Improved knapsack algorithms

K.A.Draziotis, A. Papadopoulou

25 August, 2016
ACAC'16
Athens

The Knapsack Problem

Given a list of $n+1$ positive integers $\{a_1, ..., a_n, s\}$ such that,

$$\max\{a_i\}_i \leq s \leq \sum_{i=1}^{n} a_i$$

find a binary vector

$$\mathbf{x} = (x_i)_i \text{ with } \mathbf{x} \cdot \mathbf{a} = \sum_{i=1}^{n} x_i a_i = s. \tag{1}$$

The Knapsack Problem

Given a list of $n+1$ positive integers $\{a_1, ..., a_n, s\}$ such that,

$$\max\{a_i\}_i \leq s \leq \sum_{i=1}^{n} a_i$$

find a binary vector

$$\mathbf{x} = (x_i)_i \text{ with } \mathbf{x} \cdot \mathbf{a} = \sum_{i=1}^{n} x_i a_i = s. \tag{1}$$

We define the density of the previous knapsack problem:

$$d = \frac{n}{\log_2 \max_i \{a_i\}_i}.$$

The Knapsack Problem

Given a list of $n+1$ positive integers $\{a_1, ..., a_n, s\}$ such that,

$$\max\{a_i\}_i \leq s \leq \sum_{i=1}^{n} a_i$$

find a binary vector

$$\mathbf{x} = (x_i)_i \text{ with } \mathbf{x} \cdot \mathbf{a} = \sum_{i=1}^{n} x_i a_i = s. \tag{1}$$

We define the density of the previous knapsack problem:

$$d = \frac{n}{\log_2 \max_i\{a_i\}_i}.$$

In cryptography we are interested in d s.t $d < 1$.

The Knapsack Problem

Given a list of $n+1$ positive integers $\{a_1, ..., a_n, s\}$ such that,

$$\max\{a_i\}_i \leq s \leq \sum_{i=1}^{n} a_i$$

find a binary vector

$$\mathbf{x} = (x_i)_i \text{ with } \mathbf{x} \cdot \mathbf{a} = \sum_{i=1}^{n} x_i a_i = s. \tag{1}$$

We define the density of the previous knapsack problem:

$$d = \frac{n}{\log_2 \max_i\{a_i\}_i}.$$

In cryptography we are interested in d s.t $d < 1$.

The decision version is known to be NP-complete.

An example of a knapsack cryptosystem

Merkle-Hellman (1978)

The first cryptosystem based on knapsack problem is the Merkle-Hellman, which widely broken by Shamir after three years.
Assume that we want to encrypt a message $\mathbf{m}$ of length $n-$bits, $\mathbf{m} = \{m_i\}$:

- Key Generation:
  Choose a super increasing vector $\mathbf{a} = \{a_i\}$, a number q s.t $\sum_i a_i < q$, a number r s.t $\gcd(r, q) = 1$.
- Public key: b_i such that $b_i \equiv ra_i \pmod q$
- Private key: $(\{a_i\}_i, q, r)$

An example of a knapsack cryptosystem

Merkle-Hellman (1978)

The first cryptosystem based on knapsack problem is the Merkle-Hellman, which widely broken by Shamir after three years. Assume that we want to encrypt a message $\mathbf{m}$ of length $n-$bits, $\mathbf{m} = \{m_i\}$:

- Key Generation:
  Choose a super increasing vector $\mathbf{a} = \{a_i\}$, a number q s.t $\sum_i a_i < q$, a number r s.t $\gcd(r, q) = 1$.
- Public key: b_i such that $b_i \equiv ra_i \pmod{q}$
- Private key: $(\{a_i\}_i, q, r)$
- Encryption: $C = \sum m_i b_i$ (hard knapsack)
- Decryption: $C' \equiv Cr^{-1} \equiv \sum m_i b_i r^{-1} \equiv \sum m_i a_i \pmod{q}$ (easy to solve)

An example of a knapsack cryptosystem

- Another knapsack-type cryptosystems :

An example of a knapsack cryptosystem

- Another knapsack-type cryptosystems :
- Chor-Rivest cryptosystem (1988)

An example of a knapsack cryptosystem

- Another knapsack-type cryptosystems :
- Chor-Rivest cryptosystem (1988)
- OTU cryptosystem (Okamoto, Tanaka and Uchiyama, 2000)

Lattices

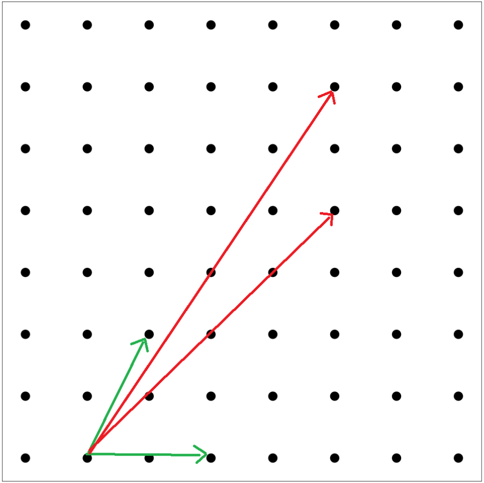
We remind some basic things about lattices.

Definition

A subset $L \subset \mathbf{R}^n$ is called a lattice if there exist linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k$ of $\mathbf{R}^n$ such that

$$L(B) = \left\{ \sum_{j=1}^{k} \alpha_j \mathbf{b}_j : \alpha_j \in \mathbb{Z}, 1 \leq j \leq k \right\} := L(\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k).$$

The set of vector vectors $\{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k\}$ is called a lattice basis of L.



