Theory Comput Syst (2015) 56:372–393
DOI 10.1007/s00224-014-9550-z

Hardness of Approximation for Knapsack Problems

Harry Buhrman · Bruno Loff · Leen Torenvliet

Published online: 15 June 2014

Abstract We show various hardness results for knapsack and related problems; in particular we will show that unless the Exponential-Time Hypothesis is false, subset-sum cannot be approximated any better than with an FPTAS. We also provide new unconditional lower bounds for approximating knapsack in Ketan Mulmuley's parallel PRAM model. Furthermore, we give a simple new algorithm for approximating knapsack and subset-sum, that can be adapted to work for small space, or in small parallel time.

Keywords Computational complexity · Knapsack problem · Subset-sum · Exponential-time hypothesis · PRAM without bit operations · Algebraic-circuit lower-bounds · Hardness of approximation

1 Introduction

The Knapsack problem is a natural example of an NP-complete optimization problem which nevertheless has a fully-polynomial time approximation scheme (FPTAS). However, there is no *a priori* reason to think that FPTAS would be the best one could hope for. In fact, in order to prove NP-hardness of knapsack we require the problem

H. Buhrman · B. Loff
CWI, Amsterdam, Netherlands

H. Buhrman
e-mail: harry.buhrman@cwi.nl

B. Loff
e-mail: bruno.loff@gmail.com

H. Buhrman · L. Torenvliet (✉)
ILLC, University of Amsterdam, Amsterdam, the Netherlands
e-mail: L.Torenvliet@uva.nl

to be solved exactly, so in principle there could exist polynomial-time approximation algorithms for knapsack with an approximation ratio strictly closer to 1 than inverse-polynomial.

We begin this work by giving evidence that inverse-polynomial is as good an approximation as we are likely to get. For example, we obtain the following:

Theorem 1 *If there is a polynomial-time algorithm to approximate knapsack with* $1 + 1/n^{\omega(1)}$ *approximation ratio, then we can decide the satisfiability of* n*-variable* NC *circuits of size* $\omega(n)$ *in time* $2^{o(n)}$.

In a way, this is a refinement of the NP-hardness of the knapsack problem. The reduction actually offers a robust trade-off, in that algorithms with a successively better approximation ratio can be used to simulate successively larger non-deterministic NC-computations.

A reduction from 3SAT instead of NC circuit satisfiability was independently discovered by Cygan et al. [7]. Their proof is more contrived, as it is the result of several intermediate reductions, and their result is slightly weaker, though some of the reduction gadgets they use have a similar flavor to our own. Our proof, however, is simple, short and direct.

From Theorem 1 and the sparsification lemma of [12], we get, in particular, a $2^{o(n)}$-time algorithm for 3SAT from the assumptions of Theorem 1. This would contradicting the Exponential–Time Hypothesis [11]. This is the first hardness-of-approximation result of its kind, since hardness of approximation has been shown for every other approximation ratio (cf. Section 4). This also follows as a consequence of [7], although in that paper the result was not clearly formulated.

The techniques in these proofs will allow us to prove a new unconditional lower bound for solving knapsack in parallel, using Mulmuley's parametric complexity technique [15]. Our reduction techniques can be used to show that the knapsack problem has high parametric complexity, even when the bit-lengths of the inputs are small. It will follow that it is impossible to approximate knapsack within a factor of $1 + \varepsilon$ in time $o\left((\log \frac{1}{\varepsilon})^{\frac{1}{4}}\right)$, and using $2^{o\left((\log \frac{1}{\varepsilon})^{\frac{1}{4}}\right)}$ processors, in Mulmuley's PRAM model without bit operations.

To complement these hardness results, we provide a simple, and to our knowledge new, algorithm for approximating knapsack and subset-sum in parallel, obtained through a mix of two standard results: the meet-in-the-middle algorithm for knapsack [10], and the PSPACE completeness of alternating time [20]. We are able to approximate the knapsack problem up to approximation ratio $1 + \varepsilon$ in space $\log \frac{1}{\varepsilon} \cdot \log n$. A linear arithmetic PRAM [see 15] can compute it in time $\log n$ using $(\frac{1}{\varepsilon})^{\log n}$ processors. Alternatively, it may be implemented by an $O(\log n)$-depth AC circuit of size $\left(\frac{1}{\varepsilon}\right)^{\log n}$.

After some preliminaries in Section 2, we study in Section 3 both old and new approximation algorithms for knapsack and related problems. In Section 4 we show how to reduce satisfiability problems to subset-sum with small bit-weights, and then proceed with the lower bounds for Mulmuley's model in Section 5.

2 Preliminaries

For a given natural number n, $[n]$ denotes the set $\{1, \ldots, n\}$, and S_n denotes the group of permutations of n letters. For $\sigma_1, \sigma_2 \in \mathsf{S}_n$, the permutation $\sigma_1\sigma_2$ is defined by $(\sigma_1\sigma_2)(i) = \sigma_2(\sigma_1(i))$ for all $i \in [n]$. For a given predicate P, we will use the notation $[P?]$ to denote 0 if P is false, and 1 if P is true.

2.1 Circuit Classes and Bounded Non-Determinism

A *width-5 permutation branching program* P over k input bits $y_1, \ldots, y_k$ is a set (of so-called *instructions*) $\{(j, z_j, \alpha_j, \beta_j)\}_{j=1}^{S}$, with $z_j \in [k]$, $\alpha_j, \beta_j \in \mathsf{S}_5$. For a given input $\bar{y} \in \{0,1\}^k$, the evaluation of the program $P(\bar{y}) \in \mathsf{S}_5$ is equal to $\gamma_1\gamma_2 \ldots \gamma_S$, where γ_j equals α_j if $y_{z_j} = 0$ and equals β_j if $y_{z_j} = 1$. We usually write $P(\bar{y}) = 1$ when $P(\bar{y}) = e$ (the identity) and write $P(\bar{y}) = 0$ to mean $P(\bar{y}) \neq e$.

We will let NC–$\mathsf{SAT}(S, k)$ denote the set of circuits C composed of k boolean input gates and S fan-in-2 NAND gates, such that $C(\bar{y}) = 1$ for some choice of $\bar{y} \in \{0,1\}^k$. We use $\exists\mathsf{NC}_1(k)$ to denote the class of sets accepted by log-depth fan-in-2 uniformly generated circuits of polynomial size which make use of k non-deterministic bits — i.e., a set A is in $\exists\mathsf{NC}_1(k)$ if there is a uniform family of polysize boolean formulae F_n such that $x \in A$ iff $F_{|x|}(x, y) = 1$ for some $y \in \{0,1\}^k$.[1] Finally, we let NC_1–$\mathsf{SAT}(S, k)$ denote the set of satisfiable size-S formulae with k boolean variables. Notice that NC_1–$\mathsf{SAT}(S, k)$ denotes a set and $\exists\mathsf{NC}_1(k)$ denotes a class of sets for which NC_1–$\mathsf{SAT}(poly(n), k)$ is a complete problem.

2.2 Mulmuley's Parallel Model of Computation

Mulmuley's model is a semi-algebraic model of parallel computation. The inputs are usually thought of as having a binary part, giving the combinatorial structure (for instance, the adjacency matrix of a graph), and an integer part, giving numerical data about this structure (such as the weights of the edges of said graph). The model treats these two types of data differently with respect to pointer jumping, which makes its full description more involved than it needs to be for our purpose. In fact, the knapsack problem has no combinatorial structure, and so we can look at a simpler form of Mulmuley's model, and leave the full details to [15].

A *(numerical) problem* in this setting is a family A_n of subsets of Z^n. The model of computation is the *arithmetic PRAM*, a device composed of a certain number of registers and processors. At the beginning of the computation, a tuple $\bar{a} = (a_1, \ldots, a_n)$ is given as input by placing each a_i in register i. Each processor is given a numbered sequence of instructions, each of which is one of the following:

- $w = u \circ v$, where w is a register and u and v are either registers or constants, and $\circ$ is one of $+, -, \times$;
- If register i is greater than zero, then go to instruction ℓ, or else go to instruction ℓ'.

[1] We may assume that such formula have logarithmic-depth [4, 5].

