Using Parallel Filtering Algorithms to Solve the 0-1 Knapsack Problem on DNA-based Computing

Sientang Tsai

Department of Information Management, Southern Taiwan University of Science and Technology, Yuan Kung District, Tainan City, Taiwan, R.O.C.
*Corresponding author: tsai@mail.stust.edu.tw

Copyright © 2013 Horizon Research Publishing All rights reserved.

Abstract It is shown first by Adleman that deoxyribonucleic acid (DNA) strands could be employed towards calculating solution to an instance of the NP-complete Hamiltonian Path Problem (HPP). Lipton also demonstrated that Adleman’s techniques could be used to solve the satisfiability (SAT) problem. In this paper, it is demonstrated how the DNA operations presented by Adleman and Lipton can be used to develop the DNA-based algorithm for solving the 0-1 Knapsack Problem.

Keywords DNA-based Computing, NP-complete Problems, 0-1 Knapsack Problem.

1. Introduction

In 1961, Feynman first offered bio-molecular computation, but his idea was not implemented by experiments for a few decades [1]. Adleman [2] in 1994 succeeded to solve an instance of the Hamiltonian path problem in a test tube, just by handling DNA strands. From [5], it was indicated that an optimal solution of every NP-complete or NP-hard problem is determined from its characteristics. DNA-based algorithms had been proposed to solve many computational problems and those consisted of the satisfiability problem [3], the maximal clique [6], three-vertex-coloring [7], the set-splitting problem [8], the set-cover problem and the problem of exact cover by 3-sets [9], the dominating-set [10], the maximum cut problem [11], the binary integer programming problem [12] and the set-partition problem [24]. One potentially significant area of application for DNA algorithms is the breaking of encryption schemes [13,14]. From [15-17], DNA-based arithmetic algorithms are proposed. Furthermore from [18], DNA-based algorithms for constructing DNA databases are also offered.

The rest of the paper is organized as follows. Section 2 introduces DNA models of computation proposed by Adleman and his co-authors in details. Section 3 introduces the DNA program to solve the 0-1 knapsack problem from solution spaces of DNA strands. Conclusions are drawn in Section 4.

2. DNA Model of Computation

In the last decade there have been revolutionary advances in the field of biomedical engineering particularly in recombinant DNA and RNA manipulating. Due to the industrialization of the biotechnology field, laboratory techniques for recombinant DNA and RNA manipulation are becoming highly standardized. Basic principles about recombinant DNA can be found in [22-26]. In this section we describe eight biological operations useful for solving the 0-1 knapsack problem. The method of constructing DNA solution space for the 0-1 knapsack problem is based on the proposed method in [20,21].

A (test) tube is a set of molecules of DNA (a multi-set of finite strings over the alphabet \{A, C, G, T\}). Given a tube, one can perform the following operations:

1. Extract: Given a tube \( T \) and a short single strand of DNA, \("s\"), produce two tubes \(+ (T, s)\) and \(−(T, s)\), where \(+ (T, s)\) is all of the molecules of DNA in \( T \) which contain the strand \("s\") as a sub-strand and \(−(T, s)\) is all of the molecules of DNA in \( T \) which do not contain the strand \("s\”).

2. Merge: Given tubes \( T_1 \) and \( T_2 \), yield \(∪(T_1, T_2)\), where \(∪(T_1, T_2) = T_1 ∪ T_2\). This operation is to pour two tubes into one, without any change of the individual strands.

3. Amplify: Given a tube \( T \), the operation \( Amplify (T, T_1, T_2) \), will produce two new tubes \( T_1 \) and \( T_2 \) so that \( T_1 \) and \( T_2 \) are a copy of \( T \) (\( T_1 \) and \( T_2 \) are identical) and \( T \) becomes an empty tube.

4. Append: Given a tube \( T \) and a short strand of DNA, \("s\"), the operation will append the short strand, \("s\"), onto the end of every strand in the tube \( T \). It is denoted by \( Append (T, s) \).

5. Append-head: Given a tube \( T \) and a short strand of DNA, \("s\"), and the operation will append the short strand, \("s\"), onto the head of every strand in the tube \( T \). It is denoted by \( Append-head (T, s) \).

6. Detect: Given a tube \( T \), say ‘yes’ if \( T \) includes at least
one DNA molecule, and say ‘no’ if it contains none. It is denoted by \( \text{Detect} (T) \).

7. Discard: Given a tube \( T \), the operation will discard the tube \( T \). It is denoted by Discard \( (T) \).

8. Read: Given a tube \( T \), the operation is used to describe a single molecule, contained in the tube \( T \). Even if \( T \) contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them. It is denoted by Read \( (T) \).

