코드 최적화 DNA-Haskell을 도입한 DNA 컴퓨팅에 의한 배낭 문제 해결

Solution for Knapsack Problem using DNA Computing with Code Optimized DNA-Haskell

김은경, 이상용
공주대학교 컴퓨터공학과, 공주대학교 정보통신공학부
Eun-Gyeong Kim, Sang-Yong Lee
Dept. of Computer Engineering, Division of Information & Communication Engineering, Kongju National University
E-mail : rotnrwk@kongju.ac.kr

요 약

배낭 문제는 조합 최적화 문제로서, 다항 시간(polynomial time)에 풀리지 않는 NP-hard 문제이다. 이 문제를 해결하기 위해 기존에는 DNA 컴퓨팅 기법과 GA 등을 사용하여 해결하였다. 하지만 기존의 방법들은 DNA의 정확한 특성을 고려하지 않아, 실제 실험과의 결과 차이가 발생하고 있다.

본 논문에서는 DNA 컴퓨팅 실험 과정에서 발생하는 DNA 조작 오류를 최소화하고, 보다 정확한 예측을 위해 함수 언어인 Haskell을 이용한 코드 최적화 DNA-Haskell을 제안한다. 코드 최적화 DNA-Haskell은 배낭 문제 중 (0,1)-배낭 문제에 적용하였고, 그 결과 기존의 DNA 컴퓨팅 방법보다 실험적 오류를 최소화 하였으며, 또한 적합한 해를 빠른 시간내에 찾을 수 있었다.

1. 서론

DNA 컴퓨팅은 실제 DNA를 계산 및 저장의 매체로 사용하고, 생물학 실험실에서 사용되는 여러 가지 실험 방법들을 연산자로 이용하는 계산 모델이다. 또한 DNA의 막대한 병렬성과 저장 능력을 이용하여 계산학적으로 어려운 문제들인 NP-complete 문제들을 해결하고 있다[1][2].

그러나 현재 DNA 컴퓨팅은 NP-complete 문제들에 적용하였을 때 다음과 같은 문제점들이 발생하고 있다. 첫 번째는 DNA에 기반한 컴퓨팅 모델이 실험실을 벗어나 일상 생활에서 적용하기에는 미흡하다. 두 번째는 이러한 모델을 프로그램으로 설계했을때 DNA 컴퓨팅의 요구조건을 충분히 만족하고 있지 않다. 세 번째는 실험실에서의 생물학적 실험 결과와 현실에 적용할 시뮬레이션 모델과의 실험 결과에 대한 평가에서 차이점이 발생하고 있다는 것이다.

이러한 문제점을 해결하기 위한 기존 연구로는 함수형 언어인 Haskell을 이용한 DNA-Haskell을 DNA 컴퓨팅에 도입한 연구가 있다. 하지만 문제점들을 완전히 해결하지는 못하고 있다.

본 논문에서는 DNA 컴퓨팅에 배낭문제를 적용했을때 발생되는 문제점들을 해결하기 위하여, DNA-Haskell을 보완한 코드 최적화 DNA-Haskell을 제안한다. 제안한 알고리즘은 NP 문제 중 (0,1)- 배낭 문제에 적용하였다. 그리고 2종류의 컴퓨터 시뮬레이션과 생물학적 DNA 실험을 비교 평가 하였다.

2. 관련연구

2.1 DNA 컴퓨팅

DNA 컴퓨팅은 합성 DNA를 정보소자로 사용하여 풀고자 하는 문제에 대한 해법을 DNA 코드로 기술하고, 주어진 DNA 조각들로부터 화학적으로 합성, 검출함으로써 답을 찾아 내는 기술을 말한다. DNA는 1㎤에 1조개의 CD보다 많은 정보를 가질 수 있으며, A(Adenine), C(Cytosine), G(Guanine), T(Thymine)의 4가지 염기로 이루워진다[3]. 또한 복잡한 염기 조합의 패턴은 하나의 유전 정보를 담고 있으며, 인체내

에서 자연 발생하는 효소에 의해 읽혀지고 있다. 효소는 생물학 실험 방법들과 함께 DNA 컴퓨팅의 연산자로 사용되고 있다.

