A Linear Time DNA Algorithm to Solve 0/1 Knapsack

Babak Khorsand[a]*, Javad Zahiri[b], Abdorreza Savadi[a]

[a] Computer Engineering Department, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
[b] Bioinformatics and Computational Omics Lab (BioCOOL), Faculty of Biological Sciences, Tarbiat Modares University, Tehran, Iran

Abstract. Many combinatorial optimization problems are known to be NP-hard, which can be solved in exponential time. Due to the vast parallelism capability of the Deoxyribonucleic Acid (DNA), DNA computing has been exploited to solve various NP-hard problems in polynomial time. The 0/1 knapsack problem is a well-known NP-hard problem, which designed to find a subset of a given set of objects with maximum profit and the total is weight less than or equal to the knapsack capacity. In this paper, we present a novel parallel DNA algorithm to solve the 0/1 knapsack problem in O(n) time complexity.

Keywords: DNA computing; Knapsack; Adelman.

1. Introduction

Deoxyribonucleic acid (DNA) is a molecule that carries most of the genetic instructions used in the development, functioning and reproduction of all known living organisms. Just 1 gram of DNA can hold about 455 Exabyte of data equal to 4 ×1021 bits. The number of compact disks (CDs) required to hold this amount of information, lined up edge to edge, would circle the Earth 375 times. With bases spaced at 0.35 nm along DNA, data density is over a million Gigabits/inch compared to seven Gigabits/inch in typical high performance hard disk drive (HDD). Considering parallelism aspect, a DNA molecule is capable of parallelizing 300 tera molecules (300 × 1012) at a time [1]. This enormous parallelism has encouraged the researchers to build a DNA-based model for solving NP-hard problems [2].

The first DNA computing model was introduced by Adelman in 1994 [3]. He described how to solve a seven-node instance of Hamiltonian path problem utilizing biological operations, and also showed the enormous parallel power of DNA computation. In 1995, Lipton used Adleman's method to solve the Satisfiablility problem [4]. In 1997, Ouyang solved Maximal clique problem [5] by designing Restriction enzyme model. In 1998 Winfree designed two new models: The Sticker model [6] and The Self-assembly model [7]. Smith also designed the Surface based model [8] in 1998. Also, in 2000, Sakomoto proposed the Hairpin model [9].

Recently, substantial efforts have been done to use these models for solving various NP-hard problems such as 0-1 knapsack [10], binpacking [11], assignment [12] and n-queen [13].

In this paper, based on a combination of Adleman-Lipton model, a parallel DNA algorithm is proposed for solving knapsack problem, which is able to solve the problem in O(n) time complexity, compared to the exponential time complexity by electronic computing algorithm.

2. Knapsack problem

Knapsack 0-1 [14] is a famous NP-hard problem. In this problem, we have a knapsack that can tolerate a specific weight (W). Moreover, there are n different objects with different weights (w_i), each with a specific value (v_i). We want to select the best possible items such that the sum of the values becomes maximized without exceeding W. Another constraint in 0-1 edition of knapsack is that it is impossible to choose a fraction of an item, so, either the whole item or none of that must be chosen.

Knapsack problem has been formulated in the following formula:

$$\sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}$$

$$i = \{1,2,\dots,n\}$$

* Corresponding author.
E-mail address: khorsand@stu-mail.um.ac.ir

3. Adelman-Lipton model

A DNA computing algorithms consists of the three following steps:

- Producing specific sequences and putting them in a test tube.
- Doing a number of operations on the DNA sequences inside the tube and removing unwanted sequences, which certainly cannot be the final answer of the problem.
- Reading and reporting the DNA sequence, if any exists in final tube.