At every time-step, each processor executes its current instruction simultaneously; we assume that concurrent reads and writes are OK, and are handled, for instance, by ordering the processors according to some priority. All instructions have unit cost, which actually means that the model can handle very large numbers. The machine eventually halts, for instance by letting processor 1 execute a special instruction, and we say that $\bar{a}$ was accepted if register 1 holds a zero (meaning $\bar{a} \in A$), and was rejected otherwise.

It should be noted that division is not part of the basic instructions. As a result of this, it can be proven that the model cannot know the least significant bit of an n-bit register in $o(\sqrt{n})$ steps and using $2^{o(\sqrt{n})}$ processors; because of this, the model is sometimes called *PRAM without bit operations*. This is an algebraic model, in that the contents of the registers at any given time-step are essentially polynomials in the numerical parameters given as the input.[2]

Now we may define *Ketan's class* $\mathsf{KC}(P, T)$ as the class of problems A which can be decided by such a device using P processors and T time. An algebraic algorithm will have P and T depend only on n, whereas a semi-algebraic algorithm will have them depend on the bit-length of the integers a_i. This model is quite powerful, and in fact it is capable of implementing every parallel algorithm that the we know of. Nevertheless, Mulmuley [15] shows that the decision version of maximum flow is not in $\mathsf{KC}(2^{\frac{\sqrt{n}}{a}}, \frac{\sqrt{n}}{a})$, for some constant a, even when the bit-lengths of the flow capacities are at most $O(n^2)$. The result extends to any numerical problem, such as the traveling salesman problem, to which max-flow reduces by a parallel algebraic reduction. But subset-sum is not known to be such a problem. [3] In the paper [19], it was shown that subset-sum is not in $\mathsf{KC}(2^{\frac{\sqrt{n}}{a}}, \frac{\sqrt{n}}{a})$, but using a different technique than Mulmuley's, which gives no limit on the bit-length of the inputs among which a hard instance will be found (we could say that the lower bound was proven for algebraic algorithms, and not for semi-algebraic algorithms). In Section 5 we prove that subset-sum is not in $\mathsf{KC}(2^{\frac{1}{a}n^{1/4}}, \frac{1}{a}n^{1/4})$, even for bit-length of the inputs bounded by $O(n^2)$.

2.3 Knapsack and Generalizations

As part of our study we will include a variant of the knapsack problem which we call the *symmetric knapsack problem*, and which is obtained by adding additional

[2] More precisely, the contents of a register at any given time-step is a polynomial in these parameters, but precisely which polynomial depends on the outcomes of previous conditionals (greater-than-zero instructions).

[3] While the max-flow problem, like any problem in P, reduces to the knapsack problem, it does not seem to be possible to have such a reduction in a way that preserves the bit-length of the numerical parameters (the flow capacities). So the lower bound for max-flow does not imply a lower bound for the knapsack problem.

Intuitively, there is good reason to believe that such a reduction does not exist. The subset-sum problem is a "purely numerical" problem, in the sense that it has no combinatorial structure whatsoever; hence the combinatorial structure of a max-flow problem (the graph itself) will need to be somehow encoded in the numbers of the subset-sum problem we would wish to reduce to; but this graph can be much larger than the bit-lengths of the edge weights.

constraints to the knapsack instance, in the form of equations over the symmetric group. It will be seen that the symmetric knapsack problem is significantly harder to solve than the knapsack problem. Our reason to study this problem here is two-fold. First of all, not many hardness results are known for problems in non-commutative algebra [cf. 16], and symmetric subset-sum is a natural problem in that setting (see Observation 1 below). Secondly, our results show that the addition of even the simplest combinatorial structure to the knapsack problem already makes it significantly harder.

In the definition below, it helps to think of $\{1, \ldots, n\}$ as a set of items, the $v(i)$ as values, $w(i)$ as weights, and $\sigma(i)$ as patterns of these items, and that our task is to fit a set of items of maximum total value into a knapsack that can carry W weight, with the added restriction that the items we pick must orderly fit together according to a certain pattern Σ.

Definition 1 The *0–1 symmetric knapsack problem*, which we denote with $\textsc{SymK}(n, a, b)$ for given integer parameters $n, a, b \geq 0$, is defined as the following optimization problem: We are given $v : [n] \to [2^a]$, $w : [n] \to [2^b]$, $\sigma : [n] \to \mathsf{S}_5$, as well as $W \in [2^b]$, $\Sigma \in \mathsf{S}_5$, and then, going over all subsets $I \subseteq [n]$, we wish to

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{i \in I} v(i) \\ \text{s.t.} \quad & \textstyle\sum_{i \in I} w(i) \leq W \\ & \textstyle\bigodot_{i \in I} \sigma(i) = \Sigma \end{aligned}$$

The notation $\odot_{i \in I} \sigma(i)$ means the product of the elements $\sigma(i)$ for $i \in I$, *in ascending order of* i, and for completion we let $\odot_{i \in \emptyset} \sigma(i)$ denote e, the identity permutation. Note that the order is important since S_5 is not commutative.

Throughout, we will assume that every $w(i) \leq W$, since of course larger items do not fit into the knapsack, and can be ignored.

The symmetric subset-sum problem, $\textsc{SymSS}(n, b)$, is the problem of deciding, when given an instance $X = (v, w, \sigma, W, \Sigma)$ of $\textsc{SymK}(n, b, b)$ with $v = w$, if there exists a feasible solution matching the bound W:

Definition 2 The symmetric subset-sum problem $\textsc{SymSS}(n, b)$ is defined as the following decision problem: We are given $w : [n] \to [2^b]$, $\sigma : [n] \to \mathsf{S}_5$, $W \in [2^b]$, and $\Sigma \in \mathsf{S}_5$, and we wish to decide if there exists an $I \subseteq [n]$ such that $\sum_{i \in I} w(i) = W$ and $\odot_{i \in I} \sigma(i) = \Sigma$.

Again it will be convenient to think of $[n]$ as a set of items, $w(i)$ as weights, and $\sigma(i)$ as patterns. Intuitively, our goal is now to find a choice of items matching a specific weight with a specific pattern. Note that subset-sum is defined as a decision problem whereas knapsack is defined as an optimization problem. If the we restrict σ_i to be the identity of S_5, we have the original knapsack and subset-sum problems:

Definition 3 The 0–1 knapsack problem $\mathrm{K}(n, a, b)$ is $\textsc{SymK}(n, a, b)$ restricted to the case when σ_i, Σ are all e. The 0–1 subset-sum problem $\mathrm{SS}(n, b)$ is defined in the same way.

Observation 1 *The symmetric subset-sum problem, as defined here, is a natural problem in the setting of non-commutative algebra, because it is a special case of the subset-sum problem over general groups (studied in* [16]*). Indeed, to an instance* (w, σ, W, Σ) *of* $\textsc{SymSS}(n, b)$ *corresponds an instance of the subset-sum problem over the symmetric group, as follows. Let* $M = p_1 \dots p_m$ *be the smallest product of the first m primes such that* $M > n2^b$*, and let* $\phi(w, \sigma) = (w \mod p_1, \dots, w \mod p_m, \sigma)$ *be the canonical (additive) group isomorphism from* $\mathsf{Z}_M \times \mathsf{S}_5$ *to* $\mathsf{Z}_{p_1} \times \cdots \times \mathsf{Z}_{p_m} \times \mathsf{S}_5$*, itself a subgroup of* S_B*, with* $B = p_1 + \ldots + p_m + 5$.

Then there exists an $I \subseteq [n]$ *such that* $\sum_{i \in I} w(i) = W$ *and* $\odot_{i \in I} \sigma(i) = \Sigma$*, if and only if there exists an* $I \subseteq [n]$ *such that* $\odot_{i \in I} \phi(w(i), \sigma(i)) = \phi(W, \Sigma)$*, and this latter problem is an instance of subset-sum over* S_B.

From $m \leq b \log n$ *and* [1, Section 2.7]*, we get* $B \leq (b \log n)^2 \ln(b \log n) + 5$.

3 Approximation Algorithms: Old and New

Given an instance $X = (v, w, \sigma, W, \Sigma)$ of symmetric knapsack, it will have a unique optimum value m^*, corresponding to one (among possibly many) optimal solution I^*.

