GENERAL APPROXIMATION ALGORITHMS FOR SOME 
ARITHMETICAL COMBINATORIAL PROBLEMS

by

Shlomo Moran

Technical Report #140

December 1978
1. INTRODUCTION

There is a large class of problems which are generally stated as follows:

On input \( a = (a_1, \ldots, a_n, b_1, \ldots, b_m) \in \mathbb{Z}^{n+m} \) (\( \mathbb{Z} \) denotes the set of nonnegative integers), find the maximal (minimal) integer \( K \) satisfying the following:

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

(b) The above operations should be executed subject to some 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:

Input: \( (a_1, \ldots, a_n, b_1, \ldots, b_n, b) \). Output: the maximal integer \( K \) satisfying:

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

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

Other examples are the "Job Sequencing with Deadline" ([Ka72], [Sa76]), a large class of scheduling problems ([Sa76], [LKB 77], [HS76], [Gr69]), the subset product problem ([M78]), etc.

When viewed as recognition problems, those problems are in \( \mathbb{NP} \) (provided that the arithmetical operations and the verification of the restrictions can be executed by a polynomial time bounded algorithms). Some such problems are also known to be \( \mathbb{NP} \) complete, and hence, probably, there are no polynomial time algorithms for solving them.
On the other hand, for some of these problems fast approximation algorithms, which guarantee a worst case relative error smaller than $\varepsilon$, for arbitrarily small positive $\varepsilon$, in time which is polynomial in both the length of the input and $\frac{1}{\varepsilon}$, have recently been found ([Ik75], [Sa76], [HS76], [GJ76a], [M78]). Those problems are denoted as "fully approximable" ([PM77]), or as problems which have a "fully polynomial time approximation scheme" ([GJ76b]). (A characterization of those problems is given in [M78].)

Moreover, all of the NP complete optimization problems, which appear in the literature (see e.g. [GJ76a]) and are known to be fully approximable are "arithmetical combinatorial problems" in the sense described above.

In [PM77] it was shown that if a problem is fully approximable, then it must satisfy a certain condition, denoted as "p-simplicity" (to be defined in the next section). The goal of this paper is to provide a general approximation technique, which uses the p-simplicity of a given problem to get a fully approximation algorithm for it.

For a large class of hard problems it will be shown that every problem in that class is either fully approximable by the technique introduced, or it is not fully approximable at all (provided $P \neq NP$); and hence this technique is the most general approximation technique for problems in that class.

2. PRELIMINARIES

The following definitions are adopted (with slight changes) from [PM77] (see also [M78]).
Definition 2.1: An NP optimization problem (NPOP) is a labeled pair $(A, t)_{\text{Ext}}$ where:

1. $\text{Ext} = \text{Max}$ or $\text{Ext} = \text{Min}$;
2. $A \subseteq \mathbb{E}$ is a polynomial time recognizable set;
3. $t$ is a function, $t: A \rightarrow P_0(Z^+)$ (set of all finite subsets of $Z^+$), such that $t$ can be computed by a nondeterministic polynomial time bounded algorithm.

For a given $a$, $\text{op}(a)$ (the optimum of $a$) denotes $\text{Ext}(t(a))$.

Definition 2.2: Sol is an algorithm which solves $(A, t)_{\text{Ext}}$ iff for all $a \in A$, $\text{Sol}(a) = \text{op}(a)$. ($\text{Sol}(a)$ denotes the output of Sol on input $a$).

In what follows, we shall assume that $\text{Ext} = \text{Max}$.

The results obtained are true for the dual problems with $\text{Ext} = \text{Min}$, too, up to some minor changes in the procedures.

Definition 2.3: $\text{Ap}$ is an $\varepsilon$ approximation algorithm for $(A, t)_{\text{Max}}$ iff 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 $\varepsilon$-approximates $(A, t)_{\text{Max}}$.

Definition 2.5: $(A, t)_{\text{Max}}$ is fully approximable if for all $\varepsilon > 0$ there is an $\varepsilon$-approximation algorithm to $(A, t)_{\text{Max}}$ with time complexity $Q(l(a), \frac{1}{\varepsilon})$, 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 rigid if for some $k \in \mathbb{Z}^+$, the set \{a $\in A$ | $\text{op}(a) \leq k$\} is not in P (provided $P \neq \text{NP}$). $(A, t)_{\text{Ext}}$ is simple if it is not rigid.
Definition 2.7: \( (A,t)^\text{Ext} \) is p-simple if for all \( k \in \mathbb{Z}^+ \), the set \( \{a \in A \mid \text{op}(a) \leq k\} \) is recognizable in \( \hat{Q}(\ell(a),k) \) time, for some polynomial \( \hat{Q} \).

