

MASTER THESIS

**Reporting Center Problem
for Interval Graphs and Trees**

Hengchin Yeh

*Department of Computer Science and Information Engineering,
National Taiwan University*

Advisor

Prof. D. T. Lee

*Department of Computer Science and Information Engineering,
National Taiwan University*

July, 2004

Abstract

In this thesis we have studied the reporting center problem for interval graphs and trees. The reporting center strategy is one of many strategies used to track the mobile users of a wireless network. The reporting center problem arises from the need to improve the efficiency, which is accomplished by balancing the cost of searching for a user against the cost of location-reporting by a user. In the reporting center strategy, a subset of the base stations of the cellular network is selected as reporting centers, to which mobile users report their locations when visiting their cells. The set of non-reporting centers associated with a reporting center is called its vicinity. To route an incoming call, the network locates a user by searching in the vicinity of the last contacted reporting center. The size of the vicinity of a reporting center determines the searching and updating cost of the cellular network. It is thus an objective to minimize the number of reporting centers subject to the constraint that the size of the vicinity of each reporting center is bounded by a constant $Z > 0$. The problem has been shown to be *NP*-complete for arbitrary graphs for $Z \geq 2$. The major contribution of this work is divided into two parts: (1) an algorithm that solves the reporting center problem for arbitrary vicinity for interval graphs, thereby improving a previous result which only solves for vicinity $Z = 2$ for interval graphs and for arbitrary vicinity for proper interval graphs, and (2) an $O(n)$ time algorithm that solves the reporting center problem for trees, which is better than the previous $O(nZ^3)$ result.

Acknowledgement

I would like to express my sincere gratitude to my graduate advisor Dr. Der-Tsai Lee for his constant support, guidance and motivation. I am also grateful to the committee members Dr. Gerard Jennhwa Chang and Dr. Kun-Mao Chao for providing valuable comments. I take this opportunity to thank the colleagues in the Institute of Information Science, Academia Sinica, especially Mr. Liao Chung Shou for helpful discussions.

Contents

Bibliography	iii
1 Introduction	1
2 Reporting Center Problem for Interval Graphs	5
2.1 The Staircase Method	5
2.2 Bricks Transformation Method	28
3 Reporting Center Problem for Trees	31
4 Conclusion	53
Bibliography	55

List of Figures

1.1	Reporting cells in a network.	2
2.1	A packet.	7
2.2	Packetizing a set of non-reporting centers.	8
2.3	Proper containment.	8
2.4	The influence region.	9
2.5	Case 1 of influence region.	9
2.6	Case 2 of influence region.	10
2.7	Three different cases for P_3	10
2.8	A staircase specified by a sequence of points.	12
2.9	A staircase.	13
2.10	An example of packet which is compatible with a staircase.	15
2.11	The updating of a staircase.	16
2.12	An illustration of a set of packets which is compatible with a staircase.	16
2.13	Lemma 2.4.	17
2.14	case 2 in the proof of Lemma 2.4.	18
2.15	H_1 dominates H_2	19
2.16	The definition of a brick.	28
2.17	Bricks transformation with $Z = 3$	29
2.18	(a)Feasible and (b)non-feasible solutions.	30
3.1	Packets in a tree.	32
3.2	The $b(x)$ value of a non-reporting center x	33
3.3	Lemma ??	36
3.4	Lemma ??	38
3.5	Case 1 in Lemma ??	39
3.6	Case 2 in Lemma ??	40
3.7	Case 1 in Lemma ??	42

3.8	Case 2 in Lemma ??	42
3.9	Case (a) of case 1 in Lemma ??	45
3.10	Case (b) of case 1 in Lemma ??	45
3.11	Case (a) of case 2 in Lemma ??	48
3.12	Case (b) of case 2 in Lemma ??	48

Chapter 1

Introduction

The reporting center problem was first introduced in the field of wireless networks, and later modeled as a graph-theoretic problem [3]. There has been a significant growth in the use of wireless networks over the past decades, creating a need for greater capacity and better coverage which is mostly addressed by cellular networks. A cellular network consists of multiple mobile users communicating wirelessly with fixed base stations, which are in turn connected by a wired backbone network. Each base station provides communication services within a geographic area referred to as a *cell*, and smooth connectivity is achieved by using overlapping cells.

For a network to route incoming calls, it is essential to know which cell contains the users because they can only be contacted through the base stations. A variety of strategies have been designed for *tracking* mobile users. To evaluate the efficiency of any particular design requires identifying the *searching cost* and the *updating cost*. The former involves maintaining complex data structures to store information of each user; the latter considers the frequency of updates necessary to achieve appropriate precision, which determines the bandwidth and energy consumption. Several strategies aiming at

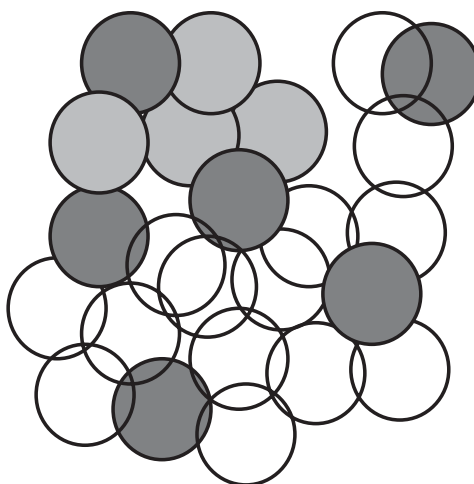


Figure 1.1: Reporting cells in a network.

balancing the searching and updating costs have been proposed [1–5, 8–10].

