

國立臺灣大學電機資訊學院資訊工程學系

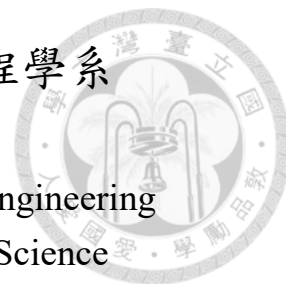
碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis



在置換生成圖中尋找最小的 k 距支配集

Finding a Minimum Distance- k Dominating Set on
Permutation Graphs

呂哲宇

Che-Yu Lu

指導教授: 陳健輝 博士

林清池 博士

Advisor: Gen-Huey Chen, Ph.D.

Ching-Chi Lin, Ph.D.

中華民國 112 年 7 月

July, 2023

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

在置換生成圖中尋找最小的 k 距支配集

Finding a Minimum Distance- k Dominating Set on
Permutation Graphs

本論文係呂哲宇君（學號 R10922119）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 112 年 7 月 27 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 27 July 2023 have examined a Master's thesis entitled above presented by LU CHE YU (student ID: R10922119) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

陳健輝

(指導教授 Advisor)

薛文蔚

林清山

傅榮勝

系主任/所長 Director:

洪士瀨

國立臺灣大學碩士學位論文

口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

在置換生成圖中尋找最小的 k 距支配集

Finding a Minimum Distance- k Dominating Set on
Permutation Graphs

本論文係呂哲宇君（學號 R10922119）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 112 年 7 月 27 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 27 July 2023 have examined a Master's thesis entitled above presented by LU CHE YU (student ID: R10922119) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

陳健輝

(指導教授 Advisor)

張貴雲

系主任/所長 Director:

洪士瀨



Acknowledgements

首先我要感謝鄭容濤學弟想出演算法的原型，讓我可以以這套原型為基礎，並以前人的論文為輔助誕生這篇論文。接下來我要感謝林清池教授，在每周的會議中，他讓我了解將論文簡單明瞭的寫出來是一件困難的事，也透過聽眾的角度，輔導我將論文寫得更好理解。舉例來說，我的演算法有一部份原本是難以閱讀的，清池老師幫助我將演算法其中兩行順序調換，整理完之後就發現閱讀性好非常多，且與前面連貫起來。像這樣將艱澀難懂的文字簡化的能力，是這兩年來我最大的收穫。最後要感謝陳健輝教授，他教導我報告時要以圖片代替文字，輔以口述，能讓聽眾更容易理解我想表達的內容。兩位教授讓我在兩年的碩士生涯學到很多，萬分感謝。



摘要

給定一個置換 π 代表置換生成圖 G ，我們用 $O(n \log n)$ 時間與 $O(n)$ 空間的演算法找到最小 k 距支配集。我們先找到一個動態規劃的規則，再參考 Farber 與 Keil [12] 在同類型的圖上對於支配集 (即 $k = 1$ 的情況) 的動態規劃演算法，將兩者合併後變成 $O(n^2)$ 時間演算法，最後引入 AVL 樹，改良成 $O(n \log n)$ 時間演算法。

關鍵字： k 距支配集；置換生成圖；動態規劃；AVL 樹



Abstract

Given a permutation π which denotes a permutation graph G . We use an $O(n \log n)$ time and $O(n)$ space algorithm to find a minimum distance- k dominating set. We first find a dynamic programming rule, then we combine it with another dynamic programming algorithm given by Ferber and Keil [12], which is used to find minimum dominating set (the case of $k = 1$) on permutation graphs. So the $O(n^2)$ time algorithm is created. Finally, we use AVL tree to reduce our time complexity to $O(n \log n)$.

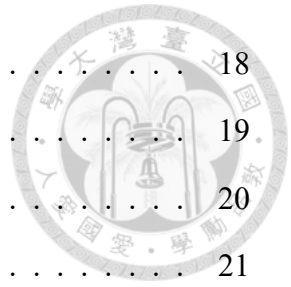
Keywords: distance- k dominating set; permutation graph; dynamic programming; AVL tree



Contents

	Page
Acknowledgements	i
摘要	ii
Abstract	iii
Contents	iv
Chapter 1 Introduction	1
1.1 Minimum Distance- k Dominating Sets	1
1.2 Permutation Graphs	1
1.3 Definitions and Notations	2
Chapter 2 Previous Work	3
2.1 Minimum Distance- k Dominating Set on Permutation Graphs	3
2.2 Related Work	3
Chapter 3 A Dynamic Programming Algorithm	5
3.1 A Data Structure	5
3.1.1 Definitions and Properties	5
3.1.2 An Algorithm to Find Trapezoids	6
3.1.3 Correctness and Complexity Analysis	8
3.2 A New Distance- k Dominating Set	11
3.2.1 Definitions and Algorithms	11
3.2.2 Correctness	12
Chapter 4 An Improved Algorithm	16
4.1 Improvement by a New Distance- k Dominating Set	16
4.1.1 Some Changes	16

4.1.2	Correctness and Complexity Analysis	18
4.2	A New Algorithm	19
4.2.1	Some Changes	20
4.2.2	Correctness and Complexity Analysis	21
4.3	Improvement by AVL Trees	24
4.3.1	One Bottleneck	25
4.3.2	Another Bottleneck	25
4.3.3	Correctness	28
Chapter 5	Conclusion	29
	References	30





Chapter 1 Introduction

In this chapter, we will define "minimum distance- k dominating set" on a graph. Finding one of such set is NP-hard [13] in general graphs. We will then introduce a special graphs named "permutation graphs". It has some good performances s.t. some hard problems become polynomial-time solvable on it. The problem we want to solve belongs them. Finally we will propose some notations which will be used often in this thesis.

1.1 Minimum Distance- k Dominating Sets

Given a graph $G = (V, E)$ and an integer $k > 0$, a vertex subset D of V is a distance- k dominating set if for each vertex $j \in V$, there is at least one vertex $i \in D$ s.t. the distance between i and $j \leq k$. Note that we define the distance from any vertex to itself to be 0. The minimum distance- k dominating set is to find such D with minimum cardinality.

1.2 Permutation Graphs

A graph $G = (V, E)$ is a permutation graph if there exists a permutation π of $\{1, 2, \dots, |V|\}$ such that $(i, j) \in E \Leftrightarrow (i - j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0$, where $\pi^{-1}(i)$ is the position of i in π .

In this thesis, we use "permutation diagram" (Figure 1.1) to denote it. In the diagram, we have two horizontal lines called top channel and bottom channel. The numbers $1, 2, \dots, |V|$ are drawn on top channel from left to right; the numbers $\pi(1), \pi(2), \dots, \pi(|V|)$ are drawn on bottom channel from left to right. Each vertex i becomes a line with both top and bottom channel = i . We say its position on top channel is i , and on bottom channel is $\pi^{-1}(i)$. Two vertices are adjacent \Leftrightarrow their corresponding lines intersect with each other.

To determine whether a given graph is a permutation graph and to find the defining permutation π if it is, Spinrad [21] give an algorithm in $O(|V|^2)$ time in 1983, and it is improved to $O(|V| + |E|)$ time by McConnell and Spinrad [18] in 1999. So we assume

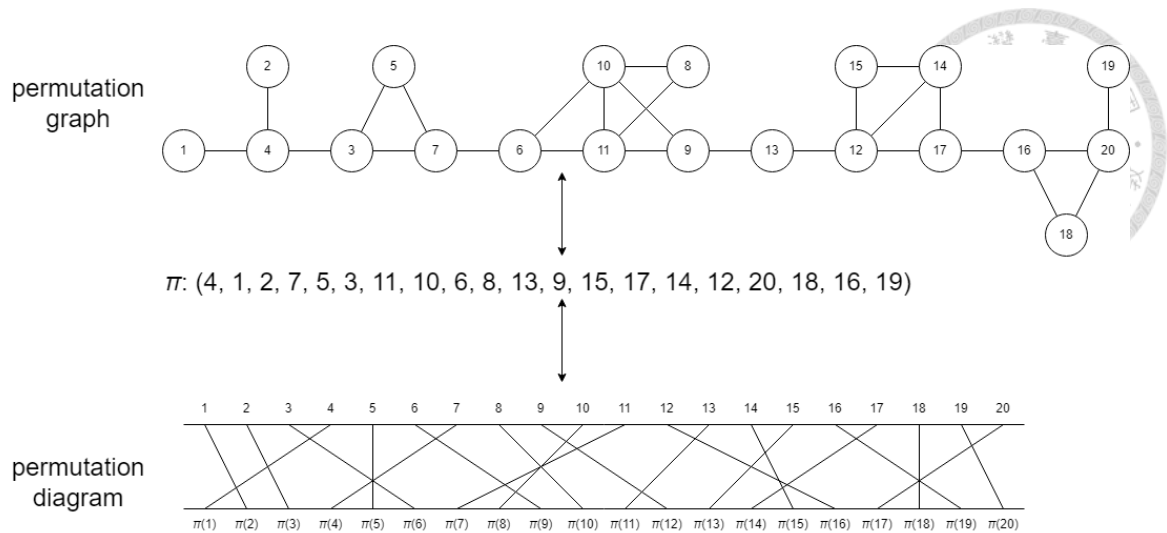


Figure 1.1: A permutation graph with permutation π and its corresponding permutation diagram.

the input is π in this thesis.

1.3 Definitions and Notations

We first define some notations used in this thesis. We use n to denote $|V|$, m to denote $|E|$. Then we use i to denote a line in permutation diagram whose position on top channel is i , on bottom channel is $\pi^{-1}(i)$, and j has similar denotation. Then we use $d(i, j)$ to denote the distance between i and j . If $(i - j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0$, then $d(i, j) = 1$, and we say the two lines i and j "intersect" with each other in permutation diagram. Finally we use t to denote a position on top channel, b to denote a position on bottom channel.



Chapter 2 Previous Work

In this chapter, we will propose some previous work.

2.1 Minimum Distance- k Dominating Set on Permutation Graphs

In this section, we will propose the previous work to solve our problem. Both $k = 1$ case and $k > 1$ case will be told.

For domination problem (which is the case of $k = 1$) on permutation graphs, Ferber and Keil [12] gave an $O(n^2)$ time dynamic programming based algorithm. Then Tsai and Hsu [22] improved it into $O(n \log \log n)$ time using a tree structure. Finally Chao, Hsu, and Lee [8] used linked list, with some adjustment on Ferber and Keil's algorithm, to construct an $O(n)$ time algorithm. Thus the optimal time is found.

For $k > 1$ case, Chang, Ho, and Ko [7] gave an $O(nm^2)$ time algorithm on AT-free graphs. Since permutation graphs is a subclass of AT-free graphs, we can use this algorithm to solve the problem. Then, Rana, Pal, and Pal [19] gave an $O(n^2)$ time algorithm on permutation graphs.

In this thesis, we create an $O(n \log n)$ time and $O(n)$ space algorithm. We first find a dynamic programming rule to solve the problem, then we combine it with Ferber and Keil's algorithm [12] to create an $O(n^2)$ algorithm. Finally, we use a data structure called AVL tree [1] to reduce our time complexity to $O(n \log n)$.

2.2 Related Work

In this section, we will propose other previous work related but not the same as our problem. We will first propose some different problem on permutation graphs, then propose domination problem on different graphs.

Weighted domination problem on permutation graphs is solved in $O(n^3)$ time by Ferber and Keil [12] in 1985, $O(n(m+n))$ time by Liang, *et al.* [17] in 1991, $O(n^2 \log^2 n)$ time by Tsai and Hsu [22] in 1993, and $O(m+n)$ time by Rhee, *et al.* [20] in 1996.

