A Fast Approximation Algorithm for the Subset-Sum Problem

Bartosz Przydatek[*]
Institute of Theoretical Computer Science, ETH Zurich

Abstract

The subset-sum problem (SSP) is defined as follows: given a positive integer bound and a set of n positive integers find a subset whose sum is closest to, but not greater than, the bound. We present a *randomized* approximation algorithm for this problem with linear space complexity and time complexity of $\mathcal{O}(n \log n)$. Experiments with random uniformly-distributed instances of SSP show that our algorithm outperforms, both in running time and average error, Martello and Toth's [MT84] quadratic greedy search, whose time complexity is $\mathcal{O}(n^2)$. We propose conjectures on the expected error of our algorithm for uniformly-distributed instances of SSP and provide some analytical arguments justifying these conjectures. We present also results of numerous tests.

1 Introduction

The subset-sum problem (SSP) is a special case of the knapsack problem and is defined as follows: given a set[1] of positive integers $\{a_1, \ldots, a_n\}$, $n \geq 1$, and a positive integer B (the bound), find a subset of the a_i's such that their sum is as close as possible to B, without exceeding B.

This problem is NP-hard (see e.g. [GJ79]), therefore finding an optimal solution for big n and big values of a_i's is currently not feasible. In practice however one is often satisfied with an approximate solution, which can be found quite efficiently (i.e. in polynomial time).

We present a new, randomized approximation algorithm for the SSP which runs in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(n)$ space. Tests with random uniformly-distributed instances of SSP show that our algorithm performs significantly better than the best algorithms known so far.

1.1 An Alternative Formulation of the SSP and Some Notation

The subset-sum problem can be defined equivalently as: given $[a_1, \ldots, a_n] \in \mathbb{N}^n$ and $B \in \mathbb{N}$, find a vector $\mathbf{x} = [x_1, \ldots, x_n]$, with $x_i \in \{0, 1\}$, $i = 1, \ldots, n$, which

$$\begin{aligned} &\text{maximizes} \quad && \sum_{i=1}^{n} a_i x_i \\ &\text{subject to} \quad && \sum_{i=1}^{n} a_i x_i \leq B \ . \end{aligned}$$

[*]e-mail address: `przydatek@inf.ethz.ch`

[1]The considered sets are actually *multi-sets* since the numbers a_i do not have to be distinct.

In this case a number a_i is in the solution subset if and only if x_i is equal 1, and the vector $\mathbf{x} = [x_1, \ldots, x_n]$ can be interpreted as a representation of this subset.

Let P denote an instance of SSP, $S_*(P)$ a subset of a_i's constituting an optimal solution of P and $V_*(P)$ its value, i.e. the sum of the numbers belonging to $S_*(P)$. Furthermore let $S_{\mathsf{A}}(P)$ and $V_{\mathsf{A}}(P)$ denote the analogous quantities obtained when a heuristic algorithm A is applied to P. The *worst-case performance ratio* of algorithm A is the largest real number $r_{wc}(\mathsf{A})$, such that

$$r_{wc}(\mathsf{A}) \leq \frac{V_{\mathsf{A}}(P)}{V_*(P)}, \quad \text{for all } P.$$

For a solution $\mathbf{x}=[x_1, \ldots, x_n]$ of the problem P the *absolute error* $\delta(\mathbf{x})$ of $\mathbf{x}$ is defined as

$$\delta(\mathbf{x}) = V_*(P) - \sum_{i=1}^{n} x_i a_i \ ,$$

and the *relative error* $\varepsilon(\mathbf{x})$ as

$$\varepsilon(\mathbf{x}) = \frac{\delta(\mathbf{x})}{V_*(P)} \ .$$

An approximation algorithm A is said to have a *relative error bound* ε if every solution $\mathbf{x}$ returned by A for any given problem P fulfills

$$\varepsilon(\mathbf{x}) \leq \varepsilon \ .$$

Finally, the *density* $\mathrm{d}(P)$ of a problem P is defined as

$$\mathrm{d}(P) = \frac{n}{\log_2 \max_{1 \leq i \leq n} a_i} \ .$$

2 The Best Known Approximation Algorithms

This section surveys some approximation algorithms for SSP found in the literature, including the best algorithms currently available.

2.1 Greedy Methods

The standard greedy algorithm G for solving the SSP starts with an empty solution-subset and examines the numbers a_i in the *decreasing* order of their values. Each considered a_i is inserted into the current solution if and only if it is smaller than the difference between B and the sum of the current solution. Its time complexity, clearly dominated by the sorting operation, is $\mathcal{O}(n \log n)$ and the space required is $\mathcal{O}(n)$. The worst-case performance ratio $r_{wc}(\mathsf{G})$ is equal to $\frac{1}{2}$.

If in the greedy algorithm G the numbers a_i are examined in *random* order, i.e. no sorting is performed, the running time reduces to $\mathcal{O}(n)$ and the resulting *randomized greedy* algorithm

RG gives often surprisingly good results (see [TS86] and Section 5). Since RG is a randomized algorithm its result is not deterministic and two independent runs on the same instance of SSP can yield different solutions. To reduce the probability of an "unlucky" solution, whose error is significantly larger than the expected error, one can perform a few independent trials of RG on the given instance and finally return the best solution. Such an algorithm with t trials is denoted by $\mathsf{RG}(t)$.

Martello and Toth [MT84] presented another greedy method: a *quadratic greedy* algorithm, QG, with running time $\mathcal{O}(n^2)$, space complexity $\mathcal{O}(n)$ and $r_{wc}(\mathsf{QG})=\frac{3}{4}$.

Kellerer *et al.* [KMS98] proposed two linear algorithms with $r_{wc}=\frac{3}{4}$ and $r_{wc}=\frac{4}{5}$. For uniformly distributed instances of SSP their algorithm gives results similar to $\mathsf{RG}(40)$, which is also linear (cf. [KMS98] and Section 5).

2.2 Approximation Schemes

An *approximation scheme* for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem but also a value $\varepsilon > 0$. For any fixed ε the scheme is an approximation algorithm with relative error bound ε.

A *polynomial-time approximation scheme* is an approximation scheme which for any fixed $\varepsilon > 0$ runs in time polynomial in the input size n.

An approximation scheme is a *fully polynomial-time approximation scheme* if its running time is polynomial both in $1/\varepsilon$ and in the input size n, where ε is the relative error bound of the scheme.

Approximation schemes are preferred over "normal" approximation algorithms, since they offer a trade-off between the computation time and the quality of the approximation, i.e. they can achieve increasingly smaller relative error bounds by using more and more time and/or space.

There exist *fully* polynomial approximation schemes for the SSP (e.g. [IK75]), which however either require a large amount of space and become infeasible for relatively small n, or are in practice outperformed by the best known polynomial approximation schemes [MT85].

Martello and Toth [MT84] presented a *polynomial* approximation scheme $\mathsf{MT}(s)$, $s \geq 2$, which for $s \geq 3$ gives $r_{wc}(\mathsf{MT}(s)) \geq (s+3)/(s+4)$. Its time complexity is $\mathcal{O}(n^s)$ and the space complexity $\mathcal{O}(n)$.

Soma et al. [SZYH95] proposed a variation of the MT scheme, denoted by $\mathsf{PS}(s, v)$, $s \geq 2$, $v \geq 1$. $\mathsf{PS}(s, v)$ can be theoretically two times slower than $\mathsf{MT}(s)$ and gives the same maximum error, but practically is as fast as $\mathsf{MT}(s)$ and produces on the average slightly smaller errors (cf. Figure 2 in Section 5 and [SZYH95]).

For $s=2$ both algorithms MT and PS are equivalent to the *quadratic* greedy algorithm QG.

3 A New Fast Algorithm

Although both $\mathsf{MT}(s)$ and $\mathsf{PS}(s, v)$ run in polynomial time, already their simplest and fastest versions, i.e. $\mathsf{MT}(2)$ resp. $\mathsf{PS}(2, 1)$, are equivalent to QG and hence have time complexity $\mathcal{O}(n^2)$. This is considerably worse than $\mathcal{O}(n \log n)$ achieved by the greedy algorithm G, which however produces significantly larger errors.

