Efficient Algorithms for Fixed-Precision Instances of Bin Packing and Euclidean TSP

David R. Karger Jacob Scott

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{karger,jhscott}@csail.mit.edu

April 7, 2008

Abstract

This paper presents new, polynomial time algorithms for bin packing and Euclidean TSP under *fixed precision*. In this model, integers are encoded as floating point numbers, each with a mantissa and an exponent. Thus, an interger i with $i = a_i 2^{t_i}$ has mantissa a_i and exponent t_i. This natural representation is the norm in real-world optimization. A set of integers I has *L-bit precision* if $\max_{i \in I} a_i < 2^L$. In this framework, we show an exact algorithm for bin packing and an FPTAS for Euclidean TSP which run in time $\text{poly}(n)$ and $\text{poly}(n + \log 1/\epsilon)$, respectively, when L is a fixed constant. Our algorithm is for the later problem is exact when distances are given by the L_1 norm. In contrast, both problems are strongly NP-Hard (and yield PTASs) when precision is unbounded. These algorithms serve as evidence to the signicance of the *class* of fixed precision polynomial time solvable problems. Taken together with algorithms for the knapsack and $Pm||C_{\max}$ problems introduced by Orlin et al. [10], we see that fixed precision defines a class incomparable to polynomial time approximation schemes, covering at least four distinct natural NP-hard problems.

1 INTRODUCTION

When faced with an NP-complete problem, algorithm designers must either settle for approximation algorithms, accept superpolynomial runtimes, or else identify natural restrictions of the given problem that are tractable. In the last category we find numerous *weakly NP-hard problems*, which have *pseudo-polynomial algorithms* that solve them in time polynomial in the problem size *and the magnitude of the numbers in the problem instance*, and are thus polynomial in the problem size when the number magnitudes are polynomial in the problem size. A more recent development is that of *fixed parameter tractibility*, another way of responding to the hardness of specific problem instances. In this paper, we explore *fixed precision tractability*. While pseudo-polynomial algorithms aim at problems whose numbers are *integers* of bounded size, we consider the case of *floating point* numbers that have bounded mantissas but arbitrary exponents. We consider such an exploration natural for a variety of reasons:

1. Floating point numbers are the norm in real-world optimization. They reflect the fact that practitioners seem to need effectively unlimited ranges for their numbers but recognize that they must tolerate limited precision. It is thus worth developing a complexity theory aligned with these types of inputs.

2. Conversely, from a complexity perspective, it remains unclear exactly *why* certain problems become easy in the pseudo-polynomial sense. Is it because the *magnitude* of those numbers is limited, or because their *precision* is? The pseudo-polynomial characterization bounds both, so it is not possible to distinguish.

3. As observed by Orlin et al. [10], a fixed-precision-tractable algorithm yields a *inverse approximation algorithm* for the problem, taking an arbitrary input and yielding an *optimal* solution to an instance in which the input numbers are perturbed by a small *relative* factor (namely, by rounding to fixed precision). This is arguably a more meaningful approximation than a PTAS or FPTAS that perturbs the value of the output.

Orlin et al., while introducing the notion of fixed precision tractibility, gave fixed precision algorithms for knapsack and three-partition. But this left open the question of how general fixed-precision tractability might be. Knapsack and partition are some of the "easiest" NP-complete problems, both exhibiting trivial fully polynomial approximation schemes. It was thus conceivable that fixed-precision tractability was a fluke arising from these problems' simplicity.

In this paper we bring evidence as to the significance of the fixed precision class, by showing the Bin Packing and Euclidean TSP are both fixed precision tractable. This is interesting because neither of the is known to have a fully polynomial approximation scheme. Orlin et al. showed that conversely there are problems with FPTASs that are not fixed precision tractable. We therefore see that fixed precision defines a class incomparable to FPTASs, covering at least 4 distinct natural NP-hard problems.

In order to demonstrate these results it is necessary for us to introduce some new, natural solution techniques. If all our input numbers have the same exponent, then we can concentrate on the mantissas of those numbers (which will be bounded integers) and apply techniques from pseudo-polynomial algorithms. The question is how to handle the varying exponents. We develop dynamic programs that scale through increasing values of the exponents, and arguments that once we have reached a certain exponent, numbers with much smaller exponents can be safely ignored. This would be trivial if we were seeking approximate solutions but takes some work as we are seeking exact solutions. We believe that our approach is a general one to developing fixed-precision algorithms.

2 BACKGROUND

In this section we give an overview of generally related previous work. Then we review the L-bit precision model to which our algorithms apply. Finally, we present relevant work on bin packing and Euclidean TSP, the problems for which we give algorithms in Sections 3 and 4.

