Theoretical Computer Science 14 (1981) 289–303
North-Holland Publishing Company

GENERAL APPROXIMATION ALGORITHMS FOR SOME ARITHMETICAL COMBINATORIAL PROBLEMS

Shlomo MORAN
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Communicated by M. Nivat
Received March 1979
Revised February 1980

Abstract. A general approximation technique for a large class of NP-hard optimization problems which involve arithmetic calculations is given. This technique guarantees a worst case relative error smaller than ε in time which is polynomial both in the size of the problem instance and $1/\varepsilon$. It is also shown that problems in that class which are not approximable by this technique are not approximable in polynomial time at all, provided $P \neq NP$, and hence this technique is the most general approximation technique applicable to this class.

1. Introduction

Let Z^+ denote the class of nonnegative integers. There is a large class of algorithmic problems which are generally stated as follows:

On input $a = (a_1, \ldots, a_n, b_1, \ldots, b_m) \in (Z^+)^{n+m}$, find the maximal (or minimal) integer k satisfying:

(a) k is the result of some specific arithmetic operations on $\{a_1, \ldots, a_n\}$, to be executed according to specific rules given together with the problem, while

(b) the above operations should be executed according to given restriction, depending on $\{b_1, \ldots, b_m\}$.

A simple and well-known example of such an 'arithmetical combinatorial problem' is the 'subset sum' problem: given $(a_1, \ldots, a_n, b_1, \ldots, b_n, b)$, find the maximal integer k satisfying:

(a) $k = \sum_{i=1}^{n} \varepsilon_i a_i$, where $\varepsilon_i \in \{0, 1\}$ for $i = 1, \ldots, n$, while

(b) $\sum_{i=1}^{n} \varepsilon_i b_i \leq b$.

Other examples are 'Job sequencing with deadlines' [7, 13], many scheduling problems [13, 9, 5, 4], the 'subset product' problem [10], etc.

When viewed as recognition problems, the problems mentioned are in NP, provided that the arithmetical operations and the verifications of the restrictions can be executed by polynomial time bounded algorithms. Some of those problems are also known to be NP-complete, and (hence) it is unlikely that there are polynomial time algorithms for solving them.

0304-3975/81/0000-0000/$02.50

On the other hand, for some of those problems fast approximation algorithms have recently been found, which guarantee a worst case relative error smaller than ε, for arbitrarily small positive ε, in time polynomial in both the length of the input and $1/\varepsilon$ [6, 13, 5, 2, 10]. Such problems are called 'fully approximable' [12], or are said to have a 'full polynomial time approximation scheme' [3]. (A characterization of those problems is given in [12].) We note that all of the NP-complete optimization problems, which appear in the literature (see e.g. [2]), and which are known to be fully approximable, are 'arithmetical combinatorial problems' in the sense described above.

In [12] it was shown that if a problem is fully approximable, then it must satisfy a certain condition, denoted as 'P-simplicity'. This condition (to be defined rigorously in the next section), is shown to be equivalent to the existence of an algorithm to find the optimal solution in time polynomial in both the length of the input and the value of the optimal solution. The goal of this paper is to produce a general approximation scheme, which will produce a full approximation algorithm for many problems which satisfy the condition of P-simplicity. This scheme is based on a technique called 'δ-condensation', which 'condense' several partial solutions of the problems which are 'within distance δ' (in a sense to be defined) each from the others, into one partial solution. The numerical value of δ is determined by the specific problem and by the desired accuracy, ε.

For a large class of optimization problems it will be shown that they are either fully approximable by the scheme introduced, or are not fully approximable at all (provided $P \neq NP$). It follows that the scheme given is the most general approximation scheme for problems in that class.

2. Preliminaries

The following definitions are adopted (with minor changes) from [12]: For a set A, let $P_0(A)$ denote the set of all finite subsets of A:

Definition 2.1. An NP optimization problem (NPOP) is a labelled pair $(A, t)_{\text{Ext}}$, where:

(1) Ext = Max or Ext = Min,

(2) $A \subset \Sigma^*$ is a polynomial time recognizable set.

(3) t is a function $t : A \to P_0(Z^+)$, which can be computed by a nondeterministic polynomial time bounded algorithm.

For a given a, Ext($t(a)$) (the optimum of a) is denoted as op(a).

Definition 2.2. An algorithm sol is said to solve $(A, t)_{\text{Ext}}$ if for all $a \in A$, sol(a) = op(a). (Sol(a) denotes the output of Sol on input a.)

In what follows, we shall assume that Ext = Max. The results obtained are true for the dual problems with Ext = Min, too, up to some minor changes in the procedures.

Definition 2.3. An algorithm Ap is said to be an 'ε approximation algorithm' for $(A, t)_{\text{Max}}$ if for all $a \in A$, $\text{Ap}(a) \in t(a)$ and $\text{Ap}(a) \geq (1-\varepsilon)\text{op}(a)$.

Definition 2.4. $(A, t)_{\text{Max}}$ is approximable if for all $\varepsilon > 0$, there is a polynomial time algorithm which ε approximate $(A, t)_{\text{Max}}$.

Definition 2.5. $(A, t)_{\text{Max}}$ is fully approximable if for all $\varepsilon > 0$ there is an ε-approximation algorithm to $(A, t)_{\text{Max}}$ with time complexity $Q(l(a), 1/\varepsilon)$, where Q is some fixed polynomial in two variables, and $l(a)$ is the length of a.

Definition 2.6. $(A, t)_{\text{Ext}}$ is simple if for all $k \in Z^+$, the set $\{a \in A \mid \text{op}(a) \leq k\}$ is in P. $(A, t)_{\text{Ext}}$ is rigid if it is not simple. (Note that if P = NP, then there are no rigid NPOPs.)

Definition 2.7. $(A, t)_{\text{Ext}}$ is P-simple if for all $k \in Z^+$, the set $\{a \in A \mid \text{op}(a) \leq k\}$ is recognizable in $Q(l(a), \text{op}(a))$ time, for some fixed polynomial Q.