The proposed new solution method for the SSP is a randomized algorithm called "Randomized Greedy with Local Improvement" (RGLI), whose time complexity is $\mathcal{O}(n \log n)$, the space complexity is $\mathcal{O}(n)$, and which on average produces much better solutions than G or QG.

One drawback of the algorithm RGLI is that it does not generalize easily to create a polynomial approximation scheme.

3.1 The Algorithm

The algorithm consists of one or more independent trials, each being a composition of two phases. In each trial a new solution is found, and the solution returned by the algorithm is the best solution from all the trials. Test runs (cf. Section 5.3) show that the maximal number of trials can be set to a small constant (~ 50), independently of n and of the magnitude of the numbers. The algorithm consisting of t independent trials of RGLI is denoted by RGLI(t).

In the first phase of a trial we choose randomly a solution vector $\mathbf{x}$, which is *permissible*, i.e. it does not exceed the bound B, and which is *maximal*, i.e. for which adding any still available number a_j would exceed the bound B. The first phase can be realized by a *random greedy* approach, i.e. starting with an empty solution subset ($\mathbf{x} = [0, \dots, 0]$), the numbers a_i are examined in *random* order and each considered a_i is inserted into the current solution if and only if it is smaller than the difference between B and the sum of the current solution.

The second phase is a *local improvement*: we examine all the numbers determined by the solution vector $\mathbf{x}$ found in the first phase, and for each considered number a_i (with $x_i = 1$) we search for the largest number *not* in the current solution, which would reduce the error $\delta(\mathbf{x})$ when taken into the solution instead of a_i. If we find such a number, we replace a_i with it and proceed with the next number in the solution.

After the second phase the improved solution is compared with the best solution found so far, and, if appropriate, the best solution is updated. Figure 1 presents the whole algorithm in pseudo-code.

Note that the first phase is equivalent to the randomized greedy algorithm RG, which, as shown by Tinhofer and Schreck [TS86], gives very good results for so called *bounded* subset-sum problems. The second phase is similar to a heuristic of Balas and Zemel [BZ80], who used it to derive an exact integer solution of a so called *approximate core knapsack problem* from an optimal (fractional) solution of a linear program associated with the problem. As we will see later this mixture of both phases gives remarkably good results.

4 Analysis of the Algorithm RGLI(t)

4.1 Time and Space Complexity

The running time of the first phase is linear since we consider each number exactly once. The random examination-order can be achieved by generating a random permutation of n elements, which also takes linear time (see e.g. [RND77]).

In the second phase we search at most $\mathcal{O}(n)$ times for a number among at most $\mathcal{O}(n)$ numbers, which are not in the current solution. If we sort those numbers (in time $\mathcal{O}(n \log n)$), each search can be executed in time $\mathcal{O}(\log n)$ (binary search), so the time complexity of the second phase is bounded by the $\mathcal{O}(n \log n)$. It follows that the total running time of the algorithm RGLI is bounded by $\mathcal{O}((\text{max-number-of-trials}) \cdot n \log n)$. If the number of trials is constant we obtain an $\mathcal{O}(n \log n)$-algorithm.

The space complexity of the algorithm is clearly linear.

```
*Input*: positive integer numbers $a_1, \ldots, a_n$, $B$
*Output*: a solution vector $[x_1, \ldots, x_n]$

$\mathbf{x}_{\text{best}} := [0, \ldots, 0]$;
for trial:=1 to (max-number-of-trials) do
    // first phase: randomized selection
    $\mathbf{x} := [0, \ldots, 0]$;
    for each $i \in_{\mathsf{R}} \{1, \ldots, n\}$ do    // in random order
        if $(a_i \le \delta(\mathbf{x}))$ then  // $\delta(\mathbf{x})$ is the absolute error of the solution $\mathbf{x}$
            $x_i := 1$;
        fi;
    od;
    // second phase: local improvement
    $\mathbb{I} := \{j : x_j = 1\}$;
    for each $i \in_{\mathsf{R}} \mathbb{I}$ do    // in random order
        if ( $\delta(\mathbf{x}) = 0$ ) then
            break;  // quit the inner "for each" loop
        fi;
        let $T$ denote the set of valid replacements for $a_i$,
        i.e. $T = \{a_l : x_l = 0 \ \wedge \ 0 < (a_l - a_i) \le \delta(\mathbf{x})\}$
        if ( $T$ is not empty ) then
            $k$:= index, such that $a_k = \max(T)$
            $x_k := 1$;
            $x_i := 0$;
        fi;
    od;
    // $\mathbf{x}_{best}$ update
    if ( $\delta(\mathbf{x}) < \delta(\mathbf{x}_{\text{best}})$ ) then
        $\mathbf{x}_{\text{best}} := \mathbf{x}$;
    fi;
    if ( $\delta(\mathbf{x}_{\text{best}}) = 0$ ) then
        break;  // quit the outer "for" loop
    fi;
od;
return $\mathbf{x}_{\text{best}}$;
```

Figure 1: The new randomized algorithm (RGLI) for the subset-sum problem

4.2 Performance Analysis

In this section we estimate the quality of approximative solutions found by the algorithm RGLI(1). Assume that the numbers a_i are *uniformly* distributed over an interval $[1..M]$, where M is a constant. We are interested in the expected error of a solution found by RGLI(1).

Note that the first phase of the algorithm determines the size of the solution set, i.e. the number of numbers chosen. The second phase *replaces* some numbers of the solution by other numbers, while keeping the size of the solution set constant. Therefore we can split the analysis into three parts:

1. estimating the expected size of the solution set
2. estimating the expected (absolute) error of the random solution chosen in the first phase
3. estimating the expected (absolute) error of the *improved* solution after the second phase.

Since the numbers a_i are uniformly distributed over an interval $[1..M]$, the first phase of the algorithm is equivalent to starting with an amount B of free space and repeating n times the following random experiment $\mathcal{A}$:

Experiment $\mathcal{A}$: let i denote the current repetition of the experiment; choose a number α_i uniform-randomly from the interval $[1..M]$ and *accept* it if it still fits into the remaining free space, *reject* otherwise; "accepting" reduces the amount of free space by α_i and "rejecting" leaves the free space unaltered.

The expected number of accepted numbers α_i after n repetitions is equal to the expected number of a_i's chosen in the first phase of the algorithm RGLI. Furthermore, the expected remaining free space after n repetitions is equal to the expected error of the solution found in the RGLI's first phase. Therefore we can handle the first two parts of the performance analysis by considering successive repetitions of the experiment $\mathcal{A}$.

Let $\varepsilon_k(B)$ denote the expected amount of free space and $\eta_k(B)$ the expected number of accepted numbers after k repetitions of the experiment $\mathcal{A}$ for a given value of B. We are interested in the expected error and the expected size of the solution set after the first phase, i.e. we would like to estimate the values of $\varepsilon_n(B)$ and $\eta_n(B)$.

If in an experiment $\mathcal{A}$ the free space is greater than M, any chosen number α_i will be surely accepted. If the free space is *not greater* than M, α_i will be accepted if and only if it is *not greater* than the free space. Therefore $\varepsilon_k(B)$ can be computed from the following recursive formula:

$$\varepsilon_k(B) = \begin{cases} \displaystyle\sum_{i=1}^{B} \frac{1}{M}\varepsilon_{k-1}(B-i) + \sum_{i=B+1}^{M} \frac{1}{M}\varepsilon_{k-1}(B) & 1 \le B \le M \\ \displaystyle\sum_{i=1}^{M} \frac{1}{M}\varepsilon_{k-1}(B-i) & B > M \end{cases} \tag{1}$$

with $\varepsilon_k(0) = 0, \;\; \varepsilon_0(B) = B \;\; \forall k, \forall B$.

Analogously we can derive a recursive formula for $\eta_k(B)$:

$$\eta_k(B) = \begin{cases} \displaystyle\sum_{i=1}^{B} \frac{1}{M}(1+\eta_{k-1}(B-i)) + \sum_{i=B+1}^{M} \frac{1}{M}\eta_{k-1}(B) & 1 \le B \le M \\ \displaystyle\sum_{i=1}^{M} \frac{1}{M}(1+\eta_{k-1}(B-i)) & B > M \end{cases} \tag{2}$$

with $\eta_k(0) = 0, \;\; \eta_0(B) = 0 \;\; \forall k, \forall B$.