Approximation algorithms, fixed parameter tractability, and inverse optimization

When faced with an NP-Hard optimization problem, the first option is most often the development of an approximation algorithm. If the best solution for an instance I of some minimization problem P has objective value $OPT(I)$, then a α-approximation algorithm for P is guaranteed for each instance I to return a solution with an objective value of no more than $\alpha \cdot OPT(I)$. Previous work in this area is vast, and we refer the interested reader to [12] for a wide overview of the field. The best one can hope for in this context is a *polynomial time approximation scheme*, a polynomial time algorithm which can provide a $(1+\epsilon)$-approximation for P, for any ϵ. If the running time of such an algorithm is polynomial in $1/\epsilon$, it is referred to as *fully* polynomial, and we say that P has an *FPTAS*. If the running time is polynomial only when ϵ is fixed, we say that P has an *PTAS*.

Another way in which NP-Hard problems can be approached is through fixed parameter tractability [6]. Here, a problem may have an algorithm that is exponential, but only in the size of a particular input parameter. Given an instance I and a parameter k, a problem P is fixed parameter tractable with respect to k if there is an (exact) algorithm for P with running time $O(f(|k|) \cdot |I|^c)$, where $|x|$ gives the length of x, f depends only on $|k|$, and c is a constant. Such problems can be tractable for small values of k. For example, Balasubramanian et al. [3] show it is possible to determine if a graph $G = (V, E)$ has a vertex cover of size k in $O\big(|V|k + (1.324718)^k k^2\big)$ time. The problem is thus tractable when the size of the vertex cover is given as a fixed parameter.

Closer to the L-bit precision regime we work in is inverse optimization, introduced by Ahuja and Orlin [1]. Consider a general minimization problem over some vector space X, $\min\{cx : x \in X\}$. If this problem is NP-Hard, an α-approximation algorithm can return a solution x^* such that $\frac{cx^* - cx}{cx} \leq \alpha$. Inverse optimization instead modifies the cost vector. That is, it searches for a solution x' and cost vector c' close to c (in, for example, L_∞ distance) so that $\min\{c'x : x \in X\} = x'$. The tightness of this approximation is then measured based on the distance from c' to c. This analysis can be more natural for some problems. In bin packing, for example, a standard approximation algorithm requires more bins than the optimal algorithm, while an inverse approximation algorithm requires an equal number of *larger* bins. All fixed precision tractable problems can be translated into this framework via rounding.

L-bit precision

Our algorithms apply to the L-bit precision model introduced by Orlin et al. in [10]. Problems in this model have their integer inputs represented in a nonstandard encoding. Each integer i is encoded as the unique pair (m, x) such that $i = m \cdot 2^x$, and m is odd. Following terminology for floating point numbers, we refer to m as the *mantissa* and x as the *exponent* of i. A problem instance has L-bit precision with respect to some subset $M \subseteq I$ of its integer inputs (encoded as above) if, $\forall (m, x) \in M, m < 2^L$. Each $(m, x) \in M$ can then be represented with $O(L + \log x)$ bits. An algorithms runtime is then computed in in relation to the total size T of its input, with numbers encoded appropriately. Two settings for L are considered. When L is constant, instances have *fixed precision*, and when $L = O(\log T)$, they have *logarithmic precision*.

Orlin et al. show polynomial time algorithms for fixed and logarithmic precision instances of the knapsack problem. They show the same for $Pm||C_{\max}$, the problem of scheduling jobs on m identical machines so as to minimize the maximum completion time. The precision here is with respect to item values for and job lengths, respectively. The algorithms work by reducing the problem to finding the shortest path on an appropriate graph. Such paths can take exponential time to find in the general case, but can be computed efficiently under L-bit precision. Our algorithms use similar techniques. We note that both of these problems are known to be NP-Complete, and also to support FPTASs.

The paper also demonstrates a polynomial time algorithm for the 3-partion problem when all numbers to be partitioned have fixed precision.[1] These instances can be reduced to integer programs with a linear number of constraints and a fixed number of variables, which in turn can be solved in linear time. Finally, a variant of the knapsack problem is shown that supports an FPTAS, but is NP-Complete for even 1-bit precision. The problem, *group knapsack*, allows items to have affinities for other items, so that item x can be placed in the knapsack iff item y is also packed. Thus, items with different exponents can express affinity for each other, essentially reconstructing items of arbitrary precision.

[1] Note that this is the problem of grouping $3n$ numbers into n 3-tuples with identical sums, not splitting them into three groups having the same sum, which is a special case of $Pm||C_{\max}$.

Bin packing

Bin packing is a classic problem in combinatorial optimization. The problem dates back to the 1970s, and there are an enormous number of variants. We restrict ourselves to the original, one dimensional, version. Here we are given a set of n items $\{x_1, x_2, \ldots x_n\}$, each with a size $s(x_i) \in [0, 1]$. We must pack each item into a unit capacity bin B_k without overflowing it – that is, $\forall k, \sum_{x_i \in B_k} s(x_i) \leq 1$. Our goal is to pack the items so that the number of bins used, K, is minimized.