Different operations in the Adelman-Lipton model:

- Merge (T_1, T_2): It will merge the content of the tube T_1 with the tube T_2 and put the mixture in the tube T_1, (the tube T_2 will be empty)
- Copy (T_1, T_2): It will copy the content of the tube T_1 to the empty tube T_2.
- Detect (T): If tube the T contains at least one sequence, then it will return Yes, otherwise No.
- Separation (T_1, X, T_2): It will move the subset of the tube T_1 which contains the X sequence and put it in the tube T_2.
- Select (T_1, X, T_2): It will move those sequences from tube T_1 which contains the sequences longer than the sequence X and put them in the tube T_2.
- Selecting (T_1, L, T_2): It will select the sequences of size L from the tube T_1 and put them in the tube T_2.
- SelectMax (T_1, T_2): It will select the longest sequence in the tube T_1 and move it into the tube T_2.
- Annealing (T): With freezing the tube converts ssDNA to dsDNA.
- Denaturing (T): With Warming the tube converts dsDNA to ssDNA.
- Discard (T): It will eliminate the content of the tube T.
- Amplify (T, T_1, T_2): make a lot of copies of the content of tube T via Polymerase Chain Reaction (PCR) [15] and put them in the tubes T_1 and T_2.
- Append (T, Z): It will put the sequence Z to the end of the all sequences in the tube T.
- Read (T): It will read a sequence from the tube T.

4. DNA algorithm for 0-1 knapsack problem

4.1. Basic idea

The basic idea of our DNA algorithm for the knapsack problem consists of two phases:

a) In the first phase we will generate all the possible selection of objects without considering any constraint. Then, we will keep the strands that have the length less than or equal to the W.

b) In the second phase, we will add some sequences to each strand equal to values vi and then evaluate all the sequences. The longest sequence will be the final solution. More precisely, the proposed algorithm consists of the following six steps:

Step 1: Build two unique 10-mer (sequence with length 10) for each item and then add a w_i-mer of G nucleotides (w_i is the weight of the corresponding item) to the end of the 10-mer.

Step 2: Construct all 2^n possible combinations of sequences.

Step 3: Eliminate all sequences which are higher than the knapsack capacity.

Step 4: Elongate the remaining sequences to have the same length as the longest one by adding spare G to the end of the sequences.

Step 5: Read each sequence and add v_i-mer of G nucleotides (v_i is the value of the corresponding item) to the end of the sequence.

Step 6: Consider the longest sequence as the optimal solution.

4.2. Strand design

First of all, for each item I_i we produce two distinct 10mer sequences, namely S_i^0 and S_i^1, to show not selection and selection of item I_i, respectively. By using three nucleotides A, T and C for a 10mer strand we can produce 59049 different strands.

Among them we select those who have the most difference. Then add number of Gs to S_i^1 (strands which assumed as that item has been selected) equal to w_i. In adding G, it is easily possible to make a rule according to that problem and to confine the number of G. As an example dividing all sequences by Great Common Divider (GCD) of all wis and W which is 20 in our example. As you can see in table 1 for eight different objects of knapsack, we use 16 different 10mer sequence and then add a G for each 20 gr to S_i^1s. As an example for creating strand for item I6 with w = 240 gr use a distinct 10mer sequence (ACTAATCTCT) for S_i^0 and another 10mer distinct sequence (CCTTAACATA) for S_i^1. Then divide GCD by its W, (240 / 20 = 12) and add 12G to S_i^1 (CCTTAACATAGGGGGGGGGGG).

In the second phase, first we make all the remaining sequences equal the maximum length and then add number of Gs equal to v_i of each of S_i^1s. As it mentioned in the first phase it is easily possible to divide all vis by GCD of all vis to reduce the number of needed G.

As an example for a possible final choice with considering 900 gr capacity for the knapsack it is possible to select I_2, I_4, I_5 and I_7 which will be coded as below in the first phase:

ACCTTATCAT CATCATTTACGGGGGG TTACATCATC TACCTCAACTGGGGGGGGG ACATTCAATCGGGGG-GGGGG ACTAATCTCT ACCCAATACTGGGGGGGGG-GGGGGG CACACATCTA

In the second phase add 5G to the above sequence to make it as high as the highest sequence among all valid remaining sequences. And finally add (20+30+8+10) / 2 = 34 G which is equal to value of the selected items divided by GCD.