Independent domination problem on permutation graphs is solved in $O(n^3)$ time by Ferber and Keil [12] in 1985, and $O(n \log^2 n)$ time by Atallah, *et al.* [4] in 1988.

Weighted independent domination problem on permutation graphs is solved in $O(n^3)$ time by Ferber and Keil [12] in 1985, $O(n^2)$ time by Brandstädt and Kratsch [6] in 1987, and $O(n \log n)$ by Atallah and Kosaraju [3] in 1989.

Connected domination problem on permutation graphs is solved in $O(n^2)$ time by Colbourn and Stewart [11] in 1990, $O(m+n)$ time by Arvind and Rangan [2] in 1992, and $O(n)$ time by Ibarra and Zheng [15] in 1994.

Weighted connected domination problem on permutation graphs is solved in $O(n^3)$ time by Colbourn and Stewart [11] in 1990, and $O(m+n \log n)$ time by Arvind and Rangan [2] in 1992.

Paired domination problem on permutation graphs is solved in $O(mn)$ time by Cheng, *et al.* [9] in 2009, and $O(n)$ time by Lappas, *et al.* [16] in 2013.

On the other hand, for domination problem:

The problem on circular-arc graphs is solved in $O(mn)$ time by Bonuccelli [5] in 1985, and $O(n)$ time by Hsu and Tsai [14] in 1991.

The problem on trees is solved in $O(n)$ time by Cockayne, *et al.* [10] in 1975.



Chapter 3 A Dynamic Programming Algorithm

In this chapter, we will propose a dynamic programming rule we find. We can simply use this rule to get the solution. First, we will construct a data structure named "trapezoid" to simplify our problem. Then, we will introduce a distance- k dominating set. We find the dynamic programming rule of it, and thus we can use it to find the solution.

3.1 A Data Structure

In this section, we find that given fixed k , for each i , there is a trapezoid, denoted by $Tz_k[i]$, s.t. $d(i, j) \leq k \Leftrightarrow Tz_k[i]$ "intersects" with line j , the word "intersect" between trapezoid and line will be formally defined below. We will propose some important properties for trapezoids, then we will propose how to find $Tz_k[i]$ for all i in $O(n \log k)$ time and $O(n)$ space. The problem thus reduce to find a minimum set $D \subseteq V$ s.t. for each line j , there is at least one line $i \in D$ s.t. $Tz_k[i]$ intersects with j . At last we will prove the theorems and prove the correctness and complexity of algorithms mentioned in this section.

3.1.1 Definitions and Properties

We now formally define the trapezoid $Tz_k[i]$ (Figure 3.1). It has two parallel lines, one is located on the top channel and the other is located on the bottom channel of the permutation diagram. It has four corners, two of them are at the top channel and two of them are at the bottom channel. We define $TR_k[i]$ to be its top right corner's position on the top channel; $BR_k[i]$ to be its bottom right corner's position on the bottom channel. $TL_k[i]$ and $BL_k[i]$ are defined similarly.

Then, we say the trapezoid $Tz_k[i]$ intersects with some line j if either $TL_k[i] \leq j \leq TR_k[i]$ or $BL_k[i] \leq \pi^{-1}(j) \leq BR_k[i]$, and $d(i, j) \leq k \Leftrightarrow Tz_k[i]$ intersects with line j .

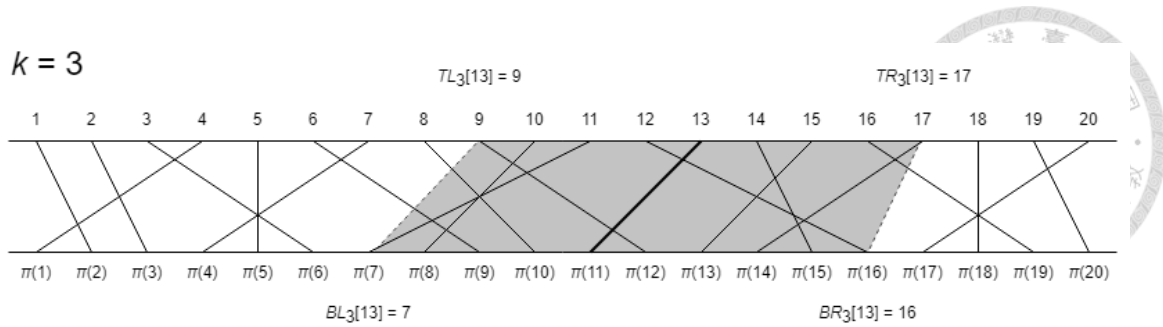


Figure 3.1: A trapezoid $Tz_3[13]$.

We then propose some properties of trapezoids we find. We find that if $k \nmid 2$, then $TL_k[i]$ and $TR_k[i]$ monotone increase as i increase. Else if $k \mid 2$, then $BL_k[i]$ and $BR_k[i]$ monotone increase as i increase (Figure 3.2). There are similar properties as $\pi^{-1}(i)$ increase. We use these properties often in the following sections. We formally write it below.

$k = 2$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$TL_2[i], TR_2[i]$	1, 4	2, 4	3, 7	1, 4	3, 7	6, 11	3, 7	8, 11	9, 13	6, 11	6, 11	12, 17	9, 13	12, 17	12, 15	16, 20	12, 17	16, 20	19, 20	16, 20
$BL_2[i], BR_2[i]$	1, 2	1, 3	1, 6	1, 6	4, 6	4, 9	4, 9	7, 10	7, 12	7, 12	7, 12	11, 16	11, 16	13, 16	13, 16	14, 19	14, 19	17, 19	17, 20	17, 20

(a)

$k = 3$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$TL_3[i], TR_3[i]$	1, 4	1, 4	1, 7	1, 7	3, 7	3, 11	3, 11	6, 11	6, 13	6, 13	6, 13	9, 17	9, 17	12, 17	12, 17	12, 20	12, 20	16, 20	16, 20	16, 20
$BL_3[i], BR_3[i]$	1, 6	1, 6	1, 9	1, 6	1, 9	4, 12	1, 9	7, 12	7, 16	4, 12	4, 12	11, 19	7, 16	11, 19	11, 16	14, 20	11, 19	14, 20	17, 20	14, 20

(b)

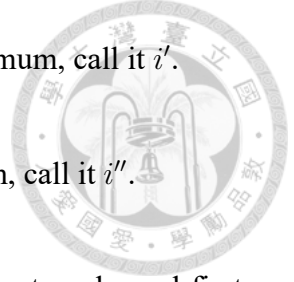
Figure 3.2: Properties of trapezoids: (a) the case of $k = 2$, (b) the case of $k = 3$.

Theorem 3.1.1. For fixed $k > 1$, for $i \leq j$, if $k \nmid 2$, then $TL_k[i] \leq TL_k[j]$ and $TR_k[i] \leq TR_k[j]$, else, then $BL_k[i] \leq BL_k[j]$ and $BR_k[i] \leq BR_k[j]$; For $\pi^{-1}(i) \leq \pi^{-1}(j)$, if $k \nmid 2$, then $BL_k[i] \leq BL_k[j]$ and $BR_k[i] \leq BR_k[j]$, else, then $TL_k[i] \leq TL_k[j]$ and $TR_k[i] \leq TR_k[j]$.

3.1.2 An Algorithm to Find Trapezoids

We first use a simple example to propose the algorithm. Let $k = 3$, given an arbitrary line i , we want to find the trapezoid $Tz_3[i]$. The method to find its top right corner's position $TR_3[i]$ is:

- (1) Collect the lines intersecting with i and i itself.



- (2) Pick a line in (1), whose position on bottom channel is maximum, call it i' .
- (3) Collect the lines intersecting with i' and i' itself.
- (4) Pick a line in (3), whose position on top channel is maximum, call it i'' .
- (5) $TR_3[i] = i''$.

To find $BR_3[i]$, we do the similar things. But find maximum on top channel first, then find maximum on bottom channel. Notice that if the final line is i''' , then its position on bottom channel is $\pi^{-1}(i''')$. So $BR_3[i] = \pi^{-1}(i''')$.

To find $TL_3[i]$ and $BL_3[i]$, it is similar as above, but replace "maximum" by "minimum".

We find that if $k = 3$, we repeat the similar steps twice. Actually we repeat the steps for arbitrary k by first on top channel, second on bottom, third on top, ..., until $k - 1$ times to find a corner. Then first on bottom channel, second on top, third on bottom, ..., until $k - 1$ times to find another corner. Then replace "maximum" by "minimum" to find remain two corners.

Now we formally propose the method. First, given an integer x , given a line i , we have a definition below.

Definition 3.1.2. $tr_x[i] = j$ with maximum j s.t. $d(i, j) \leq x$; $br_x[i] = j$ with maximum $\pi^{-1}(j)$ s.t. $d(i, j) \leq x$. $tl_x[i]$ and $bl_x[i]$ are similar, but replace "max" by "min".

If $x = 1$, we simply write $tr[i]$, $br[i]$, $tl[i]$, and $bl[i]$.

The definition leads to the following lemma, which can be used often for proof of theorems:

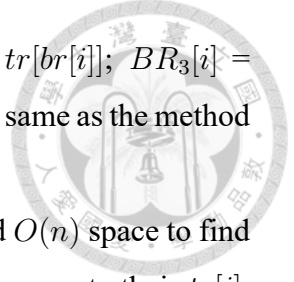
Lemma 3.1.3. If $d(i, j) \leq x$, then $tl_x[i] \leq j \leq tr_x[i]$ and $\pi^{-1}(bl_x[i]) \leq \pi^{-1}(j) \leq \pi^{-1}(br_x[i])$.

Proof. Directly by the definition. □

Then we have two theorems below, which propose that the method is correct.

Theorem 3.1.4. For arbitrary $x > 1$, $tr_x[i] = tr[br_{x-1}[i]]$; $br_x[i] = br[tr_{x-1}[i]]$; $tl_x[i] = tl[bl_{x-1}[i]]$; $bl_x[i] = bl[tl_{x-1}[i]]$.

Theorem 3.1.5. For fixed $k > 1$, $TR_k[i] = tr_{k-1}[i]$; $BR_k[i] = \pi^{-1}(br_{k-1}[i])$; $TL_k[i] = tl_{k-1}[i]$; $BL_k[i] = \pi^{-1}(bl_{k-1}[i])$.



By theorems, for $k = 3$ case, for each i , $TR_3[i] = tr_2[i] = tr[br[i]]$; $BR_3[i] = \pi^{-1}(br_2[i]) = \pi^{-1}(br[tr[i]])$; and similar for $TL_3[i]$ and $BL_3[i]$. This is same as the method we use at the beginning of the section.

The Algorithm 1 and Algorithm 2 below use $O(n \log k)$ time and $O(n)$ space to find trapezoids $Tz_k[i]$ for all lines i . In Algorithms, for every lines i , we compute their $tr[i]$, $br[i]$, $tl[i]$, $bl[i]$ first, then we use these terms, combine with double-and-add method, to find $x = k - 1$ terms, then done.

Algorithm 1 Find $tr[i]$ for each i . The algorithm to find $br[i]$, $tl[i]$, and $bl[i]$ are similar.

Input: permutation π

Output: $tr[i]$, $i = 1$ to n

$t \leftarrow n$, $b \leftarrow n$

while $t \neq 0$ **do**

while $b \geq \pi^{-1}(t)$ **do**

$tr[\pi(b)] \leftarrow t$

$b \leftarrow b - 1$

while $t \neq 0$ and $tr[t]$ has been valued **do**

$t \leftarrow t - 1$

3.1.3 Correctness and Complexity Analysis

In this section, we will prove the theorems and the correctness of algorithms above. Then we will prove that our algorithms compute $Tz_k[i]$ in $O(n \log k)$ time and $O(n)$ space for all i .