Lemma 2.1. *The following conditions are equivalent:*

(a) $(A, t)_{\text{Ext}}$ *is* P*-simple.*

(b) *For each* $a \in A$, $\text{op}(a)$ *can be found in* $\hat{Q}(l(a), \text{op}(a))$ *time, for some polynomial* $\hat{Q}$.

Proof. $(a) \to (b)$: If $(A, t)_{\text{Ext}}$ is P-simple, then there exists a polynomial $\tilde{Q}(x, y)$ which is nondecreasing in both its variables, such that for each $a \in A$, the problem 'is $\text{op}(a) > K$?' can be solved in $\tilde{Q}(l(a), k)$ time. Thus, it is easy to see that the following algorithm finds $\text{op}(a)$ in $\text{op}(a)\tilde{Q}(l(a), \text{op}(a)) = \hat{Q}(l(a), \text{op}(a))$ time:

Begin
$k \leftarrow 0$
while $\text{op}(a) > k$ **do** $k \leftarrow k + 1$
$\text{op}(a) \leftarrow k$
end

$(b) \to (a)$: suppose (b) holds. Then there is an algorithm sol which, for each $a \in A$, find $\text{op}(a)$ in $Q(l(a), \text{op}(a))$ time, where Q is nondecreasing in its both variables. The following algorithm will recognize the set $\{a \in A \mid \text{op}(a) \leq k\}$ in $\tilde{Q}(l(a), k)$ time for some polynomial $\tilde{Q}$:

Start the execution of Sol on input a
If Sol does not stop during the first $\hat{Q}(l(a), k)$ steps **then** reject
else
  if $\text{op}(a) \leq k$ **then** accept
  else reject

The following results were obtained in [12] and are given here without proofs:

Theorem 2.1. *If* $(A, t)_{\text{Ext}}$ *is rigid, then* $(A, t)_{\text{Ext}}$ *is not approximable.*

Theorem 2.2. *If* $(A, t)_{\text{Ext}}$ *is fully approximable, then* $(A, t)_{\text{Ext}}$ *is* P-*simple.*

(In [12] it is shown that P-simplicity does not imply full approximability.)

By Theorem 2.2. and Lemma 2.1, if $(A, t)_{\text{Ext}}$ is fully approximable, then there exists an algorithm Sol, which solves $(A, t)_{\text{Ext}}$ in $\hat{Q}(l(a), \text{op}(a))$ time, for some polynomial $\hat{Q}$. In this paper algorithms such as Sol above will be modified to full approximation algorithms for a large class of $(A, t)_{\text{Ext}}$ problems. This is done by using a general 'condensing' technique, which is incorporated into certain stages of Sol. This technique is a generalization of the one appearing in [10], which was itself a variant of some techniques of [13] and others. It will be shown that the problems dealt with in this paper, which are not in P and which cannot be approximated by the above technique, must be rigid (see Definition 2.6), and hence, by Theorem 2.1, are not approximable.

To simplify the exposition we shall consider problems of the following type: Given $(a, \ldots, a_n, b)$, find the maximal integer $k \leq b$, which can be obtained by certain arithmetic operations on $(a_1, \ldots, a_n)$, to be executed according to given rules. The generalization of the approximation techniques for this problem to the more general problem presented at the beginning of this paper, will usually be obvious.

Note. For the sake of consistency with previous papers on full approximation algorithms, we use the 'uniform cost' criterion in analyzing the time complexity of algorithms, in which additions and multiplications of integers require constant time [1, Chapter 1]. It should be clear that the use of a 'logarithmic cost' criterion would increase the time of computation by at most a polynomial factor, and hence would not affect the full approximability feature of the algorithms.

3. A representation of the algorithms

We first consider a problem which is a generalization of the subset sum and subset product problems. Let $(a_1, \ldots, a_k) \in (Z^+)^k$ be given, and let $\beta_1(x, y), \ldots, \beta_{k-1}(x, y)$ be $k-1$ (not necessarily distinct) binary operations defined on the integers. With each sequence $(a_1, \beta_1, a_2, \beta_2, \ldots, a_{k-1}, \beta_{k-1}, a_k)$ we associate a 'computation sequence' $(m_1, \ldots, m_k)$ defined by:

(a) $m_1 = a_1$,

(b) for $i = 1, \ldots, k-1$, $m_{i+1} = \beta_i(m_i, a_{i+1})$.

(Thus the computation sequence of $(3, x^y, 2, x+y, 1)$ is $(3, 9, 10)$.) m_k is called the result of the computation. (The output of a single element sequence (a_1) is defined to be 0.) If $\mathcal{B}$ is a set of binary operations such that for each i, $\beta_i \in \mathcal{B}$, then $(m_1, \ldots, m_k)$ is said to be a 'computation of $(a_1, \ldots, a_k)$ over $\mathcal{B}$'.

Definition 3.1. For a given set of binary operations $\mathcal{B}$, the problem Pr1($\mathcal{B}$) is the following: given $(a_1, \ldots, a_n, b)$, find the maximal integer k satisfying:

(1) for some $i_0 \in \{1, \ldots, n\}$, k is a result of a computation of $(a_{i_0}, a_{i+1_0}, \ldots, a_n)$ over $\mathcal{B}$, and

(2) $k \leq b$.

Define a binary operation u_1 by: $u_1(x, y) = x$. Pr1(u_1, $x+y$) is a version of the well-known knapsack problem, usually called the maximal subset sum problem: 'given $(a_1, \ldots, a_n, b)$, maximize $\sum \varepsilon_i a_i (\varepsilon_i \in \{0, 1\})$, subject to $\sum \varepsilon_i a_i \leq b$.' A full approximation algorithm for this problem was first obtained by [6]. Pr1(u_1, xy) is the subset product problem, for which a full approximation algorithm was obtained in [10].

Definition 3.2. A binary operation β is 'increasing' if $\beta(x, y) \geq x$ for all x, $y > 0$ ($x, y \in Z$).

The operations $x+y$, xy, x^y, y^x, u_1, are increasing, while $x-y$ and $y-x$ are not.

