Journal of Informatics and Mathematical Sciences
Vol. 10, Nos. 1 & 2, pp. 217–225, 2018
ISSN 0975-5748 (online); 0974-875X (print)
Published by RGN Publications
http://dx.doi.org/10.26713/jims.v10i1-2.611

http://www.rgnpublications.com

Research Article

Speeding Up the Partial Digest Algorithm

Hazem M. Bahig[1,2,*] and Mostafa M. Abbas[3]

[1] *College of Computer Science and Engineering, Hail University, Hail, Kingdom of Saudi Arabia*
[2] *Computer Science Division, Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt*
[3] *Qatar Computing Research institute, Hamad Bin Khalifa University, Doha, Qatar*
***Corresponding author:** h.bahig@uoh.edu.sa

Abstract. We consider the partial digest problem, which aims to find the set $X = \{x_0, x_1, \ldots, x_n\}$ such that $\Delta X = \{|x_j – x_i|, 0 \leq i < j \leq n\}$ is equal to the input of the problem which is a multiset $D = \{d_1, d_2, \ldots, d_m\}$. In bioinformatics, the lengths of DNA fragments represents the multiset D, while the set of restriction site locations represents the set $X = \{x_0, x_1, \ldots, x_n\}$. In this paper, we study experimentally the effect of increasing and decreasing the number of levels on the breadth-breadth algorithm which is the best practical algorithm for the partial digest problem. The experimental study shows that the running time of breadth-breadth method is not the minimum time. Also, we obtained the number of levels that is used in the breadth-breadth algorithm to reduce the running time.

Keywords. Partial digest problem; Exact algorithms; DNA

MSC. 92D20; 68Q25

Received: March 13, 2017 **Accepted:** July 24, 2017

1. Introduction

Deoxyribonucleic acid (DNA) is a molecule that encodes the genetic instructions used in the functioning of all known living organisms and many viruses. Most DNA molecules are double-stranded helices. The relation between these strands is strong. An essential characteristic of a DNA molecule is its linear structure, which can be viewed as a sequence of four nucleotide bases: *adenine* (A), *cytosine* (C), *guanine* (G) and *thymine* (T). Three procedures are used to determine

the linear structure of DNA. These procedures are mapping, assembling, and sequencing. In the mapping, there are many methods of map construction. One of these methods is based on exposing specific chemicals (called restriction enzymes) on DNA. These enzymes cut DNA molecules at particular patterns of nucleotides called restriction sites [3, 8, 14, 16, 17].

In 1970, Hamilton has discovered the first the restriction enzyme during studying how the bacterium *Haemophilus influenza* takes up DNA from the virus. The restriction enzyme used in the process is named HindII. This restriction enzyme recognizes and cuts DNA at two sequences GTGCAC and GTTAAC [10].

In practice, several variants of cutting approaches are used. The most common approaches are the *partial digest* (PD) and the *double digest* (DD). In the partial digest, the DNA is cut by one enzyme but with different reaction times, while in double digest we use two restriction enzymes to cut the DNA [10]. But the problem in these approaches is the information about the location of the restriction sites lost during the cutting process. So, the main goal of our problem is how to reconstruct the location of restriction sites?

In this paper, we addressed the PD problem. The mathematical definition of PD problem is as follows: Given a multiset $D = \{d_1, d_2, \ldots, d_m\}$, find the set X represents the set of restriction sites locations, i.e. $X = \{x_1, x_2, \ldots, x_n\}$, such that $\Delta X = \{|x_j - x + i|, 0 \leq i < j \leq n\} = D$, where the set D represents the multiset of lengths of DNA fragments.

Two directions have been proposed to find the solution of PD problem. The goal of the first direction is to find an exact solution for PD problem. The main challenge of this direction is the running time for each algorithm exponential time. The goal of the second direction is to find an approximate/heuristic solution for PD/DD problems. The main challenge of this direction is the output of all proposed algorithms is not always true. In this paper, we will select the first direction.