1. Proof of Theorem 3.1.1:

Claim:

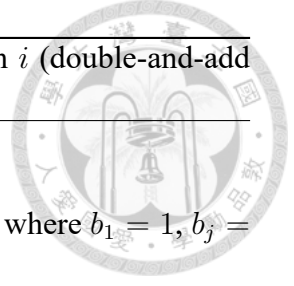
(1) If $i \leq j$, $\pi^{-1}(br[i]) \leq \pi^{-1}(br[j])$.

(2) If $\pi^{-1}(i) \leq \pi^{-1}(j)$, then $tr[i] \leq tr[j]$.

The tl , bl terms are same as tr , br terms.

Suppose the claims are correct, within the fact of Theorem 3.1.4 and 3.1.5, we can use them to prove each case of Theorem 3.1.1. For example if $i \leq j$, by (1) $BR_2[i] \leq BR_2[j]$; then by (2) $TR_3[i] \leq TR_3[j]$, and so on. The proof is similar if $\pi^{-1}(i) \leq \pi^{-1}(j)$.

We just prove (1) only, (2) are symmetry as (1). Suppose $i \leq j$, if $\pi^{-1}(br[i]) > \pi^{-1}(br[j])$, then since $\pi^{-1}(i) \leq \pi^{-1}(br[i])$, $i \geq br[i]$, so $br[i] \leq i \leq j$. And $\pi^{-1}(br[i]) > \pi^{-1}(br[j]) \geq \pi^{-1}(j)$, $br[i]$ intersects with j . This contradicts to the definition of $br[j]$.



Algorithm 2 Find $tr_{k-1}[i]$ and $br_{k-1}[i]$ given $tr[i]$ and $br[i]$ for each i (double-and-add method). The algorithm to find $tl_{k-1}[i]$ and $bl_{k-1}[i]$ are similar.

Input: integer k , $tr[i]$ and $br[i]$, $i = 1$ to n

Output: $tr_{k-1}[i] \leftarrow T[i]$ and $br_{k-1}[i] \leftarrow B[i]$, $i = 1$ to n

Use binary representation to represent $k - 1$, so $k - 1 = b_1b_2\dots b_y$, where $b_1 = 1$, $b_j =$ either 1 or 0 for $j > 1$.

```

for  $i = 1$  to  $n$  do
  // initialize
   $T[i] \leftarrow tr[i]$ ,  $B[i] \leftarrow br[i]$ 
for  $j = 2$  to  $y$  do
  // double
  if  $b_{j-1} = 1$  then
    for  $i = 1$  to  $n$  do
       $tmpT[i] \leftarrow T[B[i]]$ ,  $tmpB[i] \leftarrow B[T[i]]$ 
  else
    for  $i = 1$  to  $n$  do
       $tmpT[i] \leftarrow T[T[i]]$ ,  $tmpB[i] \leftarrow B[B[i]]$ 
  for  $i = 1$  to  $n$  do
     $T[i] \leftarrow tmpT[i]$ ,  $B[i] \leftarrow tmpB[i]$ 
  // add
  if  $b_j = 1$  then
    for  $i = 1$  to  $n$  do
       $tmpT[i] \leftarrow tr[B[i]]$ ,  $tmpB[i] \leftarrow br[T[i]]$ 
    for  $i = 1$  to  $n$  do
       $T[i] \leftarrow tmpT[i]$ ,  $B[i] \leftarrow tmpB[i]$ 

```

2. Proof of Theorem 3.1.4:

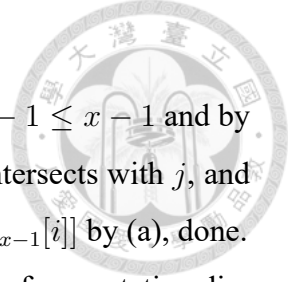
We just prove $tr_x[i] = tr[br_{x-1}[i]]$, the others have similar proof.

We prove it by checking if $tr[br_{x-1}[i]]$ meets all of the conditions of $tr_x[i]$ in Definition 3.1.2. $d(i, tr[br_{x-1}[i]]) \leq x$ since $d(i, br_{x-1}[i]) \leq x - 1$, so the remain proof is: For any arbitrary line j s.t. $d(i, j) \leq x$, j must $\leq tr[br_{x-1}[i]]$.

We claim if (a) $j > tr[br_{x-1}[i]]$, then by Lemma 3.1.3 $tr[br_{x-1}[i]] \geq br_{x-1}[i]$ and thus (b) $j > br_{x-1}[i]$. Then we split the relations of $\pi^{-1}(j)$ and $\pi^{-1}(br_{x-1}[i])$ into the following three cases. We can find that all cases lead to a contradiction. So the claim is false, the proof is finished.

(1) If $\pi^{-1}(j) < \pi^{-1}(br_{x-1}[i])$, then together with (b) j intersects with $br_{x-1}[i]$. But $j > tr[br_{x-1}[i]]$ by (a), this contradicts to the definition of tr .

(2) Else if $\pi^{-1}(j) > \pi^{-1}(br_{x-1}[i])$, let the shortest path from i to j be $i \rightarrow \dots \rightarrow j' \rightarrow j$, we claim that $j' > tr[br_{x-1}[i]]$ and j' intersects with $br_{x-1}[i]$, this contradicts to



the definition of tr .

Proof of claim: $\pi^{-1}(j') \leq \pi^{-1}(br_{x-1}[i])$ since $d(i, j') = d(i, j) - 1 \leq x - 1$ and by Lemma 3.1.3. And this leads to $\pi^{-1}(j') < \pi^{-1}(j)$. $j' > j$ since j' intersects with j , and $j' > br_{x-1}[i]$ by (b). So j' intersects with $br_{x-1}[i]$; But $j' > j > tr[br_{x-1}[i]]$ by (a), done.

(3) Else, but $j \neq br_{x-1}[i]$ by (b), this contradict to the definition of permutation diagram.

3. Proof of Theorem 3.1.5:

Claim: $d(i, j) \leq k \Leftrightarrow$ either $tl_{k-1}[i] \leq j \leq tr_{k-1}[i]$ or $\pi^{-1}(bl_{k-1}[i]) \leq \pi^{-1}(j) \leq \pi^{-1}(br_{k-1}[i])$. Then the four positions are the corners of $Tz_k[i]$ by definition of trapezoids.

We split the $d(i, j)$ into the following three cases and prove the claim above.

(1) If $d(i, j) \leq k - 1$, then the claim is correct by Lemma 3.1.3.

(2) Else if $d(i, j) = k$ ($k > 1$). Suppose $i < j$ and $\pi^{-1}(i) < \pi^{-1}(j)$, which means line j is on the right of line i , then $tl_{k-1}[i] \leq i < j$ and $\pi^{-1}(bl_{k-1}[i]) \leq \pi^{-1}(i) < \pi^{-1}(j)$. Let the shortest path from i to j be $i \rightarrow \dots \rightarrow j' \rightarrow j$, then either $j < j'$ or $\pi^{-1}(j) < \pi^{-1}(j')$. Both $j' \leq tr_{k-1}[i]$ and $\pi^{-1}(j') \leq \pi^{-1}(br_{k-1}[i])$ by Lemma 3.1.3 since $d(i, j') = k - 1$. Thus either $j < tr_{k-1}[i]$ or $\pi^{-1}(j) < \pi^{-1}(br_{k-1}[i])$. The proof is similar if line j is on the left of line i , so the claim is correct.

(3) Else, suppose line j is on the right of line i . If $i < j \leq tr_{k-1}[i]$, then j must intersect at least one line of the shortest path from i to $tr_{k-1}[i]$ since the path is continuous on the permutation diagram. Then $d(i, j)$ is at most k since the length of the path is $k - 1$, contradiction. Else if $\pi^{-1}(i) < \pi^{-1}(j) \leq \pi^{-1}(br_{k-1}[i])$, the proof similar to above leads to contradiction. So both $j > tr_{k-1}[i]$ and $\pi^{-1}(j) > \pi^{-1}(br_{k-1}[i])$. The proof is similar if line j is on the left of i .

4. Correctness of Algorithm 1:

(1) Let $t_1 \leftarrow n$, we prove that for each line i , if $\pi^{-1}(t_1) \leq \pi^{-1}(i)$, then $tr[i] \leftarrow t_1$: t_1 is maximum, so for each i , $i \leq t_1$. If $\pi^{-1}(t_1) \leq \pi^{-1}(i) \leq n$, then $d(i, t_1) \leq 1$, and again t_1 is maximum. So $tr[i] \leftarrow t_1$.

(2) Then let t_2 be the maximum number s.t. $tr[t_2]$ hasn't been valued. We prove that if $\pi^{-1}(t_2) \leq \pi^{-1}(j) < \pi^{-1}(t_1)$, then $tr[j] \leftarrow t_2$: $tr[j]$ hasn't been valued $\Leftrightarrow \pi^{-1}(j) < \pi^{-1}(t_1)$ (by (1)) and $j \leq t_2$. If $\pi^{-1}(t_2) \leq \pi^{-1}(j)$, then $d(j, t_2) \leq 1$. The remain to prove

is $d(j, i) > 1$ for $i > t_2$. This is because $j \leq t_2 < i$ and $\pi^{-1}(j) < \pi^{-1}(t_1) \leq \pi^{-1}(i)$.

(3) Repeat (2) until all $tr[i]$ are valued, then done.



5. Correctness of Algorithm 2:

By Theorem 3.1.4, for "double", if x is odd, then for each i , $tr_{2x}[i] = tr_x[br_x[i]]$, else then $tr_{2x}[i] = tr_x[tr_x[i]]$; for "add", for arbitrary x , $tr_{x+1}[i] = tr[br_x[i]]$. The other terms are similar, so the algorithm is correct.

6. Complexity Analysis:

For Algorithm 1, since both t and b runs from n to 0 once, the time complexity is $O(n)$; We input π , and save $tr[i]$ for each i , so the space complexity is $O(n)$. Since we run the Algorithm four times, the time and space complexity are same as above.

For Algorithm 2, $y = O(\log k)$, for each j we computes $tmpT[i]$, $tmpB[i]$, $T[i]$, and $B[i]$ once or twice for each i . So it takes $O(n \log k)$ time. We input $O(n)$ terms, save $tmpT[i]$, $tmpB[i]$, $T[i]$, and $B[i]$ for each i , so the space complexity are $O(n)$. Since we run the Algorithm twice, the time and space complexity are same as above.

So the two algorithms need $O(n \log k)$ time and $O(n)$ space in total.

3.2 A New Distance- k Dominating Set

In this section, given fixed $k > 1$, we will define a distance- k dominating set $D1_k(i)$. Where the "1" means one input "i". We find a dynamic programming rule of it s.t. we can easily find the solution on permutation graph. At last we will prove the theorems mentioned in this section.

3.2.1 Definitions and Algorithms

We first define $D1_k(i)$.

Definition 3.2.1. $D1_k(0) = \emptyset$, for $i = 1$ to n , $D1_k(i) \subseteq \{1, 2, \dots, i\}$ is a minimum distance- k dominating set including i which k -dominates $\{1, 2, \dots, i\}$.

Before we introduce the dynamic programming rule, we first define the *set_min*:

Definition 3.2.2. Given several sets S_1, S_2, \dots , *set_min*(S_1, S_2, \dots) output a set with minimum cardinality.

Now we have the following rules, note that they are not as trivial as it looks like.

Theorem 3.2.3. For $i = 1$ to n , $D1_k(i) = \{i\} \cup \text{set_min}_{0 \leq j < i}(D1_k(j) \mid \{i\} \cup D1_k(j))$ k -dominates $\{1, 2, \dots, i\}$.

