A Survey of the Complexity of Succinctly Encoded Problems

May 5, 2016

Abstract

Succinctness is one of the few ways that we have to get a handle on the complexity of large classes such as NEXP and EXPSPACE. The study of succinctly encoded problems was started by Galperin and Wigderson [3], who showed that many simple graph properties such as connectedness are NP-complete when the graph is represented by a succinct circuit. Papadimitriou and Yannakakis [4], building upon this work, were able to show that a large number of NP-complete graph properties, such as CLIQUE and HAMILTONIAN PATH, become NEXP-complete when the graph is given as a succinct circuit. Wagner [6] examines the relationship between succinct circuits and other forms of succinct representation and proves facts about the hardness of problems in each representation. Das, Scharpfener, and Toran [2] also look at different forms of succinct representation, specifically using more restricted types of circuit such as CNFs and DNFs, and show hardness results for Graph Isomorphism under these representations. This survey will present the main results of these papers in a unified fashion.

1 Introduction

Many computational problems have instances which can be encoded extremely efficiently. For example, many digital circuit design problems in VLSI make use of the highly regular structure of the circuits in question to achieve a smaller description of these circuits. In this case, restricting the search space to circuits which can be represented efficiently speeds up the process of finding the optimal circuit. Also, the efficient representation allows a smaller amount of space to be used. This survey will introduce the efficient representations that are most commonly used in theoretical computer science and will show that for certain problems hard instances can be represented very efficiently.

The efficient input representation that is the most well-studied in theoretical computer science is the succinct circuit representation. A succinct circuit representation (SCR) of a Boolean string (i. e. a string of 0's and 1's) is a circuit whose truth-table is the string. The truth-table of a circuit is the sequence of values it outputs when the inputs range over $\{0,1\}^n$. As a result, the size of the circuit can be logarithmic in the size of its truth-table, yielding a very large compression factor. This compression is even possible with hard instances of some computational problems, meaning that often the complexity of a problem when the input is represented as a succinct circuit is exponentially higher than when the input is represented as a Boolean string. This means that by studying the complexity of problems represented as succinct circuits, we can find problems in the exponentially harder versions of common complexity classes. In particular, in [3], there are many examples of problems such as determining whether a graph is connected which are easy for graphs represented as adjacency lists but NP-hard for graphs represented as succinct circuits, and

in [4], many NP-complete problems, including CLIQUE and INDEPENDENT SET, are shown to be NEXP-complete when the input is represented as a succinct circuit.

In addition to succinct circuits, there are other representations which also provide insight into the complexity of problems. In particular, [2] study inputs represented as the truth tables to CNFs and DNFs. [6] also looks at different representations besides the succinct circuit and succinct formula representation, specifically integer expressions and the generic hierarchical input language. An integer expression is an expression whose terms are tuples of integers and union and setwise addition are the allowed operations. Because addition of two sets A and B can produce a set of size $|A||B|$, this means that the size of sets described by integer expressions can be exponential in the description length, meaning that an integer expression is a succinct representation of some sets. The generic hierarchical input language (GHIL) was introduced to describe sets of rectangles for VLSI design, and its terms are tuples of integers, and it is similar to integer expressions except definitions can be made recursively by referring to previous definitions and addition is only allowed between a set and a term. Due to the recursive definitions, GHIL-expressions can describe sets of exponential size and are succinct input representations. These representations have different properties than succinct circuits, as some problems which are easy when the input is represented as a succinct circuit are hard when the input is represented as an integer expression or GHIL-expression, and vice versa. The following survey will define all of these succinct representations, show hardness results for succinct circuit representations, and then compare different succinct input representations.

2 Preliminaries

2.1 Succinct Representations

In this section, we will define the succinct representations covered in this survey, including succinct circuits, CNFs, DNFs, integer expressions, and GHIL-expressions. We can group these representations into two types; constructive representations such as integer expressions and GHIL-expressions, which build sets out of smaller sets, and predicative representations such as circuits, CNFs, and DNFs, which give a predicate which defines the elements of the set.