Lemma 3.1. *If β contains only increasing operations, then* Pr1(β) *is* P*-simple.*

Proof. By Lemma 2.1, it suffices to show the following:

For each $a = (a_1, \ldots, a_n, b)$, op($a$) can be found in time polynomial in $l(a)$ and op(a) (w.l.g. $a_i \leq b$ for $i = 1, \ldots, n$). The following algorithm Sol finds op(a) in $O(n \text{ op}(a) \log(\text{op}(a)))$ time:

Sol: Input $a = (a_1, \ldots, a_n, b)$. Output: op($a$)
begin
  $i \leftarrow 1$; $T \leftarrow \emptyset$
1. **if** $T \neq \emptyset$ **then** //insert 'partial computations' which contain a_i in T//
    begin
      for each s in T **do**
      $T \leftarrow T \cup \{\beta(s, a_i) \mid \beta \in B \wedge \beta(s, a_i) \leq b\}$[1]
    end
  $T \leftarrow T \cup \{a_i\}$
if $i = n$ **then** [halt; return max(T)]
else [$i \leftarrow i+1$; **goto** 1]
end

It is easy to see that at the termination of Sol, T is equal to $t(a)$, and hence $\max(T) = \text{op}(a)$. By using an appropriate data structure to represent T (e.g. 2–3 trees, see [1, 4.9–4.10]), the execution time of lines 2–5 is $O(|\mathcal{B}||T|\log|T|)$. The

[1] If $u_1 \notin B$, then s should be deleted from T before the combining of the set $\{\beta(s, a_i) \mid \beta \in \mathcal{B} \wedge \beta(s, a_i) \leq b\}$ with T.

execution time of the other lines is a constant. $|\mathscr{B}|$ is a constant, and since T contains only positive integers $\leqslant \mathrm{op}(a)$, $|T| \leqslant \mathrm{op}(a)$ all through the execution of the algorithm. It follows that the time complexity of the algorithm as a whole is $O[n\ \mathrm{op}(a) \log \mathrm{op}(a)]$.

Note. If for all $\beta \in \mathscr{B}$, β also satisfies the following: For all $x_1, x_2, y > 0$, $x_1 \geqslant x_2 \rightarrow \mathscr{B}(x_1, y) \geqslant \mathscr{B}(x_2, y)$, then the time complexity of Sol above can be reduced to $O[n\ \mathrm{op}(a)]$, by keeping T as an ordered list. (See e.g. [8, Section 2] for details.) Since all of the increasing binary operations mentioned in this paper satisfy the property above, we shall use this fact in analyzing the time complexity of the algorithms which will be given later in this paper.

Lemma 3.1 provide us with a necessary condition for full approximability. We shall show that if $\mathscr{B} \subset \{xy, x+y, u_1\}$, then algorithm Sol introduced in Lemma 3.1 can be modified to a full approximation algorithm for $\mathrm{Pr1}(\mathscr{B})$. First we need several lemmas. Let R denote the set of real numbers.

Lemma 3.2. *For all $r \geqslant 2$, for all $n > 1$ $(r \in R, n \in Z)$, if $k \leqslant r^n$, then* $(1 - 1/r^{n+1})^k > 1 - 1/r$.

Proof. Use the binomial expansion of $(1 - 1/r^{n+1})^k$.

The technique which is used to obtain a full approximation algorithm from a given algorithm Sol is the 'δ-condensation', which is defined and investigated below. The role of this technique is to keep $|T|$ relatively small, even when $\mathrm{op}(a)$ is large. This is done by deleting from T, at each stage, elements which are 'δ-close' to some other element s in T. (That is: to 'condense' those elements into s.) This δ-condensation may cause that at the end of the computation $\mathrm{Max}(T(a))$ is smaller than $\mathrm{op}(a)$, and it must be done in such a way that the difference $\mathrm{op}(a)$-$\mathrm{Max}(t(a))$ remains small relatively to $\mathrm{op}(a)$. If $\mathscr{B} \subset \{x+y, u_1\}$, then the various rounding techniques of [6, 13, 8] form an appropriate such condensation for $\mathrm{Pr1}(\mathscr{B})$, which is the subset sum problem. But those techniques fail for $\mathrm{Pr1}(\{x \cdot y, u_1\})$ (that is: for the subset product problem) [10, 11]. The technique introduced here will be shown to be efficient for $\mathrm{Pr1}(\mathscr{B})$, where $\mathscr{B}$ is any subset of $\{x+y, x \cdot y, x^y, y^x, u_1\}$, and for many other related problems.

Definition 3.3. Let $(m_1, \ldots, m_k)$ be the computation sequence of $(a_1, \beta_1, a_2, \beta_2, \ldots, a_{k-1}, \beta_{k-1}, a_k)$ and let $0 < \delta \leqslant 1$. A sequence $(m'_1, m'_2, \ldots, m'_k)$ is a δ-condensation of $(m_1, \ldots, m_k)$ if there exists a sequence $(h_1, \ldots, h_k) \in R^k$, $0 \leqslant h_i \leqslant \delta$, such that:

(a) $m'_1 = m_1(1 - h_1) = a_1(1 - h_1)$,

(b) $m'_{i+1} = [\beta_i(m'_i, a_{i+1})](1 - h_{i+1})$ $(i = 1, \ldots, k-1)$.

Lemma 3.3. *Let* $(m_1, \ldots, m_k)$ *be a computation sequence of* $(a_1, \beta_1, a_2, \ldots a_{k-1}, \beta_{k-1}, a_k)$ *and let for* $i = 1, \ldots, k-1$, $\beta_i \in \{x+y, x \cdot y, u_1\}$. *Let* r *be a real number,* $r \geq k$, $\delta = 1/r^2$. *If* $(m'_1, \ldots, m'_k)$ *is a* δ*-condensation of* $(m_1, \ldots m_k)$, *then* $1 \geq m'_k/m_k \geq 1 - 1/r$.