The goal of an approximation algorithm for this problem is to estimate the (unknown) value of m^*. The algorithm is said to achieve approximation ratio α if it always outputs a value m such that $m^* \leq \alpha m$.

3.1 Equivalence of Approximation and Exact Solution for Small Weights

A classical observation in the study of knapsack problems is that, modulo a multiplicative factor of n, solving an n-item knapsack approximately to ratio $1 + \varepsilon$ is equivalent to solving instances of knapsack having integer values bounded by $\frac{1}{\varepsilon}$. This holds the exact same way for the symmetric variant.

Theorem 2 *Let* $\varepsilon > 0$*; then each problem in the following list reduces to the one below:*

1. *Solving* $\textsc{SymK}(n, a, b)$ *with approximation ratio* $1 + \varepsilon$*;*
2. *Solving* $\textsc{SymK}(n, \log n + \log(1 + \frac{1}{\varepsilon}), b)$ *exactly;*
3. *Solving* $\textsc{SymK}(n, a, b)$ *with approximation ratio* $1 + \frac{\varepsilon}{n^2}$.

This theorem will allow us to prove both upper and lower bounds for approximation algorithms by ignoring the approximation ratio altogether, and working with instances having small values $v(i)$. The proof is standard [21], and we write it here for completeness.

Proof (1 reduces to 2) Let $X = (v, w, \sigma, W, \Sigma)$ be an instance of $\textsc{SymK}(n, a, b)$, and let $V = \max_i v(i)$. Set $t = \log(\frac{\varepsilon}{1+\varepsilon} \cdot \frac{V}{n})$, and make $v'(i) = \lfloor v(i)/2^t \rfloor$. Define X' to be X with v replaced by v'. Our approximate solution I will be an optimal solution of X'. Note that X' is an instance of $\textsc{SymK}(n, a', b)$ with $a' = \log n + \log\left(1 + \frac{1}{\varepsilon}\right)$,

and that furthermore since w, σ, W, Σ remain unchanged, then any I feasible for X' will be feasible for X also (and vice-versa).

To conclude, we prove an upper bound on the approximation ratio. So let $m = \sum_{i \in I} v(i)$ be the value of I (i.e., m' is the X-value of the X'-optimal solution), and m^* the value of an optimal solution I^* of X. Since X' is X with the t least significant bits truncated, it is easy to see that $m^* - m \leq n2^t$. Since, furthermore, $V \leq m^*$, we conclude that

$$\frac{m^* - m}{m} \leq \frac{n2^t}{V}$$

which implies, by simple algebraic manipulation and our choice of t, that

$$m^* \leq \left(\frac{V}{V - n2^t}\right) m \leq (1 + \varepsilon)m.$$

Item 2 reduces to 3 via the following more general claim:

Claim Solving $\textsc{SymK}(n, a, b)$ exactly reduces to solving $\textsc{SymK}(n, a, b)$ with approximation ratio $1 + \frac{1}{n2^a}$.

This is almost trivial to see. Note that the optimum of any instance of $\textsc{SymK}(n, a, b)$ is bounded by $m^* \leq n2^a$. Hence any value m such that $m^* \leq \left(1 + \frac{1}{n2^a}\right)m$ will have $m^* - m \leq m\left(1 - \frac{1}{1+\frac{1}{n2^a}}\right) < 1$, and so $m^* = m$ (as they are both integers). □

Observation 2 *When the inputs are given in binary, the above reductions can be computed by an* NC^0 *circuit. When the inputs are given as numerical parameters to Mulmuley's PRAM model, the reductions can be computed in time* $O\Big(\log n + \log\left(1 + \frac{1}{\varepsilon}\right)\Big)$ *using* $O(n/\varepsilon)$ *processors.*

3.2 Dynamic Programming Algorithm and a FPTAS

We now show that the original dynamic programming algorithm for classical 0–1 knapsack [3], together with its derived FPTAS, will also work for the symmetric knapsack problem. Again the proof is standard.

Theorem 3 *There is a dynamic programming algorithm for finding the optimum of* $\textsc{SymK}(n, a, b)$*, in time* $O(n^2 2^a B)$*, where* B *is the time required to sum and compare* $(b + \log n)$*-bit numbers.*

Proof Let $X = (v, w, \sigma, W, \Sigma)$ be an instance of $\textsc{SymK}(n, a, b)$, and let $V = \max_i v(i)$. Throughout the algorithm, $I = I(k, \tilde{v}, \tilde{\sigma})$, for $k \in [n]$, $\tilde{v} \in [nV]$, $\tilde{\sigma} \in \mathsf{M}$, when defined, will denote a solution which:

(1) is a subset of $[k]$, meaning $i \notin I$ for $i > k$;
(2) has value $\sum_{i \in I} v(i) = \tilde{v}$,

(3) pattern $\odot_{i\in I}\sigma(i) = \tilde{\sigma}$, and
(4) is weight-optimal, in the sense that $\sum_{i\in I} w(i)$ is minimum among the weights of all solutions obeying (1-3).

Now I is computed iteratively: for $k = 1$, we set $I(1, 0, e) = \emptyset$ and $I(1, v(1), \sigma(1)) = \{1\}$, and let $I(1, \tilde{v}, \tilde{\sigma})$ remain undefined for all other pairs of $\tilde{v}$ and $\tilde{\sigma}$ — i.e., with only one item we can either put it in I with value $v(1)$ and pattern $\sigma(1)$, or not put it in I, which results in value 0 and pattern e.

Suppose we have defined I up to $k-1$. Then let $I' = I(k-1, \tilde{v}, \tilde{\sigma})$, and $I'' = I(k-1, \tilde{v} - v(k), \tilde{\sigma}\sigma_k^{-1}) \cup \{k\}$. Then we define $I(k, \tilde{v}, \tilde{\sigma})$ to be equal to the I among I' and I'' that has minimum weight.[4] Again the same reasoning applies (we can either put k into I or not), and in this way we prove that the requirement of minimum weight is inductively maintained.

If we keep I and W as arrays in a random access memory, then each iteration can be computed in time $O(nVB) = O(n2^aB)$.

Finally, the optimum solution for X is the highest value $\tilde{v}$ for which $I = I(n, \tilde{v}, \Sigma)$ is defined, and has weight no greater than W. □

As in the original 0–1 knapsack problem, this algorithm can be used to obtain an FPTAS.

Corollary 1 *There is an algorithm for* SymK(n, a, b) *that achieves an approximation ratio of* $1+\varepsilon$ *in time* $O((1+\frac{1}{\varepsilon})n^3B)$.

3.3 A new, Simple Algorithm Using Small Space or Small Depth

We present a very simple algorithm, to our knowledge new, that solves subset-sum in $O(W^{\log n})$ time using $O(\log W \cdot \log n)$ space. Notice that if W is at least $2^{\Omega(n/\log n)}$, then we can instead run the trivial $O(n)$-space algorithm that tries every possible setting of the items, for an improved 2^n time bound. So this algorithm is only interesting for smaller values of W; but because approximation algorithms are equivalent to algorithms for small weights (Theorem 2) this makes our algorithm particularly suitable for approximation.

Our algorithm can be made to work for both the subset-sum and knapsack problems. It is a mix of ideas from two classical papers from the 70s: the *meet-in-the-middle* algorithm [10], and the PSPACE hardness of alternating time [20]. It can appropriately be called *recursive meet-in-the-middle algorithm*.

Theorem 4 (**Recursive meet-in-the-middle**) *There is an algorithm for* $SS(n, b)$ *that works in time* $O(W^{\log n})$ *and space* $O(\log W \cdot \log n)$.

Observation 3 *It is interesting to compare the time and space parameters of our algorithm with those of* [14]. *In that paper, the authors present a clever algorithm*

[4] And undefined if I' and I'' are both undefined.

for subset-sum based on the use of an adequate Fourier transform, which runs in time $\tilde{O}(n^2 W \log W)$ *and space* $\tilde{O}(n^2)$*. Our algorithm is slower, but when applying it to solve subset-sum approximately, our algorithm uses less space, because usually* $W \ll 2^{n/\log n}$ *in this scenario.*

We will present the algorithm for the subset-sum problem first, and then explain how it generalizes to the symmetric version and the knapsack problem.