Theorem 3.2.4. $\text{set_min}_{0 \leq i \leq n}(D1_k(i) \mid D1_k(i))$ k -dominates $\{1, 2, \dots, n\}$ is a minimum distance- k dominating set on given permutation graph.

By Theorem 3.2.3 and Theorem 3.2.4, we can easily construct algorithm 3 to find solution. But if we use this algorithm without any improvement, it takes $O(n^4)$ time since for each i, j , we need $O(n^2)$ time to check if $\{i\} \cup D1_k(j)$ k -dominates $\{1, 2, \dots, i\}$. So the next chapter we will propose how to improve the time complexity of this algorithm.

Algorithm 3 Find a minimum distance- k dominating set on given permutation graph.

Input: permutation π , an integer k

Output: a minimum distance- k dominating set on permutation graph denoted by π

if $k = 1$ **then**

 Use Chao, Hsu, and Lee's [8] $O(n)$ time algorithm to get the solution.

else

 Use algorithm 1 and 2 to find $Tz_k[i]$ for $i = 1$ to n .

 Initialize $D1_k(0) \leftarrow \emptyset$

for $i = 1$ to n **do**

$D1_k(i) \leftarrow \{i\} \cup \text{set_min}_{0 \leq j < i}(D1_k(j) \mid \{i\} \cup D1_k(j))$ k -dominates $\{1, 2, \dots, i\}$

 output $\text{set_min}_{0 \leq i \leq n}(D1_k(i) \mid D1_k(i))$ k -dominates $\{1, 2, \dots, n\}$

3.2.2 Correctness

In this section, we will prove two theorems above. Recall the definition of $D1_k(i)$, a set D can be a choice of $D1_k(i)$ if:

- (a) $D \subseteq \{1, 2, \dots, i\}$
- (b) $i \in D$
- (c) D k -dominates $\{1, 2, \dots, i\}$
- (d) $|D|$ is minimum

These will be used in the following proofs.

Now, we propose a lemma which is the core of proof of theorems.

Lemma 3.2.5. For each i , \exists a choice of $D1_k(i)$, say D , s.t. each two lines $\in D$ don't intersect with each other.



we set a condition (e) below:

(e) Each two lines $\in D$ don't intersect with each other.

We first use this lemma to prove theorems, then we prove the correctness of this lemma.

1. Proof of Theorem 3.2.3:

For $i = 1$ to n , we denote $Dlemma_k(i)$ to be a set described by Lemma 3.2.5, and denote $Dthm_k(i) = \{i\} \cup set_min_{0 \leq j < i} (D1_k(j) \mid \{i\} \cup D1_k(j) \text{ k-dominates } \{1, 2, \dots, i\})$. We want to prove that $Dthm_k(i)$ is a choice of $D1_k(i)$. First, $Dthm_k(i)$ must exist since $\{i\} \cup D1_k(i-1)$ meets (a), (b), (c) of $D1_k(i)$, and (d) can be found since i is finite. Then, $Dthm_k(i)$ meets (a) since $\{i\} \subseteq \{1, 2, \dots, i\}$ and $D1_k(j) \subseteq \{1, 2, \dots, j\} \subseteq \{1, 2, \dots, i\}$; $Dthm_k(i)$ meets (b) and (c) by definition. The remain to prove is (d). If $|Dthm_k(i)| \leq |Dlemma_k(i)|$, then since $|Dlemma_k(i)|$ is minimum, (d) is proved.

Let $Dlemma_k(i) = \{i_1, i_2, \dots, i_x, i\}$, where $i_1 < i_2 < \dots < i_x < i$ and $\pi^{-1}(i_1) < \pi^{-1}(i_2) < \dots < \pi^{-1}(i_x) < \pi^{-1}(i)$. We claim $Dlemma_k(i) \setminus \{i\} = \{i_1, i_2, \dots, i_x\}$ is a choice of $D1_k(i_x)$, then $|Dthm_k(i)| = |\{i\} \cup set_min_{0 \leq j < i} (D1_k(j) \mid \{i\} \cup D1_k(j) \text{ k-dominates } \{1, 2, \dots, i\})| \leq |\{i\} \cup D1_k(i_x)| = |Dlemma_k(i)|$, done.

To prove $Dlemma_k(i) \setminus \{i\}$ is a choice of $D1_k(i_x)$: (a), (b) are met immediately.

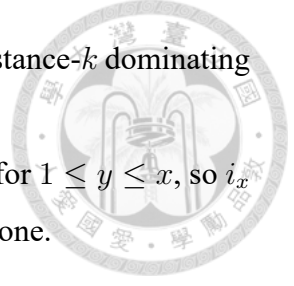
To prove (c), we know $Dlemma_k(i)$ k -dominates $\{1, 2, \dots, i\}$. If $i' \in \{1, 2, \dots, i_x\}$ is k -dominated by i , then since $i_x < i$ and $\pi^{-1}(i_x) < \pi^{-1}(i)$, by Theorem 3.1.1 $TL_k[i_x] \leq TL_k[i]$ and $BL_k[i_x] \leq BL_k[i]$, so i' is also k -dominated by i_x , done.

To prove (d), we claim $\{i_x, i\}$ can k -dominate $\{i_x + 1, i_x + 2, \dots, i\}$. Then if \exists a choice of $D1_k(i_x)$, say D' , and $|D'| < |Dlemma_k(i) \setminus \{i\}|$, then $\{i\} \cup D'$ can also be a choice of $D1_k(i)$, but it $< |Dlemma_k(i)|$, contradiction since $Dlemma_k(i)$ should be minimum. So (d) is true.

Prove the claim: if $\exists i'$ s.t. $i_x < i' < i$, $\pi^{-1}(i_x) < \pi^{-1}(i') < \pi^{-1}(i)$ and i' can't be k -dominated by $\{i_x, i\}$, then $TR_k[i_y] \leq TR_k[i_x] < i'$ and $BR_k[i_y] \leq BR_k[i_x] < \pi^{-1}(i')$ for $1 \leq y \leq x$ by Theorem 3.1.1. So $Dlemma_k(i)$ doesn't k -dominate i' , contradiction.

2. Proof of Theorem 3.2.4:

By Lemma 3.2.5, there is a minimum distance- k dominating set $\{i_1, i_2, \dots, i_x\}$ of the permutation graph meeting (e). Since $|D1_k(i_x)| \leq x$. The remain to prove is $D1_k(i_x)$ also



k -dominates $\{i_x + 1, i_x + 2, \dots, n\}$, then $D1_k(i_x)$ is also a minimum distance- k dominating set on the same graph.

By Theorem 3.1.1, $TR_k[i_y] \leq TR_k[i_x]$ and $BR_k[i_y] \leq BR_k[i_x]$ for $1 \leq y \leq x$, so i_x must k -dominate $\{i_x + 1, i_x + 2, \dots, n\}$. Since $D1_k(i_x)$ includes i_x , done.

3. Proof of Lemma 3.2.5:

We claim that given an arbitrary set $D'_k(i)$ which is a choice of $D1_k(i)$, we can transfer it into another set, say $Dlemma_k(i)$, s.t. $Dlemma_k(i)$ meets all conditions of $D1_k(i)$ and meets (e). Then the proof is done.

For convenience, we consider the subgraph with vertices = $\{1, 2, \dots, i\}$. Note that the subgraph of a permutation graph is also a permutation graph. We transfer $D'_k(i)$ by: Find two intersecting lines $j_1, j_2 \in D'_k(i)$, W.L.O.G. suppose $j_1 > j_2$ and $\pi^{-1}(j_1) < \pi^{-1}(j_2)$, then replace them by $tr[j_2], bl[j_2]$. Repeat them until each of two lines in the set don't intersecting with each other, the set is $Dlemma_k(i)$.

$Dlemma_k(i)$ meets (a): Directly since we just consider vertices = $\{1, 2, \dots, i\}$.

$Dlemma_k(i)$ meets (b): For each transfer, if $\exists j \in D'_k(i)$ intersecting with i , then since $i > j$, we choose $tr[j]$ and $bl[j]$ to replace them. Since i intersects with j and i is maximum, $tr[j] = i$. So $i \in Dlemma_k(i)$.

$Dlemma_k(i)$ meets (c): We prove that for two intersecting lines $j_1, j_2 \in D'_k(i)$ described as above, if j' is k -dominated by $\{j_1, j_2\}$, then it is also k -dominated by $\{tr[j_2], bl[j_2]\}$, so (c) is met.

(1) If either $d(j', tr[j_2]) \leq 1$ or $d(j', bl[j_2]) \leq 1$, then done.

(2) Else if $bl[j_2] \leq j' \leq tr[j_2]$ and $\pi^{-1}(bl[j_2]) \leq \pi^{-1}(j') \leq \pi^{-1}(tr[j_2])$, then since $j_2 \leq bl[j_2]$ and $\pi^{-1}(j_2) \geq \pi^{-1}(tr[j_2])$, $d(j', j_2) \leq 1$ and thus both $d(j', tr[j_2]) \leq 2$ and $d(j', bl[j_2]) \leq 2$. Since $k > 1$, done.

(3) Else, then either j' is on the right of $tr[j_2]$ or on the left of $bl[j_2]$. We just prove the former term, the latter term has similar proof. There is two cases $d(j', j_1) \leq k$ and $d(j', j_2) \leq k$.

Compare j_2 with $tr[j_2]$, $tr[j_2] = tr[tr[j_2]]$ and $\pi^{-1}(tr[j_2]) \leq \pi^{-1}(tr[tr[j_2]])$, we can use the claims in Theorem 3.1.1 to prove $TR_k[j_2] \leq TR_k[tr[j_2]]$ and $BR_k[j_2] \leq BR_k[tr[j_2]]$. So $d(j', j_2) \leq k \Rightarrow d(j', tr[j_2]) \leq k$.

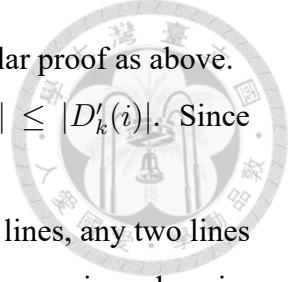
Compare j_1 with $tr[j_2]$, $tr[j_1] \leq tr[tr[j_2]] = tr[j_2]$ since $tr[j_1]$ must intersects with

j_2 ; $\pi^{-1}(br[j_1]) \leq \pi^{-1}(br[tr[j_2]])$ since $j_1 \leq tr[j_2]$, so we can do similar proof as above.

$Dlemma_k(i)$ meets (d): the transfer rule makes $|Dlemma_k(i)| \leq |D'_k(i)|$. Since $|D'_k(i)|$ is minimum, so is $|Dlemma_k(i)|$.

$Dlemma_k(i)$ meets (e): We will prove that for a set of tr and bl lines, any two lines in the set don't intersect with each other. Then when we transfer lines again and again, original lines become less and tr and bl lines become more. Since the lines are finite, the transfer will be end and finally these lines are not intersecting with each other.

We just prove for arbitrary two lines $j_1, j_2, tr[j_1]$ and $tr[j_2]$ don't intersect with each other. The other terms have similar proof. If $tr[j_1]$ intersects with $tr[j_2]$, W.L.O.G. suppose $tr[j_1] > tr[j_2]$, then $tr[j_1] > tr[j_2] \geq j_2$ and $\pi^{-1}(tr[j_1]) < \pi^{-1}(tr[j_2]) \leq j_2$. So j_2 intersects with $tr[j_1]$, contradicts to the definition of $tr[j_2]$.





Chapter 4 An Improved Algorithm

In this chapter, we will improve our algorithm based on the dynamic programming rule above. We will first propose how to improve our algorithm into $O(n^2)$ time and $O(n)$ space based on another distance- k dominating set named $D2_k(t, b)$, which is similar to Ferber and Keil's [12] data structure. Then we will introduce a simple tree structure named AVL tree [1] to improve it into $O(n \log n)$ time and $O(n)$ space.

