so is $\overline{s_c^*}$, which is a contradiction. ∎

Corollary 3.3 WEIGHTED MAX CUT *is* $0.5/d_H^N$ *s-approximable.*

Proof: Observe that the proof of Theorem 3.2 actually shows that WEIGHTED MAX CUT is $0.5|V|/d_H$ s-approximable; the result then follows from the fact that in the case of d_H and WEIGHTED MAX CUT, $d_{\max}(x) \leq |V|$. ∎

Such dumb luck does not hold in general (nor does it typically give good approximation bounds). Fortunately, as we will see in subsequent sections, we have other options.

3.1.2 Structure (In)approximability by Value (In)approximability

As noted in Section 2.3, we cannot have in general that solutions with close to optimal value also have close to optimal structure, or vice versa. However, even loose or partial relations may still be exploited in the design of approximation algorithms, as shown by the following lemmas. For the purposes of this paper, we say that a function $f : \Sigma^* \times \mathcal{N} \to \mathcal{N}$ is non-decreasing if for all $x \in \Sigma^*$ and $y, z \in \mathcal{N}$, $f(x, y) \leq f(x, z)$ if and only if $y \leq z$.

Lemma 3.4 *Given an optimization problem* Π*, a sd-function* d*, and a non-decreasing function* $f : \Sigma^* \times \mathcal{N} \to \mathcal{N}$*, if for every instance* x *of* Π *it is the case that for every candidate solution* y *of* x*,* $|optval(x) - val(x, y)| \leq f(x, d(y, optsol(x)))$*, then any* $h(|x|)/d$*-s-approx algorithm for* Π *is also a* $f(x, h(|x|))$ *v-a-approx and* $(f(x, h(|x|)) + 1)$ *v-r-approx algorithm for* Π*.*

Proof: Suppose Π is a maximization problem. By definition, the existence of a $h(|x|)/d$ s-approx algorithm A for Π implies that $d(A(x), optsol(x)) \leq h(|x|)$, which in turn implies that $f(x, d(A(x), optsol(x))) \leq f(x, h(|x|))$. However, as we also know that $|optval(x) - val(x, y)| \leq f(x, d(y, optsol(x)))$, it is the case for A that $|optval(x) - val(x, A(x))| \leq f(x, d(A(x), optsol(x)))$, which implies that A is also a $f(x, h(|x|))$ v-a-approx algorithm for Π. In a maximization problem, as $optval(x) \geq val(A(x))$, $|optval(x) - val(x, A(x))| = optval(x) - val(x, A(x))$. Hence,

$$\begin{aligned}
optval(x) - val(x, A(x)) &\leq f(x, h(|x|)) \\
\frac{optval(x) - val(x, A(x))}{val(x, A(x))} &\leq \frac{f(x, h(|x|))}{val(x, A(x))} \\
\frac{optval(x)}{val(x, A(x))} - 1 &\leq \frac{f(x, h(|x|))}{val(x, A(x))} \\
\frac{optval(x)}{val(x, A(x))} &\leq \frac{f(x, h(|x|))}{val(x, A(x))} + 1 \\
\frac{optval(x)}{val(x, A(x))} &\leq f(x, h(|x|)) + 1
\end{aligned}$$

which implies that A is also a $f(x, h(|x|)) + 1$ v-r-approx algorithm for Π. Similar proofs also hold with respect to minimization problems. ∎

The use of Lemma 3.4 and v-r-inapproximability results to prove s-inapproximability requires v-r-inapproximability for ratios that are polynomial of $|x|$.

Corollary 3.5 MAX CLIQUE *is not* $|x|^{\frac{1}{c}}/d_H$ *s-approximable for any* $c \geq 6$ *unless* $P = NP$.