The *reporting center* strategy is the one improved by the research described in this article. In this strategy, updating occurs only when users enter predetermined cells, called *reporting cells*. (The base stations of such cells are called *reporting centers*.) The space searched is restricted to the *vicinity* of the reporting centers, excluding non-reporting cells which can only be reached by traveling past any other reporting cells. Periodically, each reporting center informs users that they are within its cell by use of a specific channel. In response, users report their presence. Thus, the system knows to seek the user in the vicinity of the reporting center which received the last report. See Fig 1.1 for an illustration. The darker cells represent reporting centers while others are non-reporting centers. The lighter-shaded cells denote the vicinity of the top-left reporting center. The problem is to select reporting centers in such a way that both the size of the largest vicinity and the total number of reporting centers are minimized. These are conflicting goals since in order to decrease the size of the vicinities, the number of

reporting centers must be increased (and vice versa). The approach taken by Bar-Noy and Kessler [3] was to bound the vicinity size, and then to minimize the number of reporting centers.

Formally, let a geographic area be modelled by a *mobility graph* $G = (V, E)$ whose vertices V correspond to the cells in such area and whose edges correspond to pairs of cells that overlap. Let a vertex v be termed a *reporting center* if the user must report its position when visiting the the network cell associated with v . Otherwise, v is termed a *non-reporting center*. For a given constant Z' and a mobility graph G , the reporting center problem asks for a minimum cardinality set of reporting centers in G such that for each reporting center its vicinity has size at most Z' . Notice that in previous papers the vicinity bound integer is termed Z , but $Z - 1$ is repeatedly used in the context. In order to avoid the repeated use of $Z - 1$, we redefine Z to be their Z minus one, and give another name Z' for their Z , so that $Z = Z' - 1$.

Definition Reachability set. Consider a partition of V into R and \bar{R} , i.e., $V = R \cup \bar{R}$ and $R \cap \bar{R} = \emptyset$. Let $\Lambda_R^0(v) = v, v \in V$; and let $\Lambda_R^1(v)$ denote the set of vertices $u \in \bar{R}$ adjacent to v . Recursively we define

$$\Lambda_R^i(v) = \{w \mid \overline{u, w} \in E, u \in \Lambda_R^{i-1}(v), w \in \bar{R} \text{ and } w \notin \Lambda_R^j(v) \text{ for } j \leq i - 1\}.$$

Define $\Lambda_R^*(v) = \{w \in \bar{R} \mid u \in \Lambda_R^i(v) \text{ for some } i \geq 0\}$, which denotes the set of vertices $w \in \bar{R}$ *reachable* from v by a path passing only through vertices in \bar{R} . We will use $\Lambda_R(v)$ as a shorthand for $\Lambda_R^*(v)$ in the following text.

Definition Reporting Center Problem. Given a graph $G = (V, E)$ and an integer Z' , the *reporting center problem* involves selecting a *minimum size* set R of reporting centers such that $|\Lambda_R(v)| \leq Z'$ for all $v \in R$. For convenience, we also define the *non-reporting center problem* as the problem of selecting a *maximum size* set \bar{R} of non-reporting centers. A solution is said to be *feasible* if it does not violate the vicinity condition, and is said to be *optimal* if it achieves the minimization (maximization, respectively) condition.

The problem was shown to be *NP*-hard for arbitrary graphs and $Z \geq 2$ [3]. In the following two chapters we will study the reporting center problem for special classes of graphs, i.e. interval graphs and trees. By the physical meaning of reporting centers and its vicinity, we will consider its vicinity as its *load*, which also captures the idea of that every reporting center should not be *overloaded*, an analogy of that every reporting center should not have a vicinity greater than the given bound.

Chapter 2

Reporting Center Problem for Interval Graphs

In this chapter we have studied the reporting center problem for interval graphs. The problem was studied by Olariu et al. [7], but they only proposed results for restricted values of vicinity bound (≤ 2) for general interval graphs, or for arbitrary vicinity bound but only for proper interval graphs. We study the problem for general interval graphs without restriction to the values of vicinity bound.

2.1 The Staircase Method

Consider a set of intervals, $\mathcal{I} = \{I_i \mid i = 0, 1, \dots, n-1\}$ on the real line. Each interval I_i is represented by an ordered pair (ℓ_i, r_i) , $\ell_i \leq r_i$, denoting respectively the left and right endpoints of I_i . The intersection graph for \mathcal{I} , called *interval graph*, is a graph $G_{\mathcal{I}} = (V, E)$, in which each vertex $v \in V$ corresponds to an interval $I(v) = (a(v), b(v))$, where $a(v)$ and $b(v)$ denote respectively the left and right endpoints of $I(v)$, and there is an edge $\overline{u, v} \in E$, $u, v \in V$ if and only if $I(u)$ and $I(v)$ intersect. For a set of vertices $S \subseteq V$,

let $a(S)$ and $b(S)$ define the leftmost and rightmost endpoints, respectively, of all elements in that set, i.e. $a(S) = \min\{a(v) \mid v \in S\}$, $b(S) = \max\{b(v) \mid v \in S\}$. The *neighborhood* of $v \in V$, denoted $N(v)$, is the set of vertices adjacent to v , i.e., $N(v) = \{u \in V \mid \overline{u,v} \in E\}$. The *closed neighborhood* of v , denoted $N[v]$, is defined as $N[v] = \{v\} \cup N(v)$. The neighborhood of a set $S \subseteq V$ of vertices is defined as $N[S] = \bigcup_{v \in S} N[v]$. We assume that the set of intervals is *normalized*, i.e., the $2n$ endpoints are sorted and mapped to integers $\{1, 2, \dots, 2n\}$ for $n = |V|$. We define the terminology that we will use in the following. The intervals in \mathcal{I} and the corresponding vertices $v \in V$ in the interval graph $G_{\mathcal{I}}$ or simply G , will be used interchangeably.

Definition RMN. The *rightmost neighbor* of a vertex $v \in G$, denoted $RMN(v)$, is the vertex w such that $w \in N[v]$, and $b(w)$ is maximum. The *rightmost neighbor* of a set $S \subseteq V$ of vertices, denoted $RMN(S)$, is defined similarly, i.e., $RMN(S) = w$, such that $w \in N[S]$, and $b(w)$ is maximum.