3. The DNA Algorithms for Solving the 0-1 Knapsack Problem

3.1. Definition of the 0-1 Knapsack Problem

Assume that there are \( q \) items. Let \( S = \{ \text{item}_1, \text{item}_2, \ldots, \text{item}_q \} \), \( w_i = \text{weight of item}_i \), \( p_i = \text{profit of item}_i \), \( M = \text{maximum weight the knapsack can hold} \), where \( w_i, p_i \), and \( M \) are positive integers. Determine a subset \( A \) of \( S \) such that

\[
\sum_{i=1}^{q} p_i x_i \quad \text{maximized subject to} \quad \sum_{i=1}^{q} w_i x_i \leq M ,
\]

where \( x_i \in \{0,1\} \), \( x_i = 1 \) if the \( i \)th item in set \( S \) is selected for the knapsack and \( x_i = 0 \) otherwise. Without loss of generality we also assume that \( w_i \leq M \) for \( 1 \leq i \leq q \), so each item fits into the knapsack.

For example, suppose we have three items expressed by a finite set \( S = \{ \text{item}_1, \text{item}_2, \text{item}_3 \} \), the weight and profit for each item are listed as Table 1. The value of \( M \), the maximum weight that knapsack can carry, is 10 lb. The total subsets of \( S \) are, respectively, \( \emptyset, \{ \text{item}_1 \}, \{ \text{item}_2 \}, \{ \text{item}_3 \}, \{ \text{item}_1, \text{item}_2 \}, \{ \text{item}_1, \text{item}_3 \}, \{ \text{item}_2, \text{item}_3 \} \) and \( \{ \text{item}_1, \text{item}_2, \text{item}_3 \} \). According to the definition above of the 0-1 knapsack problem, their corresponding binary values of \( \{ x_1, x_2, x_3 \} \) are subsequently, 000, 001, 010, 100, 011, 110, 101, 111. The feasible solutions satisfying the constraint \( w_1 x_1 + w_2 x_2 + w_3 x_3 \leq 10 \) are \( \emptyset, \{ \text{item}_1 \}, \{ \text{item}_2 \}, \{ \text{item}_3 \}, \{ \text{item}_1, \text{item}_2 \}, \{ \text{item}_1, \text{item}_3 \}, \{ \text{item}_2, \text{item}_3 \} \) and \( \{ \text{item}_1, \text{item}_2, \text{item}_3 \} \). Clearly the optimal solution is to fill the knapsack with items whose profit is the largest. Then the optimal solution to this problem instance is \( A = \{ \text{item}_1, \text{item}_2 \} \), because total profit $11 is the maximum satisfied the constraint that the total weight of all selected items doesn’t exceed 10 lb.

\[
\begin{array}{|c|c|c|}
\hline
\text{Item (item\(_i\))} & \text{Weight (w\(_i\))} & \text{Profit (p\(_i\))} \\
\hline
\text{item}_1 & w_1=3 \text{ lb} & p_1=\$6 \\
\text{item}_2 & w_2=5 \text{ lb} & p_2=\$4 \\
\text{item}_3 & w_3=7 \text{ lb} & p_3=\$5 \\
\hline
\end{array}
\]

3.2. Construct the Solution Space of DNA Strands for the 0-1 Knapsack Problem

Assume that \( x_1 x_2 x_3 \ldots x_q \) is a \( q \)-bit binary number, which is applied to represent one of the \( 2^q \) subsets of a \( q \)-element set \( S \). From \([20,21]\), for every bit \( x_i \) representing the \( k \)th element in \( S \) for \( 1 \leq k \leq q \), two distinct 15-base value sequences are designed. One represents the value “0” for \( x_i \) and the other represents the value “1” for \( x_i \). For sake of convenience in our representation, assume that \( x_i^1 \), which represents the \( k \)th item selected in set \( S \), denotes the value of \( x_i = 1 \) and \( x_i^0 \), which represents the \( k \)th item not selected in set \( S \), denotes the value of \( x_i = 0 \). The following DNA-based algorithm is used to construct the solution space for \( 2^q \) possible subsets of a \( q \)-element set \( S \). The result generated by \( \text{Init} (T_0, q) \) for our example in the previous section is shown in Table 2.

\[
\begin{array}{|c|c|}
\hline
\text{Tube} & \text{The result generated by Init (T\(_0\), q)} \\
\hline
T_0 & \{x_1^0 x_2 x_1^0 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_10 x_11 x_12 x_13 x_14 x_15 x_16 x_17 x_18 x_19 x_20 x_21 x_22 \} \\
T_1 & \emptyset \\
T_2 & \emptyset \\
\hline
\end{array}
\]

**Lemma 1**: The algorithm, \( \text{Init} (T_0, q) \), is used to construct the solution space of \( 2^q \) possible subsets for a \( q \)-element set \( S \).