이러한 DNA 컴퓨팅의 특징을 살펴보면 매우 낮은 에너지로 작동되기 때문에 많은 에너지가 필요없다. 그리고 나노 수준의 막대한 병렬성을 이용하여 NP-complete에 효과적인 접근이 가능하게 되었다. 또한 계산 속도와 정보의 저장 및 처리 효율에서도 우수함을 보이고 있다[4][5].

2.2 DNA-Haskell

DNA-Haskell은 함수형 언어 Haskell을 기반으로 고안되었다. Haskell은 추상화 레벨이 갖는 문제점들을 명세화하고, 수학적인 처리를 쉽도록 하였다. 이러한 특성들은 정확성 증명과 검정 그리고 프로그램 특성의 분석이 가능하다[6]. 그래서 Stoschek는 Haskell의 특성들을 DNA 컴퓨팅에 이용하기 위해 DNA-Haskell을 고안하였으며, 그 결과 실험실에서 이루어지는 DNA 컴퓨팅 처리를 쉽게 기술할 수 있었다[7].

DNA-Haskell은 DNA 컴퓨팅 알고리즘을 보완할 수 있는 두 가지 장점을 가지고 있다. 첫 번째로 실험 결과 유사한 대량의 데이터에 대하여 모든 기능들을 표현할 수 있다. 특히 NP 문제들을 해결하기 위해 시도된 DNA 컴퓨팅의 바이오하드웨어에 대한 분자 생물학 과정들을 보충하여 시뮬레이션 할 수 있다. 두 번째로 DNA 컴퓨팅 실험에서 발생된 문제점들을 해결하기 위해 알고리즘 개발에 능동적으로 대처할 수 있다.

표 1. 데이터 기호

DNA-Haskell symbol	label (molecule or chemical group)
@	hydroxyl group (unlabled)
P	phophate group
B	Biotin
C	Cy5

표 2. 데이터 생성 규칙

```
data Base = A | T | C | G | *
data Label = P | B | C | @
data Basepair= [A,T] | [T,A] | [C,G] | [G,C] | [A,*] |
     [T,*] | [C,*] | [G,*] | [*,T] | [*,A] | [*,G] | [*,C]
data Labelpair = [@,@] | [@,P] | [@,B] | [@,C] | [P,@]
     | [B,@] | [C,@] | [P,P] | [B,B]
data Strand = NIL | Cons Basepair Strand
data Dnastrand = Labelpair ++ Strand ++ Labelpair
data Tube = NIL | Cons Dnastrand Tube
```

DNA-Haskell이 갖는 데이터 기호는 <표 1>과 같고, 데이터 생성 규칙은 <표 2>와 같이 표현된다.

2.3 배낭 문제

배낭 문제(knapsack problem)는 조합 최적화 문제로서, 다항 시간(polynomial time)에 풀리지 않는 NP-hard 문제이다. 이 문제는 사전에 무게와 이익이 알려진 품목(물체)들의 집합이 주어질 때, 무게의 한계를 초과하지 않으면서 전체 이익이 최대가 되도록 배낭을 채우는 문제를 말한다[8][9]. 즉, n개의 품목과 하나의 배낭이 있을 때, 식(1)과 같이 이득 F(x)를 최대로 하는 품목을 선택하는 문제이다.(단, 식(2)의 조건을 만족해야 한다.)

$$F(x) = \sum_{i=1}^{n} p_i x_i \quad \text{식(1)}, \qquad \sum_{i=1}^{n} w_i x_i \leq W \quad \text{식(2)}$$

이때 $x=(x_1,\cdots,x_n)$이며, x_i는 0또는 1의 값을 갖는다. 또한 p_i는 품목 i의 이득이며, w_i는 품목 i의 무게, W는 배낭의 용량을 나타낸다.

x_i의 값에 0과 1사이의 소수를 허용하면 분할 가능 배낭 문제라고 하며, 0과 1만 허용하면 (0,1)-배낭 문제라고 부른다. (0,1)-배낭 문제에서는 품목을 통채로 배낭에 넣든지 아니면 버리든지를 결정해야 하며, 욕심쟁이법과 동적계획법 등으로 풀기 어려운 문제이다. 이를 풀기 위해서는 $\Omega(2^n)$의 시간이 요구된다.