Bin packing is NP-Complete, and extensive research has been done on approximation algorithms for the problem in various settings (see [5] for a survey). The culmination of this work is the well known asymptotic PTAS of de la Vega and Lueker [7], which handles large items with a combination of rounding and brute force search, and then packs small items into the first bin in which they fit. There is a key difference between this and all other worst case approximation algorithms for bin packing and the algorithm we show in Section 3. While both run in polynomial time, the former guarantee that they can pack any set of items into a number of bins that is almost optimal. Our algorithm guarantees that for a specific class of instances, namely those whose items have fixed-precision sizes, it can pack items into *exactly* the optimal number of bins.

Euclidean TSP

The Euclidean TSP is a special case of the well known Traveling Salesman Problem. The goal of the TSP is to find the minimum cost Hamiltonian cycle on a weighted graph $G = (V, E)$. That is, to find a minimum weight tour of the graph (starting from an arbitrary vertex) that visits every other vertex exactly once before returning to the starting vertex. Determining whether or not a Hamiltonian cycle exists in a graph is one of the original twenty one problems that Karp shows to be NP-Complete in [8], and it follows immediately that TSP is NP-Complete. Indeed, this reduction implies that no polynomial time algorithm can be approximate general instances TSP to within any polynomial factor unless P=NP.

One special case of the TSP that can be approximated is metric TSP. Here there is an edge between each pair of vertices, and edge weights are required to obey the triangle inequality. A simple 2-approximation for metric TSP can be shown using a tour that "shortcuts" the cycle given by doubling every edge in a minimum spanning tree and using the resulting Eulerian tour. The best known approximation ratio for metric TSP is 3/2, and comes from Christofides [4], who augments the minimum spanning tree more wisely than doubling each edge.

Eucildean TSP, finally, is then a further restriction of metric TSP. Vertices are taken to be points in Euclidean space (throughout this paper, we restrict ourselves to the plane), and there is an edge between each pair of vertices with weight equal to their Euclidean distance. Despite this restriction, the problem remains strongly NP-Hard [11]. Arora gives what is then the best possible result (unless $P = NP$), a PTAS, for Euclidean TSP in [2]. His algorithm combines dynamic programming with the use of a newly introduced data structure, the randomly shifted quadtree, which has since found wide use in computational geometry. Our algorithm for fixed-precision Euclidean TSP uses a similar combination of recursion and brute force, but takes advantage of structural properties implied by the nature of our narrower class of inputs.

3 BIN PACKING

In this section we present an $n^{O(4^L)}$ time algorithm for L-bit precision instances of bin packing.

Preliminaries

We consider a version of bin packing that is slightly modified from that discussed in Section 2. We take the following *decision* problem: given a set of n items $S = \{u_1, u_2, \ldots u_n\}$, each with a size $s(u_i) \in \mathbb{N}$, and integers K and C, can the set S be packed into K bins of capacity C? In the standard way, one can use an algorithm for this decision problem to solve its optimization analog by doing a binary search for the minimum feasible value of K. We find it convenient to consider S to be a set of item sizes, rather than items themselves, and refer to it in this way throughout the remainder of this section.

We consider the precision of an instance with respect to the set of items S. Thus, we consider an instance of bin packing to have L-bit precision if for all $u = m2^x \in S, m < 2^L$.

The algorithm

Here, in broad strokes, is a nondeterministic algorithm $\mathcal{A}$ (that is, $\mathcal{A}$ can magically make correct guesses) for this problem. $\mathcal{A}$ runs in rounds, and is only allowed to pack items that are currently in a special *reservoir* R. It also maintains bin occupancies in a set O, such that $o_k \in O$ gives the current occupancy of (that is, the the total size of the items in) the kth bin. Rounds proceed as follows. At the start of each round, $\mathcal{A}$ moves some items from X into R. Then, it guesses which items in R to pack, and which bins to pack them into. Finally, $\mathcal{A}$ updates the occupancy of bins which received items in the round, and performs certain bookkeeping on R. After the final round, $\mathcal{A}$ reports success iff R is empty and all bins have occupancy less than C.

We now describe these steps in detail. Afterwards, we demonstrate how they can be implemented efficiently. We initialize R to be empty, and all bins to have occupancy zero. In round x, we move the subset of items $S_x \subseteq S$ from S to R, where S_x contains all items in S with exponent x. In the packing step, for each size mantissa $m \in \{1, 3, \ldots 2^L - 1\}$, we guess which bins will be packed with an odd number of items of size $m2^x$. We place exactly one item of size $m2^x$ into each such bin, removing it from R and updating O.

We note the following property of the algorithm as presented so far.