Lemma 2.1 *If S_1 and S_2 are two sets of vertices with $b(S_1) \leq b(S_2)$, then $b(RMN(S_1)) \leq b(RMN(S_2))$.*

Proof. Consider the vertex $v = RMN(S_1)$. There are two cases: either $v \in N[S_2]$ or $v \notin N[S_2]$. If $v \in N[S_2]$, then it is obvious that $b(v) \leq b(RMN(S_2))$. If $v \notin N[S_2]$, then $b(v) < a(S_2) < b(S_2) \leq b(RMN(S_2))$, because of the property of interval ordering. \square

Definition Packet. A connected component P in the induced subgraph $G[\bar{R}]$ is called a (non-reporting) *packet* (See Fig. 2.1).

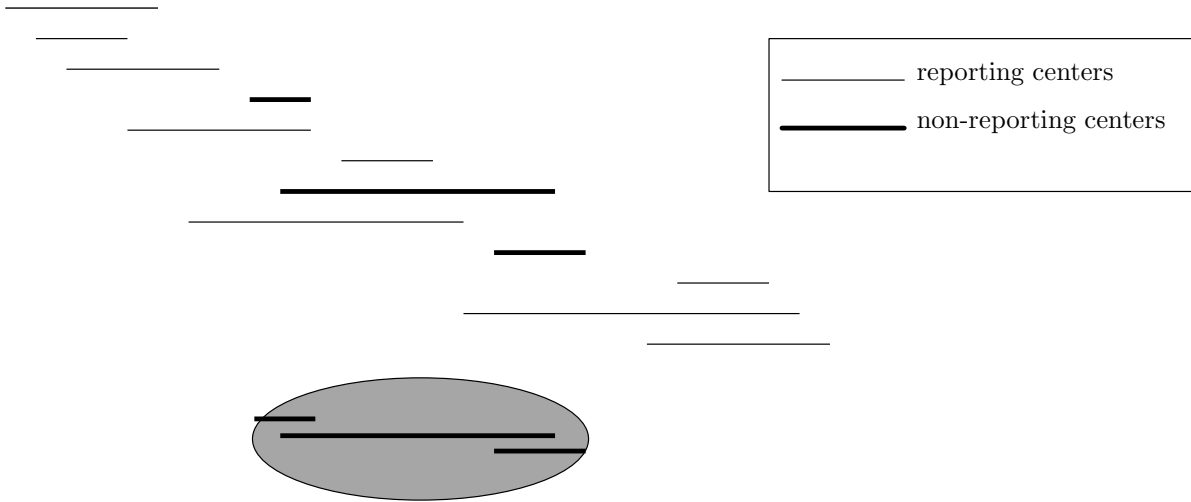


Figure 2.1: A packet.

Let V be partitioned into $R \dot{\cup} \bar{R}$. $\forall u, v \in P$, $u \in \Lambda_R(r)$ if and only if $v \in \Lambda_R(r)$, for some $r \in R$. That is to say, if an element of P is reachable from a reporting center, all other elements in that packet are also reachable from that center. Thus, a reporting center is responsible for an entire packet. Having the concept of packet in mind, the set of non-reporting centers can be decomposed into packets.

Definition Packetize. For a set $S \subseteq \bar{R}$, $Packetize(S)$ is a set of packets $\{P_1, P_2, \dots, P_m\}$ satisfying the following:

1. $S = \bigcup_{1 \leq i \leq m} P_i$, and
2. $b(P_i) < a(P_{i+1})$, $1 \leq i < m$.

See Fig. 2.2 for an illustration. We shall make a few observations that are important for subsequent discussions.

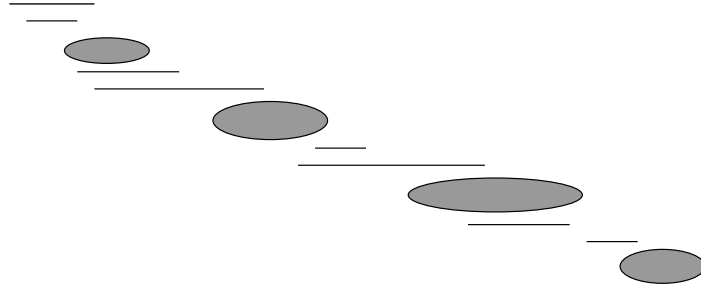


Figure 2.2: Packetizing a set of non-reporting centers.

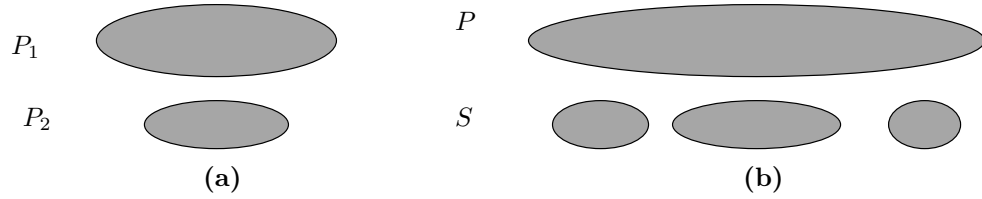


Figure 2.3: Proper containment.

Observation 2.2 Let P_1 and P_2 be two packets with $|P_1| = |P_2|$.

P_1 is said to properly contain P_2 if $a(P_1) \leq a(P_2)$ and $b(P_2) \leq b(P_1)$. For any feasible solution F containing P_1 , there exists another solution $F' = (F \setminus P_1) \cup P_2$ which is also feasible. That is, for any two packets, P_1 and P_2 , if P_1 properly contains P_2 , and $|P_1| = |P_2|$, then we can replace P_1 with P_2 in any feasible solution. See Fig. 2.3(a). Moreover, if P is a packet and S is a set with $|P| = |S|$, $a(P_1) < a(S)$ and $b(S) < b(P)$, then we can safely replace P by S . That is, for any feasible solution F containing P , $F' = F \setminus P \cup S$ is also feasible. See Fig. 2.3(b).