4.2.1 Estimating $\varepsilon_k(B)$ and $\eta_k(B)$ for $B \leq M$.

The formula (1) can be rewritten as

$$\varepsilon_k(B) = \begin{cases} \dfrac{1}{M}\displaystyle\sum_{i=1}^{B-1} \varepsilon_{k-1}(i) + \dfrac{M-B}{M}\varepsilon_{k-1}(B) & 1 \leq B \leq M \\ \\ \dfrac{1}{M}\displaystyle\sum_{i=1}^{M} \varepsilon_{k-1}(B-i) & B > M \end{cases}$$

Subtracting $\varepsilon_k(B-1)$ from $\varepsilon_k(B)$ gives

$$\varepsilon_k(B) - \varepsilon_k(B-1) = \left(1 - \frac{B}{M}\right)^k,$$

which leads directly to the (non-recursive) formula for the $\varepsilon_k(B)$ (for $B \leq M$):

$$\begin{aligned} \varepsilon_k(B) &= \varepsilon_k(B) - \varepsilon_k(0) \\ &= \sum_{i=1}^{B}(\varepsilon_k(i) - \varepsilon_k(i-1)) \\ &= \sum_{i=1}^{B}\left(1 - \frac{i}{M}\right)^k \\ &= \frac{1}{M^k}\sum_{i=1}^{B}(M-i)^k \end{aligned} \tag{3}$$

Further we get

$$\begin{aligned} \frac{1}{M^k}\int_1^B (M-x)^k \,\mathrm{d}x \ &\leq\ \varepsilon_k(B)\ \leq\ \frac{1}{M^k}\int_0^{B-1} (M-x)^k \,\mathrm{d}x \\ -\frac{1}{M^k}\left(\frac{(M-x)^{k+1}}{k+1}\right)\Bigg|_1^B \ &\leq\ \varepsilon_k(B)\ \leq\ -\frac{1}{M^k}\left(\frac{(M-x)^{k+1}}{k+1}\right)\Bigg|_0^{B-1} \\ \left(\frac{M-1}{M}\right)^k \frac{M-1}{k+1} - \frac{(M-B)^{k+1}}{M^k(k+1)} \ &\leq\ \varepsilon_k(B)\ \leq\ \frac{M}{k+1} - \frac{(M+1-B)^{k+1}}{M^k(k+1)} \end{aligned} \tag{4}$$

For $B = M$ we obtain

$$\left(\frac{M-1}{M}\right)^k \frac{M-1}{k+1} \leq\ \varepsilon_k(M)\ \leq \frac{M}{k+1},$$

which can be simplified further by using the following fact

$$0 \le \frac{M}{k+1} - \left(\frac{M-1}{M}\right)^k \frac{M-1}{k+1} \le 1 \; . \tag{5}$$

The left inequality of (5) is obvious and the right can be proved as shown below.

$$\begin{aligned}
\frac{M}{k+1} - \left(\frac{M-1}{M}\right)^k \frac{M-1}{k+1} \le 1 \;\; &\Leftrightarrow \;\; M - \left(\frac{M-1}{M}\right)^k (M-1) \le k+1 \\
&\Leftrightarrow \;\; 1 - \left(\frac{M-1}{M}\right)^{k+1} \le \frac{k+1}{M} \\
&\Leftrightarrow \;\; 1 - \frac{k+1}{M} \le \left(1 - \frac{1}{M}\right)^{k+1} \\
&\Leftrightarrow \;\; 1 - \kappa x \le (1-x)^\kappa, \quad (x = \frac{1}{M}, \; \kappa = k+1),
\end{aligned}$$

where the last inequality follows directly from the Taylor series of $(1-x)^\kappa$. Therefore (5) is indeed satisfied and we obtain

$$\varepsilon_k(M) = \frac{M}{k+1} + \mathcal{O}(1) \; . \tag{6}$$

Note that for B "close to" M, i.e. when $M - B = o(M)$, the terms subtracted on both sides of (4) are negligible[2], therefore we get

$$\varepsilon_k(B) = \varepsilon_k(M) + \mathcal{O}(1) = \frac{M}{k+1} + \mathcal{O}(1) \qquad \text{for } B \le M, \; M - B = o(M) \; . \tag{7}$$

The estimation of $\eta_k(B)$ proceeds similarly to $\varepsilon_k(B)$. Formula (2) can be equivalently written as

$$\eta_k(B) = \begin{cases} \dfrac{B}{M} + \dfrac{1}{M} \displaystyle\sum_{i=1}^{B-1} \eta_{k-1}(i) + \dfrac{M-B}{M} \eta_{k-1}(B) & 1 \le B \le M \\[2ex] 1 + \dfrac{1}{M} \displaystyle\sum_{i=1}^{M} \eta_{k-1}(B-i) & B > M \end{cases}$$

which for $B \le M$ yields

$$\eta_k(B) = H_B - \sum_{i=1}^{B} \frac{(1 - \frac{i}{M})^k}{i} \; , \tag{8}$$

where H_B is the B-th harmonic number.

[2] i.e. they can be bounded by $\mathcal{O}(1)$

Equation (8) can be transformed further as follows:

$$\begin{aligned}
\eta_k(B) &= H_B - \sum_{i=1}^{B} \frac{(1-\frac{i}{M})^k}{i} \\
&= H_B - \sum_{i=1}^{B} \frac{(1-\frac{i}{M})^{k-1}(1-\frac{i}{M})}{i} \\
&= \underbrace{H_B - \sum_{i=1}^{B} \frac{(1-\frac{i}{M})^{k-1}}{i}}_{\eta_{k-1}(B)} + \underbrace{\frac{1}{M}\sum_{i=1}^{B}(1-\frac{i}{M})^{k-1}}_{\varepsilon_{k-1}(B)} \\
&= \eta_{k-1}(B) + \frac{1}{M}\varepsilon_{k-1}(B) \\
&= \frac{1}{M}\sum_{j=0}^{k-1}\varepsilon_j(B) \; .
\end{aligned}$$

Using (7) in the above formula we get

$$\eta_k(B) = H_k + \mathcal{O}(1) = \ln k + \mathcal{O}(1) \qquad \text{for } B \le M,\ M-B = o(M) \; . \tag{9}$$

Lemma 4.1 below summarizes the results obtained hitherto.

Lemma 4.1 *For every subset-sum problem with k input numbers uniformly distributed over $[1..M]$, and a bound $B \le M$ with M-B=$o(M)$, the first phase of the algorithm RGLI(1) finds a solution set, whose expected relative error $\varepsilon_k(B)$ is given by*

$$\varepsilon_k(B) = \frac{M}{k+1} + \mathcal{O}(1) \; ,$$

and the expected number of numbers in the solution $\eta_k(B)$ by

$$\eta_k(B) = \ln k + \mathcal{O}(1) \; .$$

4.2.2 Estimating $\varepsilon_k(B)$ and $\eta_k(B)$ for Arbitrary B.

It seems that rigorous derivation of meaningful estimations of $\eta_n(B)$ and $\varepsilon_n(B)$ for arbitrary $B > M$ is a difficult task. However with a help of simple probabilistic arguments and using Lemma 4.1 we can find some convincing bounds which are sufficient for our needs.

Again, consider the first phase of the algorithm as a series of experiments $\mathcal{A}$. If B is much lager than M then a few first repetitions of $\mathcal{A}$ are always accepting — until the free space is less than M. Hence we can estimate the expected number of accepted numbers $\eta_n(B)$ by first estimating the number $\tilde{n}(B)$ of repetitions of $\mathcal{A}$ which occur until the free space reaches M. Then we add an estimation of $\eta_{n-\tilde{n}(B)}(M)$ to it using Lemma 4.1 (we neglect here "pathological" cases when $\tilde{n}(B) \ge n$). Similarly, we can estimate the expected error after the first phase $\varepsilon_n(B)$ by $\varepsilon_{n-\tilde{n}(B)}(M)$.