3. 코드 최적화 DNA-Haskell

코드 최적화 DNA-Haskell은 DNA 컴퓨팅에 보완한 DNA-Haskell을 적용하여 배낭 문제를 풀었을때 발생되는 문제점을 해결하고자 하였다.

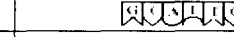
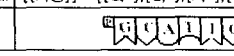
표 3. 코드 최적화 DNA-Haskell 알고리즘

```
1. Encoding
   - DNA-Haskell의 데이터 기호와 데이터 생성 규칙 적용
2. Initialization
3. Tube Synthesis
   1) Synthesis
      - 각 n개-튜브에 template와 상보적 template 삽입
   2) Union
      - n개-튜브 -> 1개의 튜브
   3) Annealing
4. Separation
   1) Cut
      - 제한 효소들로 DNA 서열 절단
   2) Labeling
      - DNA 서열 -P(Dephosphorylating)로 표지화
   3) CuttingOut
      i) Target DNA 추출하여 각각에 tube에 삽입
      ii) (TargetT_i(w_i) == max)
          TargetT_i(max)+P(phosphorylating)로 표지화
      iii) i)과 ii)를 Union -> TargetU
   4) Ligation
      - {TargetU(w_i)<W , TargetU(p_i)<P}
        -> TargetR을 추출
5. Decoding(최종해 발견)
```

코드 최적화 DNA-Haskell은 <표 3>과 같이 Encoding(표현 방법 결정), Initialization(초기화), Tube Synthesis(합성), Separation(분리), Decoding(최종해 발견)의 다섯 과정으로 구성된다.

먼저 첫 번째 과정인 Encoding은 DNA 서열을 이용하여 주어진 문제에 대해 표현하는 과정이다. 그림 1은 DNA 가닥에 대해 코드 최적화 DNA-Haskell의 표현 방법을 이용하여 unlabeling과 labeling을 표현한 예이다.

그림 1. 코드 최적화 DNA-Haskell을 이용한 표현 예

DNA strand unlabeled	5' - GCATTC - 3'
코드 최적화 DNA-Haskell	[[@,@]]++[[G,*],[C,*],[A,*],[T,*],[T,*],[C,*]]++[[@,@]]
예	
DNA strand labeled	5' P- GCATTC - 3' B
코드 최적화 DNA-Haskell	[[P,@]]++[[G,*],[C,*],[A,*],[T,*],[T,*],[C,*]]++[[B,@]]
예	

위와 같은 방법으로 Encoding 과정이 끝나면 두 번째 과정인 Initialization을 수행하여 적합한 DNA 서열들을 생성한다.

세 번째 과정은 생성된 서열을 Tube에 넣고 혼합하는 Tube Synthesis을 수행한다. Tube Synthesis는 Synthesis, Union, Annealing으로 구성된다. 먼저 Synthesis는 생성된 DNA 가닥의 문자열을 Tube에 넣는 [Char]->Tube의 구문을 갖는다. 따라서 생성된 DNA 가닥은 template와 상보적 template로 구분되며, n개-Tube안에 각각 삽입되어진다. 그 다음 Union은 n개-Tube를 1개의 Tube에 혼합하는 과정으로 Tube-> Tube-> Tube의 구문을 갖는다. 마지막으로 Annealing은 단일 가닥의 DNA 서열을 정해진 수만큼 이중 가닥으로 결합하는 과정으로 Tube->Int->Tube의 구문을 갖다.