Proof. Clearly $0 \leq m'_k \leq m_k$, and hence $1 \geq m'_k/m_k$. For the second inequality, we prove by induction for $j = 1, \ldots, k$ that $m'_j/m_j \geq (1-1/r^2)^j$. Substituting $j = k$ and using Lemma 3.2 with $n = 1$, we obtain

$$\frac{m'_k}{m_k} \geq \left(1 - \frac{1}{r^2}\right)^k > 1 - \frac{1}{r}$$

as desired.

For $j = 1$, $m'_1 = m_1(1-h_1) \geq m_1(1-1/r^2)^1$. Suppose now that

$$\frac{m'_i}{m_j} \geq \left(1 - \frac{1}{r^2}\right)^j.$$

We shall prove that $m'_{j+1}/m_{j+1} \geq (1-1/r^2)^{j+1}$.

If $\beta_j = u_1$, then

$$\frac{m'_{j+1}}{m_{j+1}} = \frac{m'_j(1-h_{j+1})}{m_j} \geq \frac{m_j(1-1/r^2)^j(1-h_{j+1})}{m_j} \geq \left(1 - \frac{1}{r^2}\right)^{j+1}.$$

If $\beta_j = xy$ then

$$\frac{m'_{j+1}}{m_{j+1}} = \frac{m'_j a_{j+1}(1-h_{j+1})}{m_j a_{j+1}} \geq \frac{m_j(1-1/r^2)^j a_{j+1}(1-h_{j+1})}{m_j a_{j+1}} \geq \left(1 - \frac{1}{r^2}\right)^{j+1}.$$

If $\beta_j = x + y$, then

$$\frac{m'_{j+1}}{m_{j+1}} = \frac{(m'_j + a_{j+1})(1-h_{j+1})}{m_j + a_{j+1}} \geq \frac{(m_j(1-1/r^2)^j + a_{j+1})(1-h_{j+1})}{m_j + a_{j+1}}$$

$$\geq \frac{(m_j + a_{j+1})(1-1/r^2)^j(1-h_{j+1})}{m_j + a_{j+1}} \geq \left(1 - \frac{1}{r^2}\right)^{j+1}.$$

Lemma 3.4. *Let* $1 \leq a_1 < a_2 < \cdots < a_t \leq b$ *be given, such that* $a_i/a_{i+1} \leq 1 - \delta$, *where* $0 < \delta \leq 1$. *Then* $t \leq \lfloor (2 \ln b)/\delta \rfloor$.

Proof. First, we note that $a_{i+1}/a_i > 1 + \delta$. Hence $(1+\delta)^{t-1} \leq a_1(1+\delta)^{t-1} \leq a_t \leq b$, or: $(1+\delta)^{t-1} \leq b$. Taking logarithms to base $1+\delta$, we get:

$$t - 1 \leq \log_{1+\delta} b = \frac{\ln b}{\ln(1+\delta)} = \frac{\ln b}{\delta - \frac{1}{2}\delta^2 + \frac{1}{3}\delta^3 \cdots} < \frac{2 \ln b}{\delta},$$

i.e. $t < (2 \ln b)/\delta + 1$.

The result follows from the fact that t is an integer.

The following algorithm, Ap1, is a fully approximation algorithm for Pr1($\mathcal{B}$), where $\mathcal{B}$ is any subset of $\{x+y, x\cdot y, u_1\}$.

Ap1: Input: $a=(a_1,\ldots,a_n,b)\in(Z^+)^{n+1}$, $\varepsilon>0$. Output: An integer $k\in t(a)$, such that $\mathrm{op}(a)\geq k\geq \mathrm{op}(a)(1-\varepsilon)$

begin

1. $r\leftarrow \mathrm{Max}(1/\varepsilon, n)$
2. $\delta\leftarrow 1/r^2$ //δ is the 'condensing' parameter//
3. $i\leftarrow 1$; $T\leftarrow\emptyset$
4. **if** $T\neq\emptyset$ **then**
   begin
5. **for** every $s\in t$ **do**
6. $T\leftarrow T\cup\{\beta(s,a_i)\mid \beta\in B\wedge\beta(s,a_i)\leq b\}$[1]
   end
7. $T\leftarrow T\cup\{a_i\}$
8. Sort T//assume $T=(s_1,\ldots,s_t)$, $s_i<s_{i+1}$//
9. **if** $i=n$ **then** [halt; **return** s_t]
10. //condensing// $j\leftarrow 1$, $k\leftarrow 2$
11. **while** $k\leq t$ *and* $s_j/s_k>1-\delta$ **do**
    begin
12. $T\leftarrow T-\{s_k\}$; $k\leftarrow k+1$
    end
13. **if** $k<t$ **then**
    begin
14. $j\leftarrow k$; $k\leftarrow k+1$
15. **go to** 11
    end
    else//The condensing is finished//
16. $i\leftarrow i+1$, **go to** 4

end

Theorem 3.1. *For each* $\varepsilon>0$, *algorithm* Ap1 *provides an* ε*-approximation to* Pr1($\mathcal{B}$), *where* $\mathcal{B}\subset\{x+y, xy, u_1\}$, *in* $O(\max\{l^4(a), l^2(a)/\varepsilon^2\})$ *time.*

Proof. By the definition of the problem, $\mathrm{op}(a)$ is an output of a computation of a sequence $(a_{i_0}, a_{i_0+1},\ldots,a_n)$ over $\mathcal{B}$, for some $1\leq i_0\leq n$.

Let the computation sequence be $(m_{i_0}, m_{i_0+1},\ldots,m_n)$, $(m_n=\mathrm{op}(a))$, and the corresponding sequence be $(a_{i_0},\beta_{i_0},a_{i_0+1},\beta_{i_0+1},\ldots,\beta_{n-1},a_n)$.

Suppose that Ap1 is carried out with $a=(a_1,\ldots,a_n,b)$ and ε as input. Denote by T_i, for $i=1,2,\ldots,n-1$, the content of T at the termination of the δ-condensation at the ith stage (i.e. just before line 16 is encountered for the ith time). T_n will denote the contents of T at the termination of the algorithm. We shall prove by induction that for $i=i_0, i_0+1,\ldots,n$, there is in T_i and element m'_i, such that the sequence m'_{i_0}, $m'_{i_0+1},\ldots,m'_n$ is a δ-condensation of $(m_{i_0},\ldots,m_n)$, (with $\delta=1/r^2$).