**Proof**: The algorithm, \( \text{Init} (T_0, q) \), is implemented via \( \text{amplify, append and merge} \) operations. Step (1) and Step (2) are subsequently applied to append DNA sequences, which represent the value “1” for \( x_q \) and the value “0” for \( x_q \) respectively, onto the end of every strand in tube \( T_1 \) and tube \( T_2 \). This means that subsets containing the \( q \)th element appear in tube \( T_1 \), and subsets not containing the \( q \)th element appear in tube \( T_2 \). Next, Step (3) is used to pour tube \( T_1 \) and \( T_2 \) into tube \( T_0 \). This indicates that DNA strands in tube \( T_0 \) include DNA sequences of \( x_q = 1 \) and \( x_q = 0 \).

Each time Step (4a) is performed, it uses \( \text{amplify} \) operation to copy the contents of tube \( T_0 \) into two new tubes, \( T_1 \) and \( T_2 \), which are copies of \( T_0 \). Tube \( T_0 \) becomes empty. Step (4b) and Step (4c) are used to subsequently append DNA sequences, respectively representing the value “1” for \( x_q \) and the value “0” for \( x_q \), onto the end of every strand in tube \( T_1 \) and tube \( T_2 \). This implies that subsets containing the \( k \)th element appear in tube \( T_1 \) and subsets without containing the \( k \)th element appear in tube \( T_2 \). Next, Step (4d) is used to pour tube \( T_1 \) and \( T_2 \) into tube \( T_0 \). This indicates that DNA strands in tube \( T_0 \) include DNA sequences of \( x_k = 1 \) and \( x_k = 0 \). After repeating execution of Step (4a) through Step (4d), it finally produces tube \( T_0 \) that consists of \( 2^q \) DNA sequences representing \( 2^q \) possible subsets. Therefore, it is inferred that \( 2^q \) possible subsets of a \( q \)-element set \( S \) can be constructed with DNA strands via this algorithm.
From Init \((T_0, q)\), it takes \((q-1)\) amplify operations, \(2 \times q\) append operations, \(q\) merge operations and three test tubes to construct the solution space for a \(q\)-element set \(S\). A \(q\)-bit binary number corresponds to a subset. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding a subset, is \(15 \times q\) bases, which is comprised of the concatenation of one value sequence for each bit.

### 3.3. Solution Space of the Value for Every Element of Each Subset for Solving the 0-1 Knapsack Problem of a Finite Set

For the purpose of appending the DNA strands that encode the weight \(w_m\) or profit \(p_m\) of all selected items, an element \(w_m\) \((p_m)\) for \(1 \leq m \leq q\) representing the size of weight (profit) of items \(m\) can be converted as a \(n\)-bit binary number, \(w_{m,n} = w_{m,n-1}, \ldots, w_{m,2}, w_{m,1}\). Suppose that \(w_{m,n}\) \((p_{m,n})\) is the most significant bit, while \(w_{m,1}\) \((p_{m,1})\) is the least significant bit. For every bit \(w_{m,k}\) \((p_{m,k})\), \(1 \leq m \leq q\) and \(1 \leq k \leq n\), from \([20, 21]\) two distinct DNA sequences are designed. One corresponds to the value "0" for \(w_{m,k}\) \((p_{m,k})\), and the other corresponds to the value "1" for \(w_{m,k}\) \((p_{m,k})\). For the sake of convenience in our representation, assume that \(w_{m,k}\) \((p_{m,k})\) denotes the value of \(w_{m,n}(p_{m,n})\) to be 1 and \(w_{m,k}\) \((p_{m,k})\) defines the value of \(w_{m,n}(p_{m,n})\) to be 0. The following algorithm is employed to construct the binary values of weight and profit for each element in \(2^n\) subsets of a \(q\)-element set \(S\). The partial result generated by ValueWT_PT \((T_0, q, n)\) for our example in subsection 3.1 is shown in Table 3.