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 \( \hat{Q}(\ell(a), \text{op}(a)) \) time, for some polynomial \( \hat{Q} \).

Outline of the Proof:

(a) \( \Rightarrow \) (b): If \( (A,t)^\text{Ext} \) is p-simple, then there exists a polynomial \( \hat{Q}(x,y) \) 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 \( \hat{Q}(\ell(a),k) \) time. Thus, it is easy to see that the following algorithm finds \( \text{op}(a) \) in \( \hat{Q}(\ell(a), \text{op}(a)) \) time:

1. \( k \leftarrow 0 \)
2. if \( \text{op}(a) > k \) then \( k \leftarrow k+1 \), repeat else \( \text{op}(a) = k \), halt.

(b) \( \Rightarrow \) (a): Suppose (b) holds. Thus there is an algorithm Sol which for each \( a \in A \), finds \( \text{op}(a) \) in \( \hat{Q}(\ell(a), \text{op}(a)) \) time, where \( \hat{Q} \) is nondecreasing in its both variables. The following algorithm recognized the set \( \{a \in A \mid \text{op}(a) \leq k\} \) in \( \hat{Q}(\ell(a),k) \) time for some polynomial \( \hat{Q} \):

1. Start the execution of Sol on input \( a \)
2. If Sol has not stopped during the first \( \hat{Q}(\ell(a),k) \) steps then reject
3. Else accept iff \( \text{op}(a) \leq k \).

Q.E.D.
Theorem 2.1: ([PM77]) If \((A,t)_{\text{Ext}}\) is rigid then \((A,t)_{\text{Ext}}\) is not approximable.

Theorem 2.2: ([PM77]). If \((A,t)_{\text{Ext}}\) is fully approximable then \((A,t)_{\text{Ext}}\) is p-simple.

(In [PM77] it is shown that p-simplicity does not imply fully 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 \(Q(\ell(a), \text{op}(a))\) time, for some polynomial \(Q\). In this paper algorithms as Sol above are extended to a fully approximation algorithms for \((A,t)_{\text{Ext}}\) where \((A,t)_{\text{Ext}}\) belongs to a large class of 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 [M78], which is itself a variant of techniques of [Sa76] and others. It will be shown that the problems dealt with in this paper, which are not in P and cannot be approximated by the above technique, are rigid (see Definition 2.6), and hence, by Theorem 2.1, are not approximable.

To simplify the exposition we shall concern problems of the following type: Input: \((a_1, \ldots, a_n, b)\). Output: the maximal integer \(k \leq b\), which is obtainable by certain arithmetic operations on \((a_1, \ldots, a_n)\), to be executed according to some given specific rules. A generalization of the approximation technique to the more general problem, as represented at the beginning of this paper, should be, in general, obvious.

Note: For the sake of consistency with previous papers on fully approximation algorithms, we use a "uniform cost" criteria in analyzing
the time complexity of algorithms, in which additions and multiplications of input integers require a constant time ([AHU] Ch.1). It should be clear that the use of a "logarithmic cost" criteria would increase the time of computation by a polynomial factor at most, and hence would not affect the fully 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 $(a_1, \ldots, a_k) \in \mathbb{Z}^k$ 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 the alternating 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 $(3, x^2, 2, x+y, 1)$ is $(3, 9, 10)$.)

$m_k$ is the result of the computation. (The output of a single element sequence $(a_1)$ is defined as 0.)

If $B$ is a set of binary operations such that for each $i$, $\beta_i \in B$, then $(m_1, \ldots, m_k)$ is a "computation of $(a_1, \ldots, a_k)$ over $B$".

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

1. For some $i_0 \in \{1, \ldots, n\}$, $k$ is a result of a computation of $(a_{i_0}, a_{i_0}+1, \ldots, a_n)$ over $B$.

2. $k \leq b$. 

Technion - Computer Science Department - Technical Report CS0140 - 1978
We define a binary operation \( u_1 \) by: \( u_1(x,y) = x \).

\( \text{PrI}(u_1, x+y) \) is the Knapsack problem: "given \( (a_1, \ldots, a_n, b) \), maximize \( \Sigma_i a_i \cdot (c_i \in \{0,1\}) \), subject to \( \Sigma_i a_i \leq b \)." A fully approximation algorithm for this problem was first obtained by [IK75].

\( \text{PrI}(u_1, x \cdot y) \) is the subset product problem, for which a fully approximation algorithm was obtained in [M78].