Different algorithms are proposed to find the exact solution of PD problem [1, 2, 4–7, 9, 11–13, 15, 18]. Some of these algorithms are impractical algorithms, while the other algorithms are practical such as [1, 9, 18]. The first practical algorithm was designed by Skiena, Smith and Lemke [18] and ran in $n^2 \log n$ in the average case. In the worst case (Zhang instances [19]), the algorithm was run in order of 2^{n-1} time. Also, Fomin [9] introduced the second practical algorithm for PD. In the case of Zhang instances, Skiena, Smith and Lemke algorithm is faster than Formin's algorithm, while in some other case Formin's algorithm is faster than the Skiena, Smith and Lemke algorithm. The third practical algorithm was presented by Abbas and Bahig [1]. The algorithm was compared with the first practical algorithm and the results show that the proposed algorithm is more efficient than Skiena, Smith, and Lemke algorithm.

In this paper, we speed up the running time of the third practical algorithm. For this objective, we organized the paper as an introduction and four sections. In Section 2, we mention the best known practical algorithm for PD. In Section 3, we introduce the method that is speed up the running time of the best known practical algorithm. In Section 4, we present the final version of the modified algorithm. Finally, the conclusion of our work is in Section 5.

2. Best Known Exact Algorithm

Abbas and Bahig [1] introduced two exact sequential algorithms for PD problem. The first algorithm, BBb, is based on traverse the solution tree by using the breadth-first strategy from level 1 to n. The running time for the BBb algorithm is less than Skiena, Smith and Lemke algorithm. The main disadvantage of the BBb algorithm is the memory required to find the exact solution. The second algorithm, BBb2, is proposed to reduce the storage of the BBb algorithm without increasing the running time. The algorithm is based on two main stages. In the first stage, the algorithm traverses the solution tree α_M levels by using the breadth search strategy. We can construct the elements of each level by using the procedure *GenerateNextLevel*. The procedure takes the values of X and D at the level i as two lists, L_D and L_X, and generates the new values of X and D for the next level, $i+1$. A subroutine called $Find\_\alpha_M$ determines the value of α_M. Determining the value of α_M is based mainly on reducing the memory consumed. The second stage is traversing each element in the α_M level by using the breadth strategy. In both stages, the algorithm deletes all repeated elements at each level. The principle steps of the BBb2 algorithm are as follows.

Algorithm BBb2

Input: A multiset of integers, D and $|D| = N$.

Output: The solution set S.

Begin

1. $S = L_D = L_X = \emptyset$
2. $width = \text{Maximum}(D)$
3. $D = D - \{width\}$
4. $X = \{0, width\}$
5. $L_D = D \cup L_D$
6. $L_X = X \cup L_X$
7. $Find\_\alpha_M(N, \alpha_M)$
8. for $i = 0$ to $\alpha_M - 1$ do
9. $\quad GenerateNextLevel(L_D, L_X, S)$
10. end for
11. for each $eD \in L_D$ do
12. $\quad eL_D = eL_D \cup e_D$
13. $\quad eL_X = eL_X \cup e_X$
14. $\quad$ while $eL_D \neq \emptyset$ do
15. $\quad\quad GenerateNextLevel(eL_D, eL_X, S)$
16. $\quad$ end while
17. end for

End

---

3. Speeding up BBb2 Algorithm

In this section, we study the effect of increasing or decreasing the number of levels in the running time for the BBb2 algorithm. In more details, what is the best value of the number of levels that leads to minimize the running time of the BBb2 algorithm?

To achieve this goal we have two subsections. The first subsection is related to the proposed method, while the second subsection is relevant to the experimental study of the proposed method.

3.1 The Proposed Method

We answer the question that is introduced at the beginning of the section by using the following steps. In the first step, we fixed the value of n, say $n = 30$, and then apply the BBb2 algorithm in the case of the number of level $\alpha = 1$. The running time for this step is denoted by t_1. In the second step, we increase the number of the level by 1, $\alpha = 2$, and apply the BBb2 algorithm. The running time for this step is t_2. We repeat the previous steps until the value of $\alpha = n - 1$. If $\alpha = n$, the BBb2 algorithm is equivalent to BBb algorithm. After that, we find the minimum value of t_i's, $1 \le i \le n-1$, say $t_{\min}$. Finally, we compare between the value of α_M (calculated by the procedure $Find\_\alpha_M$) and min (calculated experimentally).