Good and bad basis

The LLL reduction Algorithm

- In 1982, A.Lenstra, H.Lenstra and Lovasz published in their landmark paper the LLL-algorithm, which is a basis reduction algorithm for lattices, based on Hermite's inequality. The aim of LLL algorithm is to find a short vector that approximates the shortest nonzero vector of the lattice. The LLL algorithm runs in polynomial time as a function of the rank of the lattice.

The LLL reduction Algorithm

- The LLL-algorithm achieve to give us a small length vector, more specific
$$||\mathbf{b}_1|| \leq 2^{(k-1)/2}\lambda_1(L),$$
where $\lambda_1(L)$ is the length of a shortest vector of the lattice L.

The LLL reduction Algorithm

- The LLL-algorithm achieve to give us a small length vector, more specific

$$||\mathbf{b}_1|| \leq 2^{(k-1)/2}\lambda_1(L),$$

where $\lambda_1(L)$ is the length of a shortest vector of the lattice L.

- Some apllications are:
  - to the factorization of polynomials over $\mathbf{Z}[x]$ and finite fileds
  - to find integer relations between some real numbers $w_1, ..., w_k$.
  - Solve approximate closest vector problem (Babai's algorithm)
  - Cryptanalysis of RSA (Coppersmith's method)

The LLL reduction Algorithm

- LLL algorithm consists from two basic subroutines $red(k, \ell)$ and $swap(k)$.

The LLL reduction Algorithm

- LLL algorithm consists from two basic subroutines *red*(k, ℓ) and *swap*(k).
- Let $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be a basis of our Lattice and $\{\mathbf{b}_1^*, ..., \mathbf{b}_n^*\}$ the Gramm-Schimdt Orhogonal vectors.

The LLL reduction Algorithm

- LLL algorithm consists from two basic subroutines $red(k, \ell)$ and $swap(k)$.
- Let $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be a basis of our Lattice and $\{\mathbf{b}_1^*, ..., \mathbf{b}_n^*\}$ the Gramm-Schimdt Orhogonal vectors.
- $red(k, \ell)$ consists from the following steps.

  for $k = 1, 2, ..., \ell - 1$ do

  if $|\mu_{k\ell}| = \left|\frac{\mathbf{b}_k \cdot \mathbf{b}_\ell^*}{B_k}\right| < 1/2$ where $B_k = \mathbf{b}_k^{*2}$ then set

  $r = \lceil \mu_{k\ell} \rfloor, \quad \mathbf{b}_k \leftarrow \mathbf{b}_k - r\mathbf{b}_\ell$

  $\mu_{kj} \leftarrow \mu_{kj} - r\mu_{\ell j}$

The LLL reduction Algorithm

- The second subroutine is called $swap(k)$

The LLL reduction Algorithm

- The second subroutine is called *swap*(k)
- this make a swap between two vectors $\mathbf{b}_k, \mathbf{b}_{k+1}$ if the Lovasz condition (for $\delta = 3/4$) does not hold, that is :

$$\frac{3}{4}|\mathbf{b}^*_{k-1}|^2 > |\mathbf{b}^*_k + \mu_{k,k-1}\mathbf{b}^*_{k-1}|^2$$

The LLL reduction Algorithm

- The second subroutine is called *swap*(k)
- this make a swap between two vectors $\mathbf{b}_k, \mathbf{b}_{k+1}$ if the Lovasz condition (for $\delta = 3/4$) does not hold, that is :

$$\frac{3}{4}|\mathbf{b}^*_{k-1}|^2 > |\mathbf{b}^*_k + \mu_{k,k-1}\mathbf{b}^*_{k-1}|^2$$

- Now LLL alogirthm excute the following loop
  set $k = 2$
  while $k \leq n$ do
  red(k, ℓ) $\ell = k-2, ..., 1$ and then call *swap*(k).

LLL & BKZ

- Another reduction algorithm similar to LLL is the BKZ.

LLL & BKZ

- Another reduction algorithm similar to LLL is the BKZ.
- Unfortunately, BKZ is not polynomial in general.

LLL & BKZ

Consider a basis B from a lattice L. Both algorithms calculate a reduced-basis with short, nearly orthogonal vectors from B.

LLL(1982)

- terminates in polynomial time
- good basis for small dimensions
- exponential dependence on the dimension
- behave much better than the worst-case analysis according to experiments

BKZ(1987)

- non polynomial (there is no good bound on the time complexity)
- better basis for large n (> 40)
- enumerate all lattice points within a certain radius. Prune to decrease the branches

Schroeppel-Shamir algorithm

Schroeppel-Shamir algorithm

Time complexity: $\tilde{O}(2^{n/2})$
Memory requirement: $O(2^{n/4})$

Schroeppel-Shamir algorithm

Time complexity: $\tilde{O}(2^{n/2})$
Memory requirement: $O(2^{n/4})$
Basic idea:

$$S = \sum_{i=1}^{n} x_i a_i = \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4$$

We decompose the original knapsack to four smaller knapsack problems of $n/4$ elements. Now, for each value σ_M of $n/4-$bits we do the following. Construct two sorted lists $\{\sigma_2\}, \{\sigma_4\}$ and the two sets $\{\sigma_{12}\}, \{\sigma_{34}\}$ as follows.

