Mid-labeled Partial Digest Problem[1]

Hung-Lin Fu Yi-Ting Hsiao

Department of Applied Mathematics
National Chiao Tung University
Hsinchu, Taiwan 30050

Abstract

In DNA sequencing, the *partial digest problem (PDP)* plays an important role on reconstructing the locations of restriction sites in DNA. From combinatorial point of view, we can model PDP as follows. Let $X = \{x_1, x_2, ..., x_n\}$ be the set of restriction sites with $0 = x_1 < x_2 < ... < x_n$ be positive integers, and $\Delta X = \{x_j - x_i \mid 1 \leq i < j \leq n\}$ be the multi-set of distances between every two distinct restriction sites. Then, the PDP is to find the solutions X by knowing ΔX. Though the model looks simple, so far, the hardness of PDP is still unknown. In this paper, motivated by the approach of *end-labeled partial digest*, *labeled partial digest* and the idea of *probed partial digest*, we propose an idea called *mid-labeled partial digest*. As a consequence, if k mid-labels are added, then we obtain an algorithm with running time $O(n^{\frac{3}{2}} 2^{\frac{2n}{k+1}} \lg n)$ in the worst case.

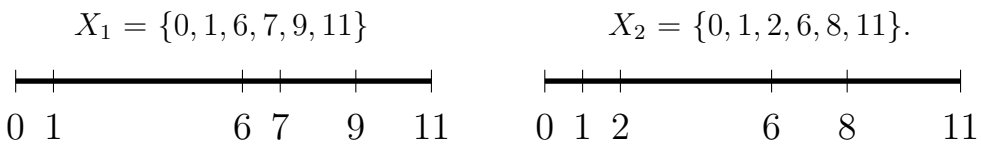
1 Introduction and Preliminaries

In computational molecular biology, our goal is to find the nucleic acid sequence on the DNA. But the DNA sequence is too long for us to read, the technique now can only rely on determining short DNA fragments. So, to cut DNA into pieces is required. The tool to cut DNA is "*restriction enzyme*", which has a strong specificity, always cut the same pattern on DNA. Different restriction enzymes cut different patterns. For example, EcoRI only recognizes GAATTC, and cuts this into two pieces G | AATTC. There may have such patterns in many places of DNA, we call these locations as *restriction sites*. Since restriction enzyme needs time to cut DNA, by controlling the reaction environment, we can have all the DNA fragments between every two restriction sites. Then, using a technique called *gel electrophoresis* to measure the lengths of them.

Suppose there are n restriction sites on the target DNA, a part of DNA that we are interested, then the *partial digest problem (PDP)* is to find the locations of these n restriction sites, by knowing the $\binom{n}{2}$ lengths between every two restriction sites. We give an example of the partial digest problem:

[1]Supported in part by NSC 100-2115-M-009-MY3.

Let $X = \{0, 1, 6, 7, 9, 11\}$ be the set of restriction sites. It is easy to find $\Delta X = \{1, 1, 2, 2, 3, 4, 5, 5, 6, 6, 7, 8, 9, 10, 11\}$, the set of distances between every two restriction sites. Then, partial digest problem is to find X, by knowing ΔX. Notice that the solution may not be unique, for the given ΔX above, there are two solutions with different structures (up to symmetry).



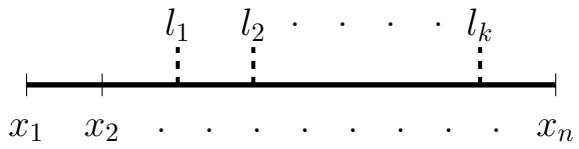
Skiena et al. [16] showed that the number of possible solutions is between $\frac{1}{2}n^{0.8107144}$ and $\frac{1}{2}n^{1.2324827}$. He also gave a *back-tracking algorithm* for finding the solutions, but the worst running time is exponential. Precisely, it is $O(n2^n \lg n)$, based on the restriction sites n. So far, the complexity of PDP is still unknown. For the error case of PDP, Cieliebak et al. proved it is NP-hard for measurement errors [3] and noisy data [4]. Skiena and Sundaram [15] analyzed back-tracking algorithm, by assuming that the location of sites are chosen under binomial distribution. Then the running time of the algorithm can be $O(n^3)$ with high probability. Furthermore, it can tolerate a relative error $O(\frac{1}{n^2})$ in the fragment lengths.