**Definition 3.2:** A binary operation \( \beta \) is "monotone" iff \( \beta(x,y) \geq x \)
for all \( x, y > 0 \). \((x,y \in \mathbb{Z})\)

The operations \( x+y, x \cdot y, x^y, y^x, u_1 \) are monotone, while \( x-y \) and \( y-x \) are not. If \( \beta \) contains only monotone operations then \( \text{PrI}(\beta) \) will be called a "monotone problem":

**Lemma 3.1.** If \( \text{PrI}(\beta) \) is monotone then it is p-simple.

**Proof.** By Lemma 2.1, it suffices to show that for each

\( a = (a_1, \ldots, a_n, b), \text{op}(a) \) can be found in time polynomial in \( \lambda(a) \) and \( \text{op}(a) \). The following algorithm \( \text{Sol} \) finds \( \text{op}(a) \) in \( O(n \cdot \text{op}(a) \log(\text{op}(a))) \)
time:

\( \text{Sol}: \) Input \( a=(a_1, \ldots, a_n, b) \) \((WLG \ a_i \leq b \ for \ i=1, \ldots, n)\). Output: \( \text{op}(a) \).
1. Begin;
2. \( i \leftarrow 1, T \leftarrow \emptyset; \)
3. for each \( s \) in \( T \) and for each \( \beta \in \beta \) do;
4. if \( \beta(s, a_i) \leq b \) then \( T \leftarrow T \cup \beta(s, a_i) \);
5. if \( i=n \) then halt and return \( \max(T) \);
6. else \( i \leftarrow i+1 \), go to 3, end.

It is easy to see that at the termination of \( \text{Sol} \), \( T \) 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 [AHU] 4.9-4.10), the execution time of lines 3-5 is $O(|B| |T| \log |T|)$. The execution time of the other lines is a constant. Since $|B|$ is a constant, and $|T| \ll \text{op}(a)$ all through the execution of the algorithm, the time complexity of the algorithm as a whole is $O(n \cdot \text{op}(a) \cdot \log \text{op}(a))$. Q.E.D.

Note: If for all $\beta \in B$, $x_1 \geq x_2 \Rightarrow \beta(x_1, y) \geq \beta(x_2, y)$ for all $x_1, x_2, y > 0$ then the time complexity can be reduced to $O(n \cdot \text{op}(a))$ time, by keeping $T$ as an ordered list. (See e.g. [La77], Sec. 2, for details.)

Lemma 3.1 provides us with a necessary condition for fully approximability. We shall show now that if $B \subseteq \{x \cdot y, x + y, u_1\}$, then algorithm $\text{Sol}$ introduced in Lemma 3.1 can be extended to a fully approximation algorithm for $\text{Pr}l(B)$. First we need several lemmas:

Fact 3.1: For all $r \geq 2$, for all $n \geq 1$ ($r \in \mathbb{R}^\times$, $n \in \mathbb{Z}$), the following holds true: If $k < r^n$, then $(1 - \frac{1}{r^{n+1}})^k > 1 - \frac{1}{r}$

To prove Fact 3.1 use the binomial expansion of $(1 - \frac{1}{r^{n+1}})^k$.

Definition 3.3: Let $(m_1, m_2, \ldots, m_k) \in \mathbb{Z}^k$ be the computation sequence of $(a_1, \beta_1, a_2, \beta_2, \ldots, a_{k-1}, \beta_{k-1}, a_k)$, and let $0 < \delta < 1$ be given. A sequence $(m'_1, m'_2, \ldots, m'_k)$ is a "$\delta$ condensation" of $(m_1, \ldots, m_k)$ iff there exists a sequence $(h_1, \ldots, h_k) \in \mathbb{R}^k$, $0 \leq h_i \leq \delta$, such that:

(a) $m'_i = m_i(1 - h_i) (= a_i(1 - h_i))$

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

Lemma 3.2. Let $(m_1, \ldots, m_k)$ be the computation sequence of $(a_1, \beta_1, a_2, \ldots, a_{k-1}, \beta_{k-1}, a_k)$.
Let \( r \) be a real number, \( r \geq k \), and let \( \delta = \frac{1}{r^2} \). If \((m'_1, \ldots, m'_k)\) is a \( \delta \) condensation of \((m_1, \ldots, m_k)\), then

\[
1 \geq \frac{m'_k}{m_k} > 1 - \frac{1}{r}.
\]

Proof. Clearly \( 0 \leq m'_k \leq m_k \), and hence \( 1 \geq \frac{m'_k}{m_k} \). To prove the second inequality, we prove by induction for \( j = 1, \ldots, k \) that \( \frac{m'_j}{m_j} \geq \left(1 - \frac{1}{r^2}\right)^j \).