Proof of Theorem 4 We show a recursive procedure for the following problem: we are given an instance $X = (w, W) \in SS(n, b)$, and a value $m \in [W]$; we will output whether there exists a subset of items $I \subseteq [n]$ such that $\sum_{i \in I} w(i) = m$. The procedure works as follows:

(1) (Base case) If $n = 1$, we return *true* iff $w(1) = m$, else we return *false*.
(2) We go through each possible value $m' \leq m$;
(3) then we consider the partial problems X_1 having the first half of the items, and X_2, having the second half;
(4) we recursively call our procedure on inputs X_1, m' and $X_2, m - m'$: if no solution was found for either of the sub-problems, we move on to the next choice of m';
(5) if solutions were found, return *true*.
(6) If we have gone through every value, we return *false*.

To see that we return *true* in case a solution exists, note that if I is a solution of X having value m, then $I_1 = I \cap \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ and $I_2 = I \setminus I_1$ will have some values, respectively m_1 and m_2. Since I has value m, it must be that $m_2 = m - m_1$; hence the algorithm will find some solution for X_1 and X_2 when calling (4) with $m' = m_1$ (here we assume inductively that solutions will be found for smaller n).

Finally, an algorithm for $SS(n, b)$ will call this procedure with $m = W$. To bound the time and space used, we notice that the recursion has depth $\log n$, and each call in the stack uses $O(\log W)$ space to store m and m'. □

By analyzing the algorithm above, we can see that it is given by an alternation of ORs and ANDs. Let $SS(X, m)$ mean that there exists a solution for X of value m. Then it is clear that

$$SS(X, m) \iff \bigvee_{m' \in [m]} \left[SS(X_1, m') \wedge SS(X_2, m - m') \right].$$

Expanding this out in a circuit gives us $O(\log n)$ layers. The bottom layer will simply contain equality tests. Hence we conclude that:

Corollary 2 *A parallel version of the algorithm can be implemented with* AC *circuits of depth* $O(\log n)$ *and size* $O(W^{\log n} b)$*, or in an arithmetic PRAM with* $O(W^{\log n})$ *processors running for* $O(\log n)$ *time.*

Observation 4 *For the knapsack problem* $\mathrm{K}[n, a, b]$*, where* w *and* v *may be distinct, each recursive call will look for solutions whose weight is* at most m*, rather than exactly* m*, and maintain in memory the maximum value among the admissible solutions found so far (rather than simply whether there exists a solution or not); i.e., the recursion is now:*

$$\mathrm{K}(X, m) = \max\{\mathrm{K}(X_1, m') + \mathrm{K}(X_2, m - m') | m' \in [m]\},$$

where $\mathrm{K}(X, m)$ *is the maximum value attainable for the instance* X *with weight up to* m*. The basic case is* $v(1)$ *if* $w(1) \leq m$*, and 0 otherwise. This is computable serially in* $O(W^{\log n})$ *time and* $(\log W + a) \log n$ *space, or by means of an* AC *circuit of depth* $O(\log n)$ *and size* $O(W^{\log n}(a^2 W^2 + b))$.[5]

4 Hardness of Approximation

In the last few decades, theoretical computer science has classified most known NP-hard optimization problems according to how close to the optimum solution a polynomial time algorithm is likely to get. For instance, if $\mathsf{P} \neq \mathsf{NP}$, then it holds that:

- For some constants $c > c' > 1$, SetCover can be approximated to ratio $c \log n$, but not $c' \log n$ [18].
- For some constants $c > c' > 1$, Max2Sat can be approximated to ratio c [13], but not c' [8].
- The problem of finding an optimal multi-processor scheduling with speed factors[6] has a PTAS, but no FPTAS [6, 9].

It is now natural to ask: how about problems that *do* have a FPTAS, can we prove that this kind of approximability is optimal, under a hardness assumption of some kind? We now show that the answer is yes, and that in this context the (much stronger) exponential-time hypothesis is an adequate choice.

4.1 Hardness of Approximation for Knapsack

Our strategy is the usual: we reduce a hard problem to the task of approximating subset-sum with ratio better than $1 + \frac{1}{poly}$.

Theorem 5 $\mathsf{NC\text{–}SAT}(S, k) \leq_m^{\mathsf{AC}_0} SS(O(S + k), O(S + k))$.

[5] To see this, notice that the maximum of W-many a-bit numbers can be computed by AC circuits of size $O(W^2 a^2)$, and the basic case can be computed by a circuit of size $O(b)$.

[6] This is the problem of scheduling tasks to processors of different speeds, while attempting to minimize execution time. Hence each task t in a set of tasks T has a length $l(t)$, and each processor p in a set of processors P has a speed factor $s(p)$. We wish to find an assignment of the tasks $f : T \to P$ minimizing completion time: $\max_{p \in P} \sum_{t : f(t) = p} l(t) s(p)$.

Proof We will show that there is a uniform AC_0 circuit which, given as input a size-S circuit $C(\bar{y})$ over k variables $\bar{y} = y_1, \ldots, y_k$, will output an instance $X = (w, W)$ of

$$\mathrm{SS}(2\,k + 3S, 2k + 4S + 1),$$

that admits a subset-sum of weight W if and only if $C(\bar{y}) = 1$ for some choice of $\bar{y}$.

Suppose C is given as a sequence $(G_1, \ldots, G_S)$ of binary NAND gates, where each gate G_j is defined by the two gates feeding into it (these could be input gates or gates $G_{j'}$ for $j' < j$). Then X will have two items $Y_i^{(0)}$ and $Y_i^{(1)}$ for each $i \in [k]$, and three items $G_j^{(00)}, G_j^{(01)}, G_j^{(10)}$ for each NAND gate $j \in [S]$. We will construct our subset-sum instance in such a way that any optimal solution must choose exactly one of the $Y_i^{(0)}, Y_i^{(1)}$ items for each $i \in [k]$, and exactly one of the G_j^{ab} items for each $j \in [S]$. For any optimal solution, a choice of the Y items will force a choice of gate items corresponding to an evaluation of the circuit.

It will help comprehension if we present the weights of the items in our instance both algebraically and in table form. For $i \in [k]$, $j \in [S]$, let $c(i, j)$ be 2 if input i is the first input of gate j, 1 if it is the second input, and 0 otherwise. Similarly with gates, for $j, j' \in [S]$, let $d(j, j')$ be 2 if gate j is the first input of gate j', 1 if it is the second input, and 0 otherwise. We will set the weights to:

$$\begin{aligned}
w(Y_i^{(0)}) &= 2^{2(k-i)+4S+1} \\
w(Y_i^{(1)}) &= 2^{2(k-i)+4S+1} + \sum_{j=1}^{S} c(i, j)2^{4(S-j)+1} \\
w(G_j^{(00)}) &= (2+1) \times 2^{4(S-j)+1} + \sum_{j'=j+1}^{S} d(j, j')2^{4(S-j')+1} + [j = S?] \\
w(G_j^{(01)}) &= (2+0) \times 2^{4(S-j)+1} + \sum_{j'=j+1}^{S} d(j, j')2^{4(S-j')+1} + [j = S?] \\
w(G_j^{(10)}) &= (0+1) \times 2^{4(S-j)+1} + \sum_{j'=j+1}^{S} d(j, j')2^{4(S-j')+1} + [j = S?]
\end{aligned}$$

(Recall that $[j = S?]$ denotes 1 when $j = S$ and 0 otherwise.) We set the maximum weight to:

$$W = \sum_{i=1}^{k} 2^{2(k-i)+4S+1} + \sum_{j=1}^{S} 3 \times 2^{4(S-j)+1} + 1$$

The DLOGTIME uniformity of the reduction follows from the simplicity of the above expressions, which make the circuit computing the bits of the instance very

easy to describe. Some global lookup is required to compute the functions c and d, which makes the reduction AC_0, rather than simply NC_0.[7]

Let us see what these numbers look like when written in binary. Since each number is less than $2^{2k+4S+1}$, we will write them in a table with $k+S+1$ blocks. The first k blocks have two bits each, the middle blocks have four bits each, and the last block has one bit.

w	1 …	i …	k	G_1 …	G_j …	$G_{j'}$ …	G_S	*out*
$Y_i^{(0)}$	00 …	01 …	00	0000 …	0000 …	0000 …	0000	0
$Y_i^{(1)}$	00 …	01 …	00	$c(i,1)$ …	$c(i,j)$ …	$c(i,j')$ …	$c(i,S)$	0
$G_j^{(00)}$	00 …	00 …	00	0000 **0**	0011 …	$d(j,j')$ …	$d(j,S)$	$[j=S?]$
$G_j^{(01)}$	00 …	00 …	00	0000 **0**	0010 …	$d(j,j')$ …	$d(j,S)$	$[j=S?]$
$G_j^{(10)}$	00 …	00 …	00	0000 **0**	0001 …	$d(j,j')$ …	$d(j,S)$	$[j=S?]$
					…			
W	01 …	01 …	01	0011 …	0011 …	0011 …	0011	1

It is vital to notice that in each column G_j there will be exactly five non-zero entries: two correspond to the inputs, having binary form 10 and 01, and three correspond to the gate G_j, and have binary form 11, 10 and 01. These are chosen so that for any setting $ab \in \{0,1\}^2$ of the items corresponding to the two inputs, then if we wish to achieve the maximum weight W, we will be forced to pick the gate item $G_j^{(ab)}$. Formally, we wish to prove the following claim, from which the theorem follows immediately.