네 번째 과정은 결합된 DNA 가닥들을 분리하는 Separation을 수행한다. 이 과정은 Cut, Labeling, CuttingOut, Ligation으로 구성된다. 먼저 Cut는 제한 효소를 이용하여 DNA 가닥을 특정 위치에서 잘라내는 것으로 Tube->[Char]-> Tube의 구문을 갖는다. 다음으로 잘라진 DNA 가닥에서 P(Phosphorylating)을 제거하는 Labeling으로 Tube->[Char]->Int->Tube의 구문을 갖는다. Labeling은 +/-P(Phosphorylating/ Dephosphorylating), +/-B(Biotinylating/Debiotinylating), +/-C(set Cy5-label/remove Cy5-label)의 세부 연산들을 포함하고 있다. Labeling이 끝난 후 Tube->Int->Tube의 구문을 갖는 CuttingOut을 수행한다. CuttingOut은 세가지 세부 연산으로 구분된다. 먼저 Labeling 된 DNA 가닥들 중에서 필요한 가닥을 추출하여 Tube에 삽입한다. 다음으로 삽입된 각각의 Tube 중 길이가 가장 큰 DNA 가닥을 선택하여 P를 분리하는 Labeling을 해 준다. 마지막으로 앞에서 수행된 Tubes를 Union해서 TargetU를 생성한다. Separation의 마지막 처리 과정인 Ligation은 TargetU에 들어 있는 DNA 가닥 중 길이가 배낭의 용량(W)을 초과하지 않으며, 이득(P)에 대하여 최대가 되는 TargetR을 추출한다.

앞의 네가지 과정을 거쳐서 추출된 DNA 가닥 TargetR은 마지막으로 겔 전기 영동법을 이용하여 결과를 확인하고 Decoding 한다. 제안한 알고리즘의 다섯가지 과정을 거친 DNA 가닥들은 배낭 문제에 해를 만족하는 결과를 얻기에 충분하다.

4. 실험 및 분석

제안한 알고리즘의 성능을 확인하기 위해 <표 4>의 배낭 문제를 기존의 DNA-Haskell 알고리즘과 비교 평가하였다. 단, 배낭의 용량(W)은 1104로 조건을 설정하고 만족하도록 하였다.

표 4. 배낭 문제의 상태값

번호(n)	이득(pi)	무게(wi)
1	50	670
2	78	434
3	34	254

시뮬레이션은 P-Ⅳ 2.4GHz, RAM 512MB의 PC에서 실험하였다. 그리고 실험실에서 DNA 실험은 pBluescript SK plasmid를 이용하였다. pBluescript SK plasmid는 총 2961bp로 구성되어 있으며, 이를 절단하기 위한 제한 효소는 Hae Ⅲ과 Xho Ⅰ을 이용하였다.

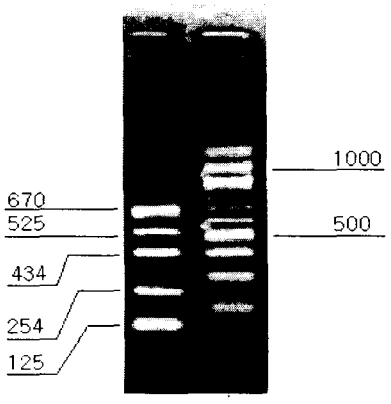


그림 2. pBluescript SK plasmid를 제한효소로 절단한 결과

그림 2의 왼쪽 수치는 pBluescript SK plasmid를 제한효소 HaeⅢ과 Xho Ⅰ로 DNA 가닥을 절단한 결과를 추정한 것이며, 오른쪽의 100bp ladder이다. 제한 효소 HaeⅢ는 pBluescript SK plasmid를 이미 곳에서 절단되었고, 제한 효소 Xho Ⅰ은 길이가 670bp 한 곳에서만 절단되었다.