**Table 3.** The Result Generated by ValueWT_PT \((T_0, q, n)\)

| Tube  | The result generated by ValueWT_PT \((T_0, q, n)\) |
|-------|---------------------------------------------------|
| \(T_0\) | \(\{k_1x_1x_1w_1s_2w_1s_1w_1s_1p_1p_1s_1w_1s_1w_2s_1w_2s_1p_1s_1w_1s_1w_2s_1w_2s_1p_1s_1w_1s_1w_2s_1w_2s_1\} \) |
| \(T_1\) | \(\{k_1x_1x_1w_1s_2w_1s_1w_1s_1p_1p_1s_1w_1s_1w_2s_1w_2s_1p_1s_1w_1s_1w_2s_1w_2s_1p_1s_1w_1s_1w_2s_1w_2s_1\} \) |
| \(T_2\) | \(\{k_1x_1x_1w_1s_2w_1s_1w_1s_1p_1p_1s_1w_1s_1w_2s_1w_2s_1p_1s_1w_1s_1w_2s_1w_2s_1p_1s_1w_1s_1w_2s_1w_2s_1\} \) |

(1) For \(m = 1\) to \(q\)
(1a) \(T_1 = + (T_0, x_1)\) and \(T_2 = - (T_0, x_1)\)
(1b) For \(k = n\) downto 1
(1c) Append \((T_1, w_{m,k})\)
(1d) Append \((T_2, w_{m,k})\)
End For

(1e) For \(k = n\) downto 1
(1f) Append \((T_1, p_{m,k})\)
(1g) Append \((T_2, p_{m,k})\)
End For

(1h) \(T_0 = \cup(T_1, T_2)\)

### End For

### End Procedure

**Lemma 2:** The binary value of weight and profit for each element in \(2^q\) subsets of a \(q\)-element set \(S\) can be constructed from the algorithm, ValueWT_PT \((T_0, q, n)\).

**Proof:** Refer to Lemma 1.

From ValueWT_PT \((T_0, q, n)\), it takes \(q\) extract operations, \(4 \times n \times q\) append operations, \(q\) merge operations and three test tubes to construct the solution space for elements in \(2^q\) subsets of a \(q\)-element set \(S\). A \(q\)-bit binary number corresponds to a choice of items and an \(n\)-bit binary number encodes the weight or profit of an item. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding the corresponding weight and profit for \(2^q\) possible choices of \(q\) items, is \(15 \times (4 \times n \times q)\) bases which consist of the concatenation of one value sequence for each bit.

### 3.4. The Construction of a Parallel One-bit Adder

A one-bit adder is a Boolean function that forms the arithmetic sum of three inputs. It includes three inputs and two outputs. Two of the input bits represent augend and addend, respectively. The third input represents the carry from the previous lower significant position. The first output gives the value of the least significant bit of the sum for augend, addend and previous carry. The second output gives the output carry transferred into the input carry of the next one-bit adder. The truth table of the one-bit adder is shown in Table 4.

**Table 4.** The Truth Table of One-Bit Adder

| Augend bit | Addend bit | Previous bit | Sum bit | Carry bit |
|-----------|-----------|-------------|--------|----------|
| 0         | 0         | 0           | 0      | 0        |
| 0         | 0         | 1           | 1      | 0        |
| 1         | 0         | 1           | 1      | 0        |
| 0         | 1         | 1           | 0      | 1        |
| 1         | 0         | 0           | 1      | 0        |
| 1         | 0         | 1           | 0      | 1        |
| 1         | 1         | 0           | 0      | 1        |
| 1         | 1         | 1           | 1      | 0        |

Suppose that two one-bit binary numbers, \(\alpha_{m-1,k}\) and \(\alpha_{m,k}\), represent the first input (addend) and the first output (sum) of a one-bit adder for \(1 \leq m \leq q\) and \(1 \leq k \leq n\), respectively. A one-bit binary number, \(\beta_{m,k}\), is applied to represent the second input (augend) of a one-bit adder. Two one-bit binary numbers, \(\gamma_{m,k-1}\) and \(\gamma_{m,k}\), are used to represent the third input (previous carry) and the second output (carry) of a one-bit adder respectively. From \([20, 21]\), two distinct DNA sequences are designed to represent the value "0" and "1" for every corresponding bit. For the sake of convenience in our representations, assume that \(\beta_{m,k}\) contains the value of \(\beta_{m,k}\) to be 1, and \(\beta_{m,k}\) contains the value of \(\beta_{m,k}\) to be 0. Also suppose that \(\alpha_{m,k}\) denotes the value of \(\alpha_{m,n-1,k}\) to be 1, and \(\alpha_{m,k}\) defines the value of \(\alpha_{m,n-1,k}\) to be 0.

Similarly, suppose that \(\gamma_{m,k}\) contains the value of \(\gamma_{m,k}\) to be 1, and \(\gamma_{m,k}\) denotes the value of \(\gamma_{m,k}\) to be 0. \(\gamma_{m,k}\) denotes the value of \(\gamma_{m,n-1,k}\) to be 1, and \(\gamma_{m,k}\) contains the value of \(\gamma_{m,n-1,k}\) to be 0.
defines the value of $\gamma_{mk}$ to be 1, and $\gamma_{mk}^0$ contains the value of $\gamma_{mk}$ to be 0. The following algorithm is proposed to perform the Boolean function of a parallel one-bit adder.