Definition 1. *A succinct circuit representation (SCR) of a string x of length 2^n is a Boolean circuit C with n inputs which has x as its truth-table; that is, for $i \in \{0,1\}^n$, $x_i = C(i)$. We denote the truth-table of a circuit C by $tt(C)$.*

Definition 2. *A CNF formula is an AND of ORs; a DNF formula is an OR of ANDs.*

Definition 3. *A succinct CNF (DNF) representation of a string x is a CNF(DNF) ϕ which has x as its truth-table.*

Note that succinct CNFs and DNFs are also succinct circuits and thus reduce to succinct circuits, but the reverse is not true; in particular, there are problems which are hard when the input is given by a circuit and easy when the circuit is given by a DNF.

Definition 4. *For sets $A, B \subseteq \mathbb{N}$, we define $A + B = \{a + b | a \in A, b \in B\}$.*

Definition 5 ([6])**.** *An integer expression is defined as follows:*

1. *$(a_1, ..., a_n)$ is an (n-dimensional) integer expression for $a_1, ..., a_n \in \mathbb{N}$.*

2. *If A and B are integer expressions, then $A \cup B$ and $A + B$ are integer expressions.*

3. *Nothing else is an integer expression.*

The integer expression $(.....((0 \cup 1)+(0 \cup 2))+(0 \cup 4)+...+(0 \cup 2^r))$ describes the set $\{0, 1, ..., 2^{r+1}-1\}$ which has size exponential in the description length, and it is easy to see that every $\cup$ or $+$ operation at most doubles the number of elements in an integer expression. Thus integer expressions are a succinct representation of sets of tuples of integers.

Definition 6 ([6]). *A General Hierarchical Input Language (GHIL)-expression is defined as follows:*

1. *$(a_1, ..., a_n)$ is a GHIL-expression for $a_1, ..., a_n \in \mathbb{N}$.*

2. *$\langle i \rangle$ is a GHIL-expression ($\langle i \rangle$ is interpreted as the ith GHIL-expression already defined*

3. *If A, B are GHIL-expressions then $A \cup B$ is a GHIL-expression.*

4. *If A is a GHIL-expression then $A+(b_1, ..., b_n) = \{(a_1+b_1, a_2+b_2, ..., a_n+b_n) | (a_1, ..., a_n) \in A\}$ is a GHIL-expression.*

5. *Nothing else is a GHIL-expression.*

The set that a GHIL-expression describes is the last GHIL-expression in the sequence. To show that a GHIL-expression is a succinct representation, we want to exhibit a set of size exponential in the description length. Consider the GHIL-expression $\{0, \langle 0 \rangle \cup (\langle 0 \rangle + 1), \langle 1 \rangle + (\langle 1 \rangle + 2), ..., (\langle r \rangle) \cup (\langle r \rangle) + 2^r\}$. By induction we can show that the ith GHIL-expression in this sequence contains the numbers from 0 to $2^i - 1$. Thus this gives a set of size 2^{r+1} with a description of size r^2. This implies that GHIL-expressions are a succinct representation of sets. It is also easy to see that the size of the set of a GHIL-expression can only grow exponentially in the description length.

There are important reducibilities between these input representations. Obviously succinct CNFs, DNFs, and formulas reduce to succinct circuits. What is not so obvious is that integer expressions reduce to GHIL-expressions.

Proposition 1 ([6]). *For every integer expression we can find a GHIL-expression which describes the same set in logarithmic space.*

We present a sketch of the proof here. Suppose we have some integer expression A. We will proceed by induction. If $A = (a_1, ..., a_n)$, then A is also a GHIL-expression. If $A = A_1 \cup A_2$, then we take the GHIL-expressions describing A_1 and A_2 and take the union, which is also a GHIL-expression. If $A = A_1 + A_2$, we do the following. We replace every instance of $(a_1, ..., a_n)$ in A_2 with $A_1 + (a_1, ..., a_n)$. To see that this works, note that every element in A_2 is the sum of tuples in the GHIL-expression for A_2, and adding A_1 to these tuples makes it so that these sums of tuples also include an element of A_1, which yields exactly $A_1 + A_2$.

In general, we will find that there is no reducibility between integer expressions or GHILs and any of boolean circuits, formulas, or CNFs unless P = NP; this is because the membership problem for GHIL-expressions and integer expressions is NP-complete while the membership problem for boolean representations is in P, and the nonemptiness problem for GHIL-expressions and integer expressions is P-complete while the nonemptiness problem is NP-complete for circuits, formulas, and CNFs.