Substituting \( j = k \) and recalling fact 3.1, with \( n = 1 \), we get

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

For \( j = 1 \), \( m'_1 = m_1(1 - h'_1) \geq m_1(1 - \frac{1}{r^2}) \). Suppose now that

\[
\frac{m'_j}{m_j} \geq \left(1 - \frac{1}{r^2}\right)^j.
\]

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

If \( \beta_j = u_{j+1} \), then

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

If \( \beta_j = x\cdot y \), then

\[
\frac{m'_{j+1}}{m_{j+1}} = \frac{m'_j a_{j+1}(1-h'_j+1)}{m_j a_{j+1}} \geq \frac{m'_j(1 - \frac{1}{r^2})^j a_{j+1}(1-h'_j+1)}{m_j a_{j+1}} \geq \left(1 - \frac{1}{r^2}\right)^{j+1}.
\]
If \( \beta_j = x + y \), then

\[
\frac{m_{j+1}'}{m_{j+1}} = \frac{(m_j' + a_{j+1})(1 - h_{j+1})}{m_j + a_{j+1}} \geq \frac{m_j (1 - \frac{1}{r^2})^j + a_{j+1} (1 - h_{j+1})}{m_j + a_{j+1}}.
\]

\[
\geq \frac{(m_j + a_{j+1})(1 - \frac{1}{r^2})^j (1 - h_{j+1})}{m_j + a_{j+1}} \geq (1 - \frac{1}{r^2})^j + 1.
\]

Q.E.D.

**Lemma 3.3.** Let \( 1 \leq a_1 < a_2 < \ldots < a_t \leq b \) be given, such that \( \frac{a_t}{a_{i+1}} \leq 1 - \delta \), where \( 0 < \delta \leq 1 \). Then \( t \leq \lceil \frac{2\ln b}{\delta} \rceil \).

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

\[
t-1 \leq \log_{1+\delta} b = \frac{\ln b}{\ln (1+\delta)} = \frac{\ln b}{\delta - \frac{\delta^2}{2} - \frac{\delta^3}{3} - \ldots}.
\]

i.e. \( t < \frac{2\ln b}{\delta} + 1 \).

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

Q.E.D.

The following algorithm, \( A_{pl} \), is a fully approximation algorithm for \( \text{Pr}l(\beta) \), where \( B \) is any subset of \( \{x+y, x'y, u_j\} \).
Input: $a = (a_1, \ldots, a_n, b) \in (\mathbb{Z})^{n+1}$, $\epsilon > 0$. Output: An integer $k \in t(a)$, such that $\text{op}(a) \geq k > \text{op}(a)(1-\epsilon)$.

1. Begin;
2. $r \leftarrow \max(\frac{1}{\epsilon}, n)$;
3. $\delta \leftarrow \frac{1}{r^2}$; // $\delta$ is the "condensing parameter"/
4. $i \leftarrow 1$, $T \leftarrow \emptyset$;
5. for every $s$ in $T$ and for every $b \in B$ do;
6. if $\beta(s, a_i) \leq b$ then $T \leftarrow T \cup \{\beta(s, a_i)\}$;
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 and return $s_t$;
10. //Condensing:// $j \leftarrow 1$, $k \leftarrow 2$;
11. while $k \leq t$ and $\frac{s_{i-1}}{s_k} > 1-\delta$ do begin.
12. $T \leftarrow \{s_k\}$, $k \leftarrow k+1$, end;
13. if $k < t$ then begin $j \leftarrow k$, $k \leftarrow j+1$, go to 11, end;
14. else // (the condensing is finished) // $i \leftarrow i+1$, go to 5, end.

Theorem 3.1: For each $\epsilon > 0$, algorithm APL provides an $\epsilon$-approximation to $P(a)$, where $B \subset \{x+y, x'y, u\}$ in $O(\max(t^4(a), \frac{2^2(a)}{\epsilon^2})$ time.

Proof. By the definition of the problem, $\text{op}(a)$ is an output of a computation of a sequence $(a_{i_0}, a_{i_0+1}, \ldots, a_n)$ over $B$, for some $1 \leq i_0 \leq n$. Let the computation sequence be $(m_0, m_{i_0+1}, \ldots, m_n)$, $(m_n = \text{op}(a))$, and the corresponding alternating sequence be $(a_{i_0}, \beta_{i_0}, a_{i_0+1}, \beta_{i_0+1}, \ldots, a_n)$.

Suppose now that APL is carried out with $a = (a_1, \ldots, a_n, b)$ and $\epsilon$ as an input. Denote by $T_i$, for $i = 1, 2, \ldots, n-1$, the content of $T$. Technion - Computer Science Department - Technical Report CS0140 - 1978.
at the termination of the condensation at the i-th stage (i.e. just before line 14 is encountered at the i-th time). $T_i$ will denote the content 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 $T_i$ an element $m^i_1$, such that the sequence $(m^i_1, m^i_1 + 1, \ldots, m^i_n)$ is a $\delta$-condensation of $(m^i_0, \ldots, m^i_n)$ (with $\delta = \frac{1}{i^2}$).