Let P_1 be the leftmost packet with closed neighborhood $N[P_1]$, shown in Fig. 2.4, where P_1 is shown shaded and $N[P_1]$ is shown as the outer empty region. The interval $(a(N[P_1]), b(N[P_1]))$ is referred to as the *influence* region of P_1 . P_1 will limit our future choices of packets in two ways:

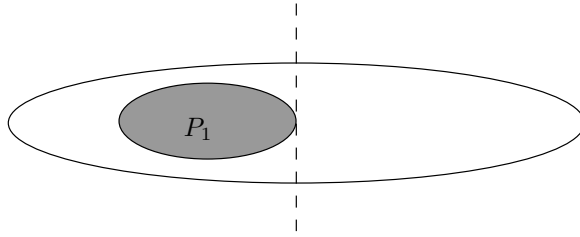


Figure 2.4: The influence region.

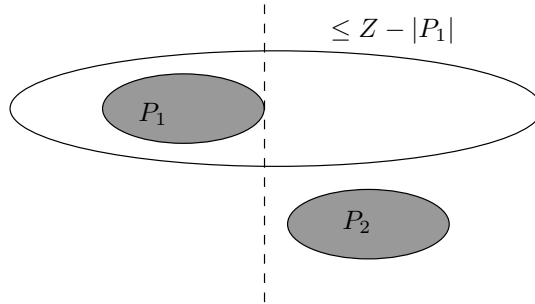


Figure 2.5: Case 1 of influence region.

1. If the next packet P_2 starts within the region $(b(P_1), b(N[P_1]))$, then it must satisfy $|P_2| \leq Z - |P_1|$. This is because the reporting center located at $RMN(P_1)$, which defines the right boundary of the influence region associated with packet P_1 , requires that its vicinity be no greater than Z . This case is shown in Fig. 2.5.
2. If the next packet lies completely to the right of $b(N[P_1])$ (or $b(RMN(P_1))$, equivalently), as shown in Fig. 2.6 then since there will be no reporting center from which to reach both P_1 and P_2 , the only restriction is the original vicinity constraint, i.e., $|P_2|$ must be no more than Z .

Suppose we have selected P_2 as the second packet, as shown in Fig. 2.7. We now have three possible cases in selecting the third packet P_3 , each with a different constraint, as described below.

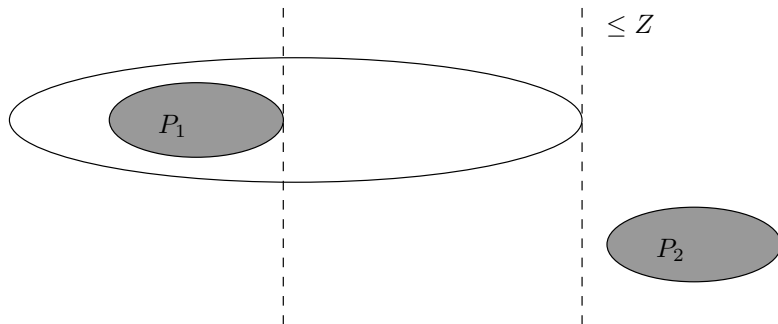


Figure 2.6: Case 2 of influence region.

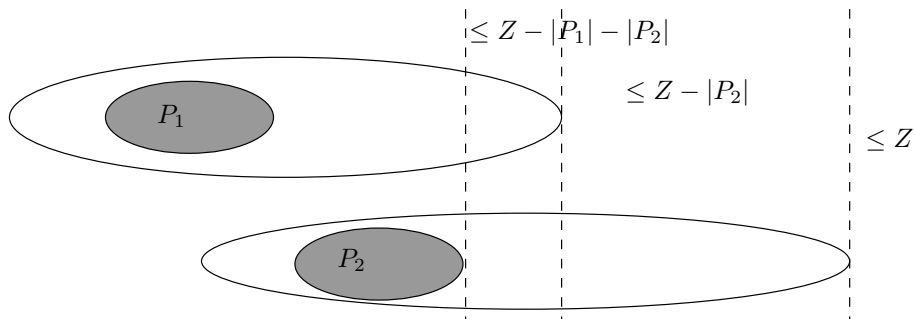


Figure 2.7: Three different cases for P_3 .

1. If $b(P_2) < a(P_3) < b(RMN(P_1))$, then P_3 must satisfy $|P_3| \leq Z - |P_1| - |P_2|$.
2. If $b(RMN(P_1)) < a(P_3) < b(RMN(P_2))$, then $|P_3| \leq Z - |P_2|$.
3. If $b(RMN(P_2)) < a(P_3)$, then $|P_3| \leq Z$.

Note that Fig. 2.7 just illustrates one possible case for P_2 . The other possibility, i.e., when $b(P_2) > b(RMN(P_1))$ will induce only the second and third cases shown above. In general we have the following observations.

Observation 2.3 1. Each packet P has a rightward influence, that limits our choice of subsequent packets. Its influence region extends exactly

to $b(\text{RMN}(P))$, beyond which the problem can be considered over.

2. Consecutive i packets have rightward influences that limit the subsequent packets in a staircase manner, e.g. first $Z - \sum_{j=1}^i |P_j|$, $Z - \sum_{j=2}^i |P_j|$, \dots , $Z - |P_i|$ and finally Z . (See Fig. 2.9)
3. After appending a packet P_k to an existing solution $\text{Packetize}(S)$, the rightward influence changes. The change is well defined, and will be discussed later.