To summarize the previous steps, we first modify the BBb2 algorithm to be M-BBb2. The main difference between the two algorithms is the value of α. Also, the *BestTime* algorithm is used to determine the number of levels that leads to minimum time.

Algorithm M-BBb2
Input: A multiset of integers, D, $|D| = N$ and the number of level α.
Output: The solution set S.
Begin
1. $S = L_D = L_X = \emptyset$
2. $width = \text{Maximum}(D)$
3. $D = D - \{width\}$
4. $X = \{0, width\}$
5. $L_D = D \cup L_D$
6. $L_X = X \cup L_X$
7. for $i = 0$ to $\alpha - 1$ do
8. $\quad GenerateNextLevel(L_D, L_X, S)$
9. end for
10. for each $e_D \in L_D$ do
11. $\quad eL_D = eL_D \cup e_D$
12. $\quad eL_X = eL_X \cup e_X$
13. $\quad$ while $eL_D \neq \emptyset$ do
14. $\quad\quad GenerateNextLevel(eL_D, eL_X, S)$
15. $\quad$ end while
16. end for
End

Algorithm BestTime
Input: A multiset of integers, D, and $N = |D|$.
Output: Best level, bL, and calculated level α_M.
Begin
1. $n = \frac{1+\sqrt{1+8N}}{2}$
2. for $\alpha = 1$ to $n-1$ do
3. $D' = D$
4. M-BBb2(D', α, S)
5. Assign the time of execution for M-BB2 to t_α
6. end for
7. $Find\_\alpha_M(N, \alpha_M)$
8. $t_{\min} = t_1$
9. for $i = 2$ to $n-1$ do
10. if $t_i < t_{\min}$ then
11. $bL = i$
12. $t_{\min} = t_i$
13. end if
14. end for
15. return bL and α_M
End.

3.2 Experimental Study

We run the proposed method on a machine with speed processor 2.5 GHz. The proposed algorithm was implemented using C++ language. The data used in the proposed algorithm is based on Zhang instances. Figure 1(a-d) represents the results of executing M-BBb2 algorithm when $n = 30, 35, 40$ and 45. The running time of the BBb2 algorithm using α_M is represented as an unfilled circle, while the running time of the BBb2 algorithm using bL is represented as an unfilled triangle.

From the Figure 1, we observed the following comments.

1. The running time of the BBb2 algorithm that is based on the value of α_M is not the minimal. There are many values of α that lead to time less than the time of the BBb2 algorithm using α_M.
2. The value of bL is greater than the value of α_M.
3. The difference between bL and α_M increases with increase the value of n.
4. When we compare the running time of the BBb2 algorithm in the case of bL and α_M, we found that the percentage of improvement for $n = 30, 35, 40$ and 45 are 7%, 15%, 15% and 28%, respectively. This means that the percentage of improvement increases with increase the value of n.
5. The behavior of the curves is based on the Zhang instances.