When $i = i_0$, it is easily checked that either $a^i_1$ is in $T_i$ (since $a^i_0$ is inserted in $T$ at line 7) which means that $a^i_0$ was not deleted from $T$ during the condensing, or, if $a^i_0$ was deleted from $T$ during the condensing, there is an integer $m^i_1$ in $T_i$ such that $1 > \frac{\varepsilon}{a^i_0} > 1 - \delta$.

In the first case we take $m^i_1$ to be $a^i_1$, and in the second case we take $m^i_1$ to be $m^i_0$. In both cases $\frac{m^i_0}{m^i_1} > 1 - \delta$.

Suppose now that $(m^i_1, m^i_1 + 1, \ldots, m^i_n)$ is a $\delta$-condensation of $(m^i_0, \ldots, m^i_n)$, and that $m^i_1 \in T_i$. By the same argument as above, either $\beta^i(m^i_1, a^i_{i+1})$ is in $T_{i+1}$, or there is an integer $m^i_{i+1}$ in $T_{i+1}$ such that

$$1 > \frac{m^i_{i+1}}{\beta^i(m^i_1, a^i_{i+1})} > 1 - \delta.$$

If we take $m^i_{i+1}$ to be $\beta^i(m^i_1, a^i_{i+1})$ in the first case, or $m^i_{i+1}$ in the second case, we have that $(m^i_1, \ldots, m^i_{i+1})$ is a $\delta$-condensation of $(m^i_0, \ldots, m^i_{i+1})$.

By Lemma 3.2,

$$\gamma > \frac{m^i_n}{m^i_n} \geq \frac{m^i_n}{\text{op}(a)} > 1 - \varepsilon.$$
Since the output of Ap1, to be denoted as \( m^*_n \), is \( \max(T_n) \geq m'_n \), we have

\[
1 \geq \frac{m^*_n}{\text{op}(a)} \geq \frac{m'_{n}}{\text{pp}(a)} > 1-\varepsilon.
\]

Hence, \( m^*_n \) is \( \varepsilon \)-approximation to \( \text{op}(a) \).

To prove the complexity, we first note that for each \( i \), \(|T_i| \leq O\left( \frac{2^2 R(a)}{\varepsilon} \right) \).

This follows from Lemma 3.3, with \( \ln(b) = O(\ell(a)) \).\(^(*)\)

It follows that \(|T| \leq O\left( \frac{2^2 R(a)}{\varepsilon} \right) \) all through the algorithm. By the note after Lemma 3.1 (p.8), the time complexity of the algorithm as a whole is \( O(n \cdot \ell(a)) \). Since \( n < \ell(a) \), this is equal to \( O(\ell(a)^4) \) if \( \ell(a) \geq \frac{1}{\varepsilon} \), and to \( O\left( \frac{\ell(a)^2}{\varepsilon^2} \right) \) if \( \ell(a) < \frac{1}{\varepsilon} \). \( \text{Q.E.D.} \)

**Note.** Algorithm Ap1 not only provides an \( \varepsilon \)-approximation to \( \text{op}(a) \), but provides an \( \varepsilon \)-approximation to each \( m \in \ell(a) \). This property, (which is not shared by the previous approximation techniques of [IK75],[Sa75]) can be used for simultaneously approximating several problems, with the same \( (a_1, \ldots, a_n) \) and \( \varepsilon \), but different \( b' \)s.

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

**Lemma 3.4.** Let \( k, b_1, b_2, \ldots, b_k \) be \( k+2 \) positive integers, where for \( i=1, \ldots, k, b_i \geq 2 \). Let a "legal sequence" be a sequence \( (f_1, \ldots, f_n) \),

\[\ldots\]

\(^(*)\) Under the logarithmic cost criteria, \( \ln(b) = O(\ell(a)) \). Under the uniform cost criteria, \( \ln(b) \leq c \), where \( c \) is some constant (\( c \) = the amount of storage in each register). (See [AHU]:1.3.)