Property 1 *After round x, for all m, the remaining number of items of size $m2^x$ that should be packed into any bin k is even.*

In light of Property 1, our bookkeeping procedure is to merge each pair of items of size $m2^x$ in R, creating a single item of size $m2^{x+1}$. Because we know that all of these items will be packed in pairs, and the size of the merged item is the sum of the sizes of its two constituents, this merging will not effect feasibility. By induction, this will mean that after round x there are no items of exponent less than x remaining in either X or R. After this step, $\mathcal{A}$ proceeds to round $x + 1$. The final round occurs when both S and R are empty, at which point all items have been packed. The following lemma bounds the number of rounds.

Lemma 2 *If a round x is skipped when both S_x and R are empty, then $\mathcal{A}$ runs at most n rounds.*

Analysis

We start by showing that the state of $\mathcal{A}$ can be represented compactly. This state can be described completely by the current round (which also dictates the items remaining in X), the items in R, and the bin occupancy O, so we represent such a state as $\{R, O, x\}$. We have already seen that g has at most n possible values. By design, every item in R at the start of round x has exponent exactly x. Thus, we can represent R by storing a count of the number of objects with each mantissa.

Expressed in this way, R has a total of $O(n^{2^{L-1}})$ possible values. The following lemma shows that O can also be represented compactly.

Lemma 3 *Consider a bin occupancy o_k at the start of round x. Then rounding o_k up to the next integer multiple of 2^x does not affect the feasible executions of algorithm A*

PROOF. Let $o_{k'}$ be equal to o_k rounded up to the next integer multiple of 2^x, and consider two bins with these occupancies. Let $e_k = C - o_k$ and $e_{k'} = C - o_{k'}$ be the space remaining in each bin. Then we have that

$$\left\lfloor \frac{e_k}{2^x} \right\rfloor = \left\lfloor \frac{e_{k'}}{2^x} \right\rfloor$$

We conclude the proof by noting that all items remaining to be packed (in both R and S) have exponent at least x, so that any set of remaining items can be packed into bin k if and only if it can be packed into bin k'.

□

At the start of round x, each bin is occupied by at most one item of each size that has exponent less than x. At this point, we thus have for each k, $o_k \leq \sum_{y=0}^{x} \sum_{w=1}^{L/2} (2w-1)2^y \leq 2^{2L+x}$. This fact coupled with Lemma 3 yields that O can also be represented by counting the number of bins of size $a \cdot 2^x$, for $a \in [0, 2^{2L})$. As there are at most n bins, the number of distinct O can then be bounded by $O(n^{4^L})$.

Now, consider a graph with nodes for each possible state $\{R, O, x\}$. Let us place a directed edge between $\{R, O, x\}$ and $\{R', O', x+1\}$ if it possible for $\mathcal{A}$ to begin round $x+1$ with reservoir R' and bin occupancy O' having started round g with reservoir R and occupancy O.[2] Then we can implement $\mathcal{A}$ by searching for a path between $\{\{\}, \{\}, 0\}$ and $\{\{\}, O^*, x_{\max}\}$ in this graph (where O^* satisfies all $o_k \leq C$ for all $o_k \in O^*$). The following lemma states that we can test if an edge should be inserted efficiently.

Lemma 4 *Given two algorithm states $S_1 = \{R, O, x\}$ and $S_2 = \{R', O', x+1\}$, it is possible to determine if $\mathcal{A}$ can transition from S_1 to S_2 in time $O\left(n \cdot n^{4 \cdot 4^L}\right)$.*

PROOF. Consider the following packing problem: given a set of n items Y, and a set of w bins with occupancies U and capacities Q (we allow bins with different capacities), is there a feasible packing of Y into the partially occupied U? We note that if all bin capacities are upper bounded by v, then this problem can be easily solved by dynamic programming in time $O(nk^{2v})$.

It should be clear that asking if a transition is possible between S_1 and S_2 can be represented as an instance of this problem. We first take $Y = R \cup S_g - R'$, noting that we may need to 'unmerge' some items in R' in order to subtract them, but that this is easy to do. Initial bin occupancies U are then given by O. Bin capacities Q are given by O', as this set describes the occupancy we would like bins to be at the start of round $x+1$.

We have one caveat here. As described above, the items in Y all have exponent 2^g, but the occupancies in O' are represented as multiples of 2^{g+1}. Thus for an occupancy $o'_k = (2i-1)2^{x+1}$, when we consider it as capacity $q_k \in Q$, we may need to increment it by 2^g. If w let $C[x]$ be the xth bit of our original capacity C, we can simply let $q_k = o'_k + C[x]2^x$.

Finally, we would like our capacities to have a small upper bound, so that the dynamic programming is efficient. Since all of the numbers here are multiples of 2^x, we scale them down

[2]We may also need some edges between states $\{R, O, x\}$ and $\{R', O', x'\}$, where $x' \neq x+1$. However, this only happens when rounds are skipped, and these edges are trivial to calculate.