$\sigma_{12} = \sigma_1 + \sigma_2 = \sigma_M \pmod{2^{n/4}}$ and
$\sigma_{34} = \sigma_3 + \sigma_4 = S - \sigma_M \pmod{2^{n/4}}$

$$\left.\right\} \begin{array}{l} \sigma_{12} = \sigma_M \mod 2^{n/4} \\ \sigma_{34} = S - \sigma_M \mod 2^{n/4} \\ \text{Searching for collision: } \{\sigma_{12}\} \text{ and} \\ \{S - \sigma_{34}\} \end{array}$$

Becker, Coron and Joux Algorithm

(Heuristic) Running time $O(2^{0.291n})$
Memory requirement: $O(2^{0.256n})$
Basic idea: $\mathbf{y}$, $\mathbf{z}$: $\in \{-1, 0, 1\}$
Decomposition:

$$\sum_{i=1}^{n} a_i y_i = \sigma_1 = R \pmod{M} \qquad \sum_{i=1}^{n} a_i z_i = \sigma_2 = S - R \pmod{M}$$

The solution in this case is:

$$x_i = \begin{cases} 0 & (y_i, z_i) = (0,0) \text{ or } (-1,1) \text{ or } (1,-1) \\ 1 & (y_i, z_i) = (1,0) \text{ or } (0,1) \\ \text{no result} & (y,z) = (1,1) \text{ or } (-1,-1) \end{cases}$$

Lattice-based attacks

- The fist attack based on lattices was given by Lagarias-Odlyzko [Solving low-density subset sum problems - 1985] Assuming a SVP-oracle, they proved that all the knapsacks of density < 0.645 can be easily solved.

Lattice-based attacks

- The fist attack based on lattices was given by Lagarias-Odlyzko [Solving low-density subset sum problems - 1985] Assuming a SVP-oracle, they proved that all the knapsacks of density < 0.645 can be easily solved.
- The previous attack was improved Coster-Joux-LaMacchia-Odlyzko-Schnorr-Stern [Improved low-density subset sum algorithms]. They managed to increase d to 0.94.

Lattice-based attacks

- The fist attack based on lattices was given by Lagarias-Odlyzko [Solving low-density subset sum problems - 1985] Assuming a SVP-oracle, they proved that all the knapsacks of density < 0.645 can be easily solved.
- The previous attack was improved Coster-Joux-LaMacchia-Odlyzko-Schnorr-Stern [Improved low-density subset sum algorithms]. They managed to increase d to 0.94.
- In practice we do not have such SVP-oracles. Only for small dimensions (≤ 40) we have algorithms which behave as SVP-oracles.

Schnorr-Shevchenko Algorithm

Hamming weight $= n/2$: $\sum_{i=1}^{n} x_i = n/2$

We consider the lattice $L(B)$ generated from the rows of the following matrix.

$$B = \begin{bmatrix} 2 & 0 & \dots & 0 & Na_1 & 0 & N \\ 0 & 2 & \dots & 0 & Na_2 & 0 & N \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & 2 & Na_n & 0 & N \\ 1 & 1 & \dots & 1 & Ns & 1 & \frac{n}{2}N \end{bmatrix} \in \mathbb{Z}^{(n+1)(n+3)} \qquad (2)$$

In our examples: $n = 80$, $a_i \in [1, 2^n]$. $N = 16 > \sqrt{n}$.

Schnorr-Shevchenko Algorithm

Hamming weight $= n/2 : \sum_{i=1}^{n} x_i = n/2$

We consider the lattice $L(B)$ generated from the rows of the following matrix.

$$B = \begin{bmatrix} 2 & 0 & \dots & 0 & Na_1 & 0 & N \\ 0 & 2 & \dots & 0 & Na_2 & 0 & N \\ . & . & \dots & . & . & . & . \\ . & . & \dots & . & . & . & . \\ . & . & \dots & . & . & . & . \\ 0 & 0 & \dots & 2 & Na_n & 0 & N \\ 1 & 1 & \dots & 1 & Ns & 1 & \frac{n}{2}N \end{bmatrix} \in \mathbb{Z}^{(n+1)(n+3)} \qquad (2)$$

If $\mathbf{b} = (b_1, b_2, \dots, b_{n+3})$ in $L(B)$ provides a solution of the knapsack problem if and only if $|b_i| = 1$ ($i = 1, \dots, n$ and $n+2$) and $b_{n+1} = b_{n+3} = 0$.

Schnorr-Shevchenko Algorithm

The solution $\mathbf{x}$, can be found by : $x_i = \frac{|b_i - b_{n+2}|}{2}$, for $i = 1, 2, \ldots, n$

How does the algorithm work?

- First 5 steps: iteratively apply BKZ-reduction to the basis B with blocksizes 2^k for $k = 1, 2, 3, 4, 5$ (no pruning). Before that, permute the rows of the matrix in order to:
  - first rows have a nonzero element in column $n + 2$
  - sort the other rows according to their norm

  Terminate if the solution has been found.

Schnorr-Shevchenko Algorithm

The solution $\mathbf{x}$, can be found by : $x_i = \frac{|b_i - b_{n+2}|}{2}$, for $i = 1, 2, \ldots, n$

How does the algorithm work?

- First 5 steps: iteratively apply BKZ-reduction to the basis B with blocksizes 2^k for $k = 1, 2, 3, 4, 5$ (no pruning). Before that, permute the rows of the matrix in order to:
  - first rows have a nonzero element in column $n + 2$
  - sort the other rows according to their norm

  Terminate if the solution has been found.
- If we have no solution in the first 5 steps, BKZ-reduce the basis independently with blocksizes: $30, 31, 32, \ldots, 62$ (32 steps) and pruning parameter $10, 11, 12, 10, 11 \ldots etc.$ Terminate if the solution has been found (in this step we assumed that $70 < n \leq 80$).