2.2 Local Reductions

What kinds of reductions extend from the setting of ordinary representations to the setting of succinct representations? If we have a problem which reduces to another problem in polynomial time, in general the same reducibility will not work when the instances are represented succinctly, as now the instances have exponential size. However, if we look at weaker reductions, we can scale these up to polynomial time reductions for succinct representations.

Definition 7. *A time $t(n)$-projection from language A to language B is a function f such that $x \in A \leftrightarrow f(x) \in B$ and there exists a machine M running in time $t(n)$ that given random access to (x, i) outputs the ith bit of $f(x)$.*

We can think of projections as very local reductions; the machine only has time to read a small fraction of the input for small values of $t(n)$ (typically we consider $t(n)$ polylogarithmic in n), so each output bit depends on a small fraction of the input bits. These local reductions are still powerful; almost all known NP-complete problems have a projection from SAT [4]. Given this reduction which works locally, we can turn it into a reduction for succinct circuits which runs in polynomial time.

Proposition 2 ([4])**.** *Suppose that A reduces to B under polylogarithmic-time projections. Then the set $\{C|tt(C) \in A\}$ reduces to $\{C|tt(C) \in B\}$ under polynomial time many-one reductions.*

The proof is simple; consider the machine M which runs in polylogarithmic time and computes the output bits of f. M runs in polynomial time on an input of exponential size, so we can turn M into a circuit, replacing random access to the tape with the succinct instance. Then, this composition is a succinct circuit representation of $f(tt(C))$.

The notion of projections gives us a useful way to lift hardness results for problems to hardness results for problems with succinct input representations. In particular, the succinct version of problems that are NP-complete under projections are NEXP-complete.

2.3 Class Operators

For some of the results, we will need to define the following operators on complexity classes.

Definition 8. *$\exists\mathcal{C}$ is the set of languages L such that $x \in L \leftrightarrow \exists y \in \{0,1\}^{p(|x|)}(x, y) \in L'$ where $L' \in \mathcal{C}$.*

Remark. *$\exists P = NP$.*

Similarly, we can define the $\forall$ operator, and we will have that $\forall$P = coNP. In addition, we will need to define the counting operator; this operator counts the number of 'proofs' that a statement is in the language, as we can consider the $\exists$ operator as saying that a string is in the language if there exists a proof such that the language of strings with proofs is in the base set.

Definition 9. **C**$\mathcal{C}$ *is the set of languages L such that $x \in L \leftrightarrow |\{y \in \{0,1\}^{p(n)} : (x, y) \in L'\}| \geq f(|x|)$ for some f computable in polynomial time and $L' \in \mathcal{C}$.*

Remark. **C**$P = PP$.

3 Results

3.1 Hardness for Succinct Circuit Representations

We will begin with looking at properties of succinct circuit representations, and then we will continue with other representations in order to highlight similarities and differences between succinct circuit representations and other representations. Our first result is that a variety of graph problems are NP-complete for succinct circuits.

Definition 10. *The succinct circuit representation of a graph G with 2^n vertices is a circuit $C(x, y)$ such that $C(x, y) = 1$ iff x and y have an edge between them in G.*

Theorem 1 ([3])**.** *The following problems are NP-hard for graphs represented as succinct circuits:*

1. *Determining whether a graph has an edge.*
2. *Determining whether a graph has a triangle.*

Proof. These proofs all proceed similarly.

1. We reduce from SAT. Consider a circuit C. Then, make the circuit C' with $C'(x, y) = 1$ iff $x = 0^{n+1}$ and $C(y_2y_3...y_{n+1}) = 1$. This graph has an edge iff C' is satisfiable, and this reduction runs in polynomial time.
2. This follows similarly to 1. This time, we make C' such that $C'(x, y) = 1$ iff $x = 0^{n+1}$ or $x = 0^n1$ and $C(y_2y_3...y_{n+1}) = 1$. Then this graph has a triangle iff C' is satisfiable.

□

We can generalize this structure as follows. The key point is that adding an edge in the right place makes the graph have the property we are checking for, and thus we can make it so that these edges exist iff the circuit is satisfiable, and then use the NP-hardness of SAT to prove NP-hardness of checking whether the graph has the property.