Claim The solutions I of X having weight exactly W, are in 1–1 correspondence with the assignments of $\bar{y}$ causing $C(\bar{y}) = 1$. □

Given a choice of $\bar{y}$, we can define the solution $I(\bar{y})$ inductively as follows:

- $\{Y_i^{(y_i)}\} \subseteq I$;
- For each gate G_j, we let
  - $a_j = 1$ if input i is the first input to gate j and $Y_i^{(1)} \in I$, or if gate ℓ is the first input to gate j and $G_\ell^{(a_\ell b_\ell)} \in I$; and $a_j = 0$ otherwise;
  - similarly for b_j and the second input to gate j.
- Then $G_j^{(a_j b_j)} \in I$ if a_j and b_j are not both 1.

This solution will have weight exactly W, if $C(\bar{y}) = 1$, and weight $W-1$, if $C(\bar{y}) = 0$. This can be seen easily by inspection of the table above: for instance, if $G_j = \mathrm{NAND}(y_i, y_{i'})$, $y_i = 1$ and $y_{i'} = 0$, then $Y_i^{(1)}$ contributes (binary) weight 10 to column G_j, $Y_i^{(0)}$ contributes weight 0, and $G_j^{(10)}$ contributes weight 01, for a total

[7] The encoding of the circuit could be changed to make this lookup easier.

of 11, which is exactly W for that column. The final column will be 1 iff the output gate G_S evaluates to 1.

To complete the proof of the claim, and hence of the theorem, it suffices to show that any solution achieving weight W must be of the form $I(\bar{y})$ for some unique choice of $\bar{y}$. Let I be any solution with total weight W. Again by inspection of the table above, we will find that, for each i, exactly one of $Y_i^{(0)}$ or $Y_i^{(1)}$ must be in I. This follows by letting i be the hypothetical first coordinate where this fails to hold: if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are in I, then the total weight will surpass W, and if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are missed, the total weight must be less than W because of the low weight of the items which are yet to be fixed.[8] So let $Y_i^{(y_i)}$, for $i \in [k]$, give us the set of chosen Y items. Then again in an inductive fashion we establish that for each gate G_j we must pick at most one item among $G_j^{(ab)}$, exactly as defined in $I(\bar{y})$: again it holds that this is the unique choice which fits the weight W exactly right.

This allows us to obtain a hardness-of-approximation result for the subset-sum problem:

Corollary 3 *For every $f(n) = \omega(1)$, there exists $g(n) = \omega(n)$, such that if the knapsack problem can be approximated to ratio $1 + n^{-f(n)}$ in polynomial time, then $\mathsf{NC}_1\text{–}\mathsf{SAT}(g(n), g(n)) \subseteq \mathsf{DTIME}(2^{o(n)})$ and the Exponential–Time Hypothesis is false.*

Proof If the knapsack problem can be approximated thus, then by Theorem 2 (more precisely, by the analogous theorem for subset-sum), we can solve $\mathrm{SS}(n, h(n)\log n)$ in time n^c, for some non-decreasing super-constant $h(n)$. By an appropriate substitution (say $n \leftarrow 2^{n/\sqrt{h(n)}}$), this implies we can solve $\mathrm{SS}(2^n, \omega(n))$ in time $2^{o(n)}$. By Theorem 5 we can now conclude that $\mathsf{NC}_1\text{–}\mathsf{SAT}(\omega(n), \omega(n))$ can be solved in time $2^{o(n)}$.

In particular this implies that the satisfiability of 3CNF formulas with $\omega(n)$-many clauses can be decided in time $2^{o(n)}$ But the sparsification lemma of [12] will then imply that the satisfiability of 3CNF formulas of any size can *also* be decided in time $2^{o(n)}$, i.e., that the Exponential–Time Hypothesis is false.

More precisely, the sparsification lemma states that there is a function f such that, for any $\varepsilon > 0$, any 3CNF formula is equivalent to the OR of $O(2^{\varepsilon n})$ 3CNF formulas in which the number of clauses is $f(\varepsilon)n$, and these formulas can be produced in time $poly(n)2^{\varepsilon n}$. Hence we let $\varepsilon = \varepsilon(n) = o(1)$ decrease slowly enough, causing $f(\varepsilon(n)) = \omega(1)$ to be sufficiently small, so that we can solve 3CNF formulas with $f(\varepsilon(n))n$ clauses in time $2^{o(n)}$. Then producing and evaluating the big OR of 3CNFs would take time $poly(n)2^{2\varepsilon n + o(n)} = 2^{o(n)}$ in total. □

[8] Here it is worth pointing out: this is the reason we need to "pad" each block of the weights with additional bits. This bears some resemblance to the superincreasing sequences used in the Merkle-Hellman knapsack cryptosystem.

Notice that this result is optimal, since for any constant c it *is* possible to approximate knapsack to ratio $1 + \frac{1}{n^c}$ in polynomial time.

4.2 Hardness of Approximation for Symmetric Knapsack

It is immediate to see that $\textsc{SymSS}(n, n)$ is NP-hard, since it includes the original subset-sum problem as a special case. However, the addition of a small amount of combinatorial structure to the problem will allow for a hardness result where the bit-length of the weights only depends logarithmically on the formula size.

Theorem 6 $\mathsf{NC}_1\text{–}\mathsf{SAT}(S, k) \leq_m^{\mathsf{AC}_0} \textsc{SymSS}(O(S^2 + k), O(k \log S))$.

Proof We will show that there is a uniform AC_0 circuit which, given as input a width-5 permutation branching program $P(\bar{y})$ over k variables $\bar{y} = y_1, \ldots, y_k$ having size S, will output an instance $X = (w, \sigma, W, \Sigma)$ of

$$\textsc{SymSS}(2(k + S), k(2 + \ell(S))), \qquad (\ell(S) = \lceil \log S + 1 \rceil)$$

that admits a subset-sum of weight W if and only if $P(\bar{y}) = e$ for some choice of $\bar{y}$. The result then follows by Barrington's theorem [2], which states that any NC^0 circuit can be evaluated by such a width-5 branching program.

So let $P = \{(j, z_j, \alpha_j, \beta_j)\}_{j=1}^{S}$, with $z_j \in [k]$, $\alpha_j, \beta_j \in \mathsf{S}_5$. We will always use i to index the y's, and j to index the instructions of P.

In our symmetric subset-sum problem, we will have two items for each possible y, denoted by $Y_i^{(1)}$ and $Y_i^{(0)}$; these items represent the possible assignments of the respective variable. We will also have two items for each instruction, denoted by A_j and B_j, which represent the choice of either α_j or β_j.

We will construct our subset-sum instance in such a way that the optimal solution will choose exactly one of the $Y_i^{(0)}, Y_i^{(1)}$ items for each i, and this choice will force the correct choice of either A_j or B_j, whenever y_i is the variable consulted in the j-th instruction of P (i.e., whenever $z_j = i$). If we choose to take item $Y_i^{(0)}$, for instance, then, in order to achieve weight W, we will be forced to always pick every A_j whenever $z_j = i$, and won't be able to pick any of the B_j.