Partial digest problem has many variants: *End-labeled partial digest problem* is to add a label at one end of target DNA, it is not hard to find the solution, but to label just only one end of target DNA has some experimental difficulty. *Simplified partial digest problem (SPDP)* is a variant of PDP. Blazewicz and Kasprzak [2] had proved that SPDP is an NP-hard problem even if there are no errors, and Blazewicz et al. [1] gave an $O(n2^n)$ algorithm for SPDP by using dynamic programming. *Labeled partial digest problem (LPDP)* looks like "end-labeled PDP", but it labels both ends. Pandurangan and Ramesh [12] gave an algorithm for error-free case LPDP, which runs in $O(n^2 \lg n)$. They also handled the error case for LPDP in the same paper, gave an $O(n^4)$ algorithm that can tolerate an absolute error $O(\frac{\Delta}{n})$, where Δ is the minimum inter-site distance. *Probed partial digest problem (PPDP)* is to add a probe (label) somewhere inside target DNA. Karp and Newberg [9] gave an $O(iN^2)$ algorithm, where $N = |M|$, and the iteration number i can be up to 3^{N-2} theoretically. For the number of solutions, Newberg and Naor [11] gave a lower bound on PPDP.

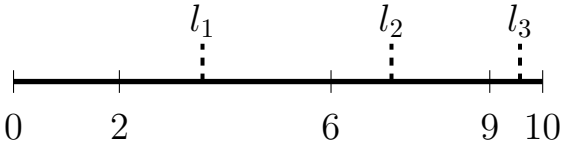
In this paper, we propose a different approach to study the PDP, namely *mid-label partial digest problem*. As a consequence, we obtain an algorithm which runs $O(n^{\frac{3}{2}}2^{\frac{2n}{k+1}} \lg n)$, where n is the number of restriction sites, k is the number of *isolated labels* in the target DNA.

2 Mid-labeled Partial Digest

Let $X = \{x_1, x_2, ..., x_n\}$ be the set of restriction sites, with $0 = x_1 < x_2 < ... < x_n$; $L = \{l_1, l_2, ..., l_k\}$, the k different *isolated labels* inside DNA, with $l_1 < l_2 < ... < l_k$; $\Delta X_\phi = \{x_j - x_i |$ if $l_v \notin (x_i, x_j), \forall 1 \leq v \leq k \}$ is the distances contain no labels; and $\Delta X_{l_s,l_t} = \{x_j - x_i | \ (x_i, x_j) \ni l_u, \forall s \leq u \leq t$ and $(x_i, x_j) \not\ni l_v$, if $v < s$ or $v > t\}$, where $1 \leq s \leq t \leq k$, the distances of DNA fragments that contain exactly $l_s, l_{s+1}, ..., l_t$ labels. Then, mid-labeled partial digest problem is to find the solution X, by knowing ΔX_ϕ, and all the labeled classes $\Delta X_{l_s,l_t}$.



We give an example for mid-labeled partial digest. Let $X = \{0, 2, 6, 9, 10\}$, and there are 3 labels $L = \{l_1, l_2, l_3\}$. l_1 is in (2,6), l_2 in (6,9), and l_3 in (9,10).



Then, we have $\Delta X_\phi = \{2\}$, $\Delta X_{l_1,l_1} = \{4, 6\}$, $\Delta X_{l_2,l_2} = \{3\}$, $\Delta X_{l_3,l_3} = \{1\}$, $\Delta X_{l_1,l_2} = \{7, 9\}$, $\Delta X_{l_2,l_3} = \{4\}$, $\Delta X_{l_1,l_3} = \{8, 10\}$.

Here we think of a partial digest with labeling k labels somewhere in the middle of target DNA, but do not know the actual locations. The labels can be radioisotope or a unique sequence in target DNA. In our algorithm, we assume that each label in L only labeled once in target DNA, and all the labels are in different base DNA fragments. Notice that if there are more than one labels in the same base fragment, we can combine them into one label, by ordering the labeled classes ΔX_A with $|A|$ labels. For convenience, these labels are called *isolated labels*. Moreover, these labels and labeled classes are sorted for use. The main idea of our algorithm is to locate the restriction sites between (l_{m_1-1}, l_{m_1}) and (l_{m_2-1}, l_{m_2}) first, and then the rest sites. The following notations are essential to our algorithm.