4.1 Improvement by a New Distance- k Dominating Set

In this section, we will first propose a theorem which makes $D1_k(i)$ easier to compute, then we will split $D1_k(i)$ into two cases. We find that for each i , Case 1 takes $O(1)$ time and $O(1)$ space. Then we will introduce another distance- k dominating set " $D2_k(t, b)$ " used for Case 2. At last we will prove the theorems mentioned in this section.

4.1.1 Some Changes

We first propose a theorem:

Theorem 4.1.1. $j < TL_k[i]$ and $\pi^{-1}(j) < BL_k[i] \Leftrightarrow j$ should be k -dominated by $D1_k(i)$ but isn't k -dominated by i .

We use $U(t, b)$ (Figure 4.1) with $t = TL_k[i] - 1$, $b = BL_k[i] - 1$ to be a set collecting lines j mentioned above. The definition of $U(t, b)$ is:

Definition 4.1.2. $U(t, b)$ is a set of lines j s.t. $j \leq t$ and $\pi^{-1}(j) \leq b$.

Then by theorem we can change $D1_k(i)$ below:

$$\begin{aligned} D1_k(i) &= \{i\} \cup \text{set_min}_{0 \leq j < i} (D1_k(j) \mid \{i\} \cup D1_k(j) \text{ } k\text{-dominates } \{1, 2, \dots, i\}) \\ &= \{i\} \cup \text{set_min}_{0 \leq j < i} (D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(TL_k[i] - 1, BL_k[i] - 1)) \end{aligned}$$

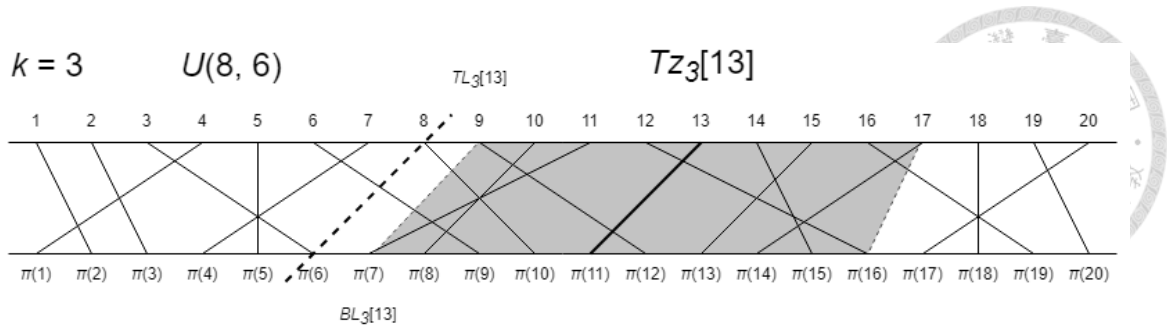


Figure 4.1: $U(t, b)$ with $t = TL_3[13] - 1, b = BL_3[13] - 1$.

Then, based on different k , we split $D1_k(i)$ into two cases. If $k \nmid 2$, we split it into $TL_k[j] = TL_k[i]$ and $TL_k[j] \neq TL_k[i]$; Else, we split it into $BL_k[j] = BL_k[i]$ and $BL_k[j] \neq BL_k[i]$.

For Case 1, we have the following theorem:

Theorem 4.1.3. $D1_k(j)$ must k -dominates $U(TL_k[i] - 1, BL_k[i] - 1)$ for Case 1.

By theorem, if $k \nmid 2$, Case 1 becomes $\{i\} \cup \text{set_min}_{0 \leq j < i, TL_k[j]=TL_k[i]}(D1_k(j))$. $k \mid 2$ is similar.

For Case 2, by Theorem 3.1.1, if $k \nmid 2$, then $0 \leq j < i$ and $TL_k[j] \neq TL_k[i] \Leftrightarrow TL_k[j] < TL_k[i]$; Else, then $0 \leq j < i$ and $BL_k[j] \neq BL_k[i] \Leftrightarrow BL_k[j] < BL_k[i]$. So if $k \nmid 2$, Case 2 becomes $\{i\} \cup \text{set_min}_{TL_k[j] < TL_k[i]}(D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(TL_k[i] - 1, BL_k[i] - 1))$. $k \mid 2$ is similar.

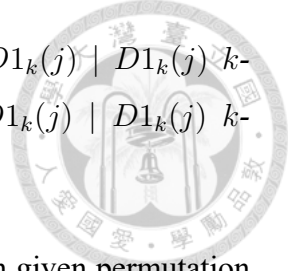
Then we define $D2_k(t, b)$ below s.t. Case 2 becomes $\{i\} \cup D2_k(TL_k[i] - 1, BL_k[i] - 1)$, where "2" means two inputs "t" and "b". We first define something which is used to define $D2_k(t, b)$.

Definition 4.1.4. Let $S = \{i_1, i_2, \dots\}$ be a set of lines, $\max TR_k(S) = \max(TR_k[i_1], TR_k[i_2], \dots)$; $\max BR_k(S) = \max(BR_k[i_1], BR_k[i_2], \dots)$.

Definition 4.1.5. Given several sets S_1, S_2, \dots , $\text{set_min_}TR_k(S_1, S_2, \dots)$ output a set S with minimum cardinality s.t. $\max TR_k(S)$ is as large as possible. $\text{set_min_}BR_k$ has similar definition.

Note that if S is a choice of $\text{set_min_}TR_k(S_1, S_2, \dots)$, then S is also a choice of $\text{set_min}(S_1, S_2, \dots)$. So is $\text{set_min_}BR_k$.

Now we can define $D2_k(t, b)$. We will propose why we use $\text{set_min_}TR_k$ and $\text{set_min_}BR_k$ instead of set_min in $D2_k(t, b)$ section.



Definition 4.1.6. If $k \nmid 2$, $D2_k(t, b) = \text{set_min_BR}_k \text{TL}_k[j] \leq t (D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(t, b))$. Else, $D2_k(t, b) = \text{set_min_TR}_k \text{BL}_k[j] \leq b (D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(t, b))$.

Note that $D2_k(n, n)$ is a minimum distance- k dominating set on given permutation graph. We can output $D2_k(n, n)$ directly in algorithm 3 instead of the original output.

Theorem 4.1.7. $D2_k(n, n)$ is a choice of $\text{set_min}_{0 \leq i \leq n} (D1_k(i) \mid D1_k(i) \text{ } k\text{-dominates } \{1, 2, \dots, n\})$.

So the $D1_k(i)$ becomes below:

$$\text{if } k \nmid 2, D1_k(i) = \{i\} \cup \text{set_min} \begin{cases} \text{set_min}_{0 \leq j < i, \text{TL}_k[j] = \text{TL}_k[i]} (D1_k(j)) \\ D2_k(\text{TL}_k[i] - 1, \text{BL}_k[i] - 1) \end{cases}$$

$$\text{else, } D1_k(i) = \{i\} \cup \text{set_min} \begin{cases} \text{set_min}_{0 \leq j < i, \text{BL}_k[j] = \text{BL}_k[i]} (D1_k(j)) \\ D2_k(\text{TL}_k[i] - 1, \text{BL}_k[i] - 1) \end{cases}$$

We will prove in section 4.1.2 that whatever k is even or odd, Case 1 use $O(1)$ time and $O(1)$ space for each i . So the complexity of $D1_k(i)$ is based on the time and space to find $D2_k(\text{TL}[i] - 1, \text{BL}[i] - 1)$.

4.1.2 Correctness and Complexity Analysis

In this section, we will prove the theorems and prove that Case 1 of $D1_k(i)$ takes $O(1)$ time and $O(1)$ space for each i .

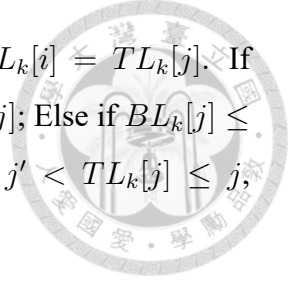
1. Proof of Theorem 4.1.1:

” \Rightarrow ” $j < \text{TL}_k[i]$ and $\pi^{-1}(j) < \text{BL}_k[i]$ implies j doesn't intersect with $Tz_k[i]$. Since $j < \text{TL}_k[i] \leq i$, j should be k -dominated by $D1_k[i]$, done.

” \Leftarrow ” j isn't k -dominated by i , so j doesn't intersect with $Tz_k[i]$. Since j should be k -dominated by $D1_k[i]$, $j \leq i$. So $j < \text{TL}_k[i]$ and $\pi^{-1}(j) < \text{BL}_k[i]$, done.

2. Proof of Theorem 4.1.3:

We just prove if $j < i$ and $\text{TL}_k[j] = \text{TL}_k[i]$, then $D1_k(j)$ must k -dominates $U(\text{TL}_k[i] - 1, \text{BL}_k[i] - 1)$. The $\text{BL}_k[j] = \text{BL}_k[i]$ case has similar proof.



Given arbitrary line $j' \in U(TL_k[i] - 1, BL_k[i] - 1)$, $j' < TL_k[i] = TL_k[j]$. If $\pi^{-1}(j') < BL_k[j]$, then by Theorem 4.1.1 j' is k -dominated by $D1_k[j]$; Else if $BL_k[j] \leq \pi^{-1}(j') \leq BR_k[j]$, then j' intersects with $Tz_k[j]$, done; Else, then $j' < TL_k[j] \leq j$, $\pi^{-1}(j') > BR_k[j] \geq j$. $d(j', j) = 1$, done.

3. Proof of Theorem 4.1.7:

We just prove $k \nmid 2$ case, the other case is similar. $D2_k(n, n) = \text{set_min_}BR_k \text{ }_{TL_k[j] \leq n}(D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(n, n))$. Since $TL_k[j]$ must $\leq n$ and $U(n, n) = \{1, 2, \dots, n\}$, $D2_k(n, n) = \text{set_min_}BR_k \text{ }_{0 \leq j \leq n}(D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } \{1, 2, \dots, n\})$. Then $D2_k(n, n)$ is also a choice of $\text{set_min}_{0 \leq i \leq n}(D1_k(i) \mid D1_k(i) \text{ } k\text{-dominates } \{1, 2, \dots, n\})$.

4. Complexity Analysis:

We just prove $k \nmid 2$ case. The other case is similar, We use the set S to save the answer of Case 1 and do the following:

(1) Initialize $S = \text{no answer}$, $i = 1$.

(2) Since there is no j s.t. $0 \leq j < i$ and $TL_k[j] = TL_k[i]$, Case 1 has no answer.

Let $S \leftarrow D1_k(i)$, $i \leftarrow i + 1$.

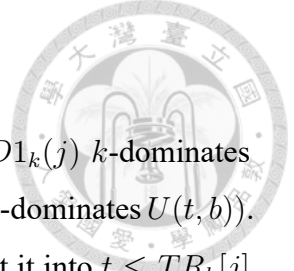
(3-1) If $TL_k[i] \neq TL_k[i - 1]$, then by Theorem 3.1.1 $TL_k[i] > TL_k[i - 1]$. Let $S = \text{no answer}$ and go to (2).

(3-2) Else, S is the answer of Case 1. then let $S \leftarrow \text{set_min}(S, D1_k(i))$, $i \leftarrow i + 1$ and go to (3).

In this method, we use $O(1)$ time to get the answer of Case 1 for each i , and we use a set S only to save Case 1, which can be saved in $O(1)$ space. (The detail of how to save $D1_k(i)$ in $O(1)$ space will be told in the complexity analysis of algorithms 4 and 5.)