Let $N(i) \leq S$ denote the number of times that y_i is consulted by P, i.e., the number of j such that $z_j = i$, and make $\ell = \lceil \log S + 1 \rceil$, so that each $N(i) < 2^\ell - 1$.

As before, we present the weights of the items in our instance both algebraically and in table form.

We will set the weights to:

$$\begin{aligned} w(Y_i^{(0)}) &= 2^{2(k-i) + 2k\ell} + N(i)2^{2(k-i)\ell} \\ w(Y_i^{(1)}) &= 2^{2(k-i) + 2k\ell} + N(i)2^{\ell+2(k-i)\ell} \\ \text{if } z_j = i, \text{ then } w(A_j) &= 2^{\ell+2(k-i)\ell} \\ \text{and } w(B_j) &= 2^{2(k-i)\ell} \end{aligned}$$

We set the maximum weight to:

$$W = \sum_{i=1}^{k} 2^{2(k-i)+2k\ell} + N(i)\left(2^{2(k-i)\ell} + 2^{\ell+2(k-i)\ell}\right).$$

The DLOGTIME uniformity of the reduction follows from the simplicity of the above expressions, which make the circuit computing the bits of the instance very easy to describe. Sum is required to compute $N(i)$, which makes the reduction AC_0, rather than simply NC_0.

Let us again see what these numbers look like in table form. Since each number is less than $2^{2k+2k\ell}$, we will write them in a table with $3k$ blocks. The first k blocks have two bits each, and the last $2k$ blocks have ℓ bits each:

w	1	…	i	…	k	$A^{(1)}$	$B^{(1)}$	…	$A^{(i)}$	$B^{(i)}$	…	$A^{(k)}$	$B^{(k)}$
$Y_i^{(0)}$	00	…	01	…	00	0	0	…	0	$N(i)$	…	0	0
$Y_i^{(1)}$	00	…	01	…	00	0	0	…	$N(i)$	0	…	0	0
$A_j\|_{z_j=i}$	00	…	00	…	00	0	0	…	1	0	…	0	0
$B_j\|_{z_j=i}$	00	…	00	…	00	0	0	…	0	1	…	0	0
							…						
W	01	…	01	…	01	$N(1)$	$N(1)$	…	$N(i)$	$N(i)$	…	$N(k)$	$N(k)$

Finally, we set $\sigma(Y_i^{(0)}) = \sigma(Y_i^{(1)}) = e$, $\sigma(A_j) = \alpha_j$ and $\sigma(B_j) = \beta_j$. Furthermore, in our sorting of the items (which is relevant for the outcome of $\odot_i \sigma(i)$), we ensure that the items A_j, B_j appear in growing order of j.

Here, our result follows from the following claim by setting $\Sigma = e$:

Claim The solutions I of X having weight exactly W and pattern Σ, are in 1–1 correspondence with the assignments of $\bar{y}$ causing $P(\bar{y}) = \Sigma$. □

To see this, notice that given a choice of $\bar{y}$, we define the solution $I(\bar{y}) = \{Y_i^{(y_i)}\} \cup \{A_j | z_j = i \wedge y_i = 0\} \cup \{B_j | z_j = i \wedge y_i = 1\}$, and this solution will have weight exactly W, and pattern $P(\bar{y})$. This can be seen easily by inspection of the table above: for instance, if $y_i = 0$, the choice of $Y_i^{(0)}$ contributes weight 1 to the i-th column of the i-th block, and weight $N(i)$ to the column $B^{(i)}$; then the choice of $N(i)$-many A_j variables will contribute with a total weight of $N(i)$ to the column $A^{(i)}$. Now the pattern of $I(\bar{y})$ will simply be $P(\bar{y})$, by construction.

To complete the proof of the claim, and hence of the theorem, it suffices to show that any solution achieving weight W must be of the form $I(\bar{y})$ for some unique choice of $\bar{y}$. Let I be any solution with total weight W. Again by inspection of the table above, we will find that, for each i, exactly one of $Y_i^{(0)}$ or $Y_i^{(1)}$ must be in I. This follows by letting i be the hypothetical first coordinate where this fails to hold: if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are missed, the total weight will be less than W, and if both $Y_i^{(0)}$ and $Y_i^{(1)}$ are in I, then the total weight will surpass W. So let $Y_i^{(y_i)}$, for $i \in [k]$, give us the set of chosen Y items. In the same way as before, it is easy to see that, for each i,

every A_j with $z_j = i$ must be picked, if $y_i = 0$, or otherwise every corresponding B_j must be picked. Hence $I = I(\bar{y})$, as intended.

From Theorems 2 and 6 we may conclude, for instance:

Corollary 4 *Symmetric knapsack cannot be approximated to ratio* $1 + 2^{-(\log n)^3}$ *in polynomial time, unless* $\exists\mathsf{NC}_1((\log n)^2) \subseteq \mathsf{P}$.

5 Lower Bounds for Parallel Time

We will now prove that the knapsack problem cannot be efficiently parallelized in Mulmuley's model. The hardness of knapsack has already been proven in [19] in the fully-algebraic setting, and now we present a proof for the optimization version of subset-sum in the semi-algebraic setting. We do this using Mulmuley's parametric complexity technique. We will stick to the least general definition that applies to the subset-sum case.

The optimization version of subset-sum is simply the knapsack problem restricted to the case where the values of the items equal their weights; i.e., we are given $(\bar{w}, W)$ and we wish to compute:

$$S(\bar{w}, W) = \max\{\bar{a} \cdot \bar{w} | \bar{a} \in \{0, 1\}^n \text{ and } \bar{a} \cdot \bar{w} \leq W\}$$

We will now refer to the problem of computing S as simply *the knapsack problem*. A *linear parametrization* for the knapsack problem on inputs of length n is a function $P_n : \mathsf{R} \to \mathsf{R}^{n+1}$ mapping some interval $[A, B]$ into instances $(\bar{w}, W)$, where W is fixed and each w_i is given by a linear function of a single parameter λ. Formally, we specify P_n by a tupple of $2n + 4$ integers $(A, B, a_0, b_0, \ldots, a_n, b_n)$; then $P_n(\lambda)$, for some real number $\lambda \in [A, B]$, is the knapsack instance with $W = a_0\lambda + b_0$, and $w_i = a_i\lambda + b_i$.

P_n is said to have bit-length $\beta = \beta(n)$ if all the a_i and b_i are integers of bit-length β. For each $\lambda \in [A, B]$, let $m(\lambda) = S(P_n(\lambda))$.[9] Then it can be seen that $m(\lambda)$ is a piecewise-linear and convex function of λ. The *complexity* $\rho(n)$ of P_n is then the number of different slopes in the pieces of $m(\lambda)$ as λ goes from A to B.

The *parametric complexity* of the knapsack problem for bit-length $\beta(n)$, $\phi(n, \beta(n))$, is equal to the maximum complexity $\rho(n)$, over all parametrizations P_n of bit-length $\beta(n)$. Now let us define the following decision version of the knapsack problem: $A_n = \{(\bar{w}, W, z) | S(\bar{w}, W) \leq z\}$. Then the following is proven in [15]:

Theorem 7 (**Theorem 3.3 of** [15]) *There exist large enough constants* a, b *such that* A_n *cannot be solved in* $\mathsf{KC}(2^{\sqrt{\log \phi(n,\beta(n))}/b}, \sqrt{\log \phi(n, \beta(n))}/b)$*; this is so even if*

[9] Here we extend the knapsack problem to the reals. This can easily be done due to it being a homogeneous optimization problem, i.e. $S(\alpha\bar{w}, \alpha W) = \alpha S(\bar{w}, W)$ for any real α. We could avoid this, except that it makes the definitions easier to visualize.

we restrict every numeric parameter in the input to be an integer with bit-length at most $a\beta(n)$.

Below we will prove exponential lower bounds on $\phi(O(n^2), O(n^2))$, which imply that A_n is not in $\mathsf{KC}(2^{n^{\frac{1}{4}}/b}, n^{\frac{1}{4}}/b)$, even for instances where each weight has at most an bits. This is a stronger setting than the results in [19], which prove that subset-sum (as a decision problem) is not in $\mathsf{KC}(2^{\sqrt{n}/b}, \sqrt{n}/b)$, but without any restriction on the bit-length at which the hard instances will be found. Unfortunately we were unable to show a parallel algebraic reduction from A_n (which is an approximation version of subset-sum, and which in this section we call *knapsack problem*) to subset-sum (as a decision problem), and hence our results hold only for the knapsack problem, rather than subset-sum.