The upper half of Fig. 2.9 shows (a part of) an interval graph, where each interval is followed by an ordered pair, which indicates the left and right endpoints, respectively of the interval. The lower half shows three packets, each with an influence region extending rightward all the way to its rightmost neighbor, and the *magnitude* of its influence being shown as the height. For example, take a look at the lower half of Fig. 2.9 from right to left. The right-end of it is a height 0 *ground*; then to the left of the ground is the last step whose height is $|P_3|$, then $|P_2| + |P_3|$, then $|P_1| + |P_2| + |P_3|$, and so forth. The left-end is an infinitely high *wall*, however, it will be shown later that a wall with height Z is enough. We call such graphical interpretation of influences a *staircase*, as the shape suggests. Before defining a staircase formally, let us first re-examine its properties. First of all, a staircase has height Z . As stated before, the left end of a staircase is an infinitely high wall. The wall is there just to prevent a selection of two overlapping packets, which must be defined as one packet instead of two. This goal can be well achieved by a wall with height Z , since a feasible solution already prevents us from selecting a packet to the left of the wall. Secondly, a staircase may have fewer than Z *steps*. However, we can always assume that there are exactly

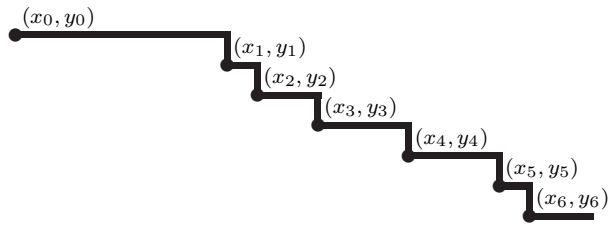


Figure 2.8: A staircase specified by a sequence of points.

Z of them, except that some of these steps have zero widths. Moreover, a staircase needs to be updated when a new packet is appended to the solution, as we have shown when P_2 is appended to P_1 in a solution.

A staircase can be specified as a sequence of points $(x_0, y_0), (x_1, y_0), (x_1, y_1), (x_2, y_1), (x_2, y_2), \dots, (x_Z, y_Z)$, where $x_0 \leq x_1 \leq x_2 \leq \dots \leq x_Z$ and $y_0 \geq y_1 \geq y_2 \geq \dots \geq y_Z$, or more concisely as $(x_0, y_0), (x_1, y_1), \dots$, as shown in Fig. 2.8. In our case we always have $y_0 = Z, y_1 = Z - 1, \dots, y_Z = 0$. We can simply represent the staircase by a Z -tuple (x_1, x_2, \dots, x_Z) with x_0 assumed to be $-\infty$.

For example, in Fig. 2.9, the staircase comprises 7 steps (in fact only 3, but as we said we pretend that there are 7 of them), each depicted as a black node along the line-drawing (bold lines) of the staircase. Right to each step i is an x_i indicating the x -coordinate of that step. The sequence of these 7 numbers, namely $\{x_1, x_2, \dots, x_7\}$, well defines the staircase. Another way to interpret these 7 numbers, is by asking the question: "How big is the room between the staircase and the ceiling (a horizontal line at height Z), at some x coordinate?" or another version: "How far can I go, as left as possible, and still have a room of height i between the staircase and the ceiling?" It can be easily seen that x_i is the answer to the latter question. Now we can give

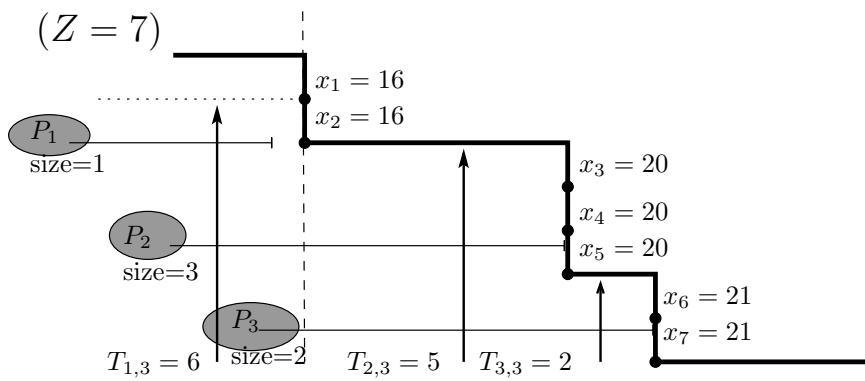
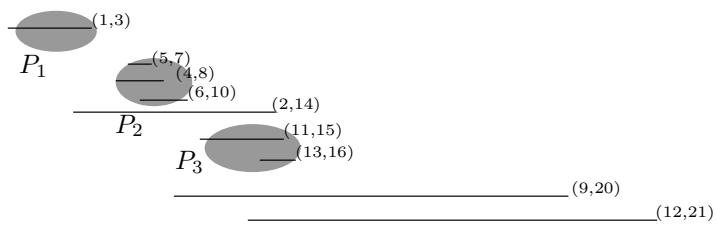


Figure 2.9: A staircase.

a formal definition of a staircase, together with some concepts showing how it serves as a limiter.

Definition Staircase. A staircase H can be encoded as a Z -dimensional vector (x_1, x_2, \dots, x_Z) , where each x_i is an integer, denoting the x -coordinate of the i -th step, which is of height $Z-i$. The i -th item of H , x_i will be denoted by $H[i]$.

The staircase of a set S is denoted by $Staircase(S)$, which can be obtained as follows:

Let $Packetize(S) = \{P_1, P_2, \dots, P_m\}$. Define $T_{l,m} = \sum_{l \leq j \leq m} |P_j|$. Let $H = Staircase(S)$, there are two cases for i to consider in order to determine the value of $H[i]$:

$$H[i] = \begin{cases} \max\{b(RMN(P_l), b(P_m))\} & \text{if } Z - T_{l,m} < i \leq Z - T_{l+1,m} \\ & \text{for some } 1 \leq l \leq m \\ b(P_m) & \text{if } i \leq Z - T_{1,m} = Z - |S| \end{cases}$$

Definition Compatibility. Let $H = (H[1], H[2], \dots, H[Z])$ be a staircase. A packet P is said to be *compatible* with H , if and only if $a(P) > H[|P|]$. Fig. 2.10 provides an illustration.

