From the Shortest Vector Problem to the Dihedral Hidden Subgroup Problem

Curtis Bright

December 29, 2011

Abstract

In *Quantum Computation and Lattice Problems* [11] Oded Regev presented the first known connection between lattices and quantum computation, in the form of a quantum reduction from the poly(n)-unique shortest vector problem to the dihedral hidden subgroup problem by sampling cosets. This article contains a summary of Regev's result.

1 Introduction

Lattice problems are widely studied in mathematics and computer science and have applications to fields such as number theory and cryptography. Their highly periodic nature suggests they might be amenable to attack by quantum computers and their ability to work in quantum superposition. Despite this, relatively few connections between lattice problems and quantum computation are known.

The first explicit connection discovered [11] was based on the dihedral hidden subgroup problem, which is interesting since at first this seems like a problem completely unconnected to lattices. The result is also of interest because it was the first known application of the dihedral hidden subgroup problem to a "useful" problem.

2 Background

In this section we discuss the background material required for Regev's result, which requires the concept of reduction and knowledge of lattices and dihedral groups. We use the notation poly(n) for $n^{O(1)}$.

2.1 Reductions

The notion of *reduction* between problems is ubiquitous in computer science. Roughly, "problem A reduces to problem B" means there is a way of solving A given some way of solving B, and as shorthand we write

$$A \leq B.$$

The idea behind this notation is that intuitively we are saying problem B is at least as hard as problem A. The reason is that if someone came up with a method of solving problem B, they could then use that method to solve problem A: they simply translate a given instance of A to an instance of B using the reduction, and then solve B to solve A. Of course, the method of translation from A to B should be efficiently computable, or else this method of solving A would not be useful.

So $A \leq B$ is roughly saying that A and B have similar difficulty, except that A might be easier, since there might be a better way of solving A than using the reduction. In general not every method of solving A is applicable to solving B; in some sense instances of A can be considered as "special cases" of instances of B.

There are many different types of reduction. The standard *Karp reduction* requires that the translation between A and B be computable by a Turing machine in polynomial time, and the "black box" or "oracle" for B can only be called once at the reduction's conclusion. The more general *Cook reduction* is similar but also allows the black box to be called multiple times.

Reductions can be even more general, for example in "white box" reductions we are also allowed to see the "source code" of the solver for B. We can also expand the model of computation to allow randomness or quantum circuits in the reduction.

2.2 Point lattices

A *point lattice* is a discrete additive subgroup of $\mathbb{R}^n$. A set of linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ generate a lattice L by taking their 'integer span', that is,

$$L = \left\{ \sum_{i=1}^{n} x_i \boldsymbol{b}_i : x_i \in \mathbb{Z} \right\}.$$

The $\boldsymbol{b}_i$ form what is called a *basis* of L. Every lattice which contains at least two linearly independent vectors has infinitely many bases, and a central lattice problem is how to compute *short* basis vectors of a given lattice.

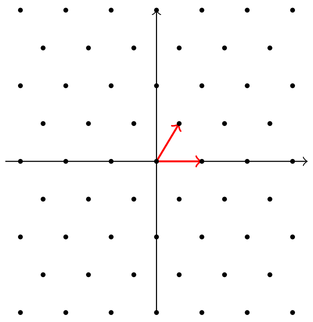


Figure 1: An example lattice in two dimensions, with one basis marked in red.

The *shortest vector problem* (SVP) is to find the shortest nonzero vector in a given lattice described by some basis vectors. (The zero vector is always excluded from being the shortest or the problem would be trivial.) In two dimensions the problem is known to be solvable in polynomial time by a generalization of Euclid's gcd algorithm [4]. Given two basis vectors, simply subtract off a multiple of the shorter one from the longer one to decrease the length of the longer one. Continue doing this until it is no longer possible to decrease the length and you will have found the shortest vector.