We interpret α_i as a *random variable* denoting the outcome of the uniform-selection of a number from the interval $[1..M]$ during the i-th experiment $\mathcal{A}$. Let $\mathbf{A}_j$ be equal to the sum $\alpha_1 + \cdots + \alpha_k$. The expected values and variances of α_i and $\mathbf{A}_j$ are given by

$$\begin{aligned} \mathbf{E}[\alpha_i] &= \frac{M+1}{2} & (10)\\ \mathbf{V}[\alpha_i] &= \frac{(M+1)(M-1)}{12} & (11)\\ \mathbf{E}[\mathbf{A}_j] &= \frac{j(M+1)}{2} & (12)\\ \mathbf{V}[\mathbf{A}_j] &= \frac{j(M+1)(M-1)}{12} & (13) \end{aligned}$$

The value of $\tilde{n}(B)$ can be estimated the number k of repetitions of experiment $\mathcal{A}$, such that the expected sum of all randomly selected α_i's , i.e. $\mathbf{E}[\mathbf{A}_k]$, is equal to $B - M$. From (12) we obtain the following equation for k (and hence an estimation of $\tilde{n}(B)$, with an error to be specified below)

$$\frac{k(M+1)}{2} = B - M\ .$$

Therefore we get

$$\tilde{n}(B) \approx k = \frac{2(B-M)}{M+1} = \frac{2B}{M+1} + \mathcal{O}(1)\ . \qquad (14)$$

To justify the above reasoning we show that $\mathbf{A}_k$ is indeed asymptotically close to $\mathbf{E}[\mathbf{A}_k]$ with high probability. More precisely, we prove that the probability $\mathrm{Prob}\,(|\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k]| < cM)$ for some small value $c \ll k$ is big, or equivalently that the probability $\mathrm{Prob}\,(|\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k]| \geq cM)$ is small.

Using *Hoeffding's inequality* [Hoe63] we get

$$\begin{aligned} \mathrm{Prob}\,(\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k] \geq cM) &\leq \exp\left(-\frac{2(cM)^2}{k(M-1)^2}\right)\\ &\leq \exp\left(-\frac{2c^2}{k}\right)\ , \end{aligned}$$

and analogously for $\mathrm{Prob}\,(\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k] \leq -cM)$:

$$\mathrm{Prob}\,(\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k] \leq -cM) \leq \exp\left(-\frac{2c^2}{k}\right)\ .$$

Hence for $c = \sqrt{k \ln k}$ we obtain

$$\mathrm{Prob}\left(|\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k]| \geq \sqrt{k \ln k}M\right) \leq \frac{2}{k^2}\ ,$$

which shows that that the probability Prob $\left(|\mathbf{A}_k - \mathbf{E}[\mathbf{A}_k]| \geq \sqrt{k \ln k} M\right)$ goes to zero if k goes to infinity. It follows that the equation (14) provides asymptotically a good approximation for $\tilde{n}(B)$ of the form

$$\tilde{n}(B) = k + \mathcal{O}(\sqrt{k \ln k}) \quad \text{with} \quad k = \frac{2B}{M+1} + \mathcal{O}(1) \; .$$

Therefore, the above arguments and Lemma 4.1 allow us to consider the following expressions as good approximations of $\eta_n(B)$ and $\varepsilon_n(B)$

$$\eta_n(B) = \tilde{n}(B) + \eta_{n-\tilde{n}(B)}(M) = \frac{2B}{M+1} + \ln\left(n - \frac{2B}{M+1}\right) \tag{15}$$

$$\varepsilon_n(B) = \varepsilon_{n-\tilde{n}(B)}(M) = \frac{M^2}{nM - 2B} \; . \tag{16}$$

Remind that the first phase of RGLI is equivalent to the randomized greedy algorithm RG. Let $B = \beta n M$, where β is fixed, $0 < \beta < \frac{1}{2}$. Assuming that (16) is exact we get the following estimations of the expected absolute and relative errors of RG(1)

$$\begin{aligned} \delta_{\mathsf{RG}} &= \frac{M}{n(1-2\beta)} \\ \varepsilon_{\mathsf{RG}} &= \frac{M}{n(1-2\beta)} \frac{1}{\beta n M} = \frac{1}{n^2} \frac{1}{\beta(1-2\beta)} \; , \end{aligned}$$

and the following conjecture

Conjecture 4.1 *For every subset-sum problem with n input numbers uniformly distributed over $[1..M]$, and a bound $B = \beta n M$ with a fixed β, $0 < \beta < \frac{1}{2}$, the expected relative error of a solution found by the algorithm RG is proportional to $\frac{1}{n^2}$.*

4.2.3 Estimating the Final Expected Error

The arguments presented in this section, although not formally precise, give some insight into the performance of RGLI and suggest an explanation of its experimental behavior.

Let k be the number of accepted numbers in the first phase and let $l = n - k$. The second phase consists of k "improvement" steps — one step per accepted number.

Assume that after the first phase the two following conditions are fulfilled:

1. all k accepted numbers are *uniformly* distributed in the interval $[1..M]$

2. all l *not* accepted numbers are also *uniformly* distributed in the interval $[1..M]$

From the above assumptions it follows that for each number a_s accepted in the first phase and for each $b > 0$ the probability that a number a_r *rejected* in the first phase lies in an interval $I = [(a_s + 1)..(a_s + b)]$ is equal to $\frac{b}{M}$ (we neglect the pathological cases, when $a_s + b > M$). Note that if b is an error of the solution after the first phase, then every number $a_r \in I$ is a potential replacement for the number a_s during the second phase.

Let a_{j_i} be an accepted number considered in the i-th improvement step and b_i a random variable denoting the error of the solution before the i-th step. We have clearly $\mathbf{E}[b_1]=\varepsilon_n(B)$.

The i-th improvement step is successful if and only if at least one of the l candidate-numbers is in the interval $I_i = [(a_{j_i}+1)..(a_{j_i}+b_i)]$. Let p_i be the probability, that none of the l numbers is in the interval I_i, i.e.

$$p_i = \left(1 - \frac{b_i}{M}\right)^l .$$

Furthermore, for all $i \geq 1$ we have

$$\begin{aligned}
\mathbf{E}[b_{i+1} | b_i] &= p_i b_i + (1-p_i)\frac{b_i}{2} \\
&= \frac{b_i}{2}(1+p_i) \\
&= \frac{b_i}{2}\left(1 + \left(1 - \frac{b_i}{M}\right)^l\right) \\
&\leq b_i - \frac{b_i^2}{2}\frac{l}{M} + \binom{l}{2}\frac{b_i^3}{2M^2} .
\end{aligned}$$

Since $b_i \leq b_1$ for all $i \geq 1$, it follows

$$\mathbf{E}[b_{i+1} | b_i] \leq b_i - b_i^2 \frac{l}{2M} + b_i^2 \frac{l^2}{4M^2} b_1 .$$

Assuming that b_1 is a constant equal to $\varepsilon_n(B)$ and having $b_1 \leq \frac{M}{l}$, as follows from (7) and (16), we get further

$$\begin{aligned}
\mathbf{E}[b_{i+1}] &= \mathbf{E}[\mathbf{E}[b_{i+1} | b_i]] \\
&\leq \mathbf{E}\left[b_i - b_i^2\left(\frac{l}{2M} - \frac{l^2}{4M^2} b_1\right)\right] \\
&\leq \mathbf{E}[b_i] - \mathbf{E}\left[b_i^2\right]\frac{l}{4M} \\
&\leq \mathbf{E}[b_i] - \mathbf{E}[b_i]^2\frac{l}{4M}
\end{aligned}$$

The last of above inequalities leads to the following lemma

Lemma 4.2 *Let $\gamma = \frac{l}{4M}$. Under the two assumptions about the uniform distribution of accepted and rejected numbers, for $b_1 = \varepsilon_n(B) \leq \frac{M}{l}$ and for all $i \geq 1$*

$$\mathbf{E}[b_i] \leq \frac{2}{i\gamma} .$$

Proof
For $i = 1, \dots, 8$ we obtain

$$\mathbf{E}[b_i] \le b_1 \le \frac{M}{l} \le \frac{2}{i}\frac{4M}{l} = \frac{2}{i\gamma} .$$