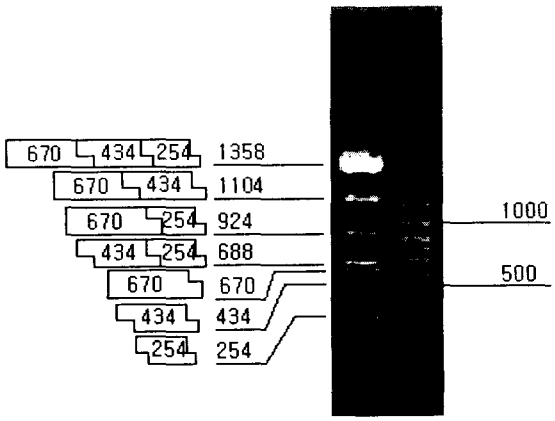


그림 3 Lagation 결과

그림 3는 제한효소에 의해 분리된 DNA 가닥들 중 길이가 각각 670bp, 434bp, 254bp인 가닥들을 대상으로 ligation 하였다. 그리고 Target DNA 가닥에 이득을 붙여 배낭의 용량에 만족하는 크기를 측정하였다. 따라서 실제 DNA 실험과 시뮬레이션은 동일한 결과 즉, 배낭의 용량 1104bp를 초과하지 않고, 이득이 큰 DNA 가닥을 얻을 수 있었다. 그리고 그 과정에서 이득에 대한 시간복잡도를 같이 평가하였다.

표 5. 시간복잡도 비교

operation	DNA-Haskell	코드 최적화 DNA-Haskell	생물학적 DNA 실험
Synthesis	O(1)	O(1)	DNA 길이에 따라 다름
Annealing	O(1)	O(1)	3.8H
Union	O(1)	O(1)	0.3H
Labeling	O(1)	O(1)	2H
Cut	O(1)	O(1)	3.1H
CuttingOut	O(1)	O(1)	3H
Ligation	O(1)	O(1)	12H
Profit	$O(log_2 n)$	O(1)	4H

실험에서 얻어진 시간복잡도의 결과는 <표 5>와 같다. 표에서 보는 것과 같이 연산자에서는 제안한 알고리즘과 DNA-Haskell이 동일한 시간복잡도를 얻을 수 있었다. 하지만 이득(Profit)에 대한 시간복잡도는 제안한 알고리즘이 DNA-Haskell 보다 우수하였다.

5. 결과

본 연구에서는 DNA-Haskell을 적용한 DNA 컴퓨팅으로 배낭 문제를 풀었을때 발생되는 문제점을 분석하고, 이를 해결하기 위해 코드 최적화 DNA-Haskell을 제안하였다. 코드 최적화 DNA-Haskell은 기존의 DNA-Haskell에서 처리하지 않은 배낭의 이득을 표현함으로써 집단의 크기와 상관없이 동일한 결과를 얻을 수 있었다. 또한 어떠한 조건을 주어도 배낭 문제의 용량을 정확히 만족하면서 이득을 최대로 발생시킬 수 있다. 그리고 빠른 시간내에 우수한 해(품목)를 찾을 수 있었다.

6. 참고문헌

[1] L.M. Adleman. "Molecular computation of solutions to combinatorial problems", Science, vol. 266, pp. 1021-1024, 1994.

[2] R.S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, and L.M. Adleman, "Solution of a 20-variable 3-SAT problem on a DNA computer", Science, vol. 296, pp. 499-502, April 2002.

[3] J.D. Watson, M. Gliman, J. Wikowski, M. Zoller, Recombinant DNA, 2nd Ed., Scientific American Books, New York, 1992.

[4] G.H. Gonnet, C. Korostensky, S.A. Benner, "Evaluation Measures of Multiple Sequence Alignments", Journal of Computational Biology, vol. 7, no. 1-2, pp. 261-276, 2000.

[5] R. Deaton, S. A. Karl, "Introduction to DNA Computing", 1999 Genetic and Evolutionary Computation Conference Tutorial Program, pp. 75-93, Orlando, Florida, July 14, 1999.

[6] P. Hudak, "The Haskell School of Expression: Learning Functional Programming through Multimedia", Cambridge University Press, New York, 2000.

[7] E.P. Stoschek, M. Sturm, and T. Hinze. "DNA-Computing - ein funktionales Modell im laborpraktischen Experiment", Informatik Forschung und Entwicklung, vol. 16, no. 1, pp. 35 - 52, 2001.

[8] 진강규, "유전알고리즘과 그 응용", 교우사, pp. 282-289, 2000.

[9] S. Khuri, T. Bäck, J. Heitkötter, "The Zero/one Multiple Knapsack Problem and Genetic Algorithms", Proc. '94 ACM Symp. on Applied Computing, Phoenix, AZ, 1994.