(a) For each $x \in X$, x provides two information $(place, right\_l)$. $x.place$ represents the location of restriction site, and $x.right\_l$ is the index of label nearest to its right.

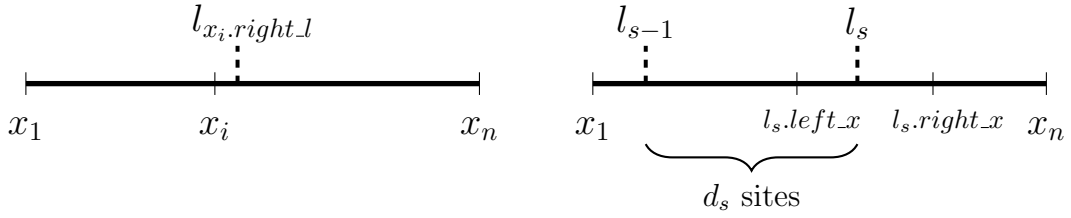
(b) Each label $l \in L$ contains two components $(left\_x, right\_x)$. $l.left\_x$ is the location of restriction site nearest to its left, and $l.right\_x$ is the location of restriction site nearest to its right.

(c) $\Delta X = (\bigcup_{1 \leq s \leq t \leq k} \{\Delta X_{l_s,l_t}\}) \cup \{\Delta X_\phi\}$.

(d) $D = \{d_1, d_2, ..., d_{k+1}\}$, d_i is the number of restriction sites between l_{i-1} and l_i.

(e) We define $\Delta(X, y) \subseteq \Delta X$, if $\forall x \in X$, the distance produced by x and y exists in the corresponding labeled class $\Delta X_{l_s,l_t}$.

For clearness, they are depicted in the following figure.



2.1 An Algorithm for Mid-labeled Partial Digest

Our algorithm contains four steps as follows, the flow chart is in Appendix I, and the pseudo code is in Appendix II.

(1) Initialization.

First, we use the number of DNA fragments in labeled classes $|\Delta X_{l_s,l_t}|$ to determine every d_i in D. By deleting the maximum length in $\Delta X_{l_1,l_k}$, i.e. the length of original DNA, we obtain the restriction sites of two end points.

(2) Find nearest sites of each label.

Since the minimum length in $\Delta X_{l_i,l_i}$ is the base fragment that only contains l_i, the maximum length in $\Delta X_{l_1,l_{i-1}}$ or $\Delta X_{l_{i+1},l_k}$ will connect the end point to this base fragment. Choose the maximum length in ΔX_ϕ instead when $k = 1$. Then we have found the nearest restriction sites of labels.

(3) Locate the sites in two intervals.

Find the minimum d_{m_1}, d_{m_2} from D, then locate the restriction sites in (l_{m_1-1}, l_{m_1}) and (l_{m_2-1}, l_{m_2}). This step is a recursive function. In each round, let the new end points be $l_{m_1-1}.right\_x$ and $l_{m_2}.left\_x$, find the maximum length in $\Delta X_{l_{m_1},l_{m_2-1}}$ and decide their locations. And the algorithm ends when we tried all the possible solutions on (l_{m_1-1}, l_{m_1}) and (l_{m_2-1}, l_{m_2}).

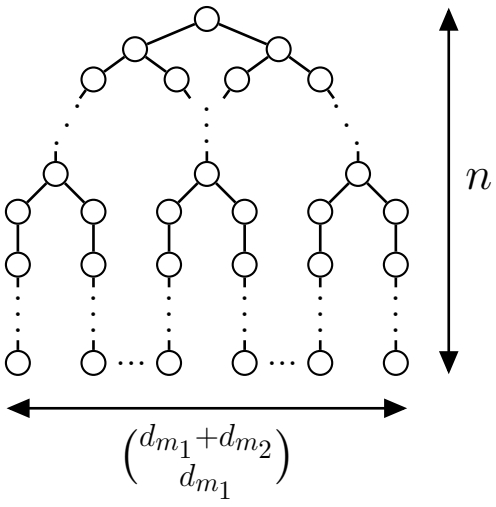
(4) Locate the rest sites.