5.1 High Parametric Complexity for Knapsack

Theorem 8 $\phi(O(n^2), O(n^2)) \geq 2^n$. *Hence* A_n *is not in* $\mathsf{KC}(2^{n^{\frac{1}{4}}/b}, n^{\frac{1}{4}}/b)$, *even for bit-lengths restricted to* $O(n)$.

The a and b above are the constants given by Theorem 7. The proofs require a careful blending of the techniques of Theorem 5 and of the paper [15].

Proof For given integers k, $K = \frac{k(k-1)}{2}$, we will construct a linear parametrization $P(\lambda)$ mapping $[0, 2^k]$ to instances of the knapsack problem with $O(k+K)$ items, and having bitlength $O(k+K)$. P will be such that the optimum $m(\lambda) = S(P(\lambda))$, as λ goes from 0 to 2^k, is a piecewise-linear function, and if we look at the slopes in the segments that make up m, there will be 2^k-many different values. This implies that $\phi(O(k+K), O(k+K)) \geq 2^k$, which proves the first statement of our Theorem up to a change in parameters. The second statement on the hardness of A_n then follows from Theorem 7.

Let $\bar{y} = y_1, \ldots, y_{k+K}$ denote binary vectors of length $k + K = k + \frac{k(k-1)}{2}$. Let $(i_1, i_2) \mapsto \pi(i_1, i_2)$ be some bijection between pairs of integers $1 \leq i_1 < i_2 \leq k$ and the integers $k+1, \ldots, k+K$. We will then use $y_{(i_1,i_2)}$ to denote $y_{\pi(i_1,i_2)}$.

Let $C(\bar{y})$ be the formula:

$$\bigwedge_{1 \leq i_1 < i_2 \leq k} \left(y_{(i_1,i_2)} = y_{i_1} \wedge \neg y_{i_2} \right), \tag{1}$$

and let $X = X_C$ be the corresponding subset-sum instance given by Theorem 5. Since C can be implemented by an NC circuit with $10K$ NAND gates, then X will have $2k + 30K$ items and the weights have bit-lengths at most $2k + 40K + 1$.

Our $P(\lambda)$ will have the same items as X, namely for each variable y_i there are two items designated $Y_j^{(0)}$ and $Y_j^{(1)}$,[10] and we will use the generic designation C_j, for

[10] We will also use the notation $Y^{(0)}_{(i_1,i_2)}$, $Y^{(1)}_{(i_1,i_2)}$ to denote the items corresponding to the variable $y_{(i_1,i_2)}$.

$j = 1, \ldots, 30K$, to denote the remaining items[11]. However, the weights of $P(\lambda)$ will be different: although the most significant bits of $P(\lambda)$ will be set in the same way as in X (and thus do not depend on λ), the least significant bits will be set so that as λ goes from 0 to 2^k, the optimum value of $P(\lambda)$ will have 2^k-many different slopes.

Again we will present $P(\lambda)$ both numerically and in table form. Let w_X and W_X denote the weights of X. We will set the weights of $P(\lambda)$ as follows:

$$\begin{aligned} w(Y_i^{(0)}) &= 2^{k(k+5)} w_X(Y_i^{(0)}) + f_i^{(0)}(\lambda) \\ w(Y_i^{(1)}) &= 2^{k(k+5)} w_X(Y_i^{(1)}) + f_i^{(1)}(\lambda) \\ w(C_j) &= 2^{k(k+5)} w_X(C_j) \\ W &= 2^{k(k+5)} W_X + 2^{k(k+5)} - 1 \end{aligned}$$

We define f as follows:

$$\begin{aligned} \text{for } 1 \le i \le k,\ f_i^{(0)}(\lambda) &= 2^{(k-i)(k+5)}(2^{k-i+1} - \lambda) \\ f_i^{(1)}(\lambda) &= 2^{(k-i)(k+5)}(2^{k-i}) \\ \text{for } 1 \le i_1 < i_2 \le k,\ f_{(i_1,i_2)}^{(0)}(\lambda) &= 0 \\ f_{(i_1,i_2)}^{(1)}(\lambda) &= 2^{(k-i_2)(k+5)} 2^{k-i_1} \end{aligned}$$

Notice that the weights w are a left shift of w_X by $k(k+5)$ bits, except for the items $Y_j^{(0)}$ and $Y_j^{(1)}$, who additionally get a least-significant value parametrized by λ. See also that the values of f, written in binary, can be partitioned into k blocks of $k+5$ bits each.

Let us illustrate this with the following table:

	$(2k + 40K + 1$ bits)	1	…	i	…	k
$Y_i^{(0)}$	Theorem 5	0	…	$2^{k-i+1} - \lambda$	…	0
$Y_i^{(1)}$	Theorem 5	0	…	2^{k-i}	…	0
$Y_{(i_1,i)}^{(0)}\|_{i_1<i}$	Theorem 5	0	…	0	…	0
$Y_{(i_1,i)}^{(1)}\|_{i_1<i}$	Theorem 5	0	…	2^{k-i_1}	…	0
W	Theorem 5	$2^{k+5} - 1$	…	$2^{k+5} - 1$	…	$2^{k+5} - 1$

The similarity with Theorem 5 ensures that we must pick exactly one of $Y_i^{(0)}$ and $Y_i^{(1)}$, and that $C(\bar{y}) = 1$ for the vector $\bar{y}$ corresponding to this choice, i.e.:

[11] By this we mean that each C_j, for $j = 1, \ldots, 30K$, is one of the three items $G_j'^{(00)}$, $G_j'^{(01)}$, $G_j'^{(10)}$ for some j', that appear in the proof of Theorem 5. We group them together because we will treat them in the exact same way.

Claim Each optimal solution of $P(\lambda)$ corresponds to a unique choice of $\bar{y}$ obeying (1). □

The proof of this claim is exactly as before. The correspondence is no longer bijective, since there might be values of $\bar{y}$ obeying (1) which have sub-optimal value due to f. However, any solution which is optimal up to the first $2k + 40K + 1$ bits must already satisfy the formula C, and in this case these bits must be set to W_X as in the proof of Theorem 5. We take this one step further, by noticing that (1) fully determines $\bar{y}$ from the first k bits $y_1, \ldots, y_k$.

So let $\hat{y}$ denote $y_1, \ldots, y_k$, and $I(\hat{y})$ be the solution containing the items $Y_i^{(y_i)}$, $Y_{(i_1,i_2)}^{(y_1 \wedge \neg y_2)}$, and the corresponding C_j's that simulate the computation of the circuit. Notice that $I(\hat{y})$ maximizes the first $2k + 10K + 1$ bits of the value to be exactly W_X: so we define $g_\lambda : \{0,1\}^k \to [2^{k(k+5)} - 1]$, to be the optimum of $I(\hat{y})$ minus $2^{k(k+5)} W_X$. Now the following claim holds:

Claim $\hat{y} \mapsto I(\hat{y})$ is a bijective correspondence between $\{0,1\}^k$ and solutions of $P(\lambda)$ having value at least $2^{k(k+5)} W_X$. Furthermore, the optimum solutions of $P(\lambda)$ are those $I(\hat{y})$ which maximize $g_\lambda(\hat{y})$. □

From this we may define $m^*(\lambda) = \max_{\hat{y} \in \{0,1\}^k} g_\lambda(\hat{y})$, and this will be the value of the optimum solution of $P(\lambda)$, minus $2^{k(k+5)} W_X$. The instance $P(\lambda)$ was constructed so that the optimum is given exactly by λ:

Claim For a given $\lambda \in [0, 2^k)$ of the form $\lambda = \lfloor \lambda \rfloor + \Delta$, $\Delta \in (\frac{1}{4}, \frac{3}{4})$, the maximum value of $g_\lambda(\hat{y})$ is attained when $\hat{y}$ is the binary expansion of $\lfloor \lambda \rfloor$. □

We now prove this claim. Fix some choice $\hat{y} = y_1, \ldots, y_k$. Then, by summing over the columns of the above table, and by (1), one sees that the following formula holds:

$$g_\lambda(\hat{y}) = \sum_{i=1}^{k} \left(y_i 2^{k-i} + (1 - y_i) \left(2^{k-i+1} - \lambda + \sum_{i_1 < i} y_{i_1} 2^{k-i_1} \right) \right) 2^{(k-i)(k+5)} \quad (2)$$

Take the binary expansion of $\lfloor \lambda \rfloor = \lambda_1 2^{k-1} + \ldots + \lambda_k 2^0$. We prove that the maximum of g_λ is achieved when $\hat{y} = \lambda_1 \ldots \lambda_k$. For any $\hat{y} \neq \lambda_1 \ldots \lambda_k$, let d be the first bit for which $\lambda_d \neq y_d$. Then we let $\tilde{y} = \lambda_1 \ldots \lambda_d y_{d+1} \ldots y_k$, and show that $g_\lambda(\tilde{y}) \geq g_\lambda(\hat{y})$.