4.2 A New Algorithm

In this section, we will propose a method to compute $D2_k(t, b)$ more efficiently, then we will propose the $O(n^2)$ time and $O(n)$ space algorithm containing $D1_k(i)$ and $D2_k(t, b)$. At last we will prove the theorems and prove the correctness and complexity of algorithms mentioned in this section.



4.2.1 Some Changes

Recall that If $k \nmid 2$, $D2_k(t, b) = \text{set_min_}BR_k_{TL_k[j] \leq t}(D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(t, b))$. Else, $D2_k(t, b) = \text{set_min_}TR_k_{BL_k[j] \leq b}(D1_k(j) \mid D1_k(j) \text{ } k\text{-dominates } U(t, b))$. Similar to $D1_k(i)$, we split $D2_k(t, b)$ into two cases. If $k \nmid 2$, we split it into $t \leq TR_k[j]$ and $TR_k[j] < t$; Else, we split it into $b \leq BR_k[j]$ and $BR_k[j] < b$.

For Case 1, we have a theorem similar to Theorem 4.1.3:

Theorem 4.2.1. $D1_k(j)$ must k -dominates $U(t, b)$ for Case 1.

By theorem, if $k \nmid 2$, the Case 1 becomes $\text{set_min_}BR_k_{TL_k[j] \leq t \leq TR_k[j]}(D1_k(j))$. $k \mid 2$ is similar. For Case 2, we have the following theorem:

Theorem 4.2.2. For $k \nmid 2$, if $D2_k(t - 1, b)$ k -dominates $U(t, b)$, then Case 2 isn't better than $D2_k(t - 1, b)$, else, then Case 2 isn't better than Case 1. It is similar for $k \mid 2$ but replace $D2_k(t - 1, b)$ by $D2_k(t, b - 1)$.

Note that this theorem is true if we use $\text{set_min_}TR_k$ and $\text{set_min_}BR_k$, but not just set_min . By theorems above, the $D2_k(t, b)$ becomes below:

$$\text{if } k \nmid 2, D2_k(t, b) = \text{set_min_}BR_k \begin{cases} \text{set_min_}BR_k_{TL_k[j] \leq t \leq TR_k[j]}(D1_k(j)) \\ D2_k(t - 1, b) \text{ if it } k\text{-dominates } U(t, b) \end{cases}$$

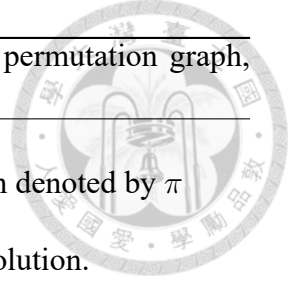
$$\text{else, } D2_k(t, b) = \text{set_min_}TR_k \begin{cases} \text{set_min_}TR_k_{BL_k[j] \leq b \leq BR_k[j]}(D1_k(j)) \\ D2_k(t, b - 1) \text{ if it } k\text{-dominates } U(t, b) \end{cases}$$

By equations of $D1_k(i)$ and $D2_k(t, b)$, we can use algorithms 4 and 5 to find the solution.

In the complexity analysis of the algorithms in section 4.2.2, for $k \nmid 2$, we find two bottlenecks below:

- (1) For each t , we find Case 1 of $D2_k(t, b)$ in $O(n)$ time. (note that Case 1 is independence from b .)
- (2) For each t , we find $D2_k(t, b)$ for all b in $O(n)$ time.

So in the next section, we will propose how to solve these two bottleneck and reduce the time complexity into $O(n \log n)$.



Algorithm 4 Find a minimum distance- k dominating set on given permutation graph, $k \nmid 2$.

Input: permutation π , a positive integer $k \nmid 2$

Output: a minimum distance- k dominating set on permutation graph denoted by π

if $k = 1$ **then**

Use Chao, Hsu, and Lee's [8] $O(n)$ time algorithm to get the solution.

else

Use algorithm 1 and 2 to find $Tz_k[i]$ for $i = 1$ to n .

Initialize $D1_k(0) \leftarrow \emptyset, i \leftarrow 1$

for $t = 0$ to n **do**

for $b = 0$ to n **do**

if $t = 0$ **then**

$D2_k(t, b) \leftarrow D1_k(0)$

else

$$D2_k(t, b) \leftarrow \text{set_min_BR}_k \begin{cases} \text{set_min_BR}_k \text{ } TL_k[j] \leq t \leq TR_k[j] (D1_k(j)) \\ D2_k(t-1, b) \text{ if it } k\text{-dominates } U(t, b) \end{cases}$$

while $TL_k[i] - 1 = t$ **do**

$$D1_k(i) \leftarrow \{i\} \cup \text{set_min} \begin{cases} \text{set_min}_{0 \leq j < i, TL_k[j] = TL_k[i]} (D1_k(j)) \\ D2_k(TL_k[i] - 1, BL_k[i] - 1) \end{cases}$$

$i \leftarrow i + 1$

output $D2_k(n, n)$

4.2.2 Correctness and Complexity Analysis

In this section, we will prove the theorems and correctness of algorithms, then we will prove that the algorithms take $O(n^2)$ time and cost $O(n)$ space. We just prove $k \nmid 2$ case, the other case has similar proof.

1. Proof of Theorem 4.2.1:

We just prove if $TL_k[j] \leq t \leq TR_k[j]$, then $D1_k(j)$ must k -dominate $U(t, b)$. The $BL_k[j] \leq b \leq BR_k[j]$ case has similar proof.

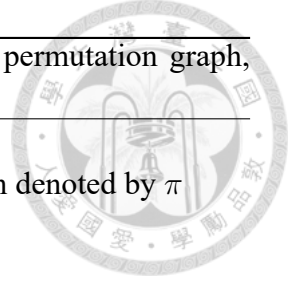
Given arbitrary line $j' \in U(t, b)$, $j' \leq t$. If $j' \geq TL_k[j]$, then j' intersects with $Tz_k[j]$, done. Else if $j' < TL_k[j]$, then the proof similar to that in Theorem 4.1.3 propose us j' must be k -dominated by $D1_k(j)$, done.

2. Proof of Theorem 4.2.2:

Recall that we split $D2_k(t, b)$ into two cases:

Case 1 = $\text{set_min_BR}_k \text{ } TL_k[j] \leq t \leq TR_k[j] (D1_k(j))$

Case 2 = $\text{set_min_BR}_k \text{ } TR_k[j] < t (D1_k[j] \mid D1_k(j) \text{ } k\text{-dominates } U(t, b))$



Algorithm 5 Find a minimum distance- k dominating set on given permutation graph, $k \mid 2$.

Input: permutation π , a positive integer $k \mid 2$

Output: a minimum distance- k dominating set on permutation graph denoted by π

Use algorithm 1 and 2 to find $TR_k[i]$ for $i = 1$ to n .

Initialize $D1_k(0) \leftarrow \emptyset, i \leftarrow 1$

for $b = 0$ to n **do**

for $t = 0$ to n **do**

if $b = 0$ **then**

$D2_k(t, b) \leftarrow D1_k(0)$

else

$D2_k(t, b) \leftarrow \text{set_min_}TR_k \begin{cases} \text{set_min_}TR_k_{BL_k[j] \leq b \leq BR_k[j]}(D1_k(j)) \\ D2_k(t, b - 1) \text{ if it } k\text{-dominates } U(t, b) \end{cases}$

while $BL_k[i] - 1 = b$ **do**

$D1_k(i) \leftarrow \{i\} \cup \text{set_min} \begin{cases} \text{set_min}_{0 \leq j < i, BL_k[j] = BL_k[i]}(D1_k(j)) \\ D2_k(TL_k[i] - 1, BL_k[i] - 1) \end{cases}$

$i \leftarrow i + 1$

output $D2_k(n, n)$

If Case 2 has no answer, then done. Else, we want to prove two things:

(1) If $D2_k(t - 1, b)$ k -dominates $U(t, b)$, then $\text{set_min_}BR_k(D2_k(t - 1, b), \text{Case 2}) = D2_k(t - 1, b)$.

(2) Else, then $\text{set_min_}BR_k(\text{Case 1}, \text{Case 2}) = \text{Case 1}$.

Proof of (1): Let $D1_k(j_0)$ be a choice of Case 2, then $TR_k[j_0] < t$. Recall that $D2_k(t - 1, b) = \text{set_min_}BR_k_{TL_k[j] \leq t-1}(D1_k[j] \mid D1_k(j) \text{ } k\text{-dominates } U(t - 1, b))$. Since $TL_k[j_0] \leq TR_k[j_0] \leq t - 1$, $D1_k(j_0)$ k -dominates $U(t, b) \supseteq U(t - 1, b)$, so $\text{set_min_}BR_k(D1_k(j_0), D2_k(t - 1, b)) = D2_k(t - 1, b)$, otherwise it contradicts to the definition of $D2_k(t - 1, b)$.

Proof of (2): $U(t, b) = \text{either } U(t - 1, b) \text{ or } U(t - 1, b) \cup \{t\}$. Since $D2_k(t - 1, b)$ k -dominates $U(t - 1, b)$ but not $U(t, b)$, $U(t, b) = U(t - 1, b) \cup \{t\}$, so (a) $b \geq \pi^{-1}(t)$, and (b) $D2_k(t - 1, b)$ doesn't k -dominates $\{t\}$. Let $D1_k(j_0)$ be a choice of Case 2 and thus (c) $TR_k(j_0) < t$. We claim:

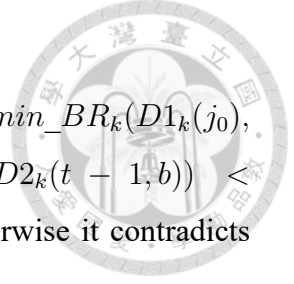
(2-1) $\text{set_min_}BR_k(D1_k(j_0), D2_k(t - 1, b) \cup \{t\}) = D2_k(t - 1, b) \cup \{t\}$

(2-2) $\text{set_min_}BR_k(D2_k(t - 1, b) \cup \{t\}, D1_k(t)) = D1_k(t)$

(2-3) $\text{set_min_}BR_k(D1_k(t), \text{Case 1}) = \text{Case 1}$

So $\text{set_min_}BR_k(D1_k(j_0), \text{Case 1}) = \text{Case 1}$.

Proof of (2-1): We claim that $|D1_k(j_0)| > |D2_k(t - 1, b)|$ and $\text{max}BR_k(D1_k(j_0)) \leq$



$BR_k[t]$, done.

$|D1_k(j_0)| > |D2_k(t-1, b)|$: By the similar proof of (1), $\text{set_min_}BR_k(D1_k(j_0), D2_k(t-1, b)) = D2_k(t-1, b)$. We claim that $\text{max}BR_k(D2_k(t-1, b)) < \text{max}BR_k(D1_k(j_0))$, then $|D1_k(j_0)|$ should $> |D2_k(t-1, b)|$, otherwise it contradicts to the definition of $D2_k(t-1, b)$.

Proof of claim: $\text{max}BR_k(D2_k(t-1, b)) < \pi^{-1}(t)$ by (b); $\pi^{-1}(t) \leq \text{max}BR_k(D1_k(j_0))$ because by (c) and Theorem 3.1.1, $TR_k[j'] < t, \forall j' \in D1_k(j_0)$, but $D1_k(j_0)$ should k -dominate t , done.

$\text{max}BR_k(D1_k(j_0)) \leq BR_k[t]: \forall j' \in D1_k(j_0), j'$ is on the left of t for $k > 1$ by (c) and Theorem 3.1.1. Then by Theorem 3.1.1 again, $BR_k[j'] \leq BR_k[t]$ for all j' , done.