Let $f(x) = x - x^2\gamma$. For $i \ge 8$ we have

$$\begin{aligned} \mathbf{E}[b_{i+1}] &\le f(\mathbf{E}[b_i]) \\ &\le f\left(\frac{2}{i\gamma}\right) , \end{aligned}$$

since for $x < \frac{1}{2\gamma}$ the function f is monotonic and increasing. Therefore we get

$$\begin{aligned} \mathbf{E}[b_{i+1}] &\le \frac{2}{i\gamma} - \frac{4}{i^2\gamma} \\ &= \frac{2}{(i+1)\gamma}\left(\frac{i+1}{i} - \frac{2(i+1)}{i^2}\right) \\ &= \frac{2}{(i+1)\gamma}\underbrace{\left(\frac{i^2-i+2}{i^2}\right)}_{\substack{\le 1, \\ \textit{for} \\ i \ge 2}} \\ &\le \frac{2}{(i+1)\gamma} . \end{aligned}$$

□

Lemma 4.2 gives us the following bound for $\mathbf{E}[b_i]$

$$\mathbf{E}[b_i] \le \frac{8M}{l}\frac{1}{i} . \tag{17}$$

Let $B = \beta n M$, for some constant β, $0 < \beta < \frac{1}{2}$. Then assuming validity of (15) and (16) the above definitions lead to

$$\begin{aligned} k &= 2\beta n + \ln(n(1-2\beta)) = 2\beta n + \mathcal{O}(\ln n) && (18) \\ l &= n(1-2\beta) + \mathcal{O}(\ln n) && (19) \\ b_1 &= \frac{M}{n(1-2\beta)} . && (20) \end{aligned}$$

Note that in this case the two assumptions about the uniform distribution of accepted and rejected numbers are pretty good fulfilled. Both sets contain $\Theta(n)$ numbers and their uniform distribution is only slightly destroyed by the $\mathcal{O}(\ln n)$ successful experiments $\mathcal{A}$, which take place after the available free space has been reduced to M.

From (17) and (18)-(20), neglecting the $\mathcal{O}(\ln n)$-terms, we obtain estimations on the expected absolute and relative errors of $\mathsf{RGLI}(1)$

$$\delta_{\mathsf{RGLI}} \leq \mathbf{E}[b_k] \leq \frac{8M}{n(1-2\beta)} \frac{1}{2\beta n} = \frac{1}{n^2} \frac{4M}{\beta(1-2\beta)} \tag{21}$$

$$\varepsilon_{\mathsf{RGLI}} \leq \frac{1}{n^2} \frac{4M}{\beta(1-2\beta)} \frac{1}{\beta n M} = \frac{1}{n^3} \frac{4}{\beta^2(1-2\beta)}, \tag{22}$$

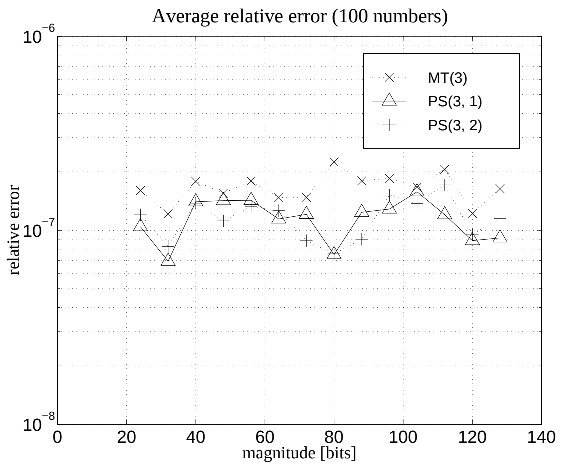
which lead to the following conjectures:

Conjecture 4.2 *For every subset-sum problem with n input numbers uniformly distributed over $[1..M]$, and a bound $B = \beta n M$ with a fixed β, $0 < \beta < \frac{1}{2}$, the expected relative error of a solution found by the algorithm RGLI(1) is proportional to $\frac{1}{n^3}$.*

Conjecture 4.3 *For every subset-sum problem with n input numbers uniformly distributed over $[1..M]$, and a bound $B = \beta n M$ with a fixed β, $0 < \beta < \frac{1}{2}$, if $M \leq \frac{1}{4} n^2 \beta (1-2\beta)$ then the algorithm RGLI(1) finds an optimal solution with high probability.*

5 Experimental Results

Although theoretical bounds for the performance of algorithms provide often a good measure for comparisons of algorithms, they are sometimes too pessimistic or too weak to help one in choosing the "best" algorithm in practice, especially when several algorithms provide similar theoretical bounds (see also [SZYH95], [MT85]). Besides, as follows from the attempts of the previous section, it is sometimes hard to derive any theoretical bounds at all. Therefore it is often reasonable to compare the performance of algorithms experimentally with some randomly-generated or especially-constructed instances of problems.



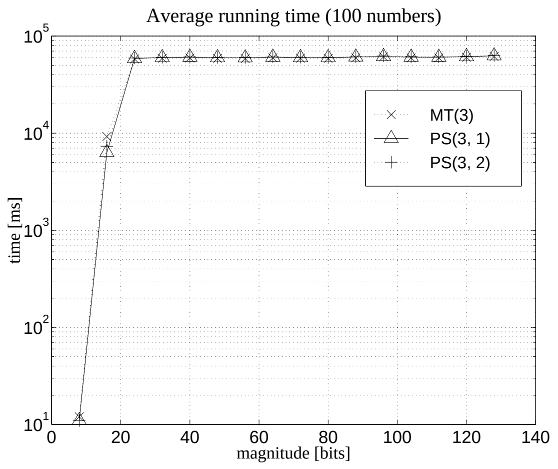


Figure 2: $\mathsf{MT}(3)$ vs. $\mathsf{PS}(3,\,1)$ and $\mathsf{PS}(3,\,2)$

This section presents results of various computational tests. The main aim was to examine the performance of the algorithm RGLI in practice. The test programs were written in Java and run on Silicon Graphics' Indy computer with a Java Virtual Machine from SGI, version "3.0.1 (Sun 1.1.3)".

Figure 2 shows a comparison of performance of the algorithms $\mathsf{MT}(s)$ and $\mathsf{PS}(s,\ v)$, where the corresponding tests were performed in the same way as for RGLI and other algorithms, as described below. As mentioned in Section 2.2 and as it is apparent from the Figure 2, the algorithms $\mathsf{MT}(s)$ and $\mathsf{PS}(s,\ v)$ have in practice roughly the same running time and provide on average similar relative errors. Furthermore, for s=2 both algorithms are *by definition* the same. Therefore in the tests comparing RGLI with other methods the algorithm $\mathsf{PS}(s,\ v)$ was omitted – it is represented by $\mathsf{MT}(s)$.

All tests were performed on randomly generated examples of the SSP. For a specified number n and a magnitude m, the numbers $a_1, \ldots, a_n$ were chosen uniformly at random from the range $[1, 2^m]$, and the bound B was set to the sum of $\frac{n}{2}$ randomly selected a_i's (the tests of Section 5.2 are an exception from this rule). This implies that the magnitude of B is roughly $\frac{n}{2} \cdot \frac{2^m}{2}$ and assures that the optimal solution has an error equal to zero. Then every tested algorithm was run on the generated problem. Since each considered algorithm, except RG, requires that the numbers a_i are in a monotone order, the numbers were sorted directly after generating. Therefore the presented running times do not include the time needed for sorting, which, using the Quick-Sort algorithm, is about 1-2ms for 100 numbers and 21-29ms for 1000 numbers (exact values depend on the length of numbers).