When a packet P is appended to a set S with $Staircase(S)$, the staircase needs to be updated. The following is a function defining this updating procedure. It can be easily verified that $Staircase(S \cup P)$ is identical to $Updatestaircase(Staircase(S), P)$, so it fulfills our purpose. See Fig. 2.11 for an illustration.

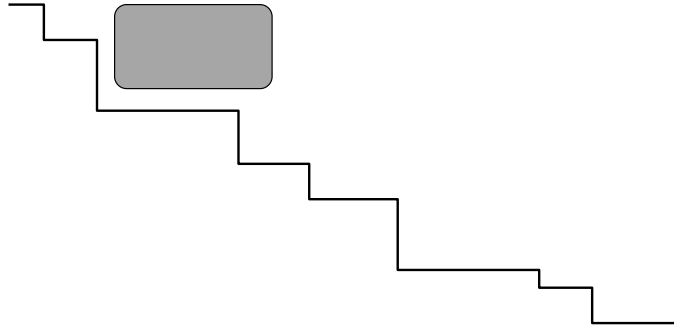


Figure 2.10: An example of packet which is compatible with a staircase.

Definition UpdateStaircase. For a staircase H and a compatible packet P $UpdateStaircase(H, P)$ is the staircase H' obtained by the following:

$$H'[i] = \begin{cases} \max\{H[i + |P|], b(P)\} & \text{for } 1 \leq i \leq Z - |P| \\ b(RMN(P)) & \text{for } Z - |P| < i \leq Z \end{cases}$$

We now define the compatibility of a sequence of packets with a staircase, as depicted in Fig. 2.12.

Definition Compatibility: for a set of packets. Let S be a set of vertices with $Packetize(S) = \{P_1, P_2, \dots, P_m\}$. Then S is said to be *compatible* with H if and only if P_i is compatible with

$$H_i = \begin{cases} UpdateStaircase(H_{i-1}, P_i) & \text{for } i > 1 \\ H & \text{for } i = 1 \end{cases}$$

Although staircases are designed to complete the task of choosing non-reporting centers, we need a solid mathematical justification. The following lemma shows how the internal structure of a set of non-reporting centers is closely related to its staircase.

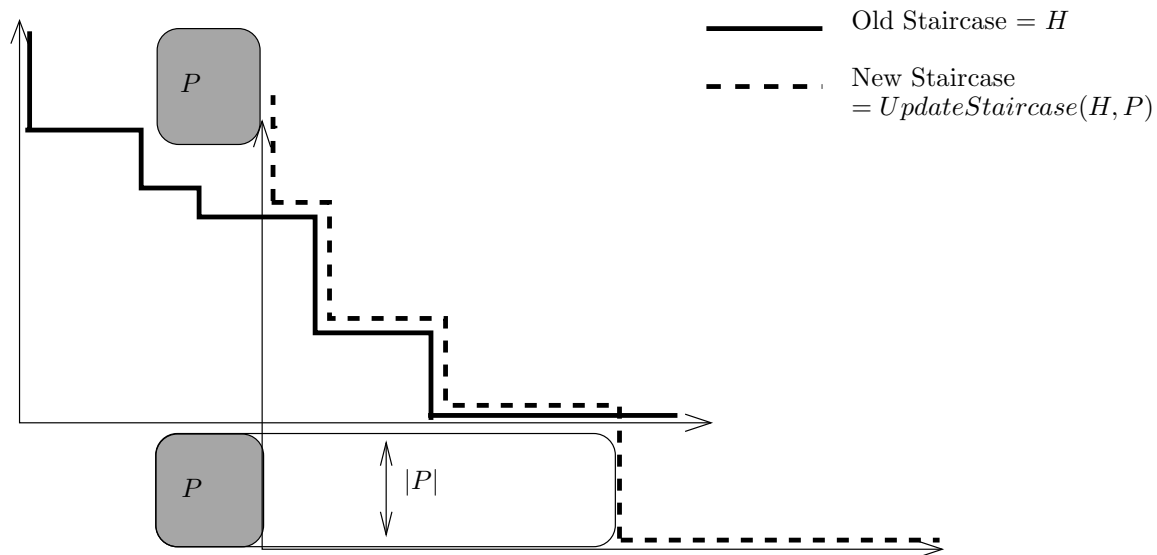


Figure 2.11: The updating of a staircase.

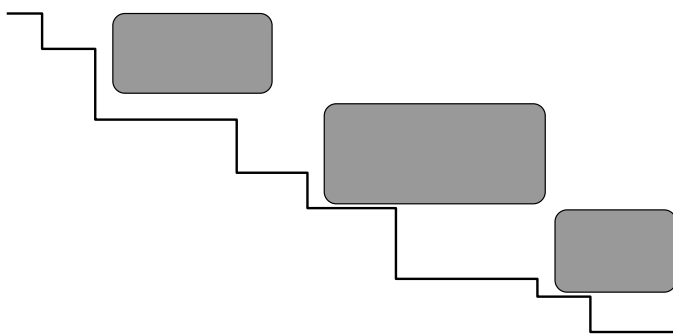


Figure 2.12: An illustration of a set of packets which is compatible with a staircase.