\(^(**)\) Note that if \( m_i \leq b_i \) then \( m^*_{i+1} \) can be computed in \( O(\log^2 b) \) time and hence also in \( O(\ell(a)) \) time, with \( a^* = (a_1, \ldots, a_n, b) \).
where \( n = k + \ell \), which satisfies the following:

(a) \( f_1 = 1; \)

(b) \( f_{j+1} = f_j + 1 \) or \( f_{j+1} = f_j \cdot b_i \) for some \( 1 \leq i \leq \ell; \)

(c) if \( f_{j_1+1} = f_{j_1} \cdot b_{i_1}, f_{j_2+1} = f_{j_2} \cdot b_{i_2}, j_1 < j_2, \) then \( i_1 < i_2. \)

Let \( f = \max(f_n | (f_1, \ldots, f_n) \) is a legal sequence). Then

\[
\ell \prod_{i=1}^{\ell} b_i, \quad (i.e. (f_1, \ldots, f_n) = (1, 2, \ldots, k, k \cdot b_1, k \cdot b_1 b_2, \ldots, k \cdot \prod_{i=1}^{\ell} b_i)).
\]

The proof of this lemma is omitted.

**Lemma 3.5:** Let \( (m_1, \ldots, m_k) \in (Z^+)^k \) and \( (a_1, \ldots, a_\ell) \in (Z^+)^{\ell} \) be given, such that \( m_i \leq m_{i+1} \) for \( i = 1, \ldots, k-1 \) and \( \ell < k. \) Assume that for some \( i_1, \ldots, i_\ell, 1 \leq i_1 < i_2 < \cdots < i_\ell < k, \) the following hold:

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

(b) \( m_{i_{j+1}} = m_{i_j} \cdot a_j. \)

Then \( \prod_{j=1}^{\ell} a_j \leq \log_2 m_k. \)

**Proof:** Since \( i_{j+1} \geq i_j \), \( m_{i_{j+1}} \geq m_{i_j} + a_j. \) Hence we have that

\[
m_{i_2} \geq m_{i_1}, \quad m_{i_3} \geq m_{i_2} + a_1 a_2, \quad \ldots, \quad m_{i_\ell} \geq m_{i_1} + a_1 a_2 \cdots a_{\ell-1}. \quad \text{Since } m_{i_1} \geq 2,
\]

we have that \( m_k \geq m_{i_1} + a_1 a_2 \cdots a_\ell \geq \prod_{j=1}^{\ell} a_j. \)

Q.E.D.

**Lemma 3.6:** Let \( (m_1', \ldots, m_k') \) be the computation sequence of

\((a_1, \beta_1, a_2, \ldots, a_{k-1}, a_k), \) where for \( i = 1, \ldots, k-1, \beta_i \in \{x+y, x \cdot y, x^y, s_{i1}\}. \)
Let $r$ be a real number, $r \geq \max(k, \log_2 m_k)$, and let $\delta = \frac{1}{r^3}$. If $(m_1',\ldots,m_k')$ is a $\delta$ condensation of $(m_1,\ldots,m_k)$ then $1 \geq \frac{m_k'}{m_k} > 1 - \frac{1}{r}$.

**Proof.** WLG we may assume that if $\beta_i = x^y$, then $m_i \geq 2$ and $a_{i+1} \geq 2$. (Otherwise replace $\beta_i$ 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) $g_{i+1} = [\text{if} \quad \beta_i \neq x^y \quad \text{then} \quad g_i+1, \quad \text{else} \quad g_i \cdot a_{i+1}+1]$.

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 $g_i \cdot a_{i+1}+1$, with the two elements $(g_i \cdot a_{i+1},g_i \cdot a_{i+1}+1)$, then we obtain a new sequence of length $k+\ell$, where $\ell = |\{i \mid \beta_i = x^y\}|$.

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 $k',a_{i+1},a_{i+1}+1,\ldots,a_{i+1}$ (where $i_1,\ldots,i_\ell$ are the indices for which $\beta_i = x^y$). Hence $g_k \leq k \prod a_{i+1}$. Moreover, the sequences $(m_1,\ldots,m_k)$ and $(a_{i+1},\ldots,a_{i+1})$ satisfy the conditions of Lemma 3.5, and hence $\prod a_{i+1} \leq \log_2 m_k$. 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 $j \neq 1,\ldots,k$,

$$m_j < \frac{(1 - \frac{1}{r^3})^j g_j}{m_j}.$$ For $j = 1$ the hypothesis is true by definition.

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

$$m_{j+1} > (1 - \frac{1}{r^3}) \frac{g_{j+1}}{m_{j+1}} = (1 - \frac{1}{r^3}) \frac{g_j+1}{m_{j+1}}$$

by an argument similar to that of Lemma 3.2.
If $\beta_j = x^y$, then

$$
\frac{m_{j+1}}{m_j} = \frac{(m_j^{a_j+1}(1-h_j))}{(m_j^{a_j+1})} = \frac{(m_j^{1 - \frac{1}{r^3}})^{a_j+1}(1-h_j)}{m_j^{a_j+1}} = (1 - \frac{1}{r^3}) \beta_j^{a_j+1}(1-h_j) \geq (1 - \frac{1}{r^3}) \beta_j^{a_j+1} = (1 - \frac{1}{r^3}) \beta_j^{a_j+1}.
$$

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

$$
\frac{m_k}{m_k} \geq (1 - \frac{1}{r^3}) \beta_k \geq (1 - \frac{1}{r^3}) \beta \geq 1 - \frac{1}{r}.
$$

(The last inequality is by fact 1, with $n=2$.)