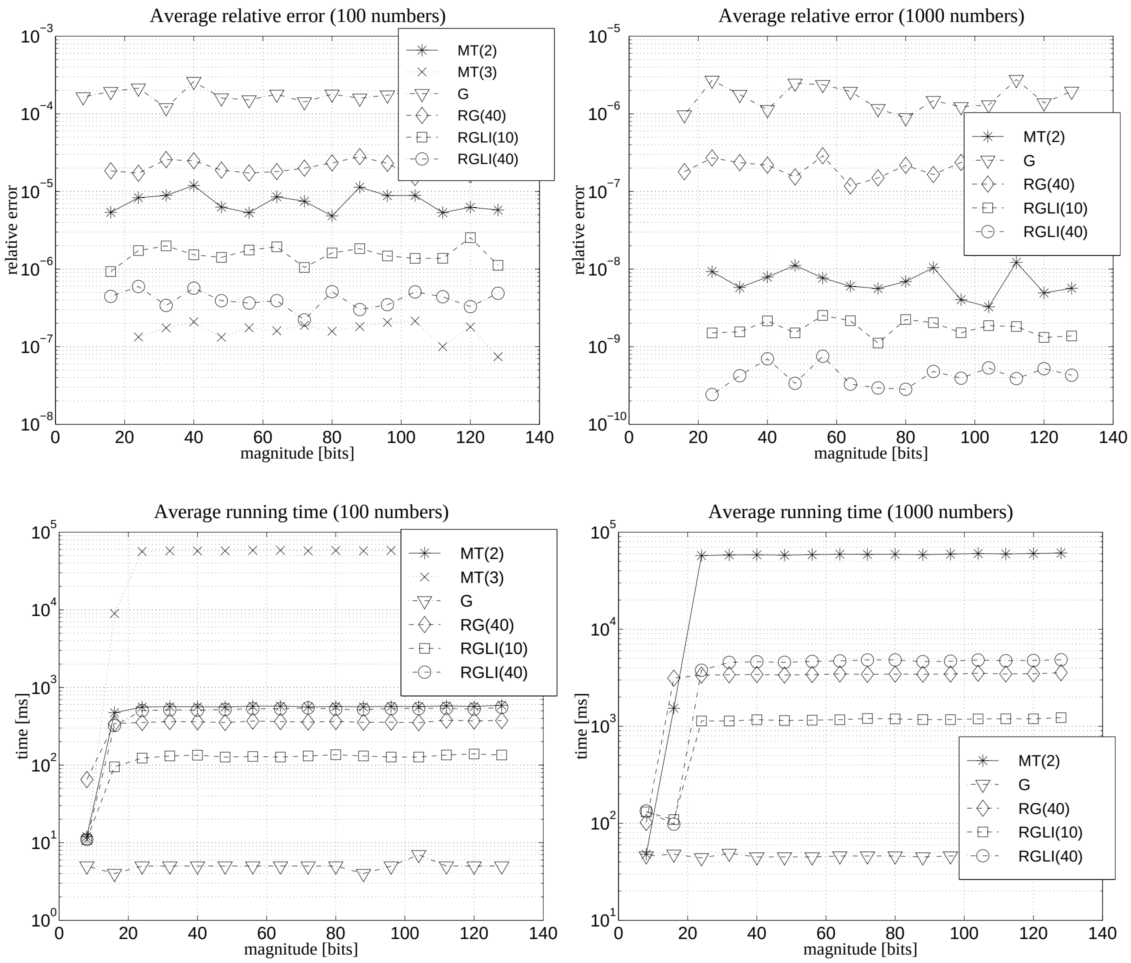


Figure 3: RGLI vs. other algorithms: approximative solving of sparse problems

5.1 Comparison of RGLI(t) with Other Algorithms

The performance of approximation algorithms for SSP depends significantly on the *density* of the specified problems. For random low-density subset-sum problems usually there exist few, if any, *exact* solutions, i.e. solutions whose error is equal to zero. However, if there are plenty of numbers of relatively small magnitude, i.e. if the density of the numbers a_i is high, then usually there exist many exact solutions. In such a case it is often possible for an approximation algorithm to find one of those exact solutions. Hence the two cases were tested separately.

The performance of the algorithm RGLI(t) was compared with other methods referred to in Section 2: G, RG(t), MT(2) (i.e. QG) and MT(3). The maximal number of trials t was set to 40 for the RG algorithm, and for RGLI two variants were tested: t=10 and t=40 (cf. Section 5.3 for more about choosing the value of t).

5.1.1 Approximative Solving of Low-Density Problems

For every chosen pair (n, m), where n denotes of number of numbers and m their magnitude, 20 random problems were generated and solved with each[3] tested algorithm.

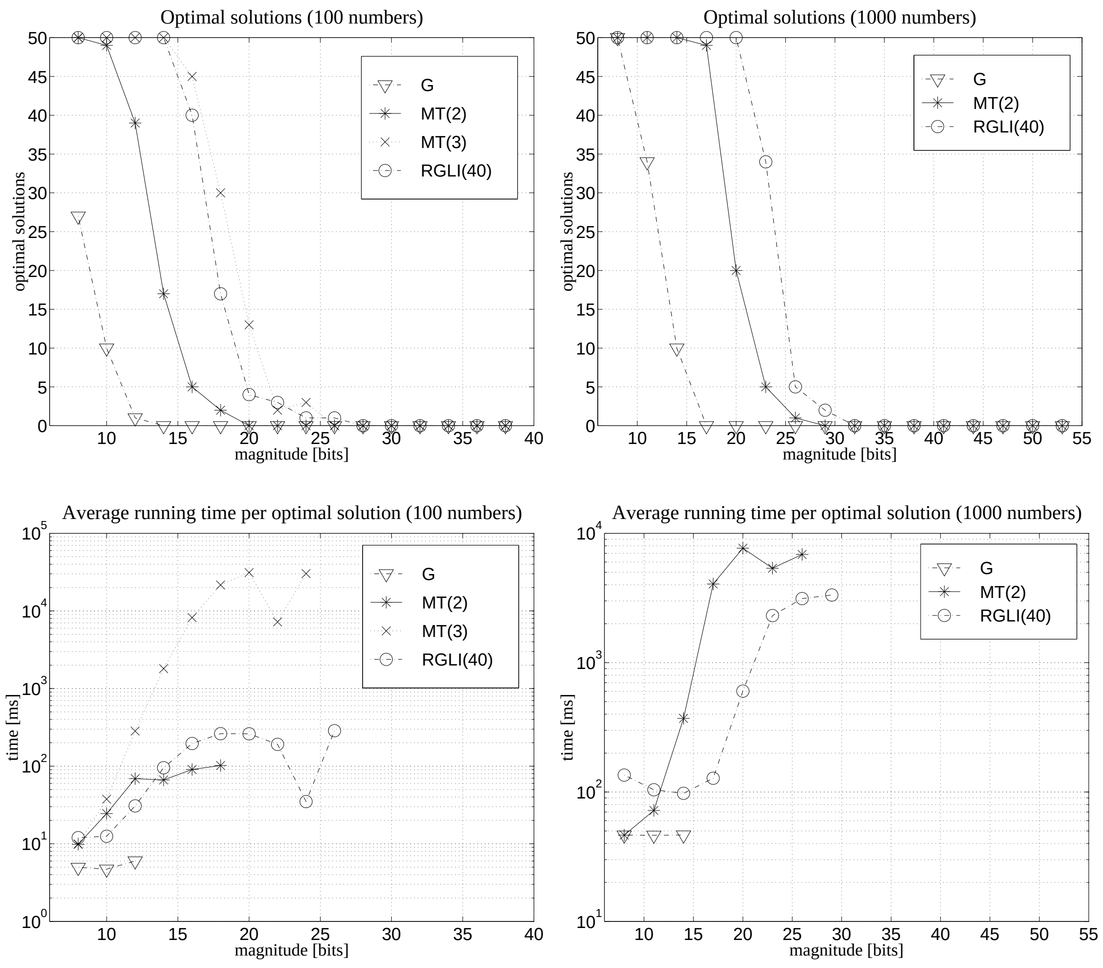


Figure 4: RGLI vs. other algorithms: exact solving of dense problems

[3]The algorithm MT(3) was excluded from the tests for n=1000 due to its immense running time.

The graphs in the Figure 3 show the averaged results of the tests. Note that with max. 10 trials the algorithm RGLI gives much smaller errors than MT(2) and the errors of RGLI(40) are comparable even with those returned by MT(3).