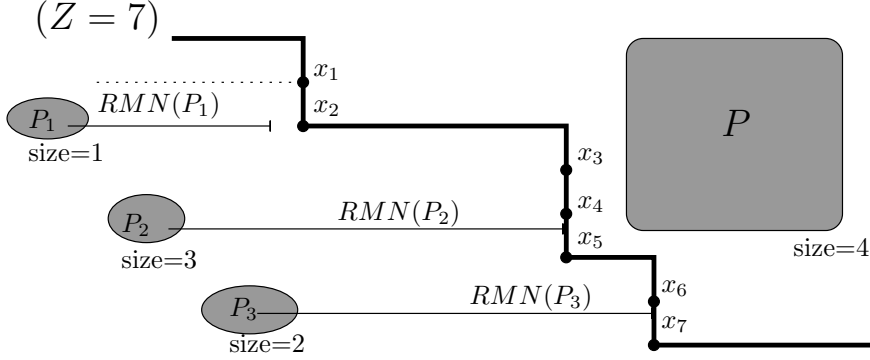


Figure 2.13: Lemma 2.4.

Lemma 2.4 *If S is a feasible solution for the non-reporting center problem and P is a packet lying totally to the right of S , i.e. $a(P) > b(S)$, then $S \cup P$ is also a feasible solution if and only if P is compatible with $Staircase(S)$.*

Proof. See Fig. 2.13. Suppose $Packetize(S) = \{P_1, P_2, \dots, P_m\}$, and $T_{l,m}$ is as defined in Definition **Staircase**. Let $Staircase(S) = H = (H[0], H[1], \dots, H[Z])$. By definition, P is compatible with $Staircase(S)$ if and only if $a(P) > H[|P|]$. Consider the two cases for $|P|$:

1. If $|P| \leq Z - T_{1,m} = Z - |S|$, then $H[|P|]$ is defined to be $b(P_m)$. Since $|S \cup P| = |S| + |P| \leq Z$, there are at most Z non-reporting centers. So none of the reporting centers will have vicinity greater than $Z + 1$, $S \cup P$ is also a feasible solution.
2. If $Z - T_{l,m} < |P| \leq Z - T_{l+1,m}$ for some $1 \leq l \leq m$, then $H[|P|]$ is set to be $\max\{b(RMN(P_l), b(P_m))\}$, $H[|P|] \geq b(RMN(P_l))$. Since P is compatible with $Staircase(S)$, $a(P) > H[|P|] \geq b(RMN(P_l))$. This means that P and P_l have no common neighbor, and that all neighbors of P situate themselves to the right of P_l . Those are the

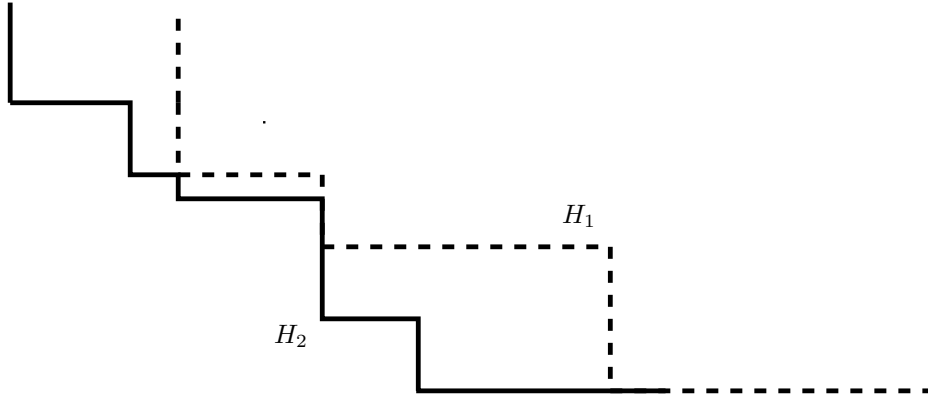


Figure 2.15: H_1 dominates H_2 .

Definition Domination. A staircase H_1 is said to *dominate* another staircase H_2 , if $H_1[i] \geq H_2[i]$, $\forall 1 \leq i \leq Z$, and is denoted by $H_1 \succeq H_2$. A staircase H_1 is said to *strictly dominate* another staircase H_2 , denoted $H_1 \succ H_2$, if $H_1 \succeq H_2$ and $H_1 \neq H_2$. A staircase H is said to be *minimal* within a set if there exists no other staircase dominated by it.

See Fig. 2.15 for an example of one staircase dominating another.

Lemma 2.5 *If H_1 and H_2 are two staircases with $H_1 \succeq H_2$, then $\pi(H_1) \leq \pi(H_2)$,*

Proof. By the definition of $\pi(H)$, the solution which is compatible with H_1 and gives $\pi(H_1)$ is also compatible with H_2 . The lemma follows immediately. \square

Definition NEXT. $NEXT(x, y)$ is the packet P satisfying the following:

1. $a(P) > x$,

2. $|P| = y$, and
3. $b(P)$ is the minimum among all those satisfying 1 and 2.

$NEXT(x, y)$ returns \emptyset if such a packet satisfying 1 and 2 doesn't exist.

Practically, we only need those $NEXT(x, y)$'s for x being the right endpoint of some vertex, and y ranging from 1 to Z , i.e. $NEXT(b(i), j)$'s, for some $i, j, 1 \leq i \leq n, 1 \leq j \leq Z$.

Lemma 2.6 *Let $P = NEXT(H[k], k)$ for a staircase H . Then for all size- k packet P' that is compatible with H , $UpdateStaircase(H, P') \succeq UpdateStaircase(H, P)$.*

Proof. P' is compatible with H if and only if $a(P') > H[k]$. Among all the P' 's compatible with H , $P = NEXT(H[k], k)$ is the one with the minimum right endpoint. By Lemma 2.1, $b(RMN(P))$ is also minimum. By the definition of $UpdateStaircase(H, P)$, the first several steps are $\max\{H[k+1], b(P)\}$, the last k steps are $b(RMN(P))$, and the steps in between are the same for all P' 's. Thus, $UpdateStaircase(H, P)$ is minimal. \square

We say that $NEXT(H[k], k)$ in Lemma 2.6 produces a minimal staircase among all size- k packets.