When $i = i_0$, it is easily checked that either a_{i_0} is in T_{i_0} (a_{i_0} is inserted in T at line 7), which means that a_{i_0} was not deleted from T during the condensing, or there is an integer $\hat{m}_{i_0}$ in T_{i_0}, such that $1 > \hat{m}_{i_0}/a_{i_0} > 1-\delta$. In the first case we take m'_{i_0} to be a_{i_0}, and in the second case we take m'_{i_0} to be $\hat{m}_{i_0}$. In both cases $m'_{i_0}/m_{i_0} > 1-\delta$.

Suppose now that $(m'_{i_0}, m'_{i_0+1}, \ldots, m'_i)$ is a δ-condensation of $(m_{i_0}, \ldots, m_i)$, and that $m'_i \in T_i$. By the same argument as above, either $\beta_i(m'_i, a_{i+1})$ is in T_{i+1}, or there is an integer $\hat{m}_{i+1}$ in T_{i+1} such that

$$1 > \frac{\hat{m}_{i+1}}{\beta_i(m'_i, a_{i+1})} > 1-\delta.$$

If we take m'_{i+1} to be $\beta_i(m'_i, a_{i+1})$ in the first case, or $\hat{m}_{i+1}$ in the second case, we have that $(m'_{i_0}, \ldots, m'_i, m'_{i+1})$ is a δ-condensation of $(m_{i_0}, \ldots, m_i, m_{i+1})$.

By Lemma 3.3,

$$1 \geqslant \frac{m'_n}{m_n} = \frac{m'_n}{\mathrm{op}(a)} > 1-\varepsilon.$$

Since the output of Ap1, denoted as m^*_n, is $\max(T_n) \geqslant m'_n$, we have

$$1 \geqslant \frac{m^*_n}{\mathrm{op}(a)} \geqslant \frac{m'_n}{\mathrm{op}(a)} > 1-\varepsilon.$$

Hence, m^*_n is ε-approximation to $\mathrm{op}(a)$.

To prove the complexity, we first note that for each i, $|T_i| \leqslant O(2l(a)/\delta)$. This follows from Lemma 3.4, with $\ln(b) = O(l(a))$.[2] It follows that $|T|$ is $O(l(a)/\delta)$ all through the algorithm. By the note after Lemma 3.1, the time complexity of the algorithm as a whole is $O(nl(a)/\delta)$. Since $n < l(a)$, this is equal to $O(l(a)^4)$ if $l(a) \geqslant 1/\varepsilon$, and to $O(l(a)^2/\varepsilon^2)$ if $l(a) < 1/\varepsilon$.

Note. Algorithm Ap1 not only provides an ε-approximation to $\mathrm{op}(a)$, but it also provides an ε-approximation to each $m \in t(a)$. This property, (which is not shared by the previous approximation techniques of [6, 13]) can be used for simultaneously approximating several problems, with the same $(a_1, \ldots, a_n)$ and ε, but different b's.

We shall show now that Ap1 can be generalized to a fully approximation algorithm for $\mathrm{Pr1}(\mathscr{B})$, where $\mathscr{B}$ is any subset of $\{x+y, x \cdot y, x^y, u_1\}$ (i.e. the operation $m_i^{a_{i+1}}$ may be executed too).[3]

Lemma 3.5. *Let k, l, b_1, $b_2, \ldots, b_l$ be $l+2$ positive integers, where $b_i \geqslant 2$ for $i = 1, \ldots, l$. Let a 'legal sequence' be a sequence $(f_1, \ldots, f_n)$ where $n = k+l$, which*

[2] Under the logarithmic cost criterion, $\ln(b) = O(l(a))$. Under the uniform cost criterion, $\ln(b) \leqslant C$, where C is some constant (C = the amount of storage in each register, see [1, 1.3]).

[3] Note that if $m_i^{a_{i+1}} \leqslant b$, then $m_i^{a_{i+1}}$ can be computed in $O(\log^2 b)$ time and hence also in $O(l(a)^2)$ time, where $a = (a_1, \ldots, a_n, b)$.

satisfies the following:

(a) $f_1 = 1$;

(b) $f_{j+1} = f_j + 1$ *or* $f_{j+1} = f_j b_i$ *for some* $1 \leq i \leq l$, *and for each* i *there is a unique* j *s.t.* $f_{j+1} = f_j b_i$.

Let $f = \max\{f_n \mid (f_1, \ldots, f_n)$ *is a legal sequence*$\}$. *Then* $f = kb_1 b_2 \cdots b_l$ (*i.e.* $(f_1, \ldots, f_n) = (1, 2, \ldots, k, kb_1, kb_1 b_2, \ldots, kb_1 \ldots b_l)$).

The easy proof of this lemma is omitted.

Lemma 3.6. *Let* $(m_1, \ldots, m_k) \in (Z^+)^k$ *and* $(a_1, \ldots, a_l) \in (Z^+)^l$ *be given, such that* $m_i \leq m_{i+1}$ *for* $i = 1, \ldots, k-1$ *and* $l < k$. *Assume that for some* $i_1, \ldots, i_l$, $1 \leq i_1 < i_2 \cdots < i_l < k$:

(a) $m_{i_1} \geq 2$;

(b) $m_{i_j+1} = m_{i_j}^{a_j}$.

Then $a_1 a_2 \cdots a_l \leq \log_2 m_k$.

Proof. Since $i_{j+1} \geq i_j + 1$, $m_{i_{j+1}} \geq m_{i_j+1} = m_{i_j}^{a_j}$. Hence we have that

$$m_{i_2} \geq m_{i_1}^{a_1},\ m_{i_3} \geq m_{i_2}^{a_2} \geq m_{i_1}^{a_1 a_2},\ \ldots,\ m_{i_l} \geq m_{i_1}^{a_1 a_2 \cdots a_{l-1}}.$$

Since $m_{i_1} \geq 2$, $m_k \geq m_{i_l+1} = m_{i_l}^{a_l}$, we have that $m_k \geq m_{i_1}^{a_1 a_2 \cdots a_l} \geq 2^{a_1 a_2 \cdots a_l}$.