Q.E.D.

Let $\text{Ap}_2$ be algorithm $\text{Ap}_1$ in which lines 2 and 3 are replaced by:

(2') $r = \max \left\{ \frac{1}{e}, n, \log_2 b \right\}$,

(3') $\delta = \frac{1}{r}$.

**Theorem 3.2:** For each $\varepsilon > 0$, $\text{Ap}_2$ provides an $\varepsilon$-approximation to $\text{Pr}_1(B)$, where $B$ is any subset of $\{x+y, x^y, x^y, y^x, u_1\}$, in $O(\max \left\{ \frac{\log_2 b}{\varepsilon}, \log_2 b \right\})$ time.

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

Though it will not be shown here, $\text{Ap}_2$ can be extended to a fully approximation algorithm for $\text{Pr}_1(B)$, where $B$ is any subset of $\{x+y, x^y, x^y, y^x, u_1\}$, which is the set of all monotone "elementary" operations on the integers.
4. GENERALIZATION

We shall now consider the following generalization of Pr1.

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

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

2. $k \leq b$.

Pr2($\{x+y, u_1\}, \mathbb{Z}^+$) = Pr2($\{x+y\}, \mathbb{Z}^+$) is the unbounded Knapsack Problem:

"Given a sequence $(a_1, \ldots, a_n, b)$, find the maximal integer $k \leq b$ such that $\sum a_ix_i = k$ has a nonnegative integer solution."

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

**Proof:** The idea of the proof is to use a "measure preserving reduction" ([PM78]), to reduce Pr2($B, \mathbb{Z}^+$) to Pr2($B', \{1\}$), which is Pr1($B$), which was shown to be fully approximable in the previous section.

Let $(A, t_1)^{\text{Ext}}$ and $(B, t_2)^{\text{Ext}}$ be NPPO's. $g: \mathbb{E}^* \rightarrow \mathbb{E}^*$ is a measure preserving reduction of the former to the latter iff:

(a) $g(a) \in B \Leftrightarrow a \in A$.

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

(This definition is a little stronger than that of [PM77], but it suffices for our purpose.) By [PM77], Lemma 6, if $(B, t_2)^{\text{Ext}}$ is fully approximable and there is a polynomial time measure preserving reduction of $(A, t_1)^{\text{Ext}}$ to $(B, t_2)^{\text{Ext}}$, then $(A, t_1)^{\text{Ext}}$ is also fully approximable.
We shall introduce a polynomial time measure preserving reduction of \( \text{Pr}2((x+y,u_1),z^+) \) to \( \text{Pr}1((x+y,u_1)) \). Other cases may be proved similarly: Let \( a = (a_1, \ldots, a_n, b) \) be an input to \( \text{Pr}2(B,z^+) \). For each \( i \), replace \( a_i \) by a sequence \( (a_1^1, 2a_1^2, 4a_1^3, \ldots, 2^i a_1^i) \), where

\[
2^i a_1^i = \lceil \log_2 \frac{b}{a_1^i} \rceil.
\]

The resulting sequence \( a' = (a_1, 2a_1, \ldots, 2^i a_1, a_2, \ldots, 2^i a_n, b) \) will be taken as an input to \( \text{Pr}1((x+y,u_1)) \). It is left to the reader to check that the reduction above is a polynomial time measure preserving reduction.

Q.E.D.

5. **COMPLEXITY RESULTS - UNAPPROXIMABLE PROBLEMS**

In some arithmetical combinatorial problems, it is required that all the integers appearing in the input sequence take part in the computation (e.g., the Partition Problem: on input \( (a_1, \ldots, a_n) \), does \( \exists x_1 a_1 = 0 \) has a \((-1,1)\) solution?) To distinguish those problems from others, we make the following convention: If \( u_1 \notin B \), then \( \text{Pr}2(B;D) \) is the following problem: on input \( (a_1, \ldots, a_n, b) \), 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 \( B \).

2) \( k \leq b \).

\( \text{Pr}2((x+y),(-1,1)) \) is the problem: on input \( (a_1, \ldots, a_n, b) \) find the maximal integer \( k \leq b \), such that \( \exists x_1 a_1 \) has a \((-1,1)\) solution.