Since the sites in (l_{m_1-1}, l_{m_1}) are all done, always find $max\Delta X_{l_i,l_{m_1-1}}$ for $i < m_1$, and $min\Delta X_{l_{m_1},l_j}$ for $j \geq m_1$ to decide the location of rest sites. Output the solution if the sites are all located. Move back to the previous step.

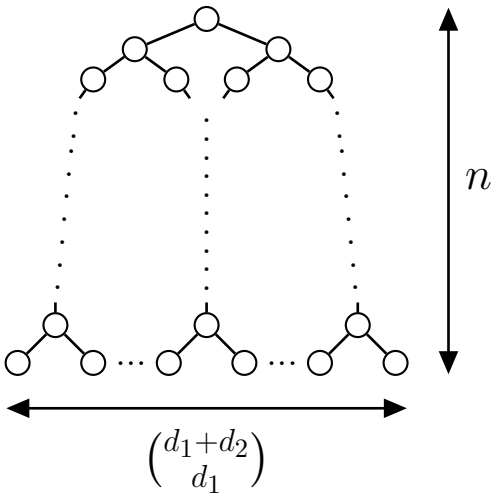
2.2 Running Time

Clearly, it will take $O(k)$ in initialization to find all the d_i's in D. Notice that to find the nearest sites of labels, we need to check the criterion "$\Delta(X, y) \subseteq \Delta X$". Since we leave an information $right\_l$, it uses $O(1)$ to find the correspond labeled class and $O(\lg n)$-time to find the length by binary search, it takes at most $O(n \lg n)$-time to check the criterion. Therefore, if there are k labels, then finding the nearest sites of labels takes $O(kn \lg n)$.

Now, for locating sites inside (l_{m_1-1}, l_{m_1}) and (l_{m_2-1}, l_{m_2}), we also need to check the criterion and thus it takes the same amount of time, i.e. $O(n \lg n)$-time to each site. Because we use d_{m_1}, d_{m_2} as the minimum sites, there are $\binom{d_{m_1}+d_{m_2}}{d_{m_1}}$ combinations in total. This implies that the time we need in locating sites inside two intervals and the rest sites can be thought as running a DFS algorithm on the binary tree with width $\binom{d_{m_1}+d_{m_2}}{d_{m_1}}$, height n, and every vertex on the tree costs $O(n \lg n)$.



Mid-labeled tree with $k > 1$.

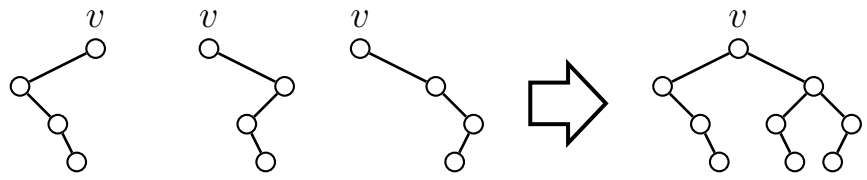


Mid-labeled tree with $k = 1$.

So, the conclusion follows by counting the number of vertices in the tree.

When $k > 1$, the tree contains at most $n\binom{d_{m_1}+d_{m_2}}{d_{m_1}}$ vertices. Since each vertex costs $O(n \lg n)$, the total cost is at most $O(n^2\binom{d_{m_1}+d_{m_2}}{d_{m_1}} \lg n)$. Notice that we choose the minimum d_{m_1}, d_{m_2} from D, $d_{m_1} + d_{m_2} \leq \frac{2n}{k+1}$. Since $\binom{n}{k}$ attends its maximum when $k = \frac{n}{2}$. The worst running time is at most $O(n^2\binom{2m}{m} \lg n)$, where $m = \frac{n}{k+1}$.

When $k = 1$, for given integers l and r, we define the tree $T_{l,r}$ which has height $l + r$, and it contains all the paths start from the root vertex that each path has exactly l left children and r right children. We give an example for $T_{1,2}$, which has 9 vertices.