Lemma 3.7. *Let* $(m_1, \ldots, m_k)$ *be the computation sequence of* $(a_1, \beta_1, a_2, \ldots, a_{k-1}, \beta_{k-1}, a_k)$, *where for* $i = 1, \ldots, k-1$, $\beta_i \in \{x+y, x \cdot y, x^y, u_1\}$. *Let* r *be a real number,* $r \geq \max\{k, \log_2 m_k\}$, *and let* $\delta = 1/r^3$.

If $(m'_1, \ldots, m'_k)$ *is a* δ*-condensation of* $(m_1, \ldots, m_k)$, *then* $1 \geq m'_k/m_k > 1 - 1/r$.

Proof. Without loss of generality we may assume that if $\beta_i = x^y$, then $m_i \geq 2$ and $a_{i+1} \geq 2$. (Otherwise replace β_i by u_1.) With the sequence $(a_1, \beta_1, \ldots, \beta_{k-1}, a_k)$, let us associate a sequence $(g_1, \ldots, g_k)$ as follows:

(a) $g_1 = 1$,

(b) $g_{i+1} = [$**if** $\beta_1 \neq x^y$ **then** $g_i + 1$, **else** $g_i a_{i+1} + 1]$.

(Thus, with $(2, x^y, 3, x+y, 6, x^y, 2)$ we associate the sequence $(1, 4, 5, 11)$.)

If we replace in the sequence $(g_1, \ldots, g_k)$, each g_{i+1} which is equal to $g_i a_{i+1} + 1$, with the two elements $(g_i a_{i+1}, g_i a_{i+1} + 1)$, then we obtain a new sequence of length $k + l$, where $l = |\{i \mid \beta_i = x^y\}|$. (Thus, the sequence $(1, 4, 5, 11)$ above will be replaced by $(1, 3, 4, 5, 10, 11)$.)

This new sequence satisfies the conditions of a 'legal sequence' of Lemma 3.4, with $k, l, a_{i_1+1}, a_{i_2+1}, \ldots, a_{i_l+1}$ (where $i_1, \ldots, i_l$ are the indices for which $B_{i_j} = x^y$). Hence $g_k \leq k a_{i_1+1} \cdots a_{i_l+1}$. Moreover, the sequences $(m_1, \ldots, m_k)$ and $(a_{a_1+1}, \ldots, a_{i_l+1})$ satisfy the conditions of Lemma 3.6, and hence $a_{i_1+1} a_{i_2+1} \cdots a_{i_l+1} \leq \log_2 m_k$. Combining the above results, we have that $g_k \leq k \log_2 m_k \leq r^2$.

We shall now prove, by induction, that for $j = 1, \ldots, k$, $m'/m_j > (1 - 1/r^3)^{g_j}$: For $j = 1$ the hypothesis is true by definition.

If the hypothesis is true for some $j \geq 1$, and $\beta_j \in \{x+y, x \cdot y, u_1\}$, then

$$g_{j+1} = g_j + 1 \quad \text{and} \quad \frac{m'_{j+1}}{m_{j+1}} \geq \left(1-\frac{1}{r^3}\right)^{g_j+1} = \left(1-\frac{1}{r^3}\right)^{g_{j+1}}$$

by an argument similar to that of Lemma 3.3.

If $\beta_j = x^y$, then

$$\frac{m'_{j+1}}{m_{j+1}} = \frac{(m_j'^{a_{j+1}}(1-h_j))}{m_j^{a_{j+1}}} \geq \frac{(m_j(1-1/r^2)^{g_j})^{a_{j+1}}(1-h_j)}{m_j^{a_{j+1}}}$$

$$= \left(1-\frac{1}{r^3}\right)^{g_j a_{j+1}}(1-h_j) \geq \left(1-\frac{1}{r^3}\right)^{g_j a_{j+1}+1} = \left(1-\frac{1}{r^2}\right)^{g_{j+1}}.$$

Substituting in the above inequality $j = k$, we obtain:

$$\frac{m'_k}{m_k} \geq \left(1-\frac{1}{r^3}\right)^{g_k} \geq \left(1-\frac{1}{r^3}\right)^{r^2} > 1-\frac{1}{r}.$$

(The last inequality is by Lemma 3.2, with $n = 2$.)

Let Ap2 be algorithm Ap1 in which lines 1 and 2 are replaced by:

(1′) $r \leftarrow \max\{1/\varepsilon, n, \log_2 b\}$,

(2′) $\delta \leftarrow 1/r^3$.

Theorem 3.2. *For each* $\varepsilon > 0$, Ap2 *provides an* ε*-approximation to* Pr1($\mathcal{B}$), *where* $\mathcal{B}$ *is any subset of* $\{x+y, x \cdot y, x^y, u_1\}$, *in* $O(\max\{l(a)^2/\varepsilon^3, l(a)^5\})$ *time.*

Proof. Let $a = (a_1, \ldots, a_n, b)$. Noting that $n = O(l(a))$ and $\log_2 b = O(l(a))$, the proof is very similar to that of Theorem 3.1, and is omitted.

Although it will not be shown here, Ap2 can be extended to a fully approximation algorithm for Pr1($\mathcal{B}$), where $\mathcal{B}$ is any subset of $\{x+y, x \cdot y, x^y, y^x, u_1\}$, which is the set of all 'elementary' increasing operations on the integers [11].

4. Generalization

We shall now consider the following generalization of Pr1:

Definition 4.1. Let $\mathcal{B}$ be a set of binary operations defined on the integers, and let D be a (possibly infinite) set of integers. Pr2 ($\mathcal{B}$, D) is the following problem: Given $(a_1, \ldots, a_n, b)$, find the maximal integer k satisfying:

(1) For some $i_0 \in \{1, 2, \ldots, n\}$ and for some $(x_{i_0}, x_{i_0+1}, \ldots, x_n) \in D^{n-i_0+1}$, k is the result of a computation of $(x_{i_0}a_{i_0}, x_{i_0+1}a_{i_0+1}, x_n a_n)$ over B.