Proof of (2-2): We claim that $|D1_k(t)| \leq |D2_k(t-1, b) \cup \{t\}|$, then since $\text{max}BR_k(D2_k(t-1, b) \cup \{t\}) = BR_k[t]$ which is proved in (2-1), and $BR_k[t] \leq \text{max}BR_k(D1_k(t))$, done.

Proof of claim: $|D1_k(t)| \leq |D2_k(TL_k[t]-1, BL_k[t]-1) \cup \{t\}|$ since the latter is Case 2 of $D1_k(t)$; $|D2_k(TL_k[t]-1, BL_k[t]-1) \cup \{t\}| \leq |D2_k(t-1, b) \cup \{t\}|$ since $U(TL_k[t]-1, BL_k[t]-1) \subseteq U(t-1, b)$ since $TL_k[t]-1 \leq t-1$ and $BL_k[t]-1 < \pi^{-1}(t) \leq b$ by (a), done.

Proof of (2-3): Immediately by definition of Case 1 since $TL_k[t] \leq t \leq TR_k[t]$.

3. Correctness of Algorithm 4:

Note that the correctness of Algorithm 5 has similar proof.

The algorithm has two problems: (1) Whether i can run from 1 to n , (2) Whether the algorithm causes circular argument. We will prove that (1) is yes and (2) is no.

(1) By Theorem 3.1.1, $TL_k[i]$ monotone increases as i increases. So when $t = 0$, we compute $D1_k(i)$ for $i = 1$ to x where $TL_k[i]-1 = 0$, then $TL_k[i]-1$ must > 0 for $i > x$. Then we increase t s.t. $TL_k[x+1]-1 = t$ and compute $D1_k(i)$ for $i = x+1$ to x' where $TL_k[i]-1 = t$, then $TL_k[i]-1$ must $> t$ for $i > x'$. Doing this again and again until $t = n$, then we compute all of the $D1_k(i)$ for $i = 1$ to n .

(2) For $k > 1$, suppose $t \leftarrow t_0$, the Case 1 of $D2_k(t, b)$ has all been computed since for each $j, TL_k[j] \leq t_0$. Take $TL_k[j_0] = t_0$ for example, this is computed when $t = t_0 - 1$ since at that time $TL_k[j_0] - 1 = t_0 - 1 = t$.

The Case 2 of $D1_k(i)$ is computed since $TL_k[i]-1 = t_0$ and for each $b, D2_k(t_0, b)$

are computed above. So the algorithm doesn't cause circular argument.



4. Complexity Analysis:

Note that the complexity of Algorithm 5 has similar proof.

For each $D1_k(i)$, we save $j < i$ s.t. $D1_k(i) = \{i\} \cup D1_k(j)$ (save -1 for $i = 0$); save $|D1_k(i)|$, and save $maxBR_k(D1_k(i))$. For each b of $D2_k(t, b)$, we save i s.t. $D2_k(t, b) = D1_k(i)$. They both cost $O(n)$ space.

Then, for each t , we use $O(n)$ time to compute Case 1 of $D2_k(t, b)$ and save it as D , which contains i s.t. $D = D1_k(i)$.

Then, for each t , for each b , we first use $O(1)$ time to check if $D2_k(t - 1, b)$ k -dominates $U(t, b)$ by checking either $maxBR_k(D2_k(t - 1, b)) \geq \pi^{-1}(t)$ or $b < \pi^{-1}(t)$. Then choose Case 1 or Case 2 to be $D2_k(t, b)$ in $O(1)$ time, save it into the place saving $D2_k(t - 1, b)$ since the latter won't be used anymore after the former has been computed.

Then, for each t , if $TL_k[i] - 1 = t$, we compute Case 1 of $D1_k(i)$ in $O(1)$ time, find Case 2 of it in $O(1)$ time. Since i runs from 1 to n once, this cost $O(n)$ times in total.

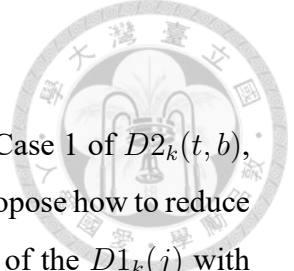
Finally, output $D2_k(n, n)$ by first find i s.t. $D2_k(n, n) = D1_k(i)$ in $O(1)$ time, then find j_1 s.t. $D1_k(i) = \{i\} \cup D1_k(j_1)$ in $O(1)$ time, then find j_2, j_3, \dots , until $j_x = 0$, thus $\{i, j_1, j_2, \dots, j_{x-1}\}$ is minimum distance- k dominating set on given permutation graph. This cost $O(n)$ time in total.

So we use $O(n^2) + O(n^2) + O(n) + O(n) = O(n^2)$ time and $O(n)$ space for algorithm.

4.3 Improvement by AVL Trees

An AVL tree, named by two inventors, G. M. Adel'son-Vel'skii and E. M. Landis [1], is a self-balancing binary search tree where the difference between heights of left and right subtrees for any node can't be more than one. For each node, its left subtree contains nodes smaller than itself; while its right subtree contains nodes larger than itself. The insert, delete, find minimum, search node of an AVL tree all take $O(\log n)$ time.

In this section, we will use the tree to reduce the time of two bottleneck parts of the algorithms above to $O(n \log n)$. We will just propose $k \nmid 2$ case, $k \mid 2$ case are similar. At last we will prove the theorems mentioned in this section.



4.3.1 One Bottleneck

Recall that for $k \nmid 2$, the first bottleneck part is: For each t , we find Case 1 of $D2_k(t, b)$, which is $set_min_BR_k\ TL_k[j] \leq t \leq TR_k[j](D1_k(j))$, in $O(n)$ time. We propose how to reduce it into $O(\log n)$ time using AVL tree. For each t , we want to put all of the $D1_k(j)$ with $TL_k[j] \leq t \leq TR_k[j]$ in the AVL tree. We can do it below:

(1) Insert $D1_k(j)$ once it is computed, which takes $O(\log n)$. Note that it is computed when $TL_k[j] - 1 = t$. Seems it doesn't meet the criterion $TL_k[j] \leq t \leq TR_k[j]$. But actually after all possible $D1_k(j)$ is inserted, we first let $t \leftarrow t + 1$, then compute the Case 1 of $D2_k(t, b)$. At this time $D1_k(j)$ meet the criterion since this time $TL_k[j] = t$.

(2) Delete $D1_k(j)$ once $t > TR_k[j]$, which takes $O(\log n)$.

Since Theorem 3.1.1, we can do them in sequence with j , so they both take $O(n \log n)$ in total. Then we propose the comparison rules in AVL tree. For two different nodes $D1_k(j_1)$ and $D1_k(j_2)$, we put $D1_k(j_1)$ on the left of the $D2_k(j_2)$ in the AVL tree if:

- (1) $|D1_k(j_1)| < |D1_k(j_2)|$
- (2) $|D1_k(j_1)| = |D1_k(j_2)|$ and $maxBR_k(D1_k(j_1)) > maxBR_k(D1_k(j_2))$
- (3) $|D1_k(j_1)| = |D1_k(j_2)|$, $maxBR_k(D1_k(j_1)) = maxBR_k(D1_k(j_2))$, and $j_1 > j_2$.

So $D1_k(j_1)$ is on the left of the $D2_k(j_2)$ in AVL tree $\Leftrightarrow set_min_BR_k(D1_k(j_1), D1_k(j_2)) = D1_k(j_1)$, and we find Case 1 of $D2_k(t, b)$ by just finding the leftmost node of AVL tree, which takes $O(\log n)$ time, done. Note that (3) just make sure there are no two different nodes in AVL tree with same value. You can define $j_1 < j_2$ if you want.

Take Figure 4.2 for example. For $k = 3$, if $t = 8$, the Case 1 of $D2_k(t, b)$ is the leftmost node of AVL tree, which is $D1_3(7)$.

Note that in practice we can just save j in AVL tree to represent $D1_k(j)$ since we already have an array s.t. for each j with computed $D1_k(j)$, it saves $|D1_k(j)|$ and $maxBR_k(D1_k(j))$. This is told in the complexity analysis of algorithms 4 and 5. So the AVL tree needs $O(n)$ space.

4.3.2 Another Bottleneck

Recall that for $k \nmid 2$, the second bottleneck part is: For each t , we find $D2_k(t, b)$ for all b in $O(n)$ time. We propose how to use another AVL tree to reduce it into $O(\log n)$ time. Before that, we first propose a theorem about $D2_k(t, b)$.

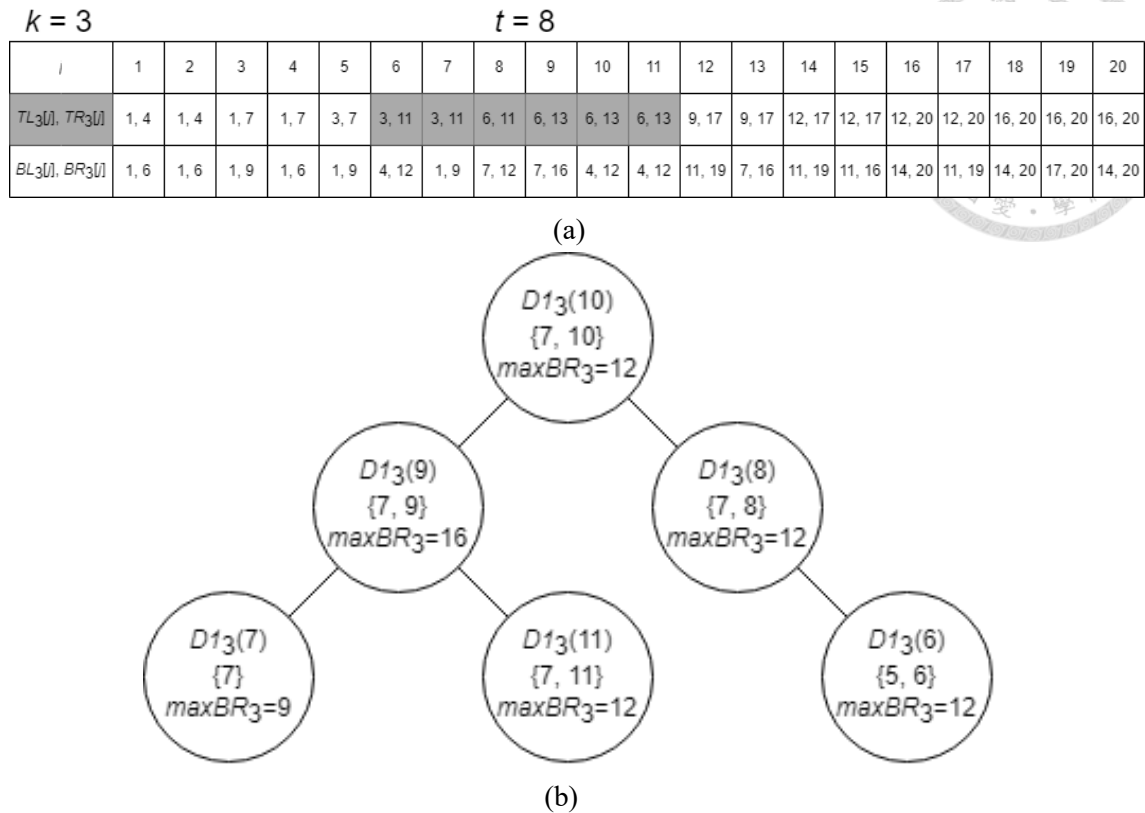


Figure 4.2: An AVL tree saving $D1_k(i)$: (a) a trapezoid table, (b) a corresponding AVL tree when $t = 8$.

Theorem 4.3.1. For $k \nmid 2$, if $b_1 < b_2$, then $set_min_BR_k(D2_k(t, b_1), D2_k(t, b_2)) = D2_k(t, b_1)$; For $k \mid 2$, if $t_1 < t_2$, then $set_min_TR_k(D2_k(t_1, b), D2_k(t_2, b)) = D2_k(t_1, b)$.