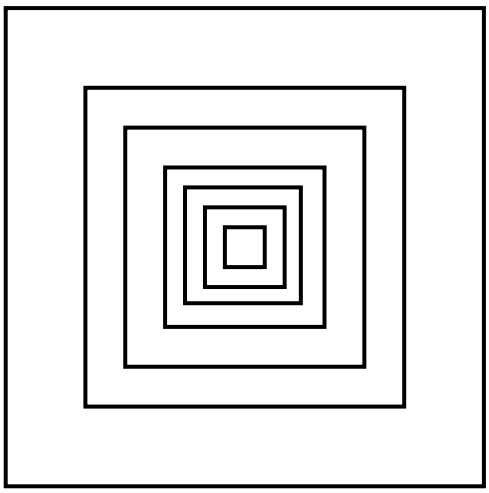

Figure 1: The initial squares $SQ_1, SQ_2, SQ_3, SQ_4, SQ_6, SQ_8$, and SQ_{12} on which points of a 2-bit precision Euclidean TSP instance can lie.

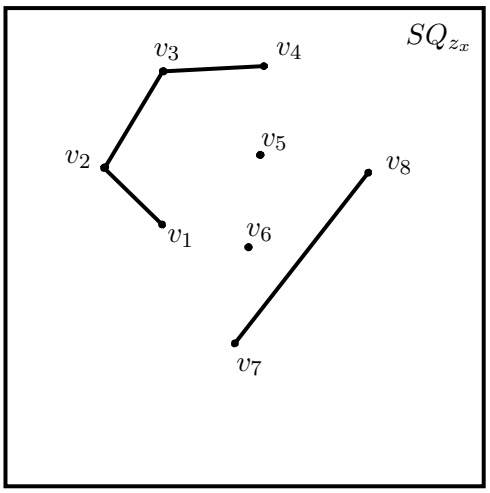

Figure 2: If the edges of W_x are given as above, then $D_x = \{0 : \{v_5, v_6\}, 1 : \{(v_1, v_4), (v_7, v_8)\}\}$. All remaining vertices (v_2 and v_3) then have degree two.

by this factor before running the dynamic programming. This gives us a maximum bin capacity of $v = 2^{2L+2}$. Thus we can test whether there should be an edge between S_1 and S_2 in time $O\Big(nk^{2^{2L+2}}\Big) = O\Big(n \cdot n^{4 \cdot 4^L}\Big)$. □

We can now state our main theorem.

Theorem 5 *There is an $n^{O(4^L)}$ time algorithm for L-bit instances of bin packing.*

PROOF. The Running time of $\mathcal{A}$ is dominated by constructing the edges in the state space graph. There are $n^{O(4^L)}$ of these edges, and each takes $n^{O(4^L)}$ time to test for inclusion.

□

4 EUCLIDEAN TSP

In this section we present an FPTAS for L-bit instances of Euclidean TSP which provides an $(1+\epsilon)$-approximation in time $n^{O(4^L)} \cdot \log 1/\epsilon$.

Preliminaries

The version of the Euclidean TSP we consider is the following. Given a set of vertices $V = \{(x, y)\}$ on the plane, what is the shortest tour that visits each point exactly once, when edge lengths are given by the L_2 distance between two endpoints? We consider the precision of an instance with respect to the coordinates of its points in following manner. We say an instance of Euclidean TSP has precision L if for every $((m_x, x_x), (m_y, x_y)) \in V$, $\max\{m_x, m_y\} < 2^L$, and both m_x and m_y are odd. Equivalently, all points in V must lie along a series of axis-aligned squares SQ_z centered on the origin, with side length $z = i \cdot 2^j$, and all $i \leq 2^L$. See Figure 1 for an example.

While we would like to show an exact algorithm for fixed-precision instances of Euclidean TSP, our desires are frustrated by the fact that the problem is not known to be in NP. Specifically, there are no known polynomial time algorithms for comparing the magnitude of sums of square roots.

Thus, given two tours with square-root edge lengths, we have no efficient way to decide which is shorter. What we do instead is to provide an algorithm that achieves a $(1+\epsilon)$-approximation with a runtime dependence on ϵ that is only $O(\log 1/\epsilon)$. We do this by rounding edge lengths to the nearest multiple of $2^{\lfloor \log \epsilon \rfloor}$. Going forward we assume that this rounding has been done.

For simplicity, we rescale the edge lengths and present our algorithm an as exact algorithm taking integral edge lengths in addition to the set of vertices V. We also give our bounds assuming that we can do operations on edge lengths in constant time. At the end of the section, we reintroduce the approximate nature of our result and the $O(\log 1/\epsilon)$ cost of doing operations on edge lengths. Note that this problem with radicals does not arise when edge lengths are given by the L_1 norm, and in this case the above rounding and approximation is unnecessary.