5.1.2 Exact Solving of High-Density Problems

In the tests investigating the behavior of the algorithms on high-density problems the algorithm RGLI(t) was tested only for t=40. Additionally, the actual number of trials till the exact solution, if found, was recorded. For each selected pair (n, m) 50 problems were generated and solved with each[4] considered algorithm.

Figure 4 shows the comparison of the algorithms' performance. Note that RGLI(40) is much better than G or MT(2), both in percentage of optimally solved problems and in running time. MT(3) finds the optimal solution slightly more often than RGLI(40), but RGLI(40) is much more efficient.

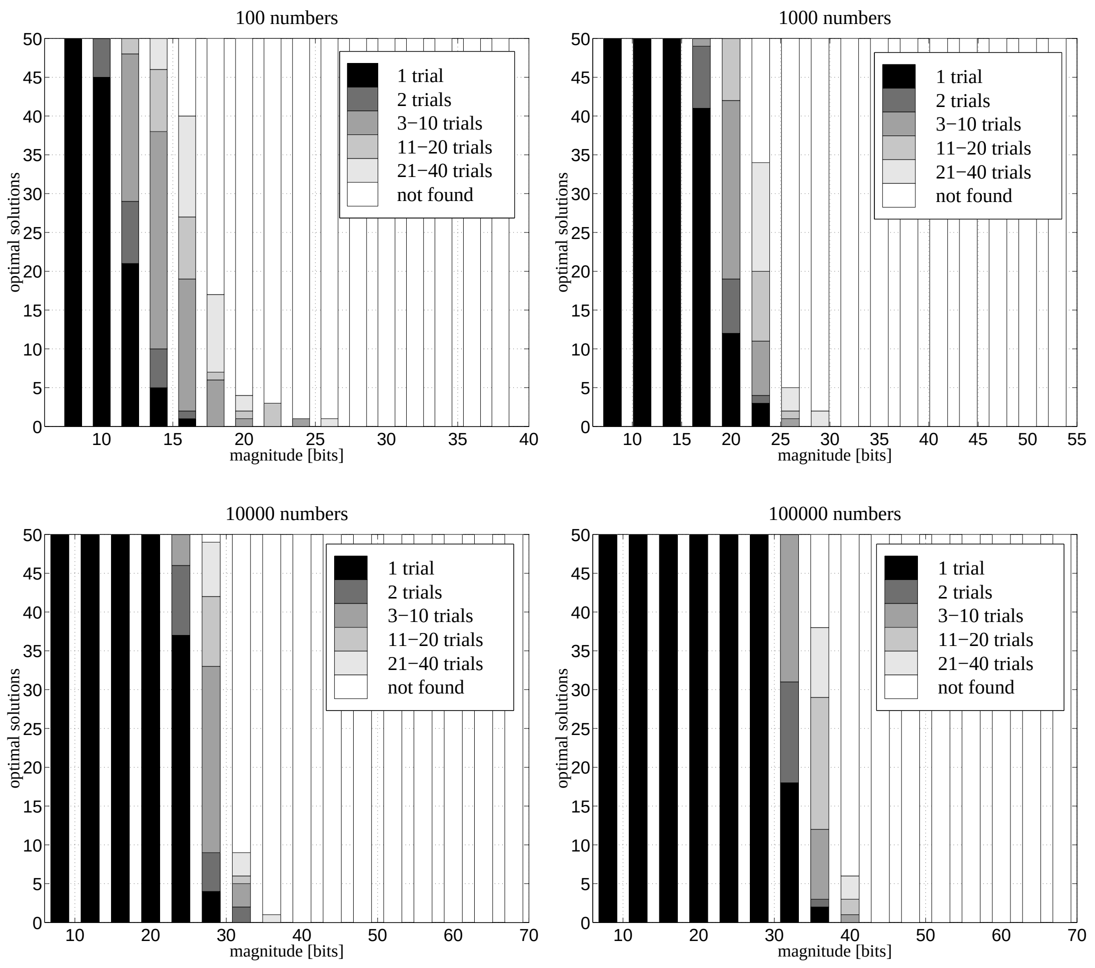


Figure 5: RGLI(40): the actual number of trials till optimal solution

Figure 5 presents the actual number of trials used by the RGLI(40) to find the optimal solutions, and the table below shows the upper bounds for M derived from the Conjecture 4.3

[4]Also here the algorithm MT(3) was excluded from the tests for n=1000.

for some values of n. Note that the conjectured values agree pretty well with the results from the Figure 5.

n	max. M for optimal solution
10^2	$\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot 10^4 \approx 2^8$
10^3	$\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot 10^6 \approx 2^{15}$
10^4	$\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot 10^8 \approx 2^{21}$
10^5	$\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot 10^{10} \approx 2^{28}$

5.2 Magnitude of B

In tests of this section the numbers a_i were generated in a usual, random way, but the bound B was always set exactly to a $\beta n M$ for some values β, $\frac{1}{16} \leq \beta \leq \frac{1}{2}$. In this case the sum of the optimal solution might be smaller than B. Since it is difficult to find the optimal solution of a random problem P, the relative errors in this section were computed by using B instead of $V_*(P)$, which gives values slightly greater than the actual relative errors. The aim of those