The number of vertices in $T_{l,r}$ is $\sum_{\substack{0\le i\le l\\ 0\le j\le r}} \binom{i+j}{j}$, each element $\binom{i+j}{j}$ represents the number of paths start from the root vertex that contain exactly i left children and j right children. And the number of these paths is equal to the number of vertices. By using the equation: $\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$, the number of vertices in $T_{l,r}$ is:

$$|T_{l,r}| \;=\; \sum_{\substack{0\le i\le l\\ 0\le j\le r}} \binom{i+j}{j} = \binom{l+r+2}{l+1} - 1.$$

The binary trees for $k = 1$, because there are two possibilities T_{d_1,d_2} and T_{d_2,d_1}, which has at most $|T_{d_1,d_2}| + |T_{d_2,d_1}| = 2\binom{d_1+d_2+2}{d_1+1} - 2$ vertices. Similarly, for $k = 1$, the total running time is at most $O(n\binom{2m}{m}\lg n)$, where $m = \frac{n+2}{2}$.

By applying Stirling's approximation: $n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$, we can rewrite the total worst running time as:

$$\begin{cases} O(\sqrt{n}2^n \lg n) & \text{if } k = 1, \text{ and} \\ O(n^{\frac{3}{2}} 2^{\frac{2n}{k+1}} \lg n) & \text{if } k > 1. \end{cases}$$

3 Conclusion

In this paper, we have provided an idea of adding mid-labels to tackle the partial digest problem. This is a possible way to do it in DNA sequencing due to the development of biological technology. Our algorithm may not be as efficient as a polynomial-time algorithm, but for n is not too large, a small number of k labels will reduce the running time to a reasonable range.

For future study, to determine the hardness of the original partial digest problem is still the most important one to pursuit. From the experience of the study, we do believe that PDP is an NP-hard problem.

References

[1] 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 Bioinformatic*, vol. 4, no. 4, pp. 668-680, 2007.

[2] J. Blazewicz and M. Kasprzak, "Combinatorial Optimization in DNA Mapping - A Computational Thread of the Simplified Partial Digest Problem," *RAIRO Operations Research*, vol. 39, no. 4, pp. 227-241, 2005.

[3] M. Cieliebak and S. Eidenbenz, "Measurement Errors Make the Partial Digest Problem NP-Hard," *Latin American Theoretical INformatics*, vol. 2976, pp. 379-390, 2004.

[4] M. Cieliebak, S. Eidenbenz and P. Penna, "Noisy Data Make the Partial Digest Problem NP-hard," *WABI*, vol. 2812, pp. 111-123, 2003.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[6] T. Dakic, On the Turnpike Problem, *PhD thesis*, Simon Fraser University, 2000.

[7] L. Goldstein and M. S. Waterman, "Mapping DNA by Stochastic Relaxation," *Advances in Applied Mathematics*, vol. 8, no. 2, pp. 194-207, 1987.

[8] N. C. Jones and P. A. Pevzner, *An Introduction to Bioinformatics Algorithms*, The MIT Press, 2004.

[9] R. M. Karp and L. A. Newberg, "An Algorithm for Analysing Probed Partial Digestion Eexperiments," *Bioinformatics/computer Applications in The Biosciences*, vol. 11, no. 3, pp. 229-235, 1995.

[10] B. Lewin, *Genes VII*, 7th ed. Oxford University Press, 1999.

[11] L. Newberg and D. Naor, "A Lower Bound on the Number of Solutions to the Probed Partial Digest Problem," *Advances in Applied Mathematics*, vol. 14, no. 2, pp. 172-183, 1993.

[12] G. Pandurangan and H. Ramesh, "The Restriction Mapping Problem Revisited," *Journal of Computer and System Sciences*, vol. 65, no. 3, pp. 526-544, 2002.

[13] J. Rosenblatt and P. D. Seymour, "The Structure of Homometric Sets," *SIAM Journal on Algebraic and Discrete Methods*, vol. 3, no. 3, 1982.

[14] J. Sambrook, E. F. Fritsch and T. Maniatis, *Molecular Cloning: A Laboratory Manual*, 2nd ed. Cold Spring Harbor Laboratory, 1989.