The algorithm

Again, we begin by giving an overview an algorithm $\mathcal{B}$ which solves the problem, and follow with an analysis of its efficiency. Like $\mathcal{A}$ discussed in the previous section, $\mathcal{B}$ is nondeterministic and runs in rounds. In each round, $\mathcal{B}$ considers points lying on a single SQ_z (as described above), in increasing order of side length. The algorithm maintains an edge set W, and for each vertex v considered in a round, the algorithm guesses all edges in an optimal tour connecting v to previously processed vertices. After the last round, when all points lying on the outermost square are processed, W will contain edges that form an optimal tour of V. The intuition behind our algorithm is similar to that of Arora [2]. Because points lie along a series of SQ_z, and z is fixed precision, we expect few edges to cross these squares, as cheaper tours can be had by 'shortcutting' such in-out edges. This allows a dynamic programming formulation that keeps as its state the entry and exit points of these squares, and gathers the (rare) crossing edges.

More formally, let T be an edge set that forms an optimal tour of V. We define $T_z \subseteq T$ to be the subset of edges in T with both endpoints on or inside SQ_z. Assume that the points in V like on a series of squares with side length $z_1 < z_2 < \ldots < z_{\max}$ (clearly, there are at most n such squares). Then in round x, $\mathcal{B}$ will add the edges in $T_{z_x} - T_{x_{z-1}}$ to W.

We use the following construction to implement $\mathcal{B}$. We define $D_x = d(W_x)$ to be an augmented degree sequence of the vertices in V induced by the edges present in W at the start of round x. We note that T_{z_x} is composed of disjoint paths; our augmentation is to also record in D_x the endpoints of these paths (in pairs). See Figure 2 for an example. This information is necessary to check for the presence of premature cycles, as we will see later. We then create a graph G_{deg} with all possible D_x as vertices. We add a directed *transition edge* with weight w between D_x and D_{x+1} if there is an edge set S_x with cumulative weight w such that $D_x + d(S_x) = D_{x+1}$. We further require that adding S_x to D_x does not create any cycles, save when $x+1 = x_{\max}$. For each pair (D_x, D_{x+1}), we keep only the lightest such transition edge, and annotate it with the set S_x. From this graph, we can recover T by taking the union of the S_x attached to a shortest path between $D_0 = (0, 0, \ldots, 0)$ and $D_{x_{\max}} = (2, 2, \ldots, 2)$.

Analysis

We now show that the number of vertices in G_{deg} is bounded, and that we can efficiently construct its edge set. We begin with the following lemma, whose proof appears in Appendix A.

Lemma 6 *Consider a L-bit precision set of vertices V on the plane. Then for all z, any optimal tour of V will cross SQ_z at most $O(4^L)$ times.*

We now consider the number of distinct feasible values of D_x, for a fixed x. Clearly, all points outside of SQ_z have degree zero. By Lemma 6, we know that at most $O(4^L)$ of the remaining

vertices have a degree that is less than two. Thus D_x can be represented compactly by listing the vertices in this set, their degree, and all path endpoints (that is, pairings of degree one vertices). This gives a bound of $n^{O(4^L)}$ on the number of distinct values.

What remains is to demonstrate that for each pair of degree sequences (D_x, D_{x+1}), the minimum weight edge set S_x whose addition to D_x yields D_{x+1}, and no cycles, can be computed in polynomial time. We note that all edges in S_x have at least one endpoint on SQ_{z_x}. We refer to edges with exactly one endpoint on SQ_{z_x} as *single* edges, and those with both endpoints on SQ_{z_x} as *double* edges. For each point v strictly inside SQ_{z_x}, we must add $D_{x+1}(v) - D_x(v)$ single edges incident to v. We enumerate all $n^{O(4^L)}$ possible non cycle inducing sets S_x^s of single edges, and claim that given such a set, the minimum weight non cycle inducing set of double edges S_x^d can be computed in polynomial time.

Lemma 7 *Let D_x and D_{x+1} be degree sequences as described above. Let X_x^s be a non cycle inducing set of single edges. Then the minimum weight non cycle inducing set of double edges S_x^d satisfying $D_x + d(S_x^s) + d(S_x^d) = D_{x+1}$ can be constructed in time $n^{O(4^L)}$.*

PROOF. All possible S_x^d can be enumerated as follows. Consider any point v along SQ_{z_x} with $D_{x+1}[v] = D_x[v] + d(S_x^s)[v] + 1$. The edges in S_x^d which v can reach form a path consisting of connected segments of the perimeter of SQ_{z_x}. This path cannot cross itself (for if this is the case, then there is a less expensive tour that does not include this crossing), so there can be at most one segment on each side of SQ_{z_x}. We can enumerate the starting and ending vertex of each segment, giving the path starting from a single vertex $O(n^8)$ possibilities. By Lemma 6, there are at most $n^{O(4^L)}$ possible starting vertices. Finally, for each set of paths, we spend $O(n)$ time to check that no cycles have been created and that the degree sequence induced is correct. □

We can now give our main lemma.