Proof: We will show that the existence of a polynomial-time $|x|^{\frac{1}{c}}/d_H$ s-approx algorithm for some $c \geq 6$ implies the existence of a polynomial-time $|V|^{(\frac{1}{2}-\epsilon)}$ v-r-approx algorithm for MAX CLIQUE for some $\epsilon > 0$. As $optval(x) - val(x, A(x)) \leq f(x, d_H(A(x), optsol(x))) = d_H(A(x), optsol(x))$ for MAX CLIQUE, by Lemma 3.4, A is also a $f(x, |x|^{\frac{1}{c}})+1 = |x|^{\frac{1}{c}}+1$ v-r-approx algorithm for MAX CLIQUE. Recall that for MAX CLIQUE, in which the input graph can be represented as $|V| \times |V|$ adjacency bit-matrix, $|x| = |V|^2$; hence A is a $(|V|^2)^{\frac{1}{c}} + 1 = |V|^{\frac{2}{c}} + 1$ s-approx algorithm for MAX CLIQUE. Choose $c' = \frac{2.6}{c}$; as $c \geq 6$, $c' < \frac{1}{2}$. Observe that $|V|^{\frac{2}{c}} + 1 \leq |V|^{c'}$ when $|V| \geq c^{\frac{c}{2}}$ and $c \geq 6$; this is so because $(c^{\frac{c}{2}})^{\frac{2}{c}} + 1 \leq (c^{\frac{c}{2}})^{\frac{2.6}{c}}$ implies that $c + 1 \leq c^{1.3}$, which is true when $c \geq 6$.

Given the above, consider the algorithm A' for MAX CLIQUE which looks up the optimal solution when $|V|$ in the given instance is less than or equal to $c^{\frac{c}{2}}$ and calls A on the given instance otherwise. By the above, A' is a polynomial-time $|V|^{(\frac{1}{2}-\epsilon)}$ v-r-approx algorithm for MAX CLIQUE for some $\epsilon > 0$; however, the existence of such an algorithm implies that $P = NP$ [11, Theorem 5.3], completing the proof. ∎

Corollary 3.6 *The version of* MAX CLIQUE *in Example 2.17 is not* $|x|^{\frac{1}{c}}/d_H$ *s-approximable for any* $c \leq 10$ *unless* $P = NP$.

Proof: This proof is similar to that for Corollary 3.5 above, but must be modified because of the non-standard value function, for which $optval(x) - val(x, A(x)) \leq f(x, d_H(A(x), optsol(x))) = 5d_H(A(x), optsol(x))$. Given this, Lemma 3.7 now implies that A is also a $f(x, |x|^{\frac{1}{c}})+1 = 5|x|^{\frac{1}{c}}+1 = 5|V|^{\frac{2}{c}}$ v-r-approx algorithm. Choose $c' = \frac{4.8}{c}$; as $c \geq 10$, $c' < \frac{1}{2}$. Observe that $5|V|^{\frac{1}{c}} + 1 \leq |V|^{c'}$ when $|V| \geq c^{\frac{c}{2}}$ and $c \geq 10$; this is because $5(c^{\frac{c}{2}})^{\frac{2}{c}} + 1 \leq (c^{\frac{c}{2}})^{\frac{4.8}{c}}$ implies that $5c + 1 \leq c^{2.4}$, which is true when $c \geq 10$. The remainder of the proof is now as in Corollary 3.5. ∎

A converse of Lemma 3.4 that may be useful for proving s-approximability also exists.

Lemma 3.7 *Given an optimization problem* Π*, a sd-function* d*, and a non-decreasing function* $f : \Sigma^* \times \mathcal{N} \to \mathcal{N}$*, if for every instance* x *of* Π *it is the case that for every candidate solution* y *of* x*,* $d(y, optsol(x)) \leq f(x, |optval(x) - val(x, y)|)$*, then*

1. *any* $h(|x|)$ *v-a-approx algorithm for* Π *is also a* $f(x, h(|x|))/d$ *s-approx algorithm for* Π*; and*
2. *any* $h(|x|)$ *v-r-approx algorithm for* Π *is also a* $f(x, optval(x)(h(|x|)-1))/d$ *s-approx algorithm for* Π.