So for fixed t , $D2_k(t, b)$ monotone increases below the definition of $set_min_BR_k$ as b increases. We use several blocks instead of an array to save $D2_k(t, b)$ s.t. if $b_1 \neq b_2$ and $D2_k(t, b_1) = D2_k(t, b_2)$, they are in the same blocks. The blocks also monotone increases as b increases.

Then, we use AVL tree to save these blocks. Each node contains the region of b with same $D2_k(t, b)$. Note that the regions don't overlap since each b has only one $D2_k(t, b)$. The node also contains j s.t. $D2_k(t, b) = D1_k(j)$. The comparison rules of this AVL tree is based on the head (or tail) of region of b , the more left is the smaller. The Figure 4.3 is an example with $k = 3, t = 12$.

Then we propose how to construct this tree. Initially when $t = 0$, the AVL tree contains only one node: $D1_k(0)$, with region $b = 0$ to n . Whenever $t \leftarrow t + 1$, Let the Case 1 of $D2_k(t, b)$ be D_{Case1} . We know that for each b :

- (1) $D2_k(t, b) = D_{Case1}$ if $set_min_BR_k(D_{Case1}, D2_k(t - 1, b)) = D_{Case1}$.
- (2) $D2_k(t, b) = D_{Case1}$ if $D2_k(t - 1, b)$ doesn't k -dominate $U(t, b)$.

$k = 3$

$D_{23}(12)$ b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	$D_{13}(0)$	$D_{13}(7)$															$D_{13}(16)$				

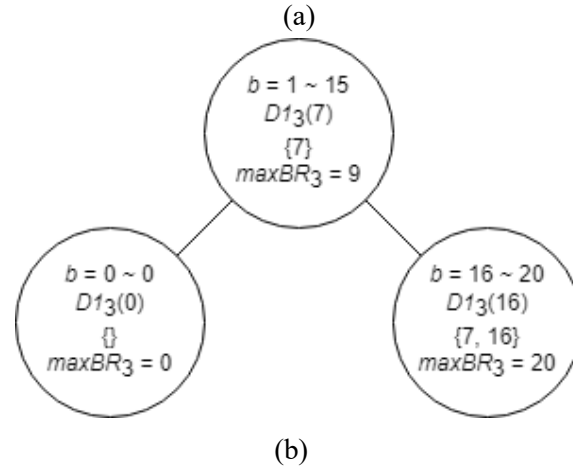


Figure 4.3: An AVL tree saving $D_{2k}(t, b)$: (a) An array saving $D_{2k}(t, b)$ with $t = 12$, (b) its corresponding AVL tree.

(3) $D_{2k}(t, b) = D_{2k}(t - 1, b)$ (no change) otherwise.

So we can continue finding the rightmost node, call the set in it = D_{Case2} and region is $b = b_1$ to b_2 , and do the following:

(1) If $set_min_BR_k(D_{Case1}, D_{Case2}) = D_{Case1}$, deletes the node and repeat.

(2) Else if D_{Case2} k -dominates t , then it must k -dominate $U(t, b)$, done. This happens when $maxBR_k(D_{Case2}) \geq \pi^{-1}(t)$.

(3) Else if $\pi^{-1}(t) < b_1$, then it must not k -dominate $U(t, b)$ for all regions, delete the node and repeat.

(4) Else, $b_1 \leq \pi^{-1}(t) \leq b_2$. D_{Case2} doesn't k -dominate $U(t, b)$ for $\pi^{-1}(t) \leq b \leq b_2$ but does for $b_1 \leq b < \pi^{-1}(t)$. Change the region to $b = b_1$ to $\pi^{-1}(t) - 1$ and done.

After the repeat is end, suppose the rightmost node has region $b = b'_1$ to b'_2 , we insert D_{Case1} to the rightmost of the tree and let its region $b = b'_2 + 1$ to n , done. Note that once the rightmost node isn't deleted, then by Theorem 4.3.1 all of the nodes won't be deleted anymore for same t , so we can then insert D_{Case1} and finished.

For each t , we insert at most one node D_{Case1} in the AVL tree, and we change at most one node's region. So these take $O(\log n)$ time. Since there are $O(n)$ nodes in the tree, we can delete $O(n)$ nodes in total. So the method above take $O(\log n)$ time in amortize to compute $D_{2k}(t, b)$ for all t and b .

Note that in practice for each node, we just save j in AVL tree to represent $D_{1k}(j)$

in it, and save head and tail of b to represent its region. So the AVL tree costs $O(n)$ space in total.

So for each t , we use $O(\log n)$ time to find Case 1 of $D2_k(t, b)$, then we use $O(\log n)$ in amortize to find $D2_k(t, b)$ for all b .

Notice that in the Case 2 of $D1_k(i)$, we need to find $D2_k(TL_k[i] - 1, BL_k[i] - 1)$, where $TL_k[i] - 1 = t$. We now can find it in $O(\log n)$ by let $b = BL_k[i] - 1$, then use second AVL tree to find a node whose region is b_1 to b_2 , s.t. $b_1 \leq b \leq b_2$. So for each i , we find $D1_k(i)$ in $O(1) + O(\log n) = O(\log n)$ time. So the algorithm takes $O(n \log n)$ time.

Since we spend $O(n)$ space to find trapezoids and save them, an $O(n)$ array to save $D1_k(i)$, and two $O(n)$ AVL trees to save $D1_k(j)$ and $D2_k(t, b)$, so the algorithm costs $O(n)$ space.

4.3.3 Correctness

In this section, we will prove Theorem 4.3.1. We just prove $k \nmid 2$ case, $k \mid 2$ case is similar.

Let $b_1 < b_2$, suppose $set_min_BR_k(D2_k(t, b_1), D2_k(t, b_2)) \neq D2_k(t, b_1)$, but since $U(t, b_1) \subseteq U(t, b_2)$, $D2_k(t, b_2)$ also k -dominates $U(t, b_1)$, and the restrict of $D1_k(j)$ of two sets are same ($TL_k[j] \leq t$), so this contradicts to the definition of $D2_k(t, b_1)$.



Chapter 5 Conclusion

In this thesis, we first find that for fixed k , for each line i , there is a trapezoid $Tz_k[i]$ s.t. $d(i, j) \leq k \Leftrightarrow Tz[i]$ intersects with j , and find $Tz[i]$ for all i in $O(n \log k)$ time and $O(n)$ space.

Then we define a distance- k dominating set $D1_k(i)$ and find that it can be computed by some $D1_k(j)$ where $j < i$.

Then we define another distance- k dominating set $D2_k(t, b)$, which is similar to that in Farber and Keil's paper [12]. We find that with the cooperation of $D1_k(i)$ and $D2_k(t, b)$, we can get the solution in $O(n^2)$ time and $O(n)$ space.

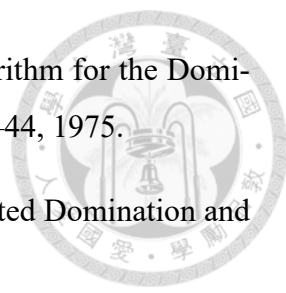
Finally, with AVL tree, we reduce the algorithm into $O(n \log n)$ time and $O(n)$ space.

The open question is whether we can solve it faster than $O(n \log n)$. Tsai and Hsu's paper [22] uses another data structure to reduce the time of $k = 1$ case into $O(n \log \log n)$. If we can use the same data structure to improve our algorithm, then our time complexity becomes $\min(O(n \log k), O(n \log \log n))$.



References

- [1] G. M. Adel'son-Vel'skii and E. M. Landis. An Algorithm for Organization of Information. *Doklady. Akademii. Nauk SSSR*, 146:263–266, 1962.
- [2] K. Arvind and C. P. Rangan. Connected Domination and Steiner Set on Weighted Permutation Graphs. *Information Processing Letters*, 41:215–220, 1992.
- [3] M. J. Atallah and S. R. Kosaraju. An Efficient Algorithm for Maxdominance, with Applications. *Algorithmica*, 4:221–236, 1989.
- [4] M. J. Atallah, G. K. Manacher, and J. Urrutia. Finding a Minimum Independent Dominating Set in a Permutation Graph. *Discrete Applied Mathematics*, 21:177–183, 1988.
- [5] M. A. Bonuccelli. Dominating Sets and Domatic Number of Circular Arc Graphs. *Discrete Applied Mathematics*, 12:203–213, 1985.
- [6] A. Brandstädt and D. Kratsch. On Domination Problems for Permutation and Other Graphs. *Theoretical Computer Science*, 54:181–198, 1987.
- [7] J. M. Chang, C. W. Ho, and M. T. Ko. Powers of Asteroidal Triple-free Graphs with Applications. *Ars Combinatoria*, 67:161–173, 2003.
- [8] H. S. Chao, F. R. Hsu, and R. C. T. Lee. An Optimal Algorithm for Finding the Minimum Cardinality Dominating Set on Permutation Graphs. *Discrete Applied Mathematics*, 102:159–173, 2000.
- [9] T. C. E. Cheng, L. Kang, and E. Shan. A Polynomial-Time Algorithm for the Paired Domination Problem on Permutation Graphs. *Discrete Applied Mathematics*, 157:262–271, 2009.

- 
- [10] E. Cockayne, S. Goodman, and S. Hedetniemi. A Linear Algorithm for the Domination Number of a Tree. *Information Processing Letters*, 4:41–44, 1975.
- [11] C. J. Colbourn and L. K. Stewart. Permutation Graphs: Connected Domination and Steiner Trees. *Discrete Mathematics*, 86:179–189, 1990.
- [12] M. Ferber and J. M. Keil. Domination in Permutation Graphs. *Journal of Algorithms*, 6:309–321, 1985.
- [13] M. R. Garey and D. S. Johnson. *A Guide to the Theory of Np-Completeness*. W. H. Freeman, 1979.
- [14] W. L. Hsu and K. H. Tsai. Linear Time Algorithms on Circular-Arc Graphs. *Information Processing Letters*, 40:123–129, 1991.
- [15] O. H. Ibarra and Q. Zheng. Some Efficient Algorithms for Permutation Graphs. *Journal of Algorithms*, 16:453–469, 1994.
- [16] E. Lappas, S. D. Nikolopoulos, and L. Palios. An $O(n)$ -Time Algorithm for the Paired Domination Problem on Permutation Graphs. *European Journal of Combinatorics*, 34:593–608, 2013.
- [17] Y. D. Liang, C. Rhee, S. K. Dhall, and S. Lakshmirarahan. A New Approach for the Domination Problem on Permutation Graphs. *Information Processing Letters*, 37:219–224, 1991.
- [18] R. M. McConnell and J. P. Spinrad. Modular Decomposition and Transitive Orientation. *Discrete Mathematics*, 201:189–241, 1999.
- [19] A. Rana, A. Pal, and M. Pal. An Efficient Algorithm to Solve the Distance k -Domination Problem on Permutation Graphs. *Journal of Discrete Mathematical Sciences and Cryptography*, 19:241–255, 2016.
- [20] C. Rhee, Y. D. Liang, S. K. Dhall, and S. Lakshmirarahan. An $O(N + M)$ -Time Algorithm for Finding a Minimum-Weight Dominating Set in a Permutation Graph. *SIAM Journal on Computing*, 25:404–419, 1996.
- [21] J. Spinrad. Transitive Orientation in $O(n^2)$ Time. *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 457–466, 1983.

- [22] K. H. Tsai and W. L. Hsu. Fast Algorithms for the Dominating Set Problem on Permutation Graphs. *Algorithmica*, 9:601–614, 1993.