Lemma 8 *There is an $n^{O(4^L)}$ time exact algorithm for L-bit precision instances of Euclidean TSP with integral edge lengths.*

PROOF. Our runtime is dominated by the time to construct G_{deg}. By Lemma 6 we have at most $n^{O(4^L)}$ possible edges in G_{deg}. By Lemma 7, the cost to check each edge is $n^{O(4^L)}$. □

We conclude by relaxing our constraints on edge lengths.

Theorem 9 *There is an FPTAS for L-bit instances of Euclidean TSP that achieves an $(1+\epsilon)$-approximation in time $n^{O(4^L)} \log 1/\epsilon$.*

5 CONCLUSION

We have given algorithms demonstrating that two natural problems, bin-packing and Euclidean TSP, are polynomial time solvable when their input numbers are given in fixed precision. We have therefore offered further evidence that the class of fixed-precision tractable NP-hard problems is a meaningful class worthy of further study. Several additional observations arise regarding this class:

1. As argued in the introduction, fixed-precision tractability seems *orthogonal* to psedu-polynomial tractability. There are problems such as Knapsack that have both pseudo-polynomial and fixed precision algorithms. Others, such as group knapsack, have pseudo-polynomial algorithms but do not have fixed-precision algorithms. Conversely, while bin-packing and Euclidean TSP do not have fully polynomial approximaiton schemes, our algorithms show them

to be fixed precision tractable, but only when the number of mantissa bits is *fixed* (note that if we could handle a logarithmic number of bits of precision, we would have a pseudo-polynomial algorithm).

2. Just how large is this class of fixed-precision tractable problems? Fixed precision tractability seems fragile, in a way the pseudo-polynomial algorithsm aren't. Add some polynomial size integers and you get another; Add two low-precision numbers with different exponents and you suddenly have a high precision number. Does this fragility mean that few problems are tractable in this way?

3. Work by Korte and Schrader [9] shows a tight connection between pseudo polynomial algorithms and fullay polynomial approximation schemes—problems that have one tend to have the other, in a formalizable sense. There is a similar coupling between algorithms with (non-fully) polynomial approximation schemes, and algorithms that can be solved by brute force enumeration when they are limited to a few input "types." Is there a similar connection betwee fixed-precision tractability and (say) inverse approximation as defined by Orlin? One direction is obvious (as is the case for FPTAS and pseudo-polynomial algorithms) but the other is not clear.

References

[1] R.K. Ahuja and J.B. Orlin. Inverse Optimization. *Operations Research*, 49(5):771–783, 2001.

[2] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[3] R. Balasubramanian, M.R. Fellows, and V. Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[4] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. *Symposium on new directions and recent results in algorithms and complexity*, page 441, 1976.

[5] EG Coffman Jr, MR Garey, and DS Johnson. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[6] R.G. Downey and M.R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[7] W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[8] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 43:85–103, 1972.

[9] B. Korte and R. Schrader. On the Existence of fast Approximation Schemes. 1982.

[10] J.B. Orlin, A.S. Schulz, and S. Sengupta. ϵ-optimization schemes and L-bit precision: alternative perspectives in combinatorial optimization (extended abstract). *ACM Symposium on Theory of Computing*, pages 565–572, 2000.

[11] C.H. Papadimitriou. Euclidean TSP is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

[12] V.V. Vazirani. *Approximation Algorithms*. Springer, 2001.

A GEOMETRIC PROOFS

Lemma 6 *Consider a L-bit precision set of vertices V on the plane. Then for all z, any optimal tour of V will cross SQ_z at most $O(4^L)$ times.*

PROOF. Each edge that crosses SQ_z has one edge strictly inside SQ_z and one edge strictly outside SQ_z. Refer to this set of edges as Q. We break the edges into two distinct subsets Q_1 and Q_2 such that $Q_1 \cup Q_2 = Q$. Edges in Q_1 have an endpoint on SQ_{z_a}, $z < z_a \leq 3z$, and edges in Q_2 have an endpoint on SQ_{z_b}, $3z < z_b$. In what follows we show that $|Q_1| \leq 9 \cdot 4^L$ and $|Q_2| \leq 192$, and the claim follows immediately.

We can construct a tour that replaces edges in Q_1 with (shortcutting) the construction shown in Figure 3. Each $SQ_{z'}$ is toured via edges with total weight $6z'$, while each edge in Q_1 with an endpoint on $SQ_{z'}$ has length at least $z'/2^L$. Since there are at most $3 \cdot 2^{L-1}$ squares between SQ_z and SQ_{3z}, this gives an upper bound of $9 \cdot 4^L$ on the size of Q_1.