ACCTTATCATCATCATTTACGGGGGGTTACATCATC-TACCTCAACTGGGGGGGGGACATTCAATCGGGGG-GGGGGACTAATCTCTACCCAATACTGGGGGGGGGG-GGGGGCACACATCTAGGGGGGGGGGGGGGGGGG-GGGGGGGGGGGGGGGGGGGG

Table 1. Sequences chosen to represent the elements of knapsack problem.

Item	Weight	Value	DNA sequence	Length
S_1^0	100gr	-	ACCTTATCAT	10
S_2^0	120gr	-	TCTATAACTA	10
S_3^0	140gr	-	TTACATCATC	10
S_4^0	180gr	-	TAACACTATT	10
S_5^0	200gr	-	TTCATACTAA	10
S_6^0	240gr	-	ACTAATCTCT	10
S_7^0	300gr	-	CCCTTATCAA	10
S_8^0	320gr	-	CACACATCTA	10
S_1^1	100gr	4$	AACTACATTTGGGGG	15
S_2^1	120gr	20$	CATCATTTACGGGGGG	16
S_3^1	140gr	6$	ATTACTCCTAGGGGGGG	17
S_4^1	180gr	30$	TACCTCAACTGGGGGGGGG	19
S_5^1	200gr	8$	ACATTCAATCGGGGGGGGGG	20
S_6^1	240gr	10$	CCTTAACATAGGGGGGGGGGGG	22
S_7^1	300gr	10$	ACCCAATACTGGGGGGGGGGGGGGG	25
S_8^1	320gr	2$	TTCTATCTATGGGGGGGGGGGGGGG	26

4.3. Sample space

Put S_1^1 in tube T and S_1^0 in T_1.
Merge (T, T_1)
For i in 1:(number of items -1)
{Amplify (T, T_1, T_2)
Append (T_1, S_i^0)
Append (T_2, S_i^1)
Merge (T_1, T_2)
Discard (T)
Copy (T_1, T)
Discard (T_1, T_2)}

4.4. Eliminate all sequences which are higher than knapsack capacity strand.

Convert W to number of nucleotide. In our example according to formula 2:

$$10n + \left(\frac{W}{20}\right)$$

Make a sequence with number of Gs equal to W and consider it as item_capacity.
Amplify (T_1)
Merge (T, T_1)
Select (T, item_capacity, T_1)
Discard (T_1)

4.5. Make all the remaining sequences the same length

SelectMax (T, T_1)
Max = Read (T_1)
While (Detect (T))
{
Append (T, G)
Select (T, Max, T1)
}
Copy (T1, T)
Discard (T1)

4.6. Calculate profit of each item for the related sequences

For i in 1:(number of items)
{Make a sequence with a number of G equal to the v_i and named it $Value_i$}
For i in 1 : (number of items)
{
Separation (T, S_i^1,T_1)
Append (T_1, $Value_i$)
Copy (T_1, T_2)
Discard (T_1)
}
Discard (T)
Copy (T_2, T)
Discard (T_2)

4.7. Final Answer

SelectMax (T, T1)
Read (T1)

5. Time complexity and feasibility

Theorem 1. The solutions of our proposed algorithm for knapsack 0/1 problem can be obtained by the bio molecular operations.

Proof. In the second step, all combinations of the n items were generated in the sample space. In addition, by the end of the third step, all the sequences longer than knapsack capacity are thrown down so the first constraint will be satisfied. By performing the fourth step we make all sequences the same length and by adding the proper G to each of the remaining sequences, the sequences will get weight equal to their value. So the longest sequence will be the most valuable set of items which would not exceed W.