First Variant

Idea: reduce the original knapsack problem to some easier problems, i.e. having smaller dimension and density

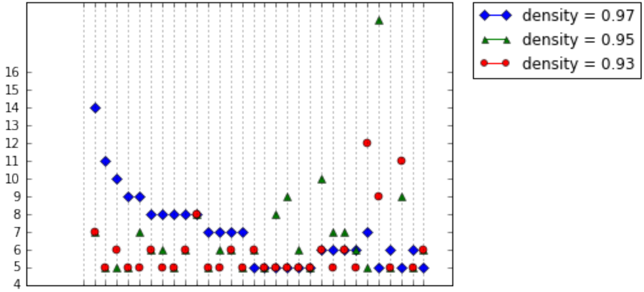


Figure : Relation of the density of random knapsacks (of dimension 72) and the number of rounds that method SS executes until success.

First Variant

Execute SS algorithm until round $= 5$

First Variant

Execute SS algorithm until round $= 5$

Brute force on the four initial bits of the solution and for each value reduce the initial knapsack to some with smaller densities and dimension. Then apply SS's method until (overall) round 11.

First Variant

Execute SS algorithm until round $= 5$

Brute force on the four initial bits of the solution and for each value reduce the initial knapsack to some with smaller densities and dimension. Then apply SS's method until (overall) round 11.

Example: we assume that the solution is of the form $[0, 0, 0, 0, ...]$. Then we consider $\sum_{j=5}^{n} a_j x_j = s$.
Run SS's method until the 11th round. If it fails, then we take $[0, 0, 0, 1]$ as the initial bits and the reduced knapsack is $\sum_{j=5}^{n} a_j x_j = s - a_4$. Execute again SS and so on, until the solution is found.

First Variant

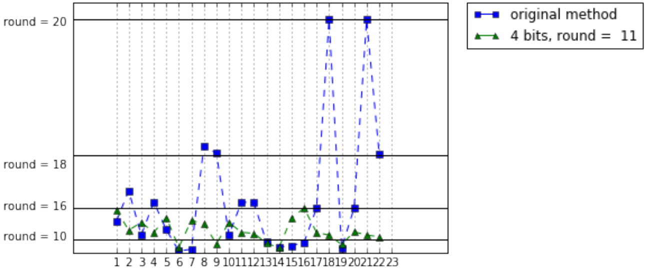


Figure : We compare the times of SS's Method and the variant for some randomly chosen instances of knapsack problem with dimension = 80 and density close to 1.

Second Variant

Remind that

$$d = \frac{n}{\log_2 \max_i \{a_i\}_i}$$

Idea: increase the denominator $\Leftrightarrow$ decrease the density.

Second Variant

Remind that

$$d = \frac{n}{\log_2 \max_i\{a_i\}_i}$$

Idea: increase the denominator $\Leftrightarrow$ decrease the density.
Let $b_n = 2^{n+6}$ be a fixed number Also assume that $x_1 = 1$.
Substitute $a_1 \leftarrow a_1 + b_n$ and $s \leftarrow s + b_n$, then the new density is:

$$d' = \frac{n}{\log_2 \max_i\{a_i\}_i} = \frac{n}{n+6} = \frac{1}{1+6/n} < 1$$

$$(because \ a_i's < 2^n, \ for \ i \geq 2)$$

Second Variant

Remind that

$$d = \frac{n}{\log_2 \max_i\{a_i\}_i}$$

Idea: increase the denominator $\Leftrightarrow$ decrease the density.
Let $b_n = 2^{n+6}$ be a fixed number Also assume that $x_1 = 1$.
Substitute $a_1 \leftarrow a_1 + b_n$ and $s \leftarrow s + b_n$, then the new density is:

$$d' = \frac{n}{\log_2 \max_i\{a_i\}_i} = \frac{n}{n+6} = \frac{1}{1+6/n} < 1$$

$$(\textit{because}\ a_i's < 2^n,\ \textit{for}\ i \geq 2)$$

For example: if $n = 80$ we get $d' \approx 0.93$.

Second Variant

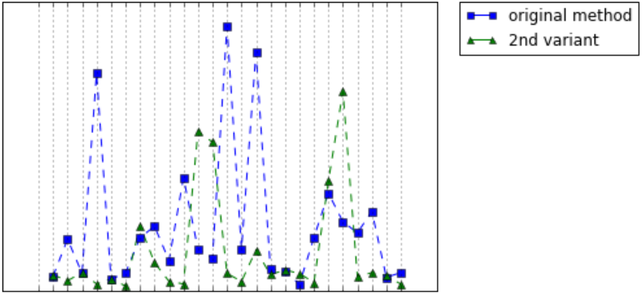


Figure : We compare the SS Method with the second variant for randomly chosen instances of knapsack problem with dimension 80 and density 1.

Compact knapsack

What is a compact knapsack?

Consider instead of the search space $\{0,1\}$ the larger space $\{0,1,...,2^{\delta n}\}$ $(\delta > 0)$, then we take a generalized knapsack problem:

$$s = \sum_{i=1}^{n} a_i x_i, \; x_i \in \{0,1,\ldots,2^{\delta n}\}$$

Compact knapsack

What is a compact knapsack?

Consider instead of the search space $\{0,1\}$ the larger space $\{0,1,...,2^{\delta n}\}$ $(\delta > 0)$, then we take a generalized knapsack problem:

$$s = \sum_{i=1}^{n} a_i x_i, \ \ x_i \in \{0,1,\ldots,2^{\delta n}\}$$

Constraints of solution: $|x_j| \leq X_j$, $(X_j \in \mathbb{Z}_{>0})$, NP-complete problem.