Now, assume that $|Q_2| > 192$. Then there must be a segment along the perimeter of SQ_{3z} of length $d/2$ that is intersected by 8 edges in Z_2. Then there are distinct vertices $H = \{a, b, c, d, e, f\}$ as in Figure 4 so that $\{(a, e)(a, f), (b, d), (b, f), (c, d), (c, e)\} \cap Q_2 = \{\}$. Thus, there must be $v_1 \in (a, b, c)$ and $v_2 \in (d, e, f)$ such that there is a path from v_1 to v_2 is neither direct nor passes through any other vertices in H. This in turn induces a graph like that found in Figure 5. Here, we can replace the edges $\{(a, d), (b, e)\}$ with edges $\{(a, b), (d, t_1), (t_1, t_2), (t_2, e)\}$ for a shorter tour, contradicting the assertion that $|Q_2| > 192$. □

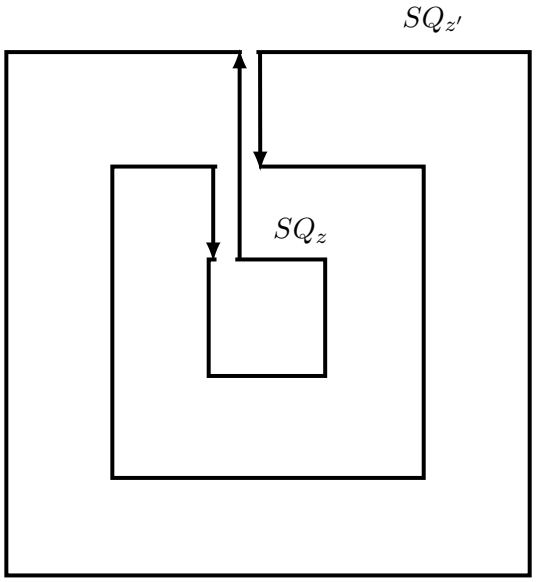

Figure 3: This is an example 'brute force' tour that will cover all points in Q_1. We take $z' \leq 3z$. Note that the horizontal distance between the vertical edges can be arbitrarily small.

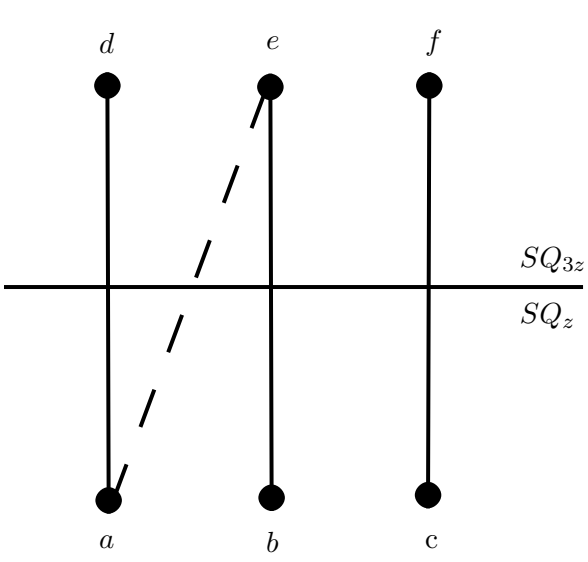

Figure 4: The solid lines between points exist in Q_2. The dashed line indicates that there is a path from a to e that does not go through b, c, d, or f. This edge is guaranteed (modulo permutations on vertex labels) to exist, because there are no other diagonal edges between these vertices in Q_2, by definition.

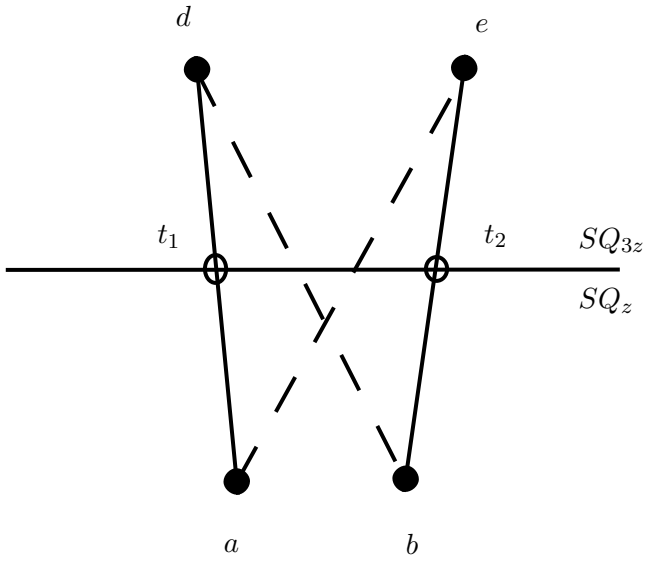

Figure 5: We have that $\text{len}(\overline{ab}) \leq \sqrt{2}z$ and $\text{len}(\overline{t_1 t_2}) \leq z/2$. Also, both $\text{len}(\overline{t_1 a})$ and $\text{len}(\overline{t_2 b})$ are at least than $2z$. Thus we can replace $\{(a,d),(b,d)\}$ with $\{(a,b),(d,t_1),(t_1,t_2),(t_2,e)\}$, and end up with a shorter tour without adding any cycles.