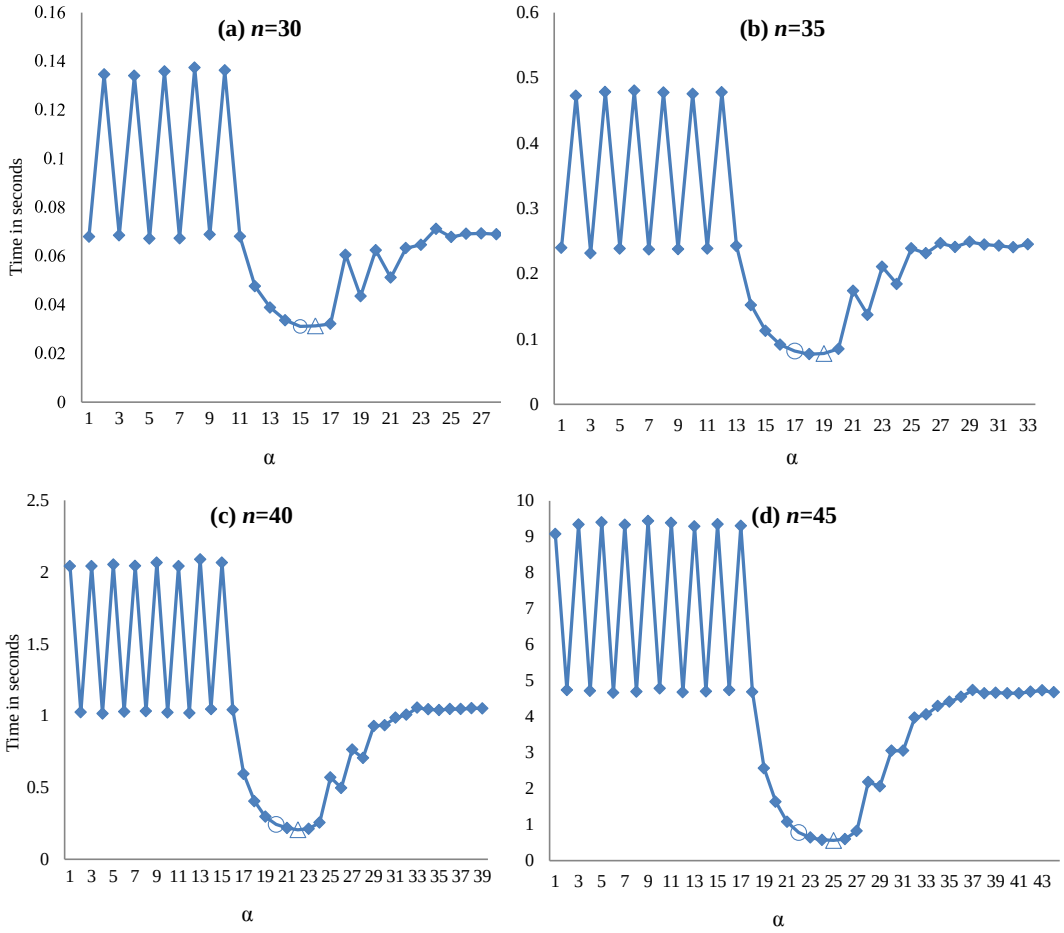

Figure 1. Running time of the BBb2 algorithm

4. Speeding up BBb2 Algorithm

From the experimental results that shown in the previous section, we can apply the BBb2 algorithm on a small range of the number of levels. The proposed range according to the experimental study for the BestTime algorithm is $[\alpha_M - 1, \alpha_M + 5]$. Therefore, we modified BestTime algorithm to be M-BestTime algoithm. We start the number of levels with $\alpha_M - 1$ to verify that the running time of the BBb2 algorithm decreases when we increase the number of levels.

Algorithm M-BestTime
Input: A multiset of integers, D, and $N = |D|$.
Output: Best level, bL, and calculated level α_M.
Begin
1. $n = \frac{1+\sqrt{1+8N}}{2}$
2. $Find\_\alpha_M(N, \alpha_M)$
3. $t_{\min} = \infty$.
4. $\alpha_{\min} = 0$.

5. for $\alpha = \alpha_M - 1$ to $\alpha_M + 5$ do
6. $D' = D$
7. M-BBb2(D', α, S)
8. Assign the time of M-BB2 to t_α
9. if $t_\alpha < t_{\min}$ then
10. $t_{\min} = t_\alpha$.
11. $bL = \alpha$.
12. else if $t_\alpha = t_{\min}$ and $\alpha_M = \alpha$ then
13. $bL = \alpha_M$
14. end if
15. end if
16. end for
17. return bL and α_M

End.