Proof: Suppose Π is a maximization problem. By definition, the existence of a $h(|x|)$ v-a-approx algorithm A for Π implies that $|optval(x) - val(x, A(x))| \leq h(|x|)$, which in turn implies that $f(x, |optval(x) - val(x, A(x))|) \leq f(x, h(|x|))$. However, as we also know that $d(y, optsol(x)) \leq f(x, |optval(x) - val(x, y)|)$, it is the case for A that $d(A(x), optsol(x)) \leq f(x, |optval(x) - val(x, A(x))|)$, which implies that A is also a $f(x, h(|x|))/d$ s-approx algorithm for Π. As for (2), note that by definition, the existence of a $h(|x|)$ v-r-approx algorithm A for Π implies that $\frac{optval(x)}{val(x, A(x))} \leq h(|x|)$. Hence,

$$
\begin{aligned}
\frac{optval(x)}{val(x,A(x))} &\leq h(|x|) \\
\frac{optval(x)}{val(x,A(x))} - 1 &\leq h(|x|) - 1 \\
\frac{optval(x) - val(x,A(x))}{val(x,A(x))} &\leq h(|x|) - 1 \\
optval(x) - val(x,A(x)) &\leq val(x,A(x))(h(|x|) - 1) \\
f(x, |optval(x) - val(x,A(x))|) &\leq f(x, val(x,A(x))(h(|x|) - 1)) \\
d(y, optsol(x)) &\leq f(x, val(x,A(x))(h(|x|) - 1)) \\
d(y, optsol(x)) &\leq f(x, optval(x)(h(|x|) - 1))
\end{aligned}
$$

which implies that A is also a $f(x, optval(x)(h(|x|)-1))/d$ s-approx algorithm for Π. Similar proofs also hold with respect to minimization problems, modulo changes introduced by switching the ratio from $\frac{optval(x)}{val(x,A(x))}$ to $\frac{val(x,A(x))}{optval(x)}$ in the equations above. ∎

Useful applications of Lemma 3.7 require v-r-approximability for ratios < 2 (and preferably $<< 2$). However, the restrictions placed on the form of $f()$ by the requirement that $d(y, optsol(x)) \leq f(x, |optval(x) - val(x,y)|)$ significantly weaken achievable results. Better results would be possible if the relationship between given and produced v-r- and s-approx ratios in the lemmas above was tighter and the relationship between solution-to-optimal distance and value was less stringent.

Open Problem #1: Prove versions of Lemmas 3.4 and 3.7 with tighter relationships between v-r-approximability and s-approximability and looser solution-to-optimal distance-value requirements.

3.1.3 Structure Inapproximability by Self-Paddability

Consider the following general way of proving s-inapproximability which is based on the "instance-copy" strategy used to show certain types of polynomial-time v-a-inapproximability, *e.g.*, [8, Theorem 6.7]. We first define a property of an optimization problem which allows the instance-copy strategy to be applied to that problem.

Definition 3.8 *Given an optimization problem Π, a sd-function d, and a non-decreasing function $h : \mathcal{N} \rightarrow \mathcal{N}$ that is computable in polynomial time, Π is* **(polynomial-time) $h(|x|)$-self-paddable with respect to** *d if the following holds:*

1. *There exists a function $f(x, h(|x|))$ such that for any instance x of Π, f creates an instance x' of Π of size $padsize(x, h(|x|))$.*
2. *There exists a function $g(f(x, h(|x|)), y)$ such that for any instance x of Π, g extracts from a solution $y \in cansol(f(x, h(|x|)))$ the set $\{y_1, y_2, \ldots, y_{h(|x|)}\}$ such that for $1 \leq i \leq h(|x|)$, $y_i \in cansol(x)$;*
3. *f and g run in time polynomial in $|x|$ and $h(|x|)$; and*

4. $d(y, optsol(f(x, h(|x|)))) = \sum_{i=1}^{h(|x|)} d(y_i, optsol(x))$.

Self-paddability is an extension of paddability as defined in [5, Definition 4.2]. Note that the instance x' created by function f above effectively encodes $h(|x|)$ instances of Π.