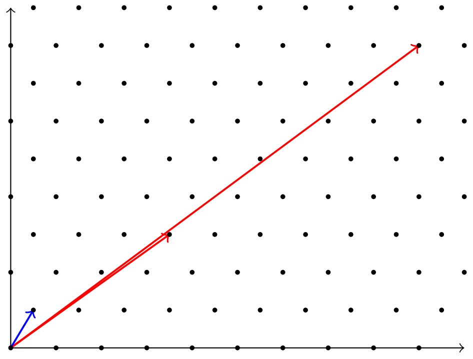


Figure 2: An example instance of the shortest vector problem. Given the red vectors, find the blue vector.

However, in n dimensions the problem seems to be extremely difficult. Indeed, all known algorithms for the SVP run in exponential time in n. When using the max-norm the problem is known to be NP-hard [13], i.e.,

$$A \leq \text{max-norm SVP}$$

for all $A \in \mathsf{NP}$, where the reduction used is the standard Karp reduction. For the more standard Euclidean norm the problem is suspected to be NP-hard and is known to be NP-hard using a RUR-reduction [2]. This reduction makes use of randomization and is only required to be accurate with high probability. That is, if you had an oracle which could solve SVP then you could solve all problems in NP provided you make use of randomization and accept a small possibility of error.

Since the SVP is too hard in general to solve exactly, we will be content with solving it approximately, i.e., finding some vector which is not too much larger than the shortest one. Approximating the shortest vector to within a constant is also known to be NP-hard under a randomized reduction [10], so it is likely we will have to accept approximation factors which grow as the dimension n increases.

Classically, the so-called LLL algorithm [8] approximates the shortest vector to within a factor of $2^{(n-1)/2}$. This is not ideal, since the approximation factor is exponential in n, but at least the algorithm provably runs in polynomial time and provably finds a vector with some approximation factor.

As a special case of approximation, we will consider solving the SVP in lattices which have an especially short shortest vector. Say a vector is $f(n)$-unique if it is a factor of $f(n)$ shorter than all other nonparallel vectors. Then the SVP in lattices with a $O(2^n)$-unique shortest vector is solvable in polynomial time, since the approximation vector that LLL finds is in fact guaranteed to be the shortest vector in this case.

Since the $O(1)$-unique SVP is probably hard, and the $O(2^n)$-unique SVP is in P, this raises the following open question: can we solve the poly(n)-unique SVP in polynomial time? This is not expected to be NP-hard, since the problem is known to be in NP and coNP [1, 9], and it being NP-hard would imply $\mathsf{NP} = \mathsf{coNP}$.

More precisely, the *decision version* of the poly(n)-unique SVP problem lies in NP and coNP. This means that given a lattice and a target bound d there is a short certificate which proves that the shortest vector is either shorter than d or longer than d. In the former case, the certificate is simply the shortest vector itself. In the latter case, the certificate is a basis of the lattice in which every vector in the Gram–Schmidt orthogonalization of the basis is longer than d.

2.3 Dihedral group

The *dihedral group of order* $2N$, denoted D_{2N}, is the set of "symmetries" of an N-sided regular polygon. More concretely, it is the collection of all the ways an N-sided regular polygon can be rotated or reflected keeping the the outline of the shape fixed.

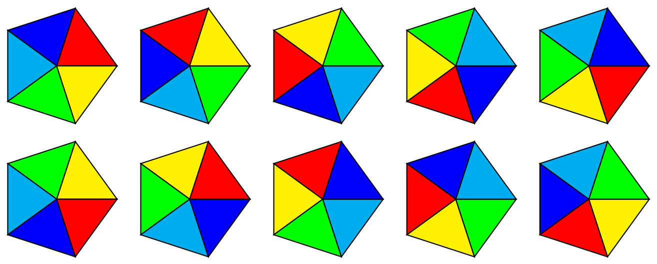


Figure 3: A diagram of D_{10}, the dihedral group of 10 elements.

There are $2N$ ways one can apply rotations or reflections in a distinct way. Every element of D_{2N} can be represented as a tuple (s, r) where $s \in \{0, 1\}$ represents the number of reflections and $0 \leq r < N$ represents the number of rotations.