Theorem 2. The time complexity of proposed algorithm for solving 0/1 knapsack problem is O(n)

Proof. Considering the complexity of every biological operation O(1) [16], the time complexity of algorithm would be sum of the time complexity of all steps:

Step 1: Seven biological operation in each step would become O(1) and in n step it would be O(n).

Step 2: Five biological operation O(1) and totally it would be O(1).
Step 3: Four biological operation O(1) and totally it would be O(1) plus 2 biological operation in each step of the loop which become O(1) and in n step it would be O(n).
Step 4: One biological operation in each step of the first loop which become O(n) plus four biological operation in each step of the loop which become O(1) and in n step it would be O(n) plus three biological operation O(1).
Step 5: Two biological operation which become O(1).

Therefore, the time complexity of the algorithm will be arrived by formula 3:

$$T(n) = O(n) + O(1) + O(n) + O(n) + O(1) = O(n) \quad (3)$$

6. Conclusion

Ultra efficient parallelism of the methods of DNA computing versus the obvious limitations in storage, speed, intelligence, and miniaturization of electronic computers is considerable. Although there are still some problems that need further study in biologic technology, it is still possible to solve a lot of NP-hard problems in linear time via DNA computing. In this article, we highlight a DNA computing model with biological operations based on Adelman-Lipton model to solve knapsack 0/1 problem with time complexity of O(n) with just 2n 10mer sequences.

We hope that in future studies, by adventing DNA computers, all DNA computing solutions become practicable and all NP-hard problems become solvable in linear time.

References

1. Amos, Martyn, et al. "Topics in the theory of DNA computing." Theoretical computer science 287.1 (2002): 3-38.
2. Amos, Martyn, et al. "Topics in the theory of DNA computing." Theoretical computer science 287.1 (2002): 3-38.
3. Adleman, Leonard M. "Molecular computation of solutions to combinatorial problems." Nature 369 (1994): 40.
4. Lipton, Richard J. "DNA solution of hard computational problems." Science 268.5210 (1995): 542.
5. Ouyang, Qi, et al. "DNA solution of the maximal clique problem." Science 278.5337 (1997): 446-449.
6. Roweis, Sam, et al. "A sticker-based model for DNA computation." Journal of Computational Biology 5.4 (1998): 615-629.
7. Winfree, Erik, et al. "Design and self-assembly of two-dimensional DNA crystals." Nature 394.6693 (1998): 539-544.
8. Smith, Lloyd M., et al. "A surface-based approach to DNA computation." Journal of computational biology 5.2 (1998): 255-267.
9. Sakamoto, Kensaku, et al. "Molecular computation by DNA hairpin formation." Science 288.5469 (2000): 1223-1226.
10. Darehmiraki, Majid, and Hasan Mishmast Nehi. "Molecular solution to the 0–1 knapsack problem based on DNA computing." Applied mathematics and computation 187.2 (2007): 1033-1037.
11. Sanches, Carlos Alberto Alonso, and Nei Yoshihiro Soma. "A polynomial-time DNA computing solution for the Bin-Packing Problem." Applied Mathematics and Computation 215.6 (2009): 2055-2062.
12. Wang, Zhaocai, et al. "A biological algorithm to solve the assignment problem based on DNA molecules computation." Applied Mathematics and Computation 244 (2014): 183-190.
13. Wang, Zhaocai, et al. "A parallel algorithm for solving the n-queens problem based on inspired computational model." BioSystems 131 (2015): 22-29.
14. Dantzig, George B. "Discrete-variable extremum problems." Operations research 5.2 (1957): 266-288.
15. Mullis, K. B., et al. "One of the first Polymerase Chain Reaction (PCR) patents." Google Patents (1987).
16. Chang, Weng-Long, et al. "Quantum algorithms for biomolecular solutions of the satisfiability problem on a quantum machine." IEEE transactions on nanobioscience 7.3 (2008): 215-222.