Theorem 3.9 *Given an NP optimization problem* Π *that is* $h(|x|)$*-self-paddable for a function* h *that is polynomially bounded, a sd-function* d*, and function* $g : \mathcal{N} \to \mathcal{N}$ *such that* $g(padsize(x, h(|x|)))$ $< h(|x|)$*, if* Π *is* $g(|x|)/d$*-s-approximable and* Π_D *is NP-hard then* $P = NP$*.*

Proof: Given an instance x of Π_D, we can run the following algorithm to solve x: Let x' be the instance of Π corresponding to x; observe that $|x'| = |x|$. Run the $g(|x|)/d$ s-approx algorithm on the instance $f(x', h(|x'|))$ of Π to create solution y. Decompose y into $\{y_1, y_2 \ldots, y_{(h(|x|))}\}$ using g, and determine, for $1 \leq i \leq h(|x|)$, which $y_i \in optsol(x)$, *i.e.*, compute all $val(x, y_i)$. As $d(y, optsol(f(x', h(|x'|)))) \leq g(|f(x', h(|x'))|) = g(padsize(x, h(|x|))) < h(|x|)$ and $d(y, optsol(f(x', h(|x'|)))) = \sum_{i=1}^{h(|x'|)} d(y_i, optsol(x'))$, at least one y_i has $d(y_i, optsol(x')) = 0$, meaning that $y_i \in optsol(x')$. We can then use this y_i to solve Π_D. As all steps of this algorithm run in time polynomial in $|x'| = |x|$, this is a polynomial-time algorithm for Π_D; however, as Π_D is NP-hard, this implies that $P = NP$, completing the proof. ∎

This theorem is useful in proving s-inapproximability both in the case of "true" optimization problems (which have existing v-r-(in)approximability results) and optimization problems corresponding to search problems (see Lemma 3.11 and 3.12 respectively).

Lemma 3.10 *Given an NP optimization problem* Π *and a sd-function* d*, if* Π_D *is NP-hard and* Π *is* $|x|^\alpha$*-self-paddable for all* $\alpha \in \mathcal{N}$*,* $\alpha \geq 1$ *such that* $padsize(x, |x|^\alpha) = |x|^{\alpha+1}$*, then* Π *is not* $|x|^{(1-\epsilon)}/d$ *s-approximable for any* $\epsilon > 0$ *unless* $P = NP$*.*

Proof: Suppose there is an algorithm A that is a $|x|^{(1-\epsilon)}/d$ s-approx algorithm for Π for some $\epsilon > 0$. As Π is $|x|^\alpha$ self-paddable for any integer $\alpha > 1$ such that $padsize(|x|, |x|^\alpha) = |x|^{\alpha+1}$, then

$$\begin{aligned}
(|x|^{(\alpha+1)})^{(1-\epsilon)} &< |x|^\alpha \\
(\alpha+1)(1-\epsilon)\log_2|x| &< \alpha \log_2 |x| \\
(\alpha+1)(1-\epsilon) &< \alpha \\
\alpha + 1 - \epsilon(\alpha+1) &< \alpha \\
-\epsilon(\alpha+1) &< -1 \\
\epsilon(\alpha+1) &\geq 1
\end{aligned}$$

holds for any $\epsilon > 0$ when $\alpha = \frac{1}{\epsilon}$, the result follows by the NP-completeness of Π_D, and Theorem 3.9. ∎

Lemma 3.11 *For all* $\alpha \in \mathcal{N}$*,* $\alpha \geq 1$*,* Weighted Max Cut *is* $|x|^\alpha$*-self-paddable with respect to* d_H *such that* $padsize(|x|, |x|^\alpha) = |x|^{\alpha+1}$*.*

Proof: Let $f(x, |x|^\alpha)$ create an instance x' of Weighted Max Cut consisting of $|x|^\alpha$ copies of x, and $g(x', y)$ return the $|x|^\alpha$ candidate solutions $\{y_1, y_2, \ldots, y_{|x|^\alpha}\}$ for the copies of x encoded in x'. Both f and g obviously run in time polynomial in $|x|$; moreover, it is also obvious that $d_H(y, optsol(x')))) = \sum_{i=1}^{|x|^\alpha} d_H(y_i, optsol(x))$. ∎

Lemma 3.12 *For all $\alpha \in \mathcal{N}$, $\alpha \geq 1$,* 3-SATISFIABILITY *is $|x|^\alpha$-self-paddable with respect to d_H such that $padsize(|x|, |x|^\alpha) = |x|^{\alpha+1}$.*