Any two elements of D_{2N} can be *composed* by performing the actions of each element in sequence. For example, the element representing a single rotation can be composed with the element representing 2 rotations to form the element representing 3 rotations. Composing a single rotation with itself N times yields a element representing N rotations, which is identical to performing no rotations. This means the set of only rotations can be thought of as equivalent to the integers mod N.

One complicating factor is that D_{2N} is a *non-commutative* group. That is, the result of a computation depends on the order the operations were done in. For example, performing a rotation followed by a reflection gives a different result than performing a reflection followed by a rotation.

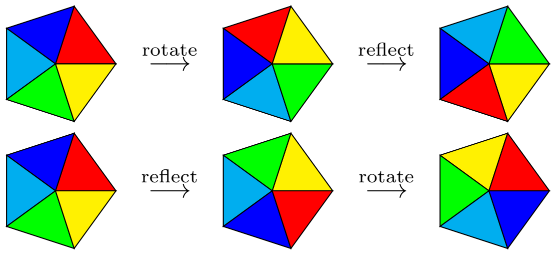


Figure 4: An example showing the non-commutativity of D_{10}.

3 Hidden subgroup problem

We say a function f on a group G *hides* a subgroup $H \subseteq G$ if it satisfies the following conditions:

- f is constant (for simplicity, say 0) on the subgroup H
- f is constant on the cosets of H and each coset has a distinct value

A coset can be thought of as a "translation" of a subgroup. For example, in the dihedral group when you apply a reflection to all elements in a subgroup you get a coset. Technically, this is a *left* coset, and for non-commutative groups we should specify either left or right cosets in the conditions on f.

An example of a subgroup-hiding function f on the dihedral group D_{10} is:

$f($⬟$) := 0$	$f($⬟$) := 1$	$f($⬟$) := 2$	$f($⬟$) := 3$	$f($⬟$) := 4$
$f($⬟$) := 3$	$f($⬟$) := 4$	$f($⬟$) := 0$	$f($⬟$) := 1$	$f($⬟$) := 2$

This function hides the subgroup {⬟,⬟} $= \{(0,0),(1,2)\}$.

The *hidden subgroup problem* (HSP) is to find H with as few queries to f as possible. We would also like the process to be efficient, namely to run in polynomial time with respect to $\log|G|$. Since $|H|$ itself might be exponential in $\log|G|$, this might only be possible by specifying H in a compact representation (for example by a set of elements which generate H).

The naïve way of solving the HSP is simply to query f on every element and output those which give the value 0, but this is not efficient. However, in some cases this is essentially the best we can do, at least classically. For example, in the worst case of the dihedral HSP, H is of size 2 and contains a flipped element which has been rotated k times. Finding k classically essentially requires a brute-force search by querying f on flipped elements with i rotations for $0 \le i < n$. When the query value is nonzero you receive no information about k other than removing one possibility for it, so in the worst case you must try all i and use $\Omega(|G|)$ queries of f.

The HSP is important to quantum computation since most quantum algorithms which run super-polynomially faster than all known classical algorithms do so by solving special cases of the HSP. For example, Shor's algorithm for factoring integers [12] solves the cyclic HSP. In fact, quantum computers can solve the HSP in polynomial time for commutative groups [3] and can solve the general HSP with only a polynomial number of queries to f, though the process itself is still exponential time [5].

3.1 Solving HSP by sampling cosets

The so-called "standard method" of solving the HSP with a quantum computer involves a process known as *sampling cosets*. A *coset state* is a state which is in a superposition over all elements in a coset. The construction of a coset state proceeds in three steps:

1. Construct superposition of over all elements of G:
$$\frac{1}{|G|}\sum_{g\in G}|g\rangle$$
2. Query f in an ancilla register:
$$\frac{1}{|G|}\sum_{g\in G}|g\rangle|f(g)\rangle$$
3. Discard the second register:
$$\frac{1}{|H|}\sum_{h\in H}|gh\rangle$$
for some random $g \in G$.