Definition 11 ([3])**.** *A t-critical graph with respect to a property Q is a graph with $O(t)$ vertices such that there are no edges between the first $t-1$ vertices and the tth vertex and adding any subset of those edges makes the graph go from not having property Q to having property Q.*

Theorem 2 ([3])**.** *For any property Q, if there exist t-critical graphs with respect to Q that can be represented succinctly for all positive integers t, then determining whether a succinct representation of a graph has property Q is NP-hard.*

We use the same idea as above to reduce from SAT. If the circuit is satisfiable, we can construct a circuit representing a t-critical graph which has the property that an edge making the graph have property Q is present iff the circuit has a satisfying assignment. Thus if we could determine whether the graph given by a succinct representation has property Q, we could solve SAT and then all of NP. This implies a large class of problems that are easy for ordinary representations of graphs are at least NP-hard for succinct representations. One can ask if the exponential relationship between the succinct circuit representation and the instance it represents shows up in the complexity of problems; [4] shows that for NP-complete problems it does by showing that succinct versions of NP-complete problems are complete for NEXP.

Theorem 3 ([4])**.** *SUCCINCT 3-SAT is complete for NEXP under polynomial-time many-one reductions.*

Clearly SUCCINCT 3-SAT is in NEXP, as we can write out the exponential-sized instance and guess an exponential-sized solution. To show that SUCCINCT 3-SAT is hard for NEXP, note that we can run the Cook-Levin reduction on a Turing machine running in exponential time to get an exponential-sized formula. It suffices to show that this formula has a succinct circuit representation and that this representation can be computed efficiently. Note that the formula produced is highly structured; it contains clauses which force a satisfying assignment to encode the input, clauses which force variables to encode the transition function of the nondeterministic Turing machine, and clauses which check that we have an accepting computation. In particular, given a clause, we can compute in polynomial time what variables this clause should have, and from this we can represent the formula as a succinct circuit.

Combining the previous theorem and 2, we get that the succinct version of every NP-complete problem that has a projection from 3-SAT is NEXP-complete under polynomial-time many-one reductions. We will show that there is a projection from 3-SAT to INDEPENDENT SET below to demonstrate what these projections look like.

Theorem 4. *There is a projection from 3-SAT to INDEPENDENT SET which runs in polylogarithmic time.*

Proof. The projection proceeds as follows. The vertices will be labeled with clauses and an index from 1 to 3. On input $z = xy$, the projection machine needs to return whether x and y have an edge, or return a bit in the representation of the size of the independent set, which will be the number of clauses (this can be found in logarithmic time using binary search). To check for an edge, we neet to check if either x and y are in the same clause or if x and y are in different clauses but their variables are negations of each other. This can easily be done in polylogarithmic time. To see that this is a reduction, note that there will be an independent set of size equal to the number of clauses iff we can pick a variable from each clause without picking two variables which are negations of each other iff there exists a satisfying assignment. Thus 3-SAT reduces to INDEPENDENT SET under projections. □

Corollary 1. *SUCCINCT INDEPENDENT SET is complete for NEXP under polynomial-time many-one reductions.*

3.2 Comparison between Succinct Circuits, Integer Expressions, and GHIL-expressions

Here we will prove that the two types of input representations are incompatible assuming P $\neq$ NP; that is, we will show that there is a problem that is P-complete in one representation but NP-complete in the other, and vice versa.

Theorem 5 ([6])**.** *The problem of determining whether x is in the set described by a succinct representation is*

1. *P-complete for succinct circuits.*
2. *NP-complete for integer expressions and GHIL-expressions.*

Proof. P-completeness of the membership problem for succinct ciruits follows from P-completeness of the circuit value problem, as determining whether x is in the set is the same as determining whether $C(x) = 1$. For integer expressions and GHIL-expressions, every element is the sum of a polynomial number of elements, and it is easy to see that membership can be decided in NP as a result. To show NP-hardness, we can reduce from SUBSET SUM; given a SUBSET SUM instance $(a_1, ..., a_n, B)$, and asked whether there is some subset of $\{a_1, ..., a_n\}$ which sums to B, we see that asking whether B is in the set $\{0, a_1\} + \{0, a_2\} + \{0, a_3\} + ... + \{0, a_n\}$ is an equivalent problem. Thus the membership problem for integer expressions is NP-hard. □