(2) $k \leq b$.

$\mathrm{Pr2}(\{x+y, u_1\}, Z^+) = \mathrm{Pr2}(\{x+y\}, Z^+)$ is the unbounded knapsack problem: Given $(a_1, \ldots, a_n, b)$, find the maximal integer $k \leq b$ such that $\sum a_i x_i = k$ has a nonnegative integer solution.

The following definition, adopted from [12], is used in the proof of the next lemma.

Definition 4.2. Let $(A, t)_{\mathrm{Ext}}$ and $(B, t_2)_{\mathrm{Ext}}$ be NPOP's. A function $g: \Sigma^* \to \Sigma^*$ is a measure preserving reduction of the former to the latter if:

(a) $g(a) \in B \leftrightarrow a \in A$;

(b) $\forall a \in A,\ t_1(a) = t_2(g(a))$.

Lemma 4.1. *If* $\mathcal{B} \subset \{x+y, x \cdot y, x^y, u_1\}$, *then* $\mathrm{Pr2}(\mathcal{B}, Z^+)$ *is fully approximable.*

Proof. By [12], if $(B, t_2)_{\mathrm{Ext}}$ is full approximable and there is a polynomial time measure preserving reduction of $(A, t_1)_{\mathrm{Ext}}$ to $(B, t_2)_{\mathrm{Ext}}$, then $(A, t_1)_{\mathrm{Ext}}$ is also fully approximable. Hence, in order to prove the lemma, it suffices to show that $\mathrm{Pr2}(\mathcal{B}, Z^+)$ is reducible to $\mathrm{Pr1}(\mathcal{B})$, which was shown to be fully approximable in the previous section, by a polynomial time measure preserving reduction.

We shall introduce a polynomial time measure preserving reduction of $\mathrm{Pr2}(\{x+y, u_1\}, Z^+)$ to $\mathrm{Pr1}(\{x+y, u_1\})$. Other cases may be proved similarly (although the proof may be a little more involved): Let $a = (a_1, \ldots, a_n, b)$ be an input to $\mathrm{Pr2}(B, Z^+)$. For each i, replace a_i by a sequence $(a_i, 2a_i, 4a_i, \ldots, 2^{l_i}a_i)$, where $l_i = \lfloor \log_2 b/a_i \rfloor$. The resulting sequence $a' = (a_1, 2a_1, \ldots, 2^{l_1}a_1, a_2, \ldots, 2^{l_n}a_n, b)$ will be taken as an input to $\mathrm{Pr1}(\{x+y, u_1\})$. It is left to the reader to check that the reduction above is a polynomial time measure preserving reduction.

5. Complexity results – nonapproximable problems

In some arithmetical combinatorial problems, it is required that all integers appearing in the input sequence take part in the computation (e.g., the Partition Problem: given $(a_1, \ldots, a_n)$, does $\sum x_i a_i = 0$ have a $\{-1, 1\}$ solution?). To distinguish these problems from others, we adopt the following convention: If $u_1 \notin \mathcal{B}$, then $\mathrm{Pr2}(\mathcal{B}, D)$ is the following problem: given $(a_1, \ldots, a_n, b)$, find the maximal integer k satisfying:

(1) For some $(x_1, \ldots, x_n) \in D^n$, k is a result of a computation of $(x_1 a_1, x_2 a_2, \ldots, x_n a_n)$ over $\mathcal{B}$.

(2) $k \leq b$.

$\mathrm{Pr2}(\{x+y\}, \{-1, 1\})$ is the problem: given $(a_1, \ldots, a_n, b)$, find the maximal integer $\kappa \leq b$, such that $k = \sum x_i a_i$ has a $\{-1, 1\}$ solution. By the NP-completeness of the Partition Problem [7], this problem can be easily shown to be rigid (provided $\mathrm{P} \neq \mathrm{NP}$), and hence, by Theorem 2.1, it is not approximable.

$\mathrm{Pr2}(\{x+y, u_1\}, \{-1, 1\})$ is the problem: On input $(a_1, \ldots, a_n, b)$ find the maximal $k \leq b$ such that $k = \sum x_i a_i$ has a $\{-1, 0, 1\}$ solution. It can be shown that this problem

is either in P, or it is rigid and (hence) not approximable [11]. On the other hand, $\mathrm{Pr2}(\{x+y\}, Z)$ is the problem: On input $(a_i, \ldots, a_n, b)$, find the maximal $k \leq b$ such that $k = \sum x_i a_i$ has an integer solution. This problem is equivalent to finding the maximal $k \leq b$ which is divisible by the greatest common divisor of $a_1, \ldots, a_n$, which by Euclid algorithm, is in P.

Table 1 summarizes results on the complexity and approximability of related problems. Some of the results were proved above, and the proofs of other results are not hard, and may be found in [10, 11].

Table 1
Complexity and approximability of arithmetical combinatorial problems

$\mathscr{B}$	D	Complexity of $\mathrm{Pr2}(\mathscr{B}, D)$	Approximability of $\mathrm{Pr2}(\mathscr{B}, D)$
$\{x+y, u_1\}$ or $\{xy, u_1\}$ or $\{x+y, xy, u_1\}$ or $\{xy, x^y, u_1\}$ or $\{x+y, x^y, u_1\}$	$\{1\}$	NPC[a]	fully approximable
$\{x^y, u_1\}$	$\{1\}$	P	
$\{x+y\} \subset \mathscr{B} \subset \{x+y, xy, x^y, u_1\}$	Z^+	NPC	fully approximable
$\{xy, x^y, u_1\}$	Z^+	P	
$\{x+y\} \subset \mathscr{B} \subset \{x+y, xy, x^y\}$	$\{-1, 1\}$	NPC	rigid[4]
$\{x+y, u_1\} \subset \mathscr{B} \subset \{x+y, xy, x^y, u_1\}$	$\{-1, 1\}$	either in P, or [not in P and rigid]	

[a] P stands for polynomial time complexity, NPC for NP-Complete.