Compact knapsack

What is a compact knapsack?

Consider instead of the search space $\{0,1\}$ the larger space $\{0,1,...,2^{\delta n}\}$ $(\delta > 0)$, then we take a generalized knapsack problem:

$$s = \sum_{i=1}^{n} a_i x_i, \ \ x_i \in \{0,1,\dots,2^{\delta n}\}$$

Constraints of solution: $|x_j| \leq X_j$, $(X_j \in \mathbb{Z}_{>0})$, NP-complete problem.

The only two methods that exist are based on lattices :

1) K. Aardal, C. Hurkens, A. Lenstra, *Solving a linear Diophantine equation with lower and upper bounds on the variables*
2) K.A. Draziotis, *Balanced Integer solutions of linear equations*

Compact knapsack-new method

We use a modification of SS algorithm $\rightarrow$ we take small and balanced integer solutions of this problem.
Consider the lattice $L(B)$,

$$B = \begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & Na_1 & 0 \\ 0 & 2 & 0 & \cdots & 0 & Na_2 & 0 \\ . & . & . & \cdots & . & . & . \\ . & . & . & \cdots & . & . & . \\ 0 & 0 & 0 & \cdots & 2 & Na_n & 0 \\ 1 & 1 & 1 & \cdots & 1 & Na_0 & 1 \end{bmatrix},$$

and apply SS method until round 11.
We get fast a better solution than the two previous methods

Algorithm

The function *check*$(B, \mathbf{v})$, for checking our candidate solution
Input. $B, \mathbf{v} = (v_1, ..., v_{n+2}) \in L(B)$
Output. Returns True if there is a solution $(x_j)_j$ such that $\sum_{j=1}^{n} a_j x_j = a_0$. Else the algorithm returns False.

if $v_{n+1} \leftarrow 0$ **then**
  if $v_{n+2} = -1$ **then**
    $x_i \leftarrow \frac{v_i - v_{n+2}}{2} \quad (i = 1, 2, ..., n)$
    return True and $(x_j)_j$
  if $v_{n+2} = 1$ **then**
    $x_i \leftarrow \frac{v_{n+2} - v_i}{2} \quad (i = 1, 2, ..., n)$
  return True and $(x_j)_j$
else
  return False

Algorithm

SS variant for compact knapsacks
Input. $(a_1, \ldots, a_n; a_0; N) \in \mathbb{Z}_{>0}^{n+2}$, $N > \sqrt{n}$.
Output. Small solutions $(x_j)_j$ of $\sum_{j=1}^{n} a_j x_j = a_0$.

for $j = 1$ *to* 11 **do**
  if $j \leq 5$ **then**
    $B_p \leftarrow$permutation(B_r, n), ($B_r = B$ at the beginning)
    $B_r \leftarrow BKZ(B_p, blocksize = 2^j)$
    $\mathbf{b}_1 \leftarrow$ first row of B_r
    if $check(B_r, \mathbf{b}_1) = True$ **then**
      $solution \leftarrow \mathbf{b}_1$
  if $j > 5$ **then**
    $B_r \leftarrow BKZ(B, blocksize = 30 + j - 6, pruning \in \{30, 31, 32\})$
    $\mathbf{b}_1 \leftarrow$ first row of B_r
    if $check(B_r, \mathbf{b}_1) = True$ **then**
      $solution \leftarrow \mathbf{b}_1$
return *solution*

Results from the experiments

In all the examples we run, the solution of the previous algorithm was better most of the times but never worst, than the solution given in references 1 and 2.

Branch and Bound algorithm (BB)

In order to find solutions in a specific range, say $2^{R-1} < x_j < 2^R$, the previous method does not work.
We shall use a *branch and bound algorithm* in order to attack this problem :

Branch and Bound algorithm (BB)

In order to find solutions in a specific range, say $2^{R-1} < x_j < 2^R$, the previous method does not work.
We shall use a *branch and bound algorithm* in order to attack this problem :

- First apply either the method of the previous section (BKZ) or one of the method 1,2. These methods return a matrix with rows the following vector,

$$L = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$$

Branch and Bound algorithm (BB)

In order to find solutions in a specific range, say $2^{R-1} < x_j < 2^R$, the previous method does not work.
We shall use a *branch and bound algorithm* in order to attack this problem :

- First apply either the method of the previous section (BKZ) or one of the method 1,2. These methods return a matrix with rows the following vector,

$$L = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$$

- Separate the last vector $\mathbf{b_n}$ of L, which provide us with a small balanced solution of the equation $\sum a_i x_i = s$.

Branch and Bound algorithm (BB)

In order to find solutions in a specific range, say $2^{R-1} < x_j < 2^R$, the previous method does not work.
We shall use a *branch and bound algorithm* in order to attack this problem :

- First apply either the method of the previous section (BKZ) or one of the method 1,2. These methods return a matrix with rows the following vector,

$$L = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$$

- Separate the last vector $\mathbf{b_n}$ of L, which provide us with a small balanced solution of the equation $\sum a_i x_i = s$.
- The other rows, $\{\mathbf{b_1}, ..., \mathbf{b_{n-1}}\}$,constitute a basis of the lattice $\{\mathbf{y} \in \mathbb{Z}^n : \sum_{j=1}^{n} a_j y_j = 0\}$.