Theorem 6. *The problem of determining whether a set described by a succinct representation is nonempty is*

1. *NP-complete for succinct circuits.*

2. *in P for integer expressions and GHIL-expressions.*

Proof. Any integer expression and GHIL-expression describes a nonempty set, so determining whether an integer expression is nonempty is the same as determining whether an integer expression is valid, which can clearly be done in polynomial time. The nonemptiness problem for succinct circuits is exactly the same as SAT; finding if there exists an element x in the set is the same as finding if there exists x such that $C(x) = 1$. □

Theorem 7. *The problem of determining whether a set has at least m elements is*

1. *PP-complete for succinct circuits.*

2. **C***NP-complete for integer expressions and GHIL-expressions.*

The first part is simple to prove: we note that counting the number of elements in a set represented by a succinct circuit is the same as counting the number of 1 outputs of this circuit, which is complete for PP. To show that this problem is **C**NP-complete for integer expressions, we note that there is a reduction from formulas to integer expressions which makes the number of elements the same as the number of assignments and which clauses the assignment satisfies determines the value of the elements. Then, the number of elements in this set is going to correspond to the number of strings that give a satisfying formula, and this problem is complete for **C**NP. More broadly, the membership problem for integer expressions and GHIL-expressions is NP-complete, so counting the number of elements in an integer expression or GHIL-expression should be **C**NP-complete.

The table below states many of the results in [6] for integer expressions, GHIL-expressions, and succinct circuits, the proofs of which of which are omitted for brevity.

Problem	Integer Expression/GHIL complexity	Succinct Circuit complexity
Membership	NP-complete	P-complete
Non-emptiness	in L	NP-complete
Critical element	NP-complete	coNP-complete
Intersection	NP-complete	NP-complete
Subset, equality	Π_2^p-complete	coNP-complete
Cardinality	**C**NP-complete	PP-complete

3.3 Hardness for CNF and DNF representations

For CNF and DNF representations, sometimes an exponential gap is not found. The reason that this is the case is that CNFs and DNFs are very restricted classes of circuits, and are not able to encode as much structure. Also, CNFs and DNFs are very structured themselves, which means that sometimes the only instances which can be encoded using small CNFs and DNFs are easy. One example of a smaller gap in complexity is the Dominating Set problem, which asks for a set of vertices for which every vertex is either in the set or has an edge to an element in the set. This problem is NEXP-complete when the input is the truth-table to a succinct circuit but PP-complete when the input is the truth-table to a DNF.

Theorem 8 ([2])**.** *Dominating Set is complete for PP when the graph is succinctly represented as a DNF.*

To see that Dominating Set is in PP when the graph is encoded as a DNF, note that for every edge in the graph, the two endpoints satisfy one of the terms of the DNF. Also, each term defines a biclique, or complete bipartite graph, as each term takes two vertices as input and outputs 1 iff the two vertices match some predetermined pattern, which means that every vertex which matches the first pattern is connected to every vertex which matches the second pattern. This means that every non-isolated vertex is in one of these bicliques, so if we pick two vertices on opposide sides of the biclique we dominate every non-isolated vertex (Note that this means that if the graph has no isolated vertices, the problem of finding a dominating set in a graph in succinct DNF form is in NP). Then what remains is to count the number of isolated vertices, which can be done in PP, and this implies that we can use nonadaptive queries to PP to compute whether there is a small dominating set. Since PP is closed under truth-table reductions, this implies that Dominating Set is in PP. To show completeness, reduce from determining whether a DNF has at least k satisfying assignments. Each assignment will correspond to a vertex in the graph, and the vertex will be connected to a central vertex iff the assignment is satisfying. This can be done by making each term of the original DNF correspond to the first vertex and making the central vertex the second vertex in the term defining an edge. Then, there are a small number of isolated vertices, and thus a small dominating set, iff there are at least k satisfying assignments to the DNF. Note that [4] implies that Dominating Set for succinct circuit representations is complete for NEXP, giving a substantial gap between the two input representations. On the other hand, for Graph Isomorphism, no such gap exists.