Each of the problems listed in Table 1 (and many others) which is not in P, is either fully approximable by algorithm Ap2, or is rigid,[4] and hence is not approximable by a polynomial time algorithm at all.

Different versions of Pr2 may be obtained by changing the order by which elements from $\{a_1, \ldots, a_n\}$ take part in the computation. Consider the following generalization of Pr2, to be denoted as Pr3; Let $\mathscr{B}$ be a set of operations and let D be a set of integers. For each positive integer n, let F_n be the set of all functions from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n\}$ and let C_n be a subset of F_n. $\mathrm{Pr3}(B, D, \{C_n\})$ is the following problem: on input $(a_1, \ldots, a_n, b)$, find the maximal integer $k \leq b$ satisfying: For some $i_0 \in \{1, 2, \ldots, n\}$,[5] for some $(x_{i_0}, \ldots, x_n) \in D^{n-i_0+1}$, and for some $\sigma \in C_n$, k is a result of a computation of $(x_{i_0} a_{\sigma(i_0)}, x_{i_0+1} a_{\sigma(i_0+1)}, \ldots, x_n a_{\sigma(n)})$ over $\mathscr{B}$.

We shall consider three cases:

(i) For each n, $C_n = \{I_n\}$, where I_n is the identity function. This case is represented by Pr2.

(ii) For each n, $C_n = F_n$.

(iii) For each n, $C_n = S_n$, the set of permutation of order n.

[4] Provided that $\mathrm{P} \neq \mathrm{NP}$.

[5] If $u_1 \notin \mathscr{B}$, then $i_0 = 1$.

Theorem 6.1. *If* $B \subset \{x+y, x\cdot y, x^y, u_1\}$ *and* $D = \{1\}$ *or* $D = Z^+$, *then* Pr3$(B, D, \{F_n\})$ *is fully approximable.*

Outline of proof. Suppose first that $D = \{1\}$. Let Ap3 be algorithm Ap2 with lines 3–7 replaced by:

($\tilde{3}$) $i \leftarrow 1$; $T \leftarrow \{a_1, \ldots, a_n, 0\}$; $A \leftarrow \{a_1, \ldots, a_n\}$.

($\tilde{4}$) **For** every s in T **and for** every $a \in A$ **do**

($\tilde{5}$) $T \leftarrow T \cup \{\beta(s, a) \mid \beta \in B \wedge \beta(s, a) \leq b\}$.[1]

Thus defined, Ap3 is a fully approximation algorithm to Pr3$(B, D, \{E\})$, with time complexity $O(\max\{l(a)^6, l(a)^3\varepsilon^3\})$. This can be proved in the same way as Theorems 3.1 and 3.2, with the exception that lines ($\tilde{4}$)–($\tilde{5}$) require $O(|T|\cdot n)$ time, instead of the $O(|T|)$ time that was required for lines (4)–(7) in Ap1 and Ap2.

If $D = Z^+$, then the same measure preserving reductions exhibited in Lemma 4.1 may be used to prove the result.

Pr3$(\mathcal{B}, D, \{S_n\})$ seems to be much harder. We do not even know whether Pr3$(\{x+y, x\cdot y, u_1\}, \{1\}, \{S_n\})$ is P-simple. The question whether this problem is fully approximable therefore remains open. Another (open) problem is the following: Is there any arithmetical combinatorial problem (in the sense of this paper), which cannot be approximated by the techniques introduced in this paper, but which is fully approximable by some other method? A third open problem is: Is the set $\{(a_1, \ldots, a_n) \mid (\forall i, a_i \in Z^+) \wedge (\sum x_i a_i = 0$ has a non trivial $\{-1, 0, 1\}$ solution$)\}$ in P? A positive answer would imply that Pr2$(\{x+y\}, \{-1, 0, 1\})$ is polynomially solvable, while a negative answer would imply that Pr2$(\{x+y\}, \{-1, 0, 1\})$ is rigid.

Acknowledgment

This paper is a part of the author's Ph.D. Thesis, supervised by Professor Azaria Paz, to whom the author wishes to express his thanks. The author also wishes to express his thanks to an anonymous referee, for his suggestions to improve the representation of the paper.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading, MA, 1974).

[2] M.R. Garey and D.S. Johnson, Approximation algorithms for combinatorial problems: an annotated bibliography, in: J. Traub, Ed., *Algorithms and Complexity* (Academic Press, New York, 1976) 41–52.

[3] M.R. Garey and D.S. Johnson, Strong NP-completeness results: motivations, examples, and applications, *J. ACM* 25 (1978) 499–508.

[4] R.L. Graham, Bounds on multiprocessing time anomalies, *SIAM J. Appl. Math.* 17 (1969) 416–429.

[5] E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *J.ACM* **23** (1975) 317–327.

[6] O.H. Ibarra and C.E. Kim, Fast approximation algorithms for knapsack and sum of subset problems, *J.ACM* **22** (1975) 463–468.

[7] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, Eds, *Complexity of Computer Computations* (Plenum Press, New York, 1972) 85–103.

[8] E.L. Lawler, Fast approximation algorithms for knapsack problems, *Proc. 18th Annual Symposium on Foundations of Computer Science* (1977) pp. 206–213.

[9] J.K. Lenstra, A.H.G. Rinooy Kan and P. Brucker, Complexity of machine scheduling problems, *Ann. Discrete Math.* **1** (1977) 343–362.

[10] S. Moran, Efficiently approximable problems, TR 135, Technion, Israel Institute of Technology, Department of Computer Science (1978).

[11] S. Moran, NP optimization problems, D.Sc. Thesis, Technion, Israel Institute of Technology.

[12] A. Paz, and S. Moran, NP optimization problems and their approximation, to appear (abridged version in: Proc. 4th *International Colloquium on Automata, Language and Programming* (1977) 370–379).

[13] S. Sahni, Algorithms for scheduling independent tasks, *J.ACM* **23** (1976) 116–127.