To see why this is the result, say the second register was measured with the result $f(g)$. Then the state collapses to a superposition of the states consistent with the second register being $f(g)$. Since f is constant on cosets, all elements of the form gh for $h \in H$ are consistent with this information. Since f was distinct on cosets, the only elements consistent with this information are of the form gh for $h \in H$. Thus the state collapses to exactly a superposition of elements of the form gh for $h \in H$, i.e., some coset state of H (namely, the coset which contains g).

Measuring the coset state gives a random element of a random coset, so is not helpful directly. For certain groups such as the integers mod N it is possible to use a quantum Fourier transform to find H. Very roughly, it removes the influence of the random factor g in the coset state. In the non-commutative case things don't work out so nicely, but there is still hope that after constructing multiple coset states one can use them together to determine H somehow.

4 Dihedral coset problem

The *dihedral coset problem* (DCP) is to find the constant integer d given a collection of states of the form

$$\tfrac{1}{\sqrt{2}}|0\rangle|x\rangle + \tfrac{1}{\sqrt{2}}|1\rangle|x+d\rangle \qquad \text{for random } x,$$

where arithmetic is done mod N.

The *dihedral* name arises because states of this form can actually be thought of as coset states of the dihedral group D_{2N} (namely, coset states of an order 2 subgroup). The first qubit of the state encodes the number of rotations of an element and the remaining register encodes the number of rotations of an element. This is the reason why arithmetic is done mod N, to ensure that N rotations is the same as 0 rotations, as they are in D_{2N}.

The problem is closely related to the dihedral HSP. In fact, if we could solve the dihedral HSP by coset sampling then we could solve the DCP. The reduction is of the "white box" form and works by modifying the source code of a procedure which solves the dihedral HSP by coset sampling.

The modified procedure will no longer construct coset samples as described in the previous section. Instead, the states which were to be constructed by coset sampling will be replaced by the states we were given in the DCP. We then run the procedure as normal, pretending the DCP states were constructed by coset sampling. This will work since as mentioned the DCP states can in fact be viewed as coset states. The output of the modified procedure will be a dihedral subgroup of the form $\{(0,0),(1,d)\}$, from which it is easy to read off the value of d and thus solve the DCP.

This shows that solving the dihedral coset problem reduces to solving the dihedral hidden subgroup problem by coset sampling,

$$\text{DCP} \leq \text{dihedral HSP by coset sampling.}$$

5 Two-point problem

As a generalization of the DCP, the *two-point problem* is to find the constant integer vector $\boldsymbol{d}$ given a collection of states of the form

$$\tfrac{1}{\sqrt{2}}|0\rangle|\boldsymbol{x}\rangle + \tfrac{1}{\sqrt{2}}|1\rangle|\boldsymbol{y}\rangle$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are random vectors which satisfy $\boldsymbol{y} - \boldsymbol{x} = \boldsymbol{d}$, where arithmetic is done mod M.

Since the DCP is a special case of the two-point problem we immediately have that

$$\text{DCP} \leq \text{two-point problem.}$$

Although the two-point problem seems harder than the DCP since $\boldsymbol{d}$ is now a vector rather than an integer, in fact the problem is no harder since a solution to the DCP can be used to solve the two-point problem, as we will now show. The key idea behind this reduction is to think of vectors as integers which are written in some base.

Consider the function f which views $\boldsymbol{x}$ (mod M) as a “base $2M$” integer,

$$f(\boldsymbol{x}) \coloneqq \sum_{i=1}^{n} x_i (2M)^{i-1}$$

where x_i is the ith entry of $\boldsymbol{x}$ (mod M) with $0 \leq x_i < M$. Applying f to the second register of a state given in the two-point problem gives

$$\tfrac{1}{\sqrt{2}}|0\rangle|f(\boldsymbol{x})\rangle + \tfrac{1}{\sqrt{2}}|1\rangle|f(\boldsymbol{y})\rangle,$$

which is a valid instance to the DCP with $N = (2M)^n$ since $f(\boldsymbol{y}) - f(\boldsymbol{x})$ is constant mod $(2M)^n$.