**Procedure ParallelOneBitAdder** ($T_0$, $a_{m,k}$, $\beta_{mk}$, $\gamma_{mk}$, $m$, $k$)

1. $T_1 = + (T_0, a_{m,k})$ and $T_2 = - (T_0, a_{m,k})$
2. $T_3 = + (T_1, \beta_{mk})$ and $T_4 = - (T_1, \beta_{mk})$
3. $T_5 = + (T_2, \beta_{mk})$ and $T_6 = - (T_2, \beta_{mk})$
4. $T_7 = + (T_3, \gamma_{mk}^1)$ and $T_8 = - (T_3, \gamma_{mk}^1)$
5. $T_9 = + (T_4, \gamma_{mk}^1)$ and $T_{10} = - (T_4, \gamma_{mk}^1)$
6. $T_{11} = + (T_5, \gamma_{mk}^1)$ and $T_{12} = - (T_5, \gamma_{mk}^1)$
7. $T_{13} = + (T_6, \gamma_{mk}^1)$ and $T_{14} = - (T_6, \gamma_{mk}^1)$
8. If (Detect ($T_7$) = "$yes") then
   Append-head ($T_7$, $a_{mk}^1$) and Append-head ($T_7$, $\gamma_{mk}^1$)
EndIf
9. If (Detect ($T_9$) = "$yes") then
   Append-head ($T_9$, $a_{mk}^0$) and Append-head ($T_9$, $\gamma_{mk}^0$)
EndIf
10. If (Detect ($T_5$) = "$yes") then
    Append-head ($T_5$, $a_{mk}^1$) and Append-head ($T_5$, $\gamma_{mk}^1$)
EndIf
11. If (Detect ($T_{10}$) = "$yes") then
    Append-head ($T_{10}$, $a_{mk}^0$) and Append-head ($T_{10}$, $\gamma_{mk}^0$)
EndIf
12. If (Detect ($T_{11}$) = "$yes") then
    Append-head ($T_{11}$, $a_{mk}^1$) and Append-head ($T_{11}$, $\gamma_{mk}^1$)
EndIf
13. If (Detect ($T_{12}$) = "$yes") then
    Append-head ($T_{12}$, $a_{mk}^1$) and Append-head ($T_{12}$, $\gamma_{mk}^1$)
EndIf
14. If (Detect ($T_{13}$) = "$yes") then
    Append-head ($T_{13}$, $a_{mk}^1$) and Append-head ($T_{13}$, $\gamma_{mk}^1$)
EndIf
15. If (Detect ($T_{14}$) = "$yes") then
    Append-head ($T_{14}$, $a_{mk}^0$) and Append-head ($T_{14}$, $\gamma_{mk}^0$)
EndIf
16. $T_0 = \cup (T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$
EndProcedure

**Lemma 3:** The algorithm, ParallelOneBitAdder ($T_0$, $a_{m,k}$, $\beta_{mk}$, $\gamma_{mk}$, $m$, $k$), can be applied to perform the Boolean function of a parallel one-bit adder.

**Proof:** Refer to Lemma 1.

From ParallelOneBitAdder ($T_0$, $a_{m,k}$, $\beta_{mk}$, $\gamma_{mk}$, $m$, $k$), it takes seven extract operations, eight detect operations, sixteen append-head operations, one merge operation, and fifteen test tubes to compute the addition of three input bits. Two output bits of a one-bit adder encode the sum and the carry to the addition of a bit. A value sequence for every output bit contains 15 base pairs. Therefore the length of a DNA strand, encoding two output bits has 30 base pairs, consists of the concatenation of one value sequence for each output bit.

### 3.5. The Construction of a Parallel N-bit Adder

The parallel one-bit adder introduced in subsection 3.4 figures out the arithmetic sum of two bits and a previous carry. Similarly, A binary parallel n-bit adder is also directly to perform the arithmetic sum for the two input operands of n-bit and the input carry by means of performing this one-bit adder n times. The following algorithm is proposed to perform the arithmetic sum for a parallel n-bit adder.

**Procedure ParallelAdder** ($T_0$, $a$, $\beta$, $\gamma$, $q$, $n$)

1. For $k = n$ downto 1
   1a. Append ($T_0$, $a_{0,k}$)
EndFor
2. For $m = 1$ to $q$
   2a. Append ($T_0$, $\gamma_{0,m}$)
   2b. For $k = 1$ to $n$
   2c. ParallelOneBitAdder ($T_0$, $a_{m-1,k}$, $\beta_{mk}$, $\gamma_{mk-1}$, $m$, $k$)
EndFor
EndProcedure

**Lemma 4:** The algorithm, ParallelAdder ($T_0$, $a$, $\beta$, $\gamma$, $q$, $n$), can be applied to perform the Boolean function to a binary parallel adder of n bits.