Theorem 9. *There are polynomial-time reductions between succinct CNF, DNF, and circuit representations of Graph Isomorphism.*

A sketch of the proof follows. Since two graphs are isomorphic iff their complements are, this gives a reduction between the CNF respresentation and the DNF representation. Also, a CNF is a type of circuit, so that leaves the reduction from circuit representations to CNF representations. First we turn the circuit into a CNF where every satisfying assignment to the circuit corresponds to a single satisfying assignment of the formula (c.f. [5]). We can think of the formula as representing a hypergraph with only 3-edges, where each 3-edge is two vertices from the original graph along with a vertex which encodes the correct values that the wires of the original circuit take when given the two vertices as input. Then, the two hypergraphs are isomorphic iff the original graphs are isomorphic. The CNF encoding the hypergraphs can then be turned into a CNF encoding regular graphs which are isomorphic iff the hypergraphs are.

Thus, to prove completeness results for succinct Graph Isomorphism, it suffices to show results under any one of these encodings. Note that since Graph Isomorphism can be done in quasipolynomial time [1], this implies that solving Graph Isomorphism on an instance of size 2^n can be done in exponential time. However, it has not been shown that succinct graph isomorphism is hard for EXP; the best lower bound we have is PSPACE.

Theorem 10. *Graph Isomorphism for succinctly represented graphs is PSPACE-hard.*

This theorem is proved by showing that there is a polylogarithmic-time projection from undirected s-t connectivity to graph non-isomorphism. Since undirected s-t connectivity is complete for L, and this reduction can be made succinct because checking whether there is an edge between two vertices is the same as checking whether one configuration follows from another which can be done locally, we can see that finding a path in a graph described by a succinct circuit is PSPACE-complete. To do the reduction, let each graph be two copies of the original graph. We describe the graph with colors, which can be constructed easily by adding large numbers of vertices connected differently. In the first graph, color s in the first copy with color 1, color t in the first copy with color 2, and color s in the second copy with color 3. In the second graph, color s in the first copy with color 1, color t in the second copy with color 2, and color s in the second copy with color 3. If these two graphs are isomorphic, then the part of the graph containing s and the part of the graph containing t are not connected, and if the two graphs are not isomorphic, the two parts are connected. Determining whether two vertices have an edge in this graph can be done in polylogarithmic time; we just check whether the edge is in the original graph or whether the edge is used in the coloring procedure. Thus succinct s-t connectivity reduces to succinct graph isomorphism and succinct Graph Isomorphism is hard for PSPACE.

4 Discussion

Succinct circuit representations of problems have many uses in theoretical computer science. Most of the NEXP-complete problems we know are succinct versions of NP-complete problems. In fact, in the proof that NEXP is not contained in ACC_0 from [7], the fact that the complete problem used is SUCCINCT SAT plays a large part in the proof, as the proof depends on the existence of succinct circuits encoding a witness to the instance of SUCCINCT SAT. In addition, the existence of succinct circuit representations means that hard instances of SAT are very easy to construct, to the point where instances can be described with only a logarithmic number of bits.

Some questions about succinct representations remain unsolved. A significant question is determining the complexity of NP-complete problems represented as integer-expressions or GHIL-expressions. Are these problems also complete for NEXP? In addition, an explicit problem that is NP-complete but is not NEXP-complete for succinct inputs has not been found, and similar questions are open for other complexity classes. In general, this requires finding a problem which is hard, but with hard instances that are not described easily; the reduction proving hardness has to be very non-local, because computation in the Turing machine model is local.

References

[1] L. Babai. Graph isomorphism in quasipolynomial time. In *Proc. 48th Annual Symposium on the Theory of Computing (STOC)*, 2016.

[2] B. Das, P. Scharpfenecker, and J. Torán. Succinct encodings of graph isomorphism. In *Language and Automata Theory and Applications*, pages 285–296. Springer, 2014.

[3] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.

[4] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[5] U. Schöning and J. Torán. *The Satisfiability Problem: Algorithms and Analyses*, volume 3. Lehmanns Media, 2013.

[6] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.

[7] R. Williams. Nonuniform acc circuit lower bounds. *Journal of the ACM (JACM)*, 61(1):2, 2014.

18.405J / 6.841J Advanced Complexity Theory

Spring 2016