Applying a black box which can solve the DCP to this instance gives the result

$$\sum_{i=1}^{n} (y_i - x_i)(2M)^{i-1},$$

and we would like to be able to extract $y_i - x_i = d_i$ from this integer. Since $x_i,\ y_i \in [0, M)$ we have $0 < y_i - x_i + M < 2M$. Thus adding $\sum_{i=1}^{n} M(2M)^{i-1}$ to the result gives a number whose base $2M$ expansion contains as “digits” $y_i - x_i + M$, from which $y_i - x_i = d_i$ can be read off.

Therefore if we could solve the DCP, or had a black box which could solve the DCP, we could find $\boldsymbol{d}$ in the two-point problem by making the base $2M$ conversion as described, which shows

$$\text{two-point problem} \leq \text{DCP.}$$

So the two problems are essentially the same difficulty, despite the two-point problem at first seeming more difficult.

6 Application to lattices

In this section we will use a solution to the two-point problem to solve the poly(n)-unique shortest vector problem. The method will exploit the ability of quantum computers to work in superposition by constructing a superposition over lattice points. The first step in this process to to create a superposition over points in $\mathbb{Z}^n$,

$$|\boldsymbol{Z}\rangle := \frac{1}{\sqrt{M^n}} \sum_{\substack{\boldsymbol{x}\in\mathbb{Z}^n \\ 0\le x_i<M}} |\boldsymbol{x}\rangle.$$

Ideally, we would like to construct a superposition over all points in $\mathbb{Z}^n$ but it will be sufficient to take M large enough, say $M = 2^{O(n)}$.

Using the definition of a lattice generated by $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$, the lattice superposition is constructed by applying the function

$$f(\boldsymbol{x}) := \sum_{i=1}^{n} x_i \boldsymbol{b}_i$$

to the state $|\boldsymbol{Z}\rangle$ and storing the result in an ancillary register.

6.1 Space partitioning

The essential component of our solution will involve partitioning $\mathbb{R}^n$ into cubes. In the optimal partitioning there will be exactly two points in each cell and the difference between the two points will be the shortest vector.

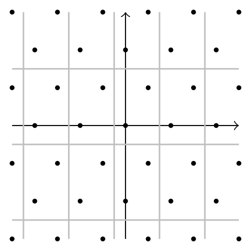


Figure 5: Example of the optimal space partitioning in two dimensions.

To make use of the partitioning, we will construct a unique cube labelling function g and apply it to the register which contains our lattice superposition. We then discard it, which collapses the state to a superposition of two points whose difference encodes the shortest vector. For example, g could be the function

$$g(\boldsymbol{x}) := \big(\lfloor x_1/r - s_1\rfloor, \lfloor x_2/r - s_2\rfloor, \ldots, \lfloor x_n/r - s_n\rfloor\big)$$

for the partitioning of cubes with length r offset by the vector $\boldsymbol{s} = (s_1, \ldots, s_n)$.

However, Figure 5 is misleading in that one already needs to know the shortest vector to be able to construct that partition. Instead, we must make do with *approximate* partitioning. For this we need an assumption that the shortest vector is a factor of $\Omega(\sqrt{n})$ shorter than all other nonparallel vectors. If the cube length is approximately the length of shortest vector then every vector in each cube will be aligned along the shortest vector, since all other vectors will be longer than the cube diagonal.