**Proof:** Refer to Lemma 1.

From ParallelAdder ($T_0$, $a$, $\beta$, $\gamma$, $q$, $n$), it takes $7 \times n \times q$ extract operations, $(n + q + 2 \times n \times q)$ append operations, $n \times q$ merge operations and fifteen test tubes to compute the sum of weight for elements in $2^n$ subsets of a q-element set S. A q-bit binary number corresponds to a subset. An n-bit binary number encodes the size of weight for an element in S. Therefore, $(q + 1) \times n$ bits correspond to the sum of weight for q elements, and one accumulator element $(\alpha)$. $q \times (n + 1)$ bits encode the carry of the sum. A value sequence for every bit contains 15 bases. Therefore, the length of a DNA strand, encoding the total weight of selected items, is $15 \times n$ base pairs consisting of the concatenation of one value sequence for each bit.

### 3.6. Parallel Comparator for Comparing the Sum of Weight Corresponding to Subsets of a Finite Set with Any Given Positive Integer

Any given positive integer, $M$, can be converted as $n$ one-bit binary numbers, $M_1M_{n-1}…M_2M_1$. The main advantage is that it is feasible for bit operations of the DNA
algorithm in a parallel comparator. **MaxiWeight** \((T_M, n)\) is proposed to construct DNA strands for encoding \(M\). **OneBitComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, k)\) is presented to finish the function of a parallel comparator for one bit and **ParallelComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, n)\) also is proposed to finish the function of a parallel comparator for comparing the total weight of selected items with the capacity of knapsack \(M\). The result of tube \(T_M\) generated by **MaxiWeight** \((T_M, n)\) is shown in Table 5.

**Procedure MaxiWeight** \((T_M, n)\)

For \(k = n \text{ downto } 1\)

(1a) **Append** \((T_M, M_k)\)

EndProcedure

Table 5. The Result Generated by MaxiWeight\((T_M, n)\)

| Tube     | The result generated by MaxiWeight\((T_M, n)\) |
|----------|---------------------------------------------|
| \(T_M\)  | \(\{M_1^0M_1^\precM_2^0M_2^\prec\}\)        |

**Lemma 5:** Any given positive integer \(M\) can be constructed with DNA strands from the algorithm, **MaxiWeight** \((T_M, n)\).

**Proof:** Refer to **Lemma 1**.

From **MaxiWeight** \((T_M, n)\), it takes \(n\) append operations and one test tube to construct the maximum weight. Any given positive integer \(M\) corresponds to \(n\) one-bit binary numbers. A value sequence for each bit contains 15 bases. Therefore, the length of DNA strands encoding \(M\) is \(15 \times n\) bases comprised the concatenation of one value sequence for each bit.

**Procedure OneBitComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, k)\)

\(1\) \(T_0^{ON} = + (T_0, W_{q,k}^1)\) and \(T_0^{OFF} = - (T_0, W_{q,k}^1)\)

\(2\) \(T_M^{ON} = + (T_M, M_q^1)\) and \(T_M^{OFF} = - (T_M, M_q^1)\)

\(3\) If (Detect \((T_M^{ON} = \text{"yes"})\) then

\(3a\) \(T_0^+ = \cup (T_0^+, T_0^{ON})\) and \(T_0^- = \cup (T_0^-, T_0^{OFF})\)

Else

\(3b\) \(T_0^+ = \cup (T_0^+, T_0^{OFF})\) and \(T_0^- = \cup (T_0^-, T_0^{ON})\)

EndIf

\(4\) \(T_M^{ON} = \cup (T_M^{ON}, T_M^{OFF})\)

Endprocedure

**Lemma 6:** The algorithm, **OneBitComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, k)\) can be applied to finish the function of a one-bit parallel comparator.

**Proof:** Refer to **Lemma 1**.

From **OneBitComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, k)\), it takes two extract operation, one detect operations, three merge operations and seven test tubes to carry out the function of a one-bit comparator.

**Procedure ParallelComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, n)\)

For \(k = n \text{ downto } 1\)

(1a) **OneBitComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, k)\)

(1b) If (Detect \((T_0^\prec) = \text{"yes"}\) then

(1c) \(T_0 = \cup (T_0^\prec, T_0)\)

Else

(1d) Terminate the execution of loop

EndIf

EndFor

EndProcedure

**Lemma 7:** The algorithm, **ParallelComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, n)\) can be applied to finish the function of an \(n\)-bit parallel comparator.