Proof: Let $f(x, |x|^\alpha)$ create an instance x' of 3-SATISFIABILITY consisting of a 3CNF formula that is the conjunction of $|x|^\alpha$ copies of the 3CNF formula in x, such that the variables in each copy have been relabeled to be distinct and $g(x', y)$ return the $|x|^\alpha$ candidate solutions $\{y_1, y_2, \ldots, y_{|x|^\alpha}\}$ for the copies of x encoded in x'. Both f and g obviously run in time polynomial in $|x|$; moreover, it is also obvious that $d_H(y, optsol(x'))) = \sum_{i=1}^{|x|^\alpha} d_H(y_i, optsol(x))$. ∎

Corollary 3.13 WEIGHTED MAX CUT *and* 3-SATISFIABILITY *are not $|x|^{(1-\epsilon)}/d_H$ s-approximable for any $\epsilon > 0$ unless $P = NP$.*

Proof: Follows from the NP-hardness of WEIGHTED MAX CUT$_D$ and 3-SATISFIABILITY$_D$ and Lemmas 3.10, 3.11, and 3.12. ∎

Results derived using self-paddability as defined above, while good, are not totally satisfactory as they cannot address issues of $\epsilon|x|$ s-inapproximability (this is so because $\epsilon(|x|^{c\alpha}) \leq |x|^\alpha$ implies that $\epsilon(|x|^c) \leq 1$ which does not hold for any $c, \epsilon > 0$).

> **Open Problem #2**: Can instance-copy arguments be modified to prove $\epsilon|x|$ s-inapproximability, $0 < \epsilon < 1$?

3.1.4 s-FPTAS Inapproximability

We can also extend other techniques for showing v-inapproximability to show corresponding kinds of s-inapproximability, such as the use of Strong NP-completeness to show FPTAS-inapproximability [8, Theorem 6.8].

Lemma 3.14 *Given an optimization problem Π and a normalized sd-function d^N such that the sd-normalization factor $d_{\max} \leq q(|x|)$ for some polynomially-computable function q, if Π has a d^N-s-FPTAS and Π_D is NP-hard then $P = NP$.*

Proof: Given an instance x of Π_D, solve this instance by the following algorithm: Create the instance x' of Π corresponding to x and run the d^N-s-FPTAS on x' with $k = q(|x|) + 1$ to create solution y. Observe that $d^N(y, optsol(x')) \leq \frac{1}{k}$ implies that $\frac{d(y, opytsol(x'))}{d_{\max}} \leq \frac{1}{k}$, which in turn implies that $d(y, optsol(x')) \leq \frac{d_{\max}}{k} \leq \frac{q(|x|)}{q(|x|)+1} < 1$. As errors are discrete, y is optimal and can be used to solve x' (and hence x) in polynomial time; however, as Π_D is NP-hard, this implies that $P = NP$, completing the proof. ∎

Corollary 3.15 *Neither* MAX CLIQUE, WEIGHTED MAX CUT, MAX k-SAT *($k \geq 2$),* LCS, *or* k-SAT *($k \geq 3$) have d^N-s-FPTAS (relative to $d = d_H^N$ or d_E^N as appropriate) unless $P = NP$.*

Proof: These results follow from the NP-hardness of the standard decision problems associated with these problems and Lemma 3.14. ∎

This rules out the existence of s-FPTAS for NP-hard problems, *cf.* the existence of FPTAS for several NP-hard optimization problems. It would be interesting to know if there is an optimization problem that is neither polynomial-time solvable nor NP-hard and has a s-FPTAS.

> **Open Problem #3**: Is there a natural non-polynomial-time-solvable optimization problem (with associated normalized sd-function d^N) that has a d^N-s-FPTAS?

3.1.5 Structure (In)approximability by Reduction

Another way to show s-approximability and -inapproximability results would be to define an s-approximability preserving reducibility and appropriate classes of problem/sd-function pairs, and then define hardness and completeness results for those classes relative to that reducibility. Consider the following classes, which, for the purposes of simplicity, are restricted to use Hamming distance as the sd-function. Let PO and NPO be the classes of polynomial-time solvable and verifiable optimization problems, respectively [1, Section 1.4.2].