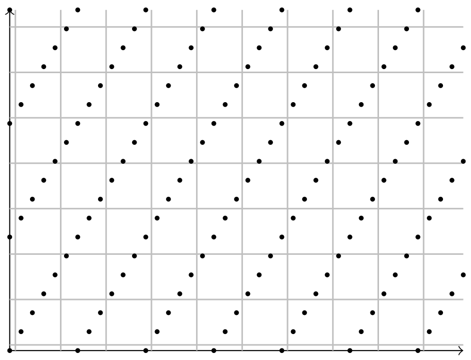
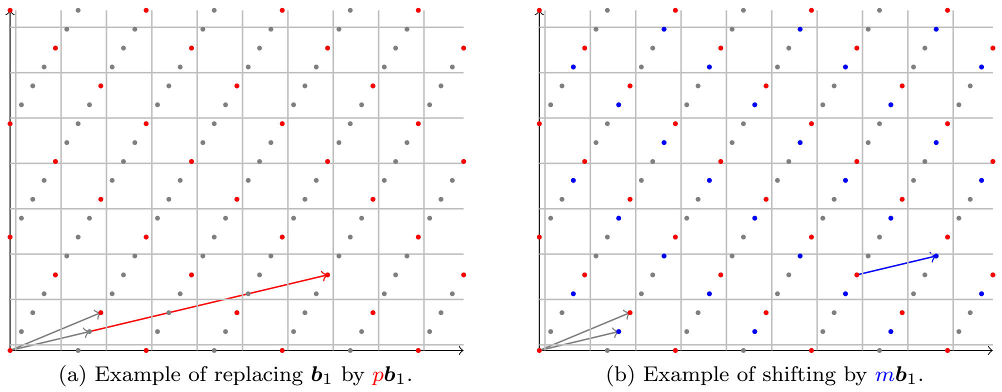


Figure 6: All vectors in each cube are aligned along the shortest vector.

However, this partitioning is inappropriate since there may be multiple points aligned along the shortest vector inside each cube. To remedy this, we scale up one of the basis vectors (say $\boldsymbol{b}_1$) by a factor p to form a superposition over a subset of the points. This ensures that there is at most one point in each cube, though we would like two points in each cube. To achieve this we shift the points over by $m\boldsymbol{b}_1$ (for some appropriate factor m) based on a Boolean variable t. By placing t in a superposition of $|0\rangle$ and $|1\rangle$ we can form a superposition of lattice points with and without the shift.

The effect of the scale and shift modifications is demonstrated in Figure 7. The parameter p controls the red points; the larger p is the more spread out the "slices" of red points will be. The parameter m controls the blue points and the amount they will be shifted; the important thing is to shift so that the red points and blue points differ by the shortest vector.



(a) Example of replacing $\boldsymbol{b}_1$ by $p\boldsymbol{b}_1$.

(b) Example of shifting by $m\boldsymbol{b}_1$.

Figure 7: Example of scaling with $p = 4$ and shifting with $m = 1$. The red points represent $t = |0\rangle$ (no shift) and the blue points represent $t = |1\rangle$ (shift).

In summary, the process of constructing the lattice superposition starts from the state $|+\rangle|\boldsymbol{Z}\rangle$ and applies to it the lattice function modified with a scale and shift,

$$f(t, \boldsymbol{x}) \coloneqq x_1(p\boldsymbol{b}_1) + t(m\boldsymbol{b}_1) + \sum_{i=2}^{n} x_i\boldsymbol{b}_i.$$

The result is stored in an ancillary register.

If the parameters have been chosen correctly then it will be highly likely that cubes are either empty or contain exactly two points (one red and one blue) whose difference is the shortest vector. As Figure 7 shows, the partition may split a pair of red and blue points and thus there will be cubes which contain just one point. However, a careful analysis shows that this will not occur too often.

Since with high probability after partitioning and collapsing the state will be in a superposition of two vectors whose difference encodes the shortest vector, we can use a solution to the two-point problem to find this difference, and thus the shortest vector itself, so that

$$\text{poly}(n)\text{-unique SVP} \leq \text{two-point problem.}$$

6.2 Choosing parameters

To complete the description of the reduction, we must also describe how to determine proper values for parameters such the cube length and shift amount. In fact, they can mostly be determined by brute force.

The cube length needs to be at least the length of the shortest vector and at most twice the length of the shortest vector. This requires $O(n)$ cases to check, since LLL finds an $O(2^n)$ approximation to the shortest vector. We first run the above procedure with the approximation length LLL finds as the cube length, and note the result. Then we divide the length by 2 and try it again, and repeat this $O(n)$ times. There will be one iteration which has cube length within a factor of 2 of the shortest vector, as required. We merely need to compare the results of each iteration and choose the shortest vector found, and possibly discard nonlattice vectors, depending on what our two-point problem solution outputs on illegal input.