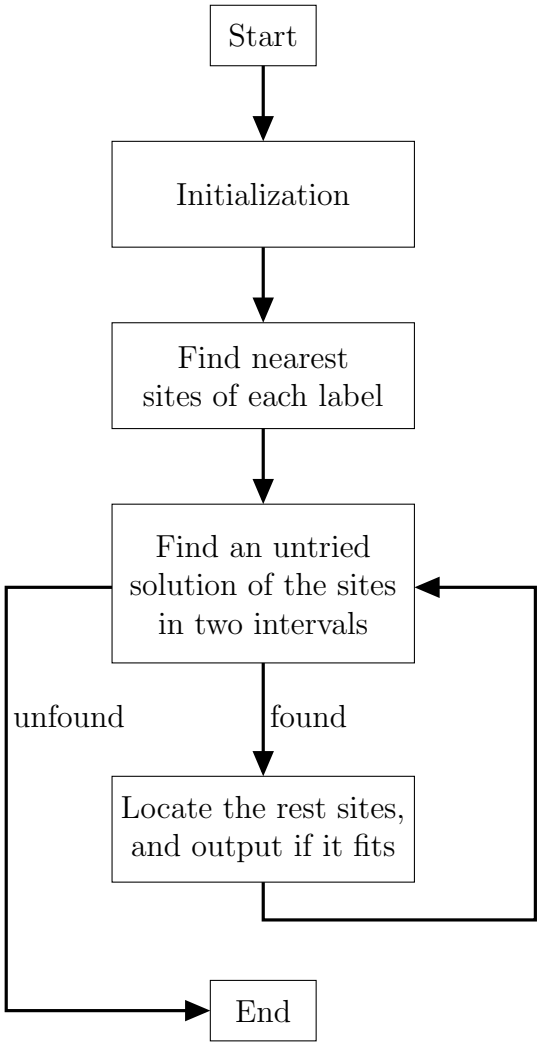
[15] S. S. Skiena and G. Sunderam, "A Partial Digest Approach to Restriction Site Mapping," *Bulletin of Mathematical Biology*, vol. 56, no. 2, pp. 275-294, 1994.

[16] S. S. Skiena, W. D. Smith and P. Lemke, "Reconstructing Sets from Interpoint Distances," In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pp. 332-339, 1990.

[17] D. B. West, *Introduction to Graph Theory*, 2nd ed. Pearson, 2000.

[18] Z. Zhang, "An Exponential Example for a Partial Digest Mapping Algorithm," *Journal of Computational Biology*, vol. 1, no. 3, pp. 235-239, 1994.

Appendix I


Start
Initialization
Find nearest sites of each label
Find an untried solution of the sites in two intervals
unfound
found
Locate the rest sites, and output if it fits
End

Appendix II

Algorithm 1 Mid-Labeled Partial Digest

Input: ΔX
Output: X

if k =1 **then** # Initialization.
  $d_1 = \frac{n+\sqrt{n^2+4|\Delta X_{l_1,l_1}|}}{2}, \quad d_2 = \frac{n-\sqrt{n^2-4|\Delta X_{l_1,l_1}|}}{2}$
else
  $d_1 = \sqrt{\frac{|\Delta X_{l_1,l_1}||\Delta X_{l_1,l_2}|}{|\Delta X_{l_2,l_2}|}}$
  for $s = 2$ to $k+1$ **do**
    $d_s = \frac{|\Delta X_{l_{s-1},l_{s-1}}|}{d_{s-1}}$
  end for
end if
$T = max\Delta X_{l_1,l_k}, \quad X = \{(0,1),(T,k+1)\}$
Delete T from $\Delta X_{l_1,l_k}$
$d_1 = d_1 - 1, \quad d_{k+1} = d_{k+1} - 1$
$l_0 = (0,0)$
$l_{k+1} = (T,T)$
if $k = 1$ **then** # Find nearest sites of labels.
  $M = max\Delta X_{\phi}, \quad m = min\Delta X_{l_1,l_1}$
  $l_1 = (T - M - m, T - M)$
else
  $M = max\Delta X_{l_2,l_k}, \quad m = min\Delta X_{l_1,l_1}$
  $l_1 = (T - M - m, T - M)$
  for $s = 2$ to k **do**
    $M = max\Delta X_{l_1,l_{s-1}}, \quad m = min\Delta X_{l_s,l_s}$
    $l_s = (M, M + m)$
  end for
end if
for $s = 1$ to k **do**
  $y = (l_s.left\_x, s)$
  if $y \notin X$ and $!(ADD\_SITE(X, y, \Delta X, L, D))$ **then**
    return
  end if
  $y = (l_s.right\_x, s + 1)$
  if $y \notin X$ and $!(ADD\_SITE(X, y, \Delta X, L, D))$ **then**
    return
  end if
end for
Choose the minimum d_{m_1}, d_{m_2} from D, where $m_1 < m_2$.
if $k = 1$ **then** # Find solutions X.
  $PLACE(X, \Delta X, m_1, m_2, L, D)$
  $SWAP(d_1, d_2)$
  $PLACE(X, \Delta X, m_1, m_2, L, D)$
else
  $PLACE(X, \Delta X, m_1, m_2, L, D)$
end if