Definition 3.16 *$sAPX$ is the class of all optimization problem-sd pairs (Π, d_H^N) such that Π is both in NPO and ϵ/d_H^N s-approximable for some ϵ, $0 < \epsilon < 1$.*

Definition 3.17 *$sPTAS$ is the class of all optimization problem-sd pairs (Π, d_H^N) such that Π is both in NPO and has a d_H^N-s-PTAS.*

Definition 3.18 *$sFPTAS$ is the class of all optimization problem-sd pairs (Π, d_H^N) such that Π is both in NPO and has a d_H^N-s-FPTAS.*

These classes are the s-approximability analogues of the v-r-approximability classes APX, $FPTAS$, and $PTAS$ defined in [1, Chapter 3]. By definition, we have the following class inclusions:

$$PO \subseteq sFPTAS \subseteq sPTAS \subseteq sAPX \subseteq NPO$$

At present, only one of these inclusions is known to be proper.

Corollary 3.19 *If $P \neq NP$ then $sFPTAS \subset sAPX$.*

Proof: Follows from Corollaries 3.15 and 3.3. ∎

Open Problem #4: Which of the following are true (assuming $P \neq NP$):

- $PO \subset sFPTAS$?
- $sFPTAS \subset sPTAS$?
- $sPTAS \subset sAPX$?
- $sAPX \subset NPO$?

Open Problem #5: How are $sAPX$, $sFPTAS$, and $sPTAS$ as defined above related to the versions of those classes that allow arbitrary metric sd-functions?

Open Problem #6: How are the s- and v-r-approximability classes related?

To show hardness and completeness for these classes, we need an appropriate s-approximability preserving reducibility. By analogy with the L-reducibility between optimization problems defined by Papadimitriou and Yannakakis [15] for the purposes of investigating value-approximation, consider the following.

Definition 3.20 *Given a pair of NPO problem-normalized sd function pairs $A = (\Pi, d_H^N)$ and $B = (\Pi', d_{H'}^N)$ with sd-normalization factors $d_{\max}(x)$ and $d'_{\max}(x')$, A* **structurally L-reduces to** *B ($A \leq_L^s B$) if there is a quadruple (f, g, α, β) such that the following hold:*

1. *Function f transforms an instance x of Π into an instance x' of Π' and is computable in time polynomial in $|x|$;*
2. *Function g transforms a candidate solution for an instance of Π' into a candidate solution for an instance of Π,* i.e.*, for all $x \in I_\Pi$ and $y' \in consol_{\Pi'}(f(x))$, $g(f(x), y') \in cansol_\Pi(x)$, and is computable in time polynomial in $|f(x)|$ and $|y'|$;*
3. *There is a constant $\alpha > 0$ such that for all $x \in I_\Pi$ and $y' \in cansol_{\Pi'}(f(x))$, $\alpha d_H^N(g(f(x), y'), optsol(x)) \leq d_{H'}^N(y', optsol(f(x)))$;*
4. *There is a constant $\beta > 0$ such that for all $x \in I_\Pi$, $d'_{\max}(f(x)) \leq \beta d_{\max}(x)$; and*
5. $0 < \frac{\beta}{\alpha} \leq 1$

Lemma 3.21 *Given a pair of NPO problem-normalized sd function pairs $A = (\Pi, d_H^N)$ and $B = (\Pi', d_{H'}^N)$, if $A \leq_L^s B$ relative to (α, β) and Π' is $\epsilon/d_{H'}^N$-approximable, then Π is $\frac{\beta\epsilon}{\alpha}/d_H^N$-approximable.*