Determining the proper shift amount m requires checking the cases $m \in [1, p)$. Shifting only matters mod p, since shifting by $p\boldsymbol{b}_1$ is equivalent to no shift. We can take $p \approx n^2$ so this still only requires poly(n) cases to check, and it is very likely that there will be one shift which reveals the shortest vector. The one case where it will not work is when the shortest vector's $\boldsymbol{b}_1$-coefficient (in the basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$) is divisible by p. Geometrically, this means that the shortest vector itself lies in the slices of the lattice formed by scaling up $\boldsymbol{b}_1$, so there is no way to shift the slice to reveal the shortest vector.

To guard against this, p should be chosen to be prime and we should apply the same scale-and-shift procedure on each $\boldsymbol{b}_i$ for $1 \leq i \leq n$; this adds an extra factor of n to the running time. At least one case must work, since if p divides a vector's $\boldsymbol{b}_i$-coefficient for each i then that vector divided by p is still in the lattice, but the shortest vector divided by p is not in the lattice.

Finally, there are other details which we have not dealt with here. For example, we should make sure that the shortest vector's $\boldsymbol{b}_i$-coefficients are small enough to appear in the $|\boldsymbol{Z}\rangle$ state (the coefficients will be absolutely bounded by 4^n assuming LLL has been run to reduce the basis). Also, the inputs to the DCP and two-point problem should be flexible enough to allow a possibility of some states not conforming to the input specifications, since even when all parameters have been chosen correctly we can only guarantee states are correct to high probability. This will have an effect on the poly(n) approximation factor achievable; the lower the approximation factor the more flexible the DCP and two-point problems need to be.

7 Conclusion

In summary, we have outlined the following reductions in a quantum context:

$$\begin{aligned}\text{poly}(n)\text{-unique SVP} &\leq \text{two-point problem}\\ &\leq \text{dihedral coset problem}\\ &\leq \text{dihedral HSP by coset sampling}\end{aligned}$$

That is, if we had a quantum computer and a way of efficiently solving the dihedral HSP by coset sampling then we could solve the poly(n)-unique SVP in poly(n) time with a high probability of success.

Although this result is of theoretical interest, it is currently unclear if it will ever be useful in practice. Even if quantum computers become available, the best known algorithm for solving the dihedral HSP [7] runs in time $2^{O(\sqrt{\log N})}$. Using the reduction as we have described would require $N = 2^{O(n^2)}$, and so this yields a $2^{O(n)}$ time algorithm for solving the poly(n)-unique SVP, like the best known classical algorithm [6]. However, there is still hope that a better algorithm for the dihedral HSP will be found and make this reduction more applicable.

References

[1] D. Aharonov and O. Regev. Lattice problems in NP $\cap$ coNP. *J. ACM*, 52:749–765, September 2005.

[2] M. Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 10–19, New York, NY, USA, 1998. ACM.

[3] A. M. Childs and W. van Dam. Quantum algorithms for algebraic problems. *Rev. Mod. Phys.*, 82:1–52, Jan 2010.

[4] H. Daudé, P. Flajolet, and B. Vallée. An average-case analysis of the gaussian algorithm for lattice reduction. *Comb. Probab. Comput.*, 6:397–433, December 1997.

[5] M. Ettinger, P. Høyer, and E. Knill. The quantum query complexity of the hidden subgroup problem is polynomial. *Inf. Process. Lett.*, 91:43–48, July 2004.

[6] R. Kumar and D. Sivakumar. On polynomial approximation to the shortest lattice vector length. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, SODA '01, pages 126–127, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[7] G. Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35:170–188, July 2005.

[8] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982. 10.1007/BF01457454.

[9] V. Lyubashevsky and D. Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 577–594, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM J. Comput.*, 30:2008–2035, December 2001.

[11] O. Regev. Quantum computation and lattice problems. *SIAM J. Comput.*, 33:738–760, March 2004. `http://arxiv.org/abs/cs/0304005`.

[12] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26:1484–1509, October 1997.

[13] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. 1981. Technical Report 8104.