**Proof:** Refer to **Lemma 1**.

From **ParallelComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, n)\), it takes \(2 \times n\) extract and detect operations, \(4 \times n\) + 1 merge operations, one discard operation and seven test tubes to carry out the function of an \(n\)-bit parallel comparator.

3.7. Search and Calculation of the Maximum Total Profit under the Restriction That the Total Weight Does Not Exceed the Capacity of Knapsack \(M\)

The following algorithm is applied to determine which strand has the maximum profit among those chosen items satisfying the constraint of problem. The strand that remains in tube \(T_0\) has the greatest binary value of profit after performing this algorithm.

**Procedure SearchMaxi** \((T_0, q, n)\)

\(1\) For \(k = n \text{ downto } 1\)

\(1a\) \(T_0^{ON} = + (T_0, P_{q,k}^1)\) and \(T_0^{OFF} = - (T_0, P_{q,k}^1)\)

\(1b\) If (Detect \((T_0^{ON}) = \text{"yes"})\) then

\(1c\) \(T_0 = \cup (T_0^{ON})\) and Discard \((T_0^{OFF})\)

Else

\(1d\) \(T_0 = \cup (T_0^{OFF})\)

EndIf

EndFor

EndProcedure

**Lemma 8:** The algorithm, **SearchMaxi** \((T_0, q, n)\) can be used to search the strands that encode the maximum value of profit.

**Proof:** Refer to **Lemma 1**.

From **SearchMaxi** \((T_0, q, n)\), it takes \(n\) extract operations, \(n\) detect operations, \(2 \times n\) merge operations and \(n\) discard operations, and three test tubes to carry out the function of searching the maximum total profit subject to the condition that the total weight of selected items must not exceed the capacity of the knapsack.

3.8. DNA Algorithm for Solving the 0-1 Knapsack Problem

**Algorithm 1:** Solving the 0-1 Knapsack Problem

**Init** \((T_0, q)\)

**ValueWT_PT** \((T_0, q, n)\)

**ParallelAdder** \((T_0, W, w, z, q, n)\)

**MaxiWeight** \((T_M, n)\)

**ParallelComparator** \((T_0, T_M, T_0^+, T_0^-, T_0^\prec, q, n)\)

**ParallelAdder** \((T_0, P, p, r, q, n)\)

**SearchMaxi** \((T_0, q, n)\)

**Theorem 1:** From those steps in Algorithm 1, the
0-1 Knapsack Problem can be solved.

**Proof:** Refer to Lemma 1.

### 3.9. The Complexity of Algorithm 1

**Theorem 2:** Suppose that a finite set \( S \) is \( \{\text{item}_1, \text{item}_2, \ldots, \text{item}_q\} \). The 0-1 Knapsack Problem for \( S \) can be solved with \( O(q \times n) \) biological operations, \( O(2^n) \) library strands, \( O(1) \) tubes and the longest library strands, \( O(q \times n) \), Where \( n \) is the number of bits for representing the size of corresponding weight or profit of element in set \( S \).

**Proof:** Refer to Algorithm 1.

### 4. Conclusions

The knapsack problem proved to be the NP-complete problem by restriction has been solved by a number of different algorithms with exponential time complexity in conventional silicon-based computer [27]. Here the proposed algorithm for solving the 0-1 knapsack problem is based on basic biological operations. The number of tubes, the number of biological operations, the number of memory strands and the longest length of memory strands, respectively, are \( O(1) \), \( O(q \times n) \), \( O(2^n) \) and \( O(q \times n) \).

The presented algorithm has several advantages lying in its massive parallelism as described below. First, these biological operations needed in the proposed algorithm are experimentally feasible in lab level [20,21]. Second, the DNA-based algorithms of polynomial time complexity are proposed for searching the maximum profit among a group of items. Third, the Adleman program [20,21] can be applied to generate good DNA sequences for the constructing solution space of our problem. It demonstrates that the proposed algorithm has a lower rate of errors for hybridization. Fourth, the contribution of this study is that DNA-based algorithms developed herein can be applied to solve addition-related problems.

### Acknowledgements

I am deeply grateful to Dr. W.L. Chang for his kind and constructive suggestions to improve this research.

### REFERENCES

[1] R. P. Feynman. In miniaturization, D.H. Gilbert, Ed., Reinhold Publishing Corporation, New York, 282-296,1961.

[2] L. Adleman. Molecular computation of solutions to combinatorial problems, Science, 266, 1021-1024, 1994.