Proof: Consider the algorithm for A that for any instance $x \in I_\Pi$, applies f to x, runs the $\epsilon/d_{H'}^N$ s-approx algorithm for Π' to $f(x)$ to produce y', and applies g to y' to produce an approximate solution for Π. By the definitions of structural L-reducibility and ϵ/d^N s-approximability, we have the following:

$$\begin{aligned}
d_{H'}^N(y', optsol(x')) &\leq \epsilon \\
\frac{d_{H'}(y', optsol(x'))}{d'_{\max}(x')} &\leq \epsilon \\
\frac{d_{H'}(y', optsol(x'))}{\beta d_{\max}(x)} &\leq \epsilon \\
\frac{\alpha d_H(g(f(x), y'), optsol(x))}{\beta d_{\max}(x)} &\leq \epsilon \\
\frac{\alpha}{\beta} d_H^N(g(f(x), y'), optsol(x)) &\leq \epsilon \\
d_H^N(g(f(x), y'), optsol(x)) &\leq \frac{\beta\epsilon}{\alpha}
\end{aligned}$$

The result then follows from the definition ϵ/d^N s-approximability. ∎

Lemma 3.22 *Reducibility $\leq_L^s$ is transitive.*

Corollary 3.23 *Given a pair of NPO problem-normalized sd function pairs A and B, if $A \leq_L^s B$ and $B \in \mathbf{X}$, $\mathbf{X} \in \{sFPTAS, sPTAS, sAPX\}$, then $A \in \mathbf{X}$.*

Open Problem #7: Find complete problems for the s-approximability classes under $\leq_L^s$.

3.1.6 Discussion

Many of the techniques for showing s-inapproximability given in this section have been based on techniques for showing v-a- or v-r-inapproximability. There are many other such v-inapproximability techniques that may yet be useful (see [1, 20] and references).

> **Open Problem #8**: Can any other v-inapproximability techniques be modified to show s-inapproximability? If so, which ones?

The glaring lack of techniques for showing non-trivial forms of s-approximability also begs the question of whether v-approximability techniques can be modified (see [1, 20] and references) or special-purpose s-approximability techniques can be developed.

> **Open Problem #9**: Can any v-approximability algorithm design techniques be modified to design efficient and effective s-approximation algorithms? If so, which ones?

> **Open Problem #10**: Develop a s-approximability algorithm design "toolkit".

3.2 Parameterized Structure-Approximation

It is possible that problems that are not polynomial-time s-approximable may yet be s-approximable in fixed-parameter time relative to some parameter. To this end, in a manner analogous to the way in which polynomial-time value-approximation has been parameterized (see [13] and references), the framework described in Section 2.2 for structure-approximation can be modified. Consider the following first steps in this direction.

Definition 3.24 *Given an optimization problem* Π *with non-value parameter* p, i.e., *a parameter that does not include the value or any bound on the value being optimized by* Π, *a sd-function* d, *and a non-decreasing function* $h : \mathcal{N} \rightarrow \mathcal{N}$, *an algorithm* A *is a* $\langle p \rangle$**-fixed-parameter** $h(|x|)/d$ **structure-approximation (s-**$\langle p \rangle$**-fp-approx) algorithm** *if for every instance* x *of* Π, $d(A(x), optsol(x)) \leq h(|x|)$ *and* A *runs in time* $f(p)|x|^{\alpha}$ *for some function* $f()$ *and constant* $\alpha > 0$.

Lemma 3.25 *Given an optimization problem* Π *with parameter non-value* p, *a sd-function* d, *and an integer constant* $c > 0$, *if* Π *is* c/d *s-*$\langle p \rangle$*-fp-approximable then* $\langle p \rangle$-Π_D *is FPT.*

Proof: Suppose there is a fixed-parameter c/d s-approx algorithm A for Π relative to parameter p for some integer constant c. Consider the following algorithm for Π_D: Run A on the given instance x of Π_D, generate all $O(|y|^c)$ solutions within distance c of y, and find the solution y' with lowest or highest value in this set, according to the goal of Π. By definition, as y was generated by A, at least one optimal solution for x is within distance c of y, and hence y' will be optimal for x. Use y' to solve x. This algorithm runs in time $f(p)|x|^{\alpha}$ for some function $f()$ and constant $\alpha > 0$, completing the proof. ∎

Corollary 3.26 *Given an optimization problem* Π *with non-value parameter* p, *a sd-function* d, *and an integer constant* $c > 0$, *if* $\langle p \rangle$-Π_D *is* $W[1]$*-hard then* Π *is not* c/d *s-*$\langle p \rangle$*-fp-approximable unless* $FPT = W[1]$.