Let $g_\lambda^{(i)}(\hat{y}) 2^{(k-i)(k+5)}$ be the i-th term in the summand of (2), and $G_i = g_\lambda^{(i)}(\tilde{y}) - g_\lambda^{(i)}(\hat{y})$. Then $G_i = 0$ for $i < d$, and so

$$g_\lambda(\tilde{y}) - g_\lambda(\hat{y}) = G_d 2^{(k-d)(k+5)} + \sum_{i>d} G_i 2^{(k-i)(k+5)}.$$

We may lower bound G_d as follows:

$$\begin{aligned} G_d &= (\lambda_d - y_d)2^{k-d} + (y_d - \lambda_d)\left(2^{k-d+1} - \lambda + \sum_{i_1<d} \lambda_{i_1} 2^{k-i_1}\right) \\ &= (\lambda_d - y_d)\left(\lambda_d 2^{k-d} - 2^{k-d} + \Delta + \sum_{i>d} \lambda_i 2^{k-i}\right) \\ &\geq \min(\Delta, 1-\Delta) \geq \frac{1}{4} \end{aligned}$$

Since every $|G_i|$ is upper bounded by 2^{k+3}, it now follows that

$$g_\lambda(\tilde{y}) - g_\lambda(\hat{y}) \geq G_d 2^{(k-d)(k+5)} - \sum_{i>d} |G_i| 2^{(k-i)(k+5)} > 0,$$

and the claim is proven.

Finally, we establish that the parametric complexity of knapsack is $\Omega(2^k)$:

Claim The slope of $m^*(\lambda)$ is unique for each interval

$$(\lfloor\lambda\rfloor + \frac{1}{4}, \lfloor\lambda\rfloor + \frac{3}{4}) \subset [0, 2^k].$$

□

Looking at (2), from the previous claim it follows that $m^*(\lambda) = g_\lambda(\lfloor\lambda\rfloor)$, which simplifies to the linear form:

$$m^*(\lambda) = g_\lambda(\lambda_1, \ldots, \lambda_k) = \sum_{i:\lambda_i=1} 2^{(k-i)(k+5)}\lambda + A_{\lfloor\lambda\rfloor}$$

for some constant $A_{\lfloor\lambda\rfloor}$. And so each binary expansion of $\lfloor\lambda\rfloor$ gives rise to a unique slope.

We now state the consequences for parallel algorithms attempting to approximate the knapsack problem.

Corollary 5 *Knapsack instances with n items cannot be approximated to ratio* $1+\varepsilon$ *in time* $(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a$*, and using* $2^{(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a}$ *processors, for some positive constant a, in Mulmuley's PRAM model without bit operations.*

Proof By the same proof as in Claim 3.1 we can show that optimally solving a knapsack instance (having $v = w$) with n items and bit-length n reduces to solving the same knapsack instance with approximation ratio $1 + \frac{1}{n2^n}$. So setting $\frac{1}{\varepsilon} = n2^n$, by Theorem 8 this cannot be done in $\mathsf{KC}(2^{n^{\frac{1}{4}}/b}, n^{\frac{1}{4}}/b)$ for some b, and hence neither in $\mathsf{KC}(2^{(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a}, (\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a)$ for some larger a. □

From our construction we also derive a new lower-bound for solving knapsack exactly, which matches the lower bound Mulmuley [15] proved for max-flow.

Corollary 6 *Knapsack instances with n items and bit-length $O(n^2)$ cannot be solved in time $\sqrt{n}/a$, using $2^{\sqrt{n}/a}$ processors, for some positive constant a, in Mulmuley's PRAM model without bit operations.*

6 Open Problems

We consider the following two open problems worthy of further research. First, we have proven a lower bound for the optimization version of the subset-sum problem (the problem of computing $S(\bar{w}, W)$ exactly), even in the case of small input bit-lengths. Can we prove a lower bound for the decision version of subset-sum? Surprisingly, there is a reduction from the optimization version to the decision version [cf. 17], but this reduction is not algebraic, or even semi-algebraic, and so the question remains open.

Second, we have proven that we can approximate knapsack up to error $1+\varepsilon$ in $\log n$ time with $(\frac{1}{\varepsilon})^{\log n}$ processors, and that we cannot do it in time $(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a$, and using $2^{(\log \frac{1}{\varepsilon})^{\frac{1}{4}}/a}$ processors. Can this gap be closed?

Acknowledgments We thank Ulle Endriss for asking the question that led to this paper, as well as Karl Bringmann, Radu Curticapean, and Dominik Scheder for pointing out that the sparsification lemma would give the full contradiction with the ETH.

References

1. Bach, E., Shallit, J.: Algorithmic Number Theory I: Efficient Algorithms. Cambridge University Press (1996)
2. Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comput. Syst. Sci. **38**(1), 150–164 (1989)
3. Bellman, R.: Bottleneck problems and dynamic programming. Proc. Natl. Acad. Sci. USA **39**(9), 947–951 (1953)
4. Buss, S.R.: The boolean formula value problem is in ALOGTIME. In: Proceedings of the 19th STOC, pp. 123–131 (1987)
5. Buss, S.R., Cook, S.A., Gupta, A., Ramachandran, V.: An optimal parallel algorithm for formula evaluation. SIAM J. Comput. **21**, 755–780 (1992)
6. Crescenzi, P., Kann, V.: A compendium of NP optimization problems. http://www.nada.kth.se/viggo/problemlist/compendium.html (2005)
7. Cygan, M., Dell, H., Lokshtanov, D., Marx, D., Nederlof, J., Okamoto, Y., Paturi, R., Saurabh, S., Wahlström, M.: On problems as hard as CNF-SAT. In: Proceedings of the 27th CCC, pp. 74–84 (2012)
8. Håstad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)
9. Hochbaum, S.S., Shmoys, D.B.: A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approach. SIAM J. Comput. **17**, 539–551 (1988)
10. Horowitz, E., Sahni, S.: Computing partitions with an application to the subset-sum problem. J. ACM **20**(2), 277–292 (1974)
11. Impagliazzo, R., Paturi, R.: The complexity of k-SAT. In: Proceedings of the 14th CCC, pp. 237–240 (1999)

12. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
13. Lewin, M., Livnar, D., Zwick, U.: Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In: Proceedings of the 9th ICPO, pp. 67–82 (2002)
14. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: Proceedings of the 42nd STOC, pp. 321–330 (2010)
15. Mulmuley, K.: Lower bounds in a parallel model without bit operations. SIAM J. Comput. **28**(4), 1460–1509 (1999)
16. Myasnikov, A., Nikolaev, A., Ushakov, A.: Knapsack problems in groups. ArXiv:1302.5671 (2013)
17. Nederlof, J., Leeuwen, E.J.v., Zwaan, R.v.d.: Reducing a target interval to a few exact queries. In: Proceedings of the 37th MFCS, pp. 718–727 (2012)
18. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: Proceedings of the 29th STOC, pp. 475–484 (1997)
19. Sen, S.: The hardness of speeding-up knapsack. Technical report, BRICS (1998)
20. Stockmeyer, LJ., Meyer, AR.: Word problems requiring exponential time. In: Proceedings of the 5th STOC, pp. 1–9 (1973)
21. Vazirani, V.V.: Approximation Algorithms. Springer (2004)