[3] R. J. Lipton. DNA solution of hard computational problems, Science, 268, 542-545, 1995.

[4] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, Paul W.K. Rothemund, and L. M. Adleman. A sticker based model for DNA computation, 2nd annual workshop on DNA Computing, Princeton University. Eds. L. Landweber and E. Baum, DIMACS: series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1-29,1999.

[5] M. Guo, W. L. Chang, M. Ho, J. Lu and J. Cao. Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-based computing. Biosystems, Vol. 80, No. 1, 71-82, 2005.

[6] O. Quyang, P.D. Kaplan, S. Liu, and A. Libchaber. DNA solution of the maximal clique problem, Science, 278,446-449, 1997.

[7] M. Amos. DNA Computation, Ph.D Thesis, department of computer science, the University of Warwick, 1997.

[8] W. L. Chang, M. Guo, and M. Ho. Towards solution of the set-splitting problem on gel-based DNA computing. Future Generation Computer Systems, Vol.20, Iss. 5, 875-885, 2004.

[9] W. L. Chang and M. Guo. Solving the Set-cover Problem and the Problem of Exact Cover by 3-Sets in the Adleman-Lipton's Model. BioSystems, Vol. 72, No. 3, 263-275, 2003.

[10] W. L. Chang, M. Ho, and M. Guo. Fast parallel molecular solution to the dominating-set problem on massively parallel bio-computing, Parallel Computing (Elsevier Science), Vol. 30, No. 9&10, 1109-1125, 2004.

[11] D. Xiao, W. Li, Z. Zhang, and L. He. Solving the maximum cut problems in the Adleman–Lipton model, BioSystems, Vol. 82, 203-207, 2005.

[12] C. W. Yeh, C. P. Chu and K. R. Wu. Molecular solutions to the binary integer programming problem based on DNA computation, Biosystems, Vol. 83, Iss. 1, 56-66, 2006.

[13] Boneh, C. Dunworth, and R. J. Lipton. Breaking DES using a molecular computer". In Proceedings of the 1st DIMACS Workshop on DNA Based Computers, 1995, American Mathematical Society. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 27, 37-66, 1996.

[14] L. Adleman, P. W. K. Rothemund, S. Roweis, and E. Winfree. On applying molecular computation to the Data Encryption Standard, The 2nd annual workshop on DNA Computing, Princeton University, DIMACS: series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 31-44, 1999.

[15] F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add, Science, Vol. 273, 220–223, 1996.

[16] M. Ho. Fast parallel molecular solutions for DNA-based supercomputing: the subset-product problem, BioSystems, Vol. 80, 233–250, 2005.

[17] H. Ahrabian and A. Nowzari-Dalini. DNA simulation of nand Boolean circuits, Advanced Modeling and Optimization, Vol. 6, No. 2, 33-41, 2004.

[18] A. Schuster. DNA databases, BioSystems, Vol. 81, 234–246, 2005.

[19] W.L. Chang. Fast Parallel DNA-based Algorithms for
Using Parallel Filtering Algorithms to Solve the 0-1 Knapsack Problem on DNA-based Computing

Molecular Computation: the Set-Partition Problem, IEEE Transactions on Nanobioscience, Vol. 6, No. 1, 346 – 353, 2007.

[20] R. S. Braich, C. Johnson, P. W.K. Rothemund, D. Hwang, N. Chelyapov, and L. M. Adleman. Solution of a satisfiability problem on a gel-based DNA computer. Proceedings of the 6th International Conference on DNA Computation in the Springer-Verlag Lecture Notes in Computer Science series, 1-27, 2001.

[21] L. M. Adleman, R. S. Braich, C. Johnson, P. W. K. Rothemund, D. Hwang and N. Chelyapov. Solution of a 20-Variable 3-SAT Problem on a DNA Computer. Science, Vol. 296, Iss. 5567, 499-502, 2002.

[22] J. Watson, M. Gilman, J. Witkowski, and M. Zoller. Recombinant DNA (2nd edition). Scientific American Books, W. H. Freeman and Co., 1992.

[23] J. Watson, N. Hoplins, J. Roberts, and et. al. Molecular Biology of the Gene. Benjamin/Cummings Menlo Park CA, 1987.

[24] G. M. Blackburn and M. J. Gait. Nucleic Acids in Chemistry and Biology. IRL Press, 1990.

[25] F. Eckstein. Oligonucleotides and Analogues. Oxford University Press. 1991.

[26] M. Amos. Theoretical and Experimental DNA Computation. Springer, 2005.

[27] M. R. Garey, and D. S. Johnson. Computer and intractability. Freeman, San Fransico, CA, 1979.