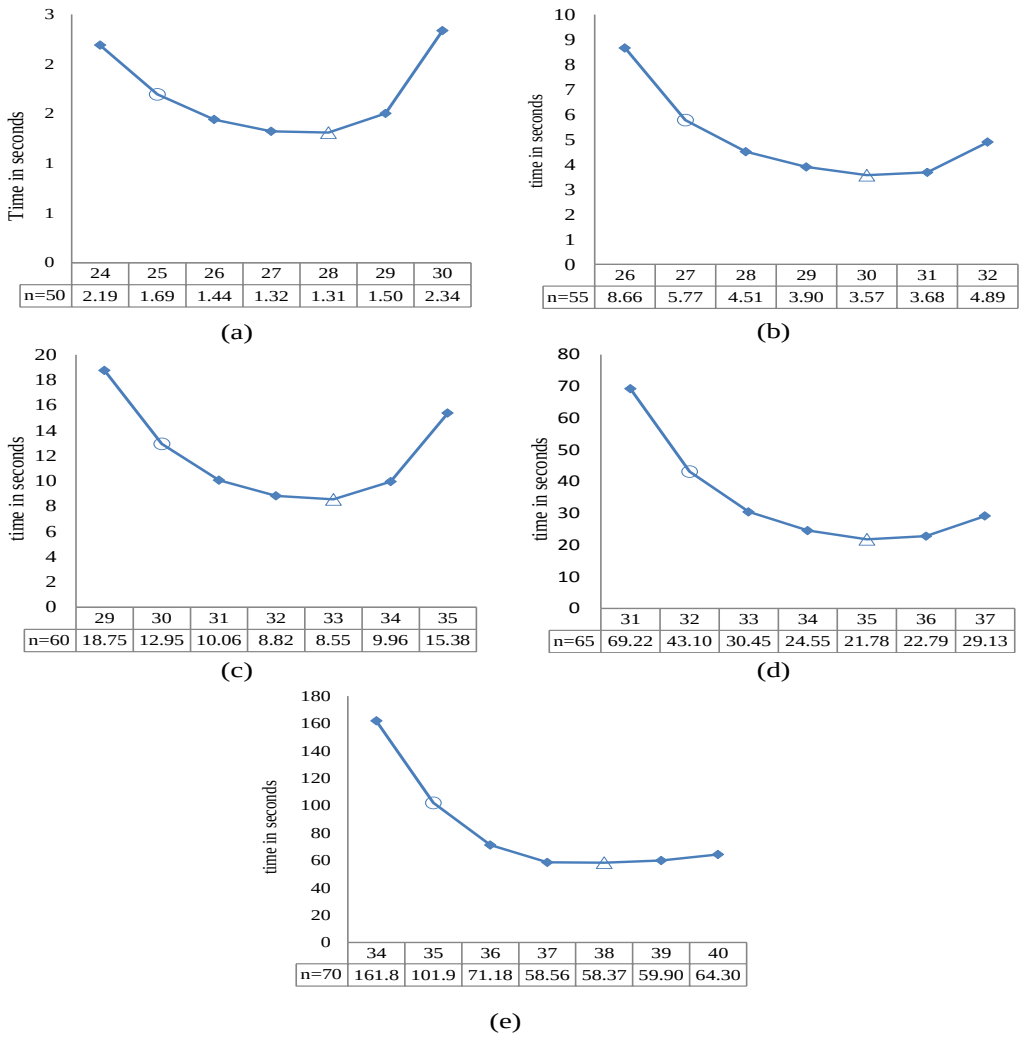

Figure 2. Running time of the M-BestTime algorithm

We are used the same platform to test the modified algorithm, *M-BestTime*. Figure 2 represents the results of applying *M-BestTime* on $n = 50, 55, 60, 65$ and 70. The first row of each figure represents the value of α, while the second row represents the running time of the algorithm at α. From the figure, we observed the following.

1. All comments in Section 3.2 are true.
2. The percentage of improvement increases with increase the value of n. For example, the percentage of improvement in the case of $n = 70$ is 42.7%.
3. The running time of the M-BB2 algorithm using bL is also less than the running time of BBb2 using α_M.
4. The values of bL that minimizes the running time of BBb2 algorithm for the studied cases are as follows.