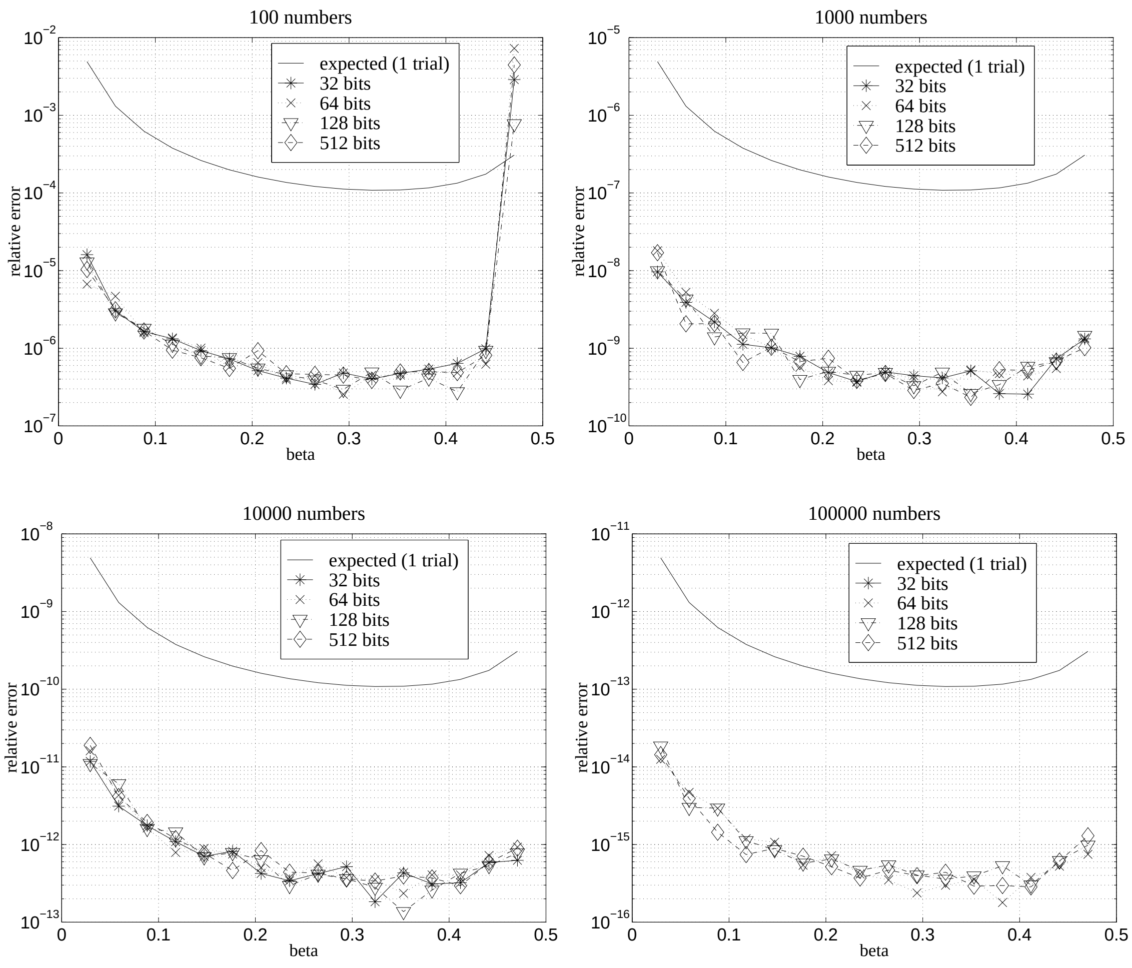
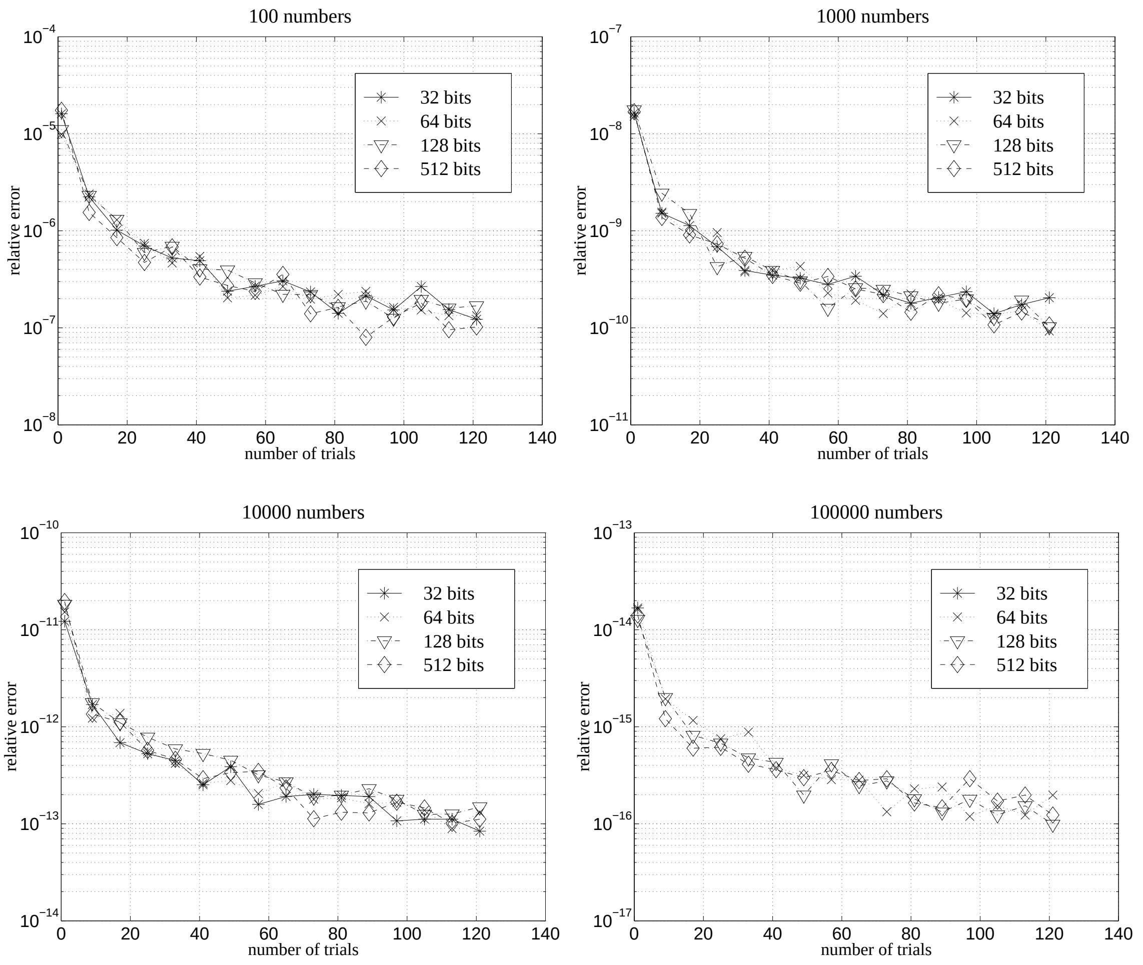


Figure 6: RGLI(40): relative error as a function of B's magnitude

tests was to check two following hypotheses (under an assumption of uniform distribution of numbers a_i):

1. the relative error of RGLI does not depend on the existence of an exact solution
2. the relative error of RGLI is roughly the same for all "usual" values of β, i.e. $0.1 \leq \beta \leq 0.4$

For each triple (n, m β) 20 random sets of a_i's were generated and solved with RGLI(40). Figure 6 shows the averaged results which apparently agree with above hypotheses. Worse performance in degenerated cases, i.e. $\beta \approx 0$ or $\beta \approx \frac{1}{2}$, can be explained by the fact, that in such cases there exist usually less "good" solutions, if any at all.



Note: for 100000 numbers and $t \geq 9$ all 32-bit-problems were solved exactly, which is in agreement with Conjecture 4.3

Figure 7: Average relative error of RGLI as a function of number of trials

The graphs of Figure 6 present also expected errors of RGLI(1) as estimated in Section 4.2 by the formula (21). The similarity of the behaviour of the estimated and measured errors is quite remarkable. The evident shift between the expected and the actual errors can be easily explained by the fact that the tested algorithm was not RGLI(1) but RGLI(40). Indeed, this argument is strongly supported by the tests presented in the next section.

5.3 Determining the Sufficient Number of Trials

The difference of the relative errors of RGLI(10) and RGLI(40) suggests that by increasing the maximal number of trials one could achieve smaller and smaller errors. Although this is definitely a correct conclusion, the experimental results presented in the Figure 7 indicate that it is worthwhile to allow maximally about 40 to 80 trials — larger numbers of trials provide only a slight improvement of the solution while increasing the running time significantly (linearly in t).

Note that the above upper limit on the number of trials apparently does not depend on the number of numbers n or on their magnitude, i.e. t can be considered as a constant in the algorithm RGLI. Therefore the time complexity of RGLI can be bounded by $\mathcal{O}(n \log n)$, as mentioned in Section 4.1.

6 Summary

We have presented a new *randomized* approximation algorithm for the subset-sum problem with time complexity $\mathcal{O}(n \log n)$ and space complexity $\mathcal{O}(n)$. Experiments with random uniformly-distributed examples of SSP show that our algorithm outperforms, both in running time and average error, Martello and Toth's quadratic greedy search, whose time complexity is $\mathcal{O}(n^2)$.

We have proposed also some conjectures on the expected error of our algorithm for uniformly-distributed examples of SSP and provided some analytical and experimental arguments justifying those conjectures.

7 Acknowledgements

I am very grateful to Emo Welzl for giving me the opportunity of working under his supervision, for the encouragement and for the invaluable hints and comments. I would like to thank also Gyula Karolyi and Nei Yoshihiro Soma for their interest and suggestions. Finally, last but not least, I want to thank Nicola Galli for his enormous help, for many corrections, improvements and stimulating discussions.

References

[BZ80] Egon Balas and Eitan Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 25:1130–1154, 1980.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[IK75] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.

[KMS98] Hans Kellerer, Renata Mansini, and Maria Grazia Speranza. Two linear approximation algorithms for the subset-sum problem. to appear in *EJOR*, 1998.

[MT84] Silvano Martello and Paolo Toth. Worst case analysis of greedy algorithms for the subset-sum problem. *Mathematical Programming*, 28:198–205, 1984.

[MT85] Silvano Martello and Paolo Toth. Approximation schemes for the subset-sum problem: Survey and experimental analysis. *European Journal of Operational Research*, 22:56–69, 1985.

[RND77] Edward M. Reingold, Jürg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms. Theory and Practice.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[SZYH95] Nei Yoshihiro Soma, Alan Solon Ivor Zinober, Horacio Hideki Yanasse, and Peter John Harley. A polynomial approximation scheme for the subset sum problem. *Discrete Applied Mathematics*, 57:243–253, 1995.

[TS86] G. Tinhofer and H. Schreck. The bounded subset sum problem is almost everywhere randomly decidable in O(n). *Information Processing Letters*, 23:11–17, 1986.