Branch and Bound algorithm (BB)

In order to find solutions in a specific range, say $2^{R-1} < x_j < 2^R$, the previous method does not work.
We shall use a *branch and bound algorithm* in order to attack this problem :

- First apply either the method of the previous section (BKZ) or one of the method 1,2. These methods return a matrix with rows the following vector,

$$L = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$$

- Separate the last vector $\mathbf{b_n}$ of L, which provide us with a small balanced solution of the equation $\sum a_i x_i = s$.
- The other rows, $\{\mathbf{b_1}, ..., \mathbf{b_{n-1}}\}$,constitute a basis of the lattice $\{\mathbf{y} \in \mathbb{Z}^n : \sum_{j=1}^{n} a_j y_j = 0\}$.
- Our algorithm uses this basis to get a new solution $\mathbf{x}'$, that satisfies the bound constraints.

Branch and Bound algorithm (BB)

We use the following heuristic. We set $e = \max_j x_j - \min_j x_j$ $(j = 1, 2, ..., n)$.

Branch and Bound algorithm (BB)

We use the following heuristic. We set $e = \max_j x_j - \min_j x_j$ $(j = 1, 2, ..., n)$.

- Starting from a solution $\mathbf{x}$ that does not meet the constraints, we update it using the relation $\mathbf{x}' \leftarrow \mathbf{x} + k\mathbf{b}_j$ for $1 \leq j \leq n-1$ and for some $k \in \mathbb{Z} - \{0\}$.

Branch and Bound algorithm (BB)

We use the following heuristic. We set $e = \max_j x_j - \min_j x_j$ $(j = 1, 2, ..., n)$.

- Starting from a solution $\mathbf{x}$ that does not meet the constraints, we update it using the relation $\mathbf{x}' \leftarrow \mathbf{x} + k\mathbf{b}_j$ for $1 \leq j \leq n-1$ and for some $k \in \mathbb{Z} - \{0\}$.
- In every step we calculate the number v of all $x_j < 2^{R-1}$ (number of lower violated bounds) and the number z of all $x_j > 2^R - 1$ (number of upper violated bounds).

Branch and Bound algorithm (BB)

We use the following heuristic. We set $e = \max_j x_j - \min_j x_j$ $(j = 1, 2, ..., n)$.

- Starting from a solution $\mathbf{x}$ that does not meet the constraints, we update it using the relation $\mathbf{x}' \leftarrow \mathbf{x} + k\mathbf{b}_j$ for $1 \leq j \leq n-1$ and for some $k \in \mathbb{Z} - \{0\}$.
- In every step we calculate the number v of all $x_j < 2^{R-1}$ (number of lower violated bounds) and the number z of all $x_j > 2^R - 1$ (number of upper violated bounds).
- If the new e', v', z' satisfy $e' \leq e, v' \leq v, z' \leq z$, then we update the solution $\mathbf{x}$ and increase (or decrease) k for a better solution, else we use another vector $\mathbf{b}_j$ from the basis.

Branch and Bound algorithm (BB)

Algorithm

The function *Update*

Input. $L = \{\mathbf{b_j}\}_j, \mathbf{x}, n, v, z, K, flag \in \{-1, 1\}$, where

$v =$number of lower violated bounds

$z =$number of upper violated bounds

L is the lattice basis of the homogeneous linear equation and K is an upper bound for $|k|$.

Output. A solution $\mathbf{x}$ that satisfies more constraints than the initial solution $\mathbf{x}$.

Branch and Bound algorithm (BB)

$k \leftarrow flag$
$i \leftarrow 1$
while $i \leq n-1$ *and* $|k| \leq K$ **do**
  $e \leftarrow \max_j(x_j) - \min_j(x_j)$ $(j = 1, 2, \ldots, n)$
  $\mathbf{x}' \leftarrow \mathbf{x} + k\mathbf{b}_i$
  find the new e', v', z'
  if $e' \leq e$ *and* $v' \leq v$ *and* $z' \leq z$ **then**
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $k \leftarrow k + flag$
  else
    $i \leftarrow i + 1$
    $k \leftarrow flag$
  end
end

Branch and Bound algorithm (BB)

Algorithm

Branch and bound for compact knapsacks
Input. $L = \{\mathbf{b_j}\}_j, \mathbf{x}, n, R, K$
Output. A solution $\mathbf{x}$ s.t. $2^{R-1} \leq x_j \leq 2^R - 1$ or a better solution which satisfies more constraints.

h=1
while $v > 0$ *or* $z > 0$ *or* $h < K$ **do**
  $h \leftarrow h + 1$
  UPDATE$(L, \mathbf{x}, v, z, K, 1)$
  if $v > 0$ *or* $z > 0$ **then**
    UPDATE$(L, \mathbf{x}, v, z, K, -1)$
  end
end
return $\mathbf{x}$

Branch and Bound algorithm (BB)

We studied 150 random instances for some n and R, with $K = 50$, which gave us the following remarks:

n	10	10	20	20	40	40	55
R	20	25	20	25	20	25	20
Suc. rate	53%	56%	28%	20%	5%	4%	3%

Table : We randomly choose the parameters $(a_j)_j$ having $R-$bits

n	10	10	10	20	20	20		20	20	26	30
R	20	25	30	20	25	30		30	36	36	36
Suc. rate	25%	12%	10%	5%	4%	0%		5%	3%	2%	0%

Table : For the first six columns, we chose the parameters $(a_j)_j$ such that, half of them have $R-$ bits and the other half have $R/2$-bits. For the rest columns, we chose the parameters $(a_j)_j$ such that, half of them

An ID System based on compact knapsack

Finally, we provide a three move id-scheme based on compact knapsack problem. We shall prove that this system is sound, which is the minimal notion of security for id-schemes.