Consider the LONGEST COMMON SUBSEQUENCE (LCS) problem in which solutions are represented as strings over the alphabet Σ. The standard value function is the length of the solution-string; let the sd-function d be the edit distance between the given solution-strings.

Corollary 3.27 LCS *is not* c/d_E *s-*$\langle m \rangle$*- or s-*$\langle m, |\Sigma| \rangle$*- fp-approximable unless* $FPT = W[t]$, $t \geq 1$.

Proof: Follows from the $W[t]$-hardness, $t \geq 1$, of $\langle m \rangle$-LCS$_D$ [3] and $\langle m, |\Sigma| \rangle$-LCS$_D$ [2] and Corollary 3.26. ∎

Open Problem #11: Develop a parameterized s-(in)approximability theory and "toolkit".

References

[1] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. (1999) *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer; Berlin.

[2] Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hallett, M.T., and Wareham, H.T. (1995) Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, **11(1)**, 49–57.

[3] Bodlaender, H.L., Downey, R.G., Fellows, M.R., and Wareham, H.T. (1994) The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, **147(1-2)**, 31–54.

[4] Chater, N., Tenenbaum, J.B., and Yuille, A. (2006) Probabilistic models of cognition: Conceptual foundations. *Trends in Cognitive Sciences*, **10(7)**, 287-291.

[5] Chen, Z.-Z. and Toda, S. (1991) "On the Complexity of Computing Optimal Solutions." *International Journal of Foundations of Computer Science*, **2(3)**, 207–220.

[6] Feige, U. and Goesmans, M.X. (1995) Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the 3rd Israel Symposium on Theory of Computing Systems*. IEEE Computer Society. 182–189.

[7] Frixione, M. (2001) Tractable competence. *Minds and Machines*, **11**, 379–397.

[8] Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco.

[9] Goldreich, O. (1998) Combinatorial Property Testing – a survey. In Pardalso, P., Rajasekaran, S., and Rolim, J.D.P. (eds.) *Randomozation Methods in Algorithm Design*. AMS-DIMACS. 45–60.

[10] Gusfield, D. (1997) *Algorithms on Strings, Trees and Sequences*. Cambridge University Press.

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

[12] Marko, S. and Ron, D. (2006) Distance Approximation in Bounded-Degree and General Sparse Graphs. In Diaz, J., Jansen, K., Rolim, J.D.P., and Zwick, U. (eds.) *Approximation, Randomization, and Combinatorial Optimization – Algorithms and Techniques: Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006) and the 10th International Workshop on Randomization and Computation (RANDOM 2006)*. Lecture Notes in Computer Science no. 4110. Springer-Verlag; Berlin. 475–486.

[13] Marx, D. (2007) Parameterized complexity and approximation algorithms. *Computer Journal*, to appear.

[14] Millgram, E. (2000) Coherence: The price of the ticket. *Journal of Philosophy*, **97**, 82–93.

[15] Papadimitriou, C. and Yannakakis, M. (1991) Optimization, Approximation, and Complexity Classes. *Journal of Computer and System Sciences*, **43**, 425–440.

[16] Pothos, E. M. and Chater, N. (2002) A simplicity principle in unsupervised human categorization. *Cognitive Science*, **26**, 303–343.

[17] Tenenbaum, J.B., Griffiths, T. L., and Niyogi, S. (2007) Intuitive theories as grammars for causal inference. In A. Gopnik and L. Schulz (eds.) *Causal learning: Psychology, philosophy, and computation*. Oxford University Press.

[18] Thagard, P. and Verbeurgt, K. (1998) Coherence as Constraint Satisfaction. *Cognitive Science*, **22(1)**, 1–24.

[19] van Rooij, I. (2003) *Tractable Cognition: Complexity Theory in Cognitive Psychology*. Ph.D. thesis, Department of Psychology, University of Victoria.

[20] Vazirani, V.V. (2004) *Approximation Algorithms*. Springer; Berlin.