Table 1. Comparison between α and bL

n	30	35	40	45	50	55	60	65	70
α_M	15	17	20	22	25	27	30	32	35
bL	10	19	22	25	28	30	33	35	38

5. Conclusion

In this paper, we addressed one of the important problems in bioinformatics which is partial digest problem. In this problem, given a multiset $D = \{d_1, d_2, \ldots, d_m\}$, find the set $X = \{x_1, x_2, \ldots, x_n\}$, such that $\Delta X = \{|x_j - x_i|, 0 \leq i < j \leq n\} = D$. We determine experimentally a new value for the number levels that can be used in the best practical algorithm, breadth-breadth. This new value will reduce the running time of the best practical known algorithm to solve PD. The percentage of improvements increases with increase the value of n. For data set used in this paper, the maximum number of improvement is 42% and we expect that this number will increase with increase n. In our study, the expected value for the number of levels is $\alpha_M + 3$ for $45 \leq n \leq 70$.

Acknowledgement

This research was supported by Research Deanship, Hail University, KSA, on grant # 0150358.

Competing Interests

The authors declare that they have no competing interests.

Authors' Contributions

All the authors contributed significantly in writing this article. The authors read and approved the final manuscript.

References

[1] M.M. Abbas and H.M. Bahig, A fast exact sequential algorithm for the partial digest problem, *BMC Bioinformatics* **17** (2016), 1365.

[2] H. Ahrabian, M. Ganjtabesh, A. Nowzari-Dalini and Z. Razaghi-Moghadam-Kashani, Genetic algorithm solution for partial digest problem, *International Journal of Bioinformatics Research and Applications* **9** (6) (2013), 584–594.

[3] M. Baker, Gene-editing nucleases, *Nature Methods* **9** (1) (2012), 23–26.

[4] J. Blazewicz, E.K. Burke, M. Kasprzak, A. Kovalev and M.Y. Kovalyov, Simplified partial digest problem: enumerative and dynamic programming algorithms, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **4** (4) (2007), 668–680.

[5] J. Błażewicz, P. Formanowicz, M. Kasprzak, M. Jaroszewski and W.T. Markiewicz, *Construction of DNA restriction maps based on a simplified experiment, Bioinformatics* **17** (5) (2001), 398–404.

[6] M. Cieliebak, S. Eidenbenz, P. Penna, *Noisy Data Make the Partial Digest Problem NP-Hard*, Springer, Berlin—Heidelberg (2003).

[7] A. Daurat, Y. Gŕard and M. Nivat, Some necessary clarifications about the chords' problem and the partial digest problem, *Theoretical Computer Science* **347** (1-2) (2005), 432–436.

[8] P.H. Dear, Genome Mapping, *eLS* (2001).

[9] E. Fomin, A simple approach to the reconstruction of a set of points from the multiset of n2 pairwise distances in n2 steps for the sequencing problem: II Algorithm, *Journal of Computational Biology* **23** (2016), 1–7.

[10] N.C. Jones and P. Pevzner, *An Introduction to Bioinformatics Algorithms*, MIT Press (2004).

[11] R.M. Karp and L.A. Newberg, An algorithm for analysing probed partial digestion experiments, *Computer Applications in the Biosciences* **11** (3) (1995), 229–235.

[12] P. Lemke and M. Werman, On the complexity of inverting the autocorrelation function of a finite integer sequence, and the problem of locating n points on a line, given the (nC2) unlabelled distances between them, *Preprint* 453 (1988).

[13] R. Nadimi, H.S. Fathabadi and M. Ganjtabesh, A fast algorithm for the partial digest problem, *Japan Journal of Industrial and Applied Mathematics* **28** (2) (2011), 315–325.

[14] P. Narayanan, *Bioinformatics: A Primer*, New Age International (2005).

[15] G. Pandurangan and H. Ramesh, The restriction mapping problem revisited, *Journal of Computer and System Sciences* **65** (3) (2002), 526–544.

[16] P. Pevzner, DNA physical mapping and alternating Eulerian cycles in colored graphs, *Algorithmica* **13** (1-2) (1995), 77–105.

[17] J. Sambrook, E.F. Fritsch and T. Maniatis, *Molecular Cloning: A Laboratory Manual*, 2nd edition, Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY (1989), 1.63–1.70.

[18] S.S. Skiena, W.D. Smith and P. Lemke, Reconstructing sets from interpoint distances, in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, ACM (1990), 332–339.

[19] Z. Zhang, An exponential example for a partial digest mapping algorithm, *Journal of Computational Biology* **1** (3) (1994), 235–239.