An ID System based on compact knapsack

Finally, we provide a three move id-scheme based on compact knapsack problem. We shall prove that this system is sound, which is the minimal notion of security for id-schemes.

In public key Id (Identification) protocols an entity (the prover) holding a secret key wants to prove its identity to an entity (the verifier) holding only the public key. We are interested in three moves Id-schemes.

An ID System based on compact knapsack

Finally, we provide a three move id-scheme based on compact knapsack problem. We shall prove that this system is sound, which is the minimal notion of security for id-schemes.

In public key Id (Identification) protocols an entity (the prover) holding a secret key wants to prove its identity to an entity (the verifier) holding only the public key. We are interested in three moves Id-schemes.

Here Alice (the prover) holds a secret key and sends to Bob (the verifier) a message which we call *commitment*. Bob responds with a random string which we call it *challenge* (or *exam*). Alice provides a *response*.

An ID System based on compact knapsack

Finally, Bob applies a verification algorithm which has as input, the public key of Alice and the previous conversation, in order to decide if he will accept or reject the id of Alice. The length of the challenge is the security parameter.

An ID System based on compact knapsack

Finally, Bob applies a verification algorithm which has as input, the public key of Alice and the previous conversation, in order to decide if he will accept or reject the id of Alice. The length of the challenge is the security parameter.

We shall provide an id scheme which is not based on discrete logarithm or factorization problem, but on the compact knapsack problem i.e. we provide a proof of knowledge for the compact knapsack problem.

An ID System based on compact knapsack

We consider the compact knapsack problem,

$$\sum_{j=1}^{n} a_j x_j = b, \qquad (3)$$

for $x_j \in I_{R+\ell}$, where $I_{R+\ell}$ the set of integers having $(R+\ell)-$bits (R, ℓ are positive integers) and a_j positive integers with length at most $R-$bits.

An ID System based on compact knapsack

Alice is the prover and Bob is the verifier of the scheme. Alice picks a vector $\mathbf{a}$ with positive integer numbers as entries and a vector $\mathbf{x} \in I^n_{R+\ell}$, such that $\mathbf{a} \cdot \mathbf{x} = b$.

Public key: $(\mathbf{a}, b, R, \ell)$. Private key: $\mathbf{x}$

The following scheme is repeated $t-$times (we'll see below that t$\approx$ 80 in order the system be secure):

An ID System based on compact knapsack problem

- Alice picks a random vector $\mathbf{k} \in I_R^n$ (where I_R is the set of positive integers with $R-$ bits). Then she computes

$$\mathbf{a} \cdot \mathbf{k} = r$$

and sends it to Bob (*commitment*).

An ID System based on compact knapsack problem

• Alice picks a random vector $\mathbf{k} \in I_R^n$ (where I_R is the set of positive integers with $R-$ bits). Then she computes

$$\mathbf{a} \cdot \mathbf{k} = r$$

and sends it to Bob (*commitment*).
$\diamondsuit$ Bob picks a random integer $e \in \{0, 1\}$ and sends it to Alice (*challenge*).

An ID System based on compact knapsack problem

• Alice picks a random vector $\mathbf{k} \in I_R^n$ (where I_R is the set of positive integers with $R-$ bits). Then she computes

$$\mathbf{a} \cdot \mathbf{k} = r$$

and sends it to Bob (*commitment*).
♢ Bob picks a random integer $e \in \{0, 1\}$ and sends it to Alice (*challenge*).
• Alice computes

$$\mathbf{s} = \mathbf{k} + e\mathbf{x}$$

and sends $\mathbf{s}$ to Bob (*response*).

An ID System based on compact knapsack problem

• Alice picks a random vector $\mathbf{k} \in I_R^n$ (where I_R is the set of positive integers with $R-$ bits). Then she computes

$$\mathbf{a} \cdot \mathbf{k} = r$$

and sends it to Bob (*commitment*).

$\diamondsuit$ Bob picks a random integer $e \in \{0, 1\}$ and sends it to Alice (*challenge*).

• Alice computes

$$\mathbf{s} = \mathbf{k} + e\mathbf{x}$$

and sends $\mathbf{s}$ to Bob (*response*).

$\diamondsuit$ Bob verifies the equality $\mathbf{a} \cdot \mathbf{s} = r + eb$ and that $\mathbf{s} \in I_R^n$ if $e = 0$. Now, if $e = 1$ Alice can choose from the beginning R, ℓ such that $\mathbf{s} \in I_{R+\ell}^n$ with large probability (≈ 1).

An ID System

Proof of correctness.

$$\mathbf{a} \cdot \mathbf{s} \equiv \mathbf{a} \cdot \mathbf{k} + e\mathbf{a} \cdot \mathbf{x} \equiv r + eb.$$

Also, $\mathbf{s}$ satisfies the constraints of the scheme. $\square$

Remark

To be precise the previous scheme is a probabilistic id-scheme, since Bob is convinced with high probability.

Soundness

The minimal notion of the security in id-schemes is the security under passive attacks. The purpose of this attack is to find the secret key knowing only the public key of the system.

Soundness

The minimal notion of the security in id-schemes is the security under passive attacks. The purpose of this attack is to find the secret key knowing only the public key of the system.

Let Eve be an adversary. The scheme is sound if Eve knowing only the public key, can pass the verification test with only negligible probability.

Soundness

The minimal notion of the security in id-schemes is the security under passive attacks. The purpose of this attack is to find the secret key knowing only the public key of the system.

Let Eve be an adversary. The scheme is sound if Eve knowing only the public key, can pass the verification test with only negligible probability.

The soundness of the scheme depends on the value t.