By the NP-completeness of the partition problem ([Ka72]), this problem can be easily shown to be rigid (provided \( P \neq \text{NP} \)), and hence, by Theorem 2.1, is not approximable.
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 a_i has a \{-1,0,1\} solution. It can be shown that either this problem is in P, or is rigid, and hence not approximable.

Pr2((x+y),Z) is the problem: on input (a_1, \ldots, a_n, b), find the maximal k \leq b such that k = \sum a_i has an integer solution. This problem is equivalent to the problem of finding the greatest common divisor of a_1, \ldots, a_n, which, by Euclid algorithm, is in P.

The next table summarizes results on the complexity and approximability of related problems. P stands for polynomial time complexity, NPC for NP-complete. Some of the results were proved above, and others are not hard and left to the reader.
### Complexity and approximability of arithmetical combinatorial problems

**Table A**

<table>
<thead>
<tr>
<th>$B$</th>
<th>$D$</th>
<th>Complexity of Pr2($B, D$)</th>
<th>Approximability of Pr2($B, D$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x+y, u_1)$ or $(x\cdot y, u_1)$ or (x+y, x\cdot y, u_1) or (x+y, x\cdot y, u_1)</td>
<td>{1}</td>
<td>NPC</td>
<td>fully approximable</td>
</tr>
<tr>
<td>$(x^y, u_1)$</td>
<td>{1}</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>$(x+y) \subseteq B \subseteq (x+y, x\cdot y, x^y, u_1)$</td>
<td>$\mathbb{Z}^+$</td>
<td>NPC</td>
<td>fully approximable</td>
</tr>
<tr>
<td>$(x\cdot y, x^y, u_1)$</td>
<td>$\mathbb{Z}^+$</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>$(x+y) \subseteq B \subseteq (x+y, x\cdot y, x^y)$</td>
<td>{-1, 1}</td>
<td>NPC</td>
<td>rigid (*)</td>
</tr>
<tr>
<td>$(x+y, u_1) \subseteq B \subseteq (x+y, x\cdot y, x^y, u_1)$</td>
<td>{-1, 1}</td>
<td>either in P, or [not in P and rigid]</td>
<td></td>
</tr>
</tbody>
</table>

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

*(*) Provided that $P \neq NP$.
6. COMMENTS ON RELATED PROBLEMS

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 \( B \) and \( D \) be, as before, sets of operations and integers respectively. For each positive integer \( n \), let \( C_n \) be a subset of \( F_n^* \), the set of all functions from \( \{1, 2, \ldots, n\} \) to \( \{1, 2, \ldots, n\} \). 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 the following: For some \( i_0 \in \{1, 2, \ldots, n\} \), for some \( (x_{i_0}, \ldots, x_n) \in D^{-i_0+1} \)
and for some \( \sigma \in C_n \), \( k \) is a result of a computation of \( (x_{i_0}^{a_0}(i_0), x_{i_0}^{a_0}(i_0+1), \ldots, x_n^{a_0}(n)) \) over \( B \).

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 permutations of order \( n \).

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

Outlines of the Proof. Suppose first that \( D = \{1\} \). Let Ap3 be algorithm Ap2 (p. 16 and p. 11), with lines 4-6 replaced by:

(4) \( i + 1, T \rightarrow \{a_1, \ldots, a_n, 0\}, A \rightarrow \{a_1, \ldots, a_n\} \).

(5) for every \( s \in T \), for every \( a \in A \), and for every \( \beta \in B \) do

(6) if \( \beta(s,a) \leq b \) then \( T \rightarrow T \cup \{\beta(s,a)\} \).

(*) If \( u_1 \in B \), then \( i_0 = 1 \).
Thus defined, Ap3 is a fully approximation algorithm to Pr3(E,D,\{F_n\}), with time complexity $O(\max\{\varepsilon(a)^6, \frac{\varepsilon(a)^3}{\varepsilon^3}\})$. This can be proved along the same line as the proof of theorems 3.1 and 3.2, with the exception that lines (5)-(6) require now $O(|T| \cdot n)$ time, instead of the $O(|T|)$ time that was required for lines (5)-(6) in Ap1 and Ap2.

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

Q.E.D.

Pr3(E,D,\{S_n\}) seems to be much harder. We don't even know if Pr3(\{x+y,x\cdot y,u_1\},\{1\},\{S_n\}) is \textit{P}-simple. The question whether this problem is fully approximable is, therefore, still 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 is fully approximable by some other method?

A third open problem is the following: Is the set \{(a_1, \ldots, a_n) | (\forall i, a_i \in \mathbb{Z}^+) \wedge (\exists_1 a_1 = 0 \text{ has a non trivial } (-1,0,1) \text{ solution}) \} is in \textit{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.

ACKNOWLEDGEMENT

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.
REFERENCES