Algorithm 2 $PLACE(X, \Delta X, m_1, m_2, L, D)$

if $\Delta X_{l_{m_1}, l_{m_2-1}} = \phi$ **then**
  $PLACE\_REST(X, \Delta X, m_1, m_2, L, D)$
  return
else
  $M = max\Delta X_{l_{m_1}, l_{m_2-1}}$
  $x_1 = l_{m_2-1}.right\_x, \quad x_2 = l_{m_2}.left\_x$
  $y = (l_{m_1-1}.right\_x + M, m_2)$
  if $(x_1 < y.place < x_2)$ and $(ADD\_SITE(X, y, \Delta X, L, D))$ **then**
    $PLACE(X, \Delta X, m_1, m_2, L, D)$
    $DELETE\_SITE(X, y, \Delta X, L, D)$
  end if
  $x_1 = l_{m_1-1}.right\_x, \quad x_2 = l_{m_1}.left\_x$
  $y = (l_{m_2}.left\_x - M, m_1)$
  if $(x_1 < y.place < x_2)$ and $(ADD\_SITE(X, y, \Delta X, L, D))$ **then**
    $PLACE(X, \Delta X, m_1, m_2, L, D)$
    $DELETE\_SITE(X, y, \Delta X, L, D)$
  end if
end if
return

Algorithm 3 $PLACE\_REST(X, \Delta X, m_1, m_2, L, D)$

for $s = 1$ to $m_1 - 1$ **do**
  while $\Delta X_{l_s, l_{m_1-1}} \neq \phi$ **do**
    $M = max\Delta X_{l_s, l_{m_1-1}}$
    $x_1 = l_{s-1}.right\_x, \quad x_2 = l_s.left\_x$
    $y = (l_{m_1}.left\_x - M, s)$
    if $!(x_1 < y.place < x_2)$ or $!(ADD\_SITE(X, y, \Delta X, L, D))$ **then**
      $RECOVER(X, \Delta X, m_1, m_2, L, D)$
      return
    end if
  end while
end for
for $s = m_1 + 1$ to $k + 1$ **do**
  while $\Delta X_{l_{m_1}, l_{s-1}} \neq \phi$ **do**
    $m = min\Delta X_{l_{m_1}, l_{s-1}}$
    $x_1 = l_{s-1}.right\_x$, $x_2 = l_s.left\_x$
    $y = (l_{m_1}.left\_x + m, s)$
    if $!(x_1 < y.place < x_2)$ or $!(ADD\_SITE(X, y, \Delta X, L, D))$ **then**
      $RECOVER(X, \Delta X, m_1, m_2, L, D)$
      return
    end if
  end while
end for
Output X
$RECOVER(X, \Delta X, m_1, m_2, L, D)$
return

Algorithm 4 $ADD\_SITE(X, y, \Delta X, L, D)$

1: **if** $d_s > 0$ and $\Delta(X, y) \subseteq \Delta X$ **then**
2: Delete $\Delta(X, y)$ from ΔX, and add y to X
3: $d_s = d_s - 1$
4: return TRUE
5: **end if**
6: return FALSE

Algorithm 5 $DELETE\_SITE(X, y, \Delta X, L, D)$

1: Delete y from X, and add $\Delta(X, y)$ to ΔX
2: $d_s = d_s + 1$

Algorithm 6 $RECOVER(X, \Delta X, m_1, m_2, L, D)$

1: **for** $x \in X$ and ($x.right\_l \neq m_1$ or m_2) **do**
2: $DELETE\_SITE(X, x, \Delta X, L, D)$
3: **end for**