Soundness

Assume for simplicity $t = 1$, i.e. we apply the scheme one time only. Say that Eve, by tossing up a fair coin, picks the right $e' \in \{0, 1\}$. Then, she computes a random vector $\mathbf{s} \in I_R^n$ if $e' = 0$ else she chooses $\mathbf{s} \in I_{R+\ell}^n$. Then she sends to Bob the pair $(r = \mathbf{a} \cdot \mathbf{s}, \mathbf{s})$ if $e' = 0$ and $(r = \mathbf{a} \cdot \mathbf{s} - b, \mathbf{s})$ if $e' = 1$. In each case this pair passes the verification test since $\mathbf{a} \cdot \mathbf{s} = r + e'b$. The success rate is $1/2$. In general is 2^{-t}. So for $t = 80$ the success rate is negligible.

Soundness

To prove that the system is sound, under the assumption that compact knapsack problem is difficult for some parameters $(n, (a_j)_j, b)$, we need to prove that if Eve chooses with probability $\varepsilon > 2^{-t}$ again she can not improve the success rate of the previous attack.

Soundness

To prove that the system is sound, under the assumption that compact knapsack problem is difficult for some parameters $(n, (a_j)_j, b)$, we need to prove that if Eve chooses with probability $\varepsilon > 2^{-t}$ again she can not improve the success rate of the previous attack.

The first result we need is the special soundness. That is, if we have two queries of the form $(\mathbf{r}, \mathbf{e}, \mathbf{s}_i)$, $(\mathbf{r}, \mathbf{e}', \mathbf{s}_i')$ then, we can find a solution of $\sum_{i=1}^{n} a_i x_i = b$, with $\mathbf{x} \in I_{R+\ell}^{n}$. This can be done for suitable choice of the parameter ℓ.

We use first the following Lemma.

Lemma

Let $\mathbf{k}$ *and* $\mathbf{x}$ *are randomly chosen from* $I_R^n \times I_{R+\ell}^n$. *Let also* $\ell = \lfloor \log_2 3 - 1 - \log_2(1 - p^{1/n}) \rfloor$ *and* $p \approx 1$, *but* $p < 1$. *It turns out*

$$Pr(\mathbf{x} - \mathbf{k} \in I_{R+\ell}^n), Pr(\mathbf{x} + \mathbf{k} \in I_{R+\ell}^n) \approx p.$$

Then, we easily get special soundness.

Proposition

(special soundness). If we have two queries $(\mathbf{r}, \mathbf{e}, \mathbf{s}_i)$, $(\mathbf{r}, \mathbf{e}', \mathbf{s}_i')$ $(i = 1, 2, ..., t)$ *and* $\mathbf{e} \neq \mathbf{e}'$, *then for suitable choice of* ℓ *we can efficiently find a solution in* Σ_b *with high probability (where the set* $\Sigma_b = \{\mathbf{x} \in I_{R+\ell} : \sum_{i=1}^n a_i x_i = b\}$*).*

Soundness

Then, to prove the soundness property of the system we make use of a parallel Monte Carlo algorithm $\mathcal{A}$. This algorithm accepts as input the public key and a random vector $\mathbf{e}$ and outputs $t-$ passing pairs with probability $\varepsilon > 2^{-t+1}$.

Soundness

Then, to prove the soundness property of the system we make use of a parallel Monte Carlo algorithm $\mathcal{A}$. This algorithm accepts as input the public key and a random vector $\mathbf{e}$ and outputs $t-$ passing pairs with probability $\varepsilon > 2^{-t+1}$.

We can prove this by following a variant of the the soundness proof of Schnorr id scheme.

Soundness

Theorem

Let $\mathcal{A}$ be a probabilistic algorithm having as inputs the public key of the id-scheme and a random vector $\mathbf{e} \in \{0,1\}^t$, and output $t-$passing pairs $(\mathbf{r}, (\mathbf{s}_i)_i) = ((r_i)_i, (\mathbf{s}_i)_i)$, with probability $\varepsilon > 2^{-t+1}$. Suppose that $\ell = \lfloor 0.58 - \log_2(1 - 0.99^{1/n}) \rfloor$. Then, with constant probability and running time $O(|\mathcal{A}|/\varepsilon)$ we can find an element of Σ_b (which is equivalent to knowing the secret key $\mathbf{x}$).

Conclusions

- In this work we addressed knapsack problem and some of its variants. We optimized the SS algorithm presented using some heuristics that are working in practice. These two variants are better in average than the original method.

Conclusions

- In this work we addressed knapsack problem and some of its variants. We optimized the SS algorithm presented using some heuristics that are working in practice.These two variants are better in average than the original method.
- We considered the compact knapsack problem and we adapt the SS algorithm in the problem of finding small solutions in linear diophantine equations. Furthermore, we provided a branch and bound algorithm for the compact knapsack problem.

Conclusions

- In this work we addressed knapsack problem and some of its variants. We optimized the SS algorithm presented using some heuristics that are working in practice.These two variants are better in average than the original method.
- We considered the compact knapsack problem and we adapt the SS algorithm in the problem of finding small solutions in linear diophantine equations. Furthermore, we provided a branch and bound algorithm for the compact knapsack problem.
- We used compact knapsack problem to construct a three move id-scheme. This scheme is lightweight , because it does not use exponentiations. Also, is potentially quantum resistant since does not use factorization or discrete logarithms.

Conclusions

- Furthermore, the system can be easily transformed to a digital signature using the Fiat-Shamir transformation. Here, someone can apply forking lemma to check the security of the resulting signature scheme.

Thank you!