Lemma 2.7 *$\Gamma(H) = P^j \cup H_j$, such that $|P^j| + |H_j|$ is maximum for $1 \leq j \leq Z$ and $P^j \neq \emptyset$, gives an optimal solution, i.e. $|\Gamma(H)| = \pi(H)$, where $P^j = NEXT(H[j], j)$, and $H_j = UpdateStaircase(H, P^j)$.*

Proof. Suppose for any $H' \succ H$, $|\Gamma(H')| = \pi(H')$. Let $\Gamma^*(H)$ be an optimal solution with $|\Gamma^*(H)| = \pi(H)$, we want to prove that $\Gamma(H)$ is also an optimal solution. Let Q be the first packet of $Packetize(\Gamma^*(H))$ with

$|Q| = q$. Let $L = \text{UpdateStaircase}(H, Q)$. After Q , the rest of the packets of $\Gamma^*(H)$ must be compatible with L and are denoted by $\Gamma^*(L)$. Thus,

$$\begin{aligned}\Gamma^*(H) &= Q \cup \Gamma^*(L). \\ |\Gamma^*(H)| &\leq q + |\Gamma(L)| \\ &\leq |P^q| + |\Gamma(H_q)| \\ &\leq |\Gamma(H)|.\end{aligned}$$

The first inequality holds because L is a staircase with $L \succ H$, by our hypothesis $\Gamma(L)$ is optimal. The second inequality holds because according to Lemma 2.6, $P^q = \text{NEXT}(H[q], q)$ is the one that produces the minimal staircase among all size q packets that are compatible with H . Therefore $L \succeq H_q$, which in turn implies that $\pi(L) \leq \pi(H_q)$, by Lemma 2.5. So $\Gamma(H)$ is also optimal. \square

The above lemma implies that there exists an optimal solution which consists of only *NEXT*-packets, and can be found by a series of calls to the *NEXT* function. However, in order to make the algorithm clearer and more efficient, a modified version of *NEXT* is introduced. The following is the definition of the *NEXT2* function, followed by a lemma showing that *NEXT2* is enough for our purpose.

Definition NEXT2. Let V_x denote a sequence whose elements $V_x[i]$, $i = 1, 2, \dots$, are all the vertices lying right of x , i.e. $\{v \in V \mid a(v) > x\}$ and sorted according to their right endpoints. Define $A_{x,y}$ to be the set that collects the first y elements of V_x . $A_{x,y} = \bigcup_{1 \leq i \leq y} V_x[i]$, and $A_{x,y}$ can be interpreted as *the leftmost y vertices lying immediately to the right of x* . Then,

$$\text{NEXT2}(x, y) = \begin{cases} A_{x,y} & \text{if } A_{x,y} \text{ is connected} \\ \emptyset & \text{if } A_{x,y} \text{ is not connected} \end{cases}$$

Lemma 2.8 *There is an optimal solution that consists of only NEXT2-packets.*

Proof. We first prove that the following is true:

If Γ is an optimal solution and $NEXT(x, y) \in Packetize(\Gamma)$, and $NEXT(x, y) \neq A_{x,y}$, then there exists another optimal solution $\Gamma' = \Gamma \setminus NEXT(x, y) \cup A_{x,y}$.

Compare the definition of $NEXT$ and $NEXT2$, the only chance that $NEXT(x, y) \neq A_{x,y}$ is when $A_{x,y}$ is not connected, at which point $NEXT(x, y)$ properly contains $A_{x,y}$. By Obs2.2, $|\Gamma'| \leq |\Gamma \setminus NEXT(x, y) \cup A_{x,y}|$. Secondly, to elucidate the relationship between this fact and Lemma 2.8, let $Packetize(A_{x,y}) = \{P_1, P_2, \dots, P_m\}$. It can be checked that $P_{i+1} = NEXT2(b(P_i), |P_{i+1}|)$ for $i > 1$ and $P_1 = NEXT2(H[|P_1|], |P_1|)$. Since we can always replace $NEXT(x, y)$ with a series of $NEXT2$ -packets, the lemma follows. \square

Now we are ready to present the algorithm.

Algorithm NRCI. A non reporting center problem for interval graphs.

Input An interval graph $G = (V, E)$, and a vicinity constraint (an integer)
 $Z' \geq 1$.

Output A maximum size non-reporting center set for vicinity Z' .

Method

$Z \leftarrow Z' - 1$;

Compute $RMN[v]$, $\forall v \in V$;

Compute $NEXT2(b(i), j)$, for $0 \leq i < n$, $1 \leq j \leq Z$;

Let H_0 be a Z -dimensional vector $(0, 0, \dots, 0)$;

return $\Gamma(H_0)$;

The $\Gamma(H)$ is defined to be a recursive function as shown below, followed by the process computing *NEXT2*. The *UpdateStaircase* function is already defined in the text.

Algorithm $\Gamma(H)$.

Input A staircase H .

Output A maximum size non-reporting center that is compatible with H .

Method

```

for  $j = 1$  to  $Z$ 
     $P^j \leftarrow \text{NEXT2}(H[j], j)$ ;
    if ( $P^j \neq \emptyset$ ) then
         $H_j \leftarrow \text{UpdateStaircase}(H, P^j)$ ;
         $Sol_j \leftarrow P^j \cup \Gamma(H_j)$ ;
    else  $Sol_j \leftarrow \emptyset$  ;
end for
if ( all  $Sol_j = \emptyset$  ) then return  $\emptyset$ ;
else return  $Sol_j$  with the maximum size  $|Sol_j|$ , for  $1 \leq j \leq Z$ ;

```

Below is the description of the variables that are used during the computation of *NEXT2*.

L : a doubly linked-list of the vertices that supports the following operation in constant time:

SetBegin(L): resets the pointer to the beginning of the list.

Read(L): reads the item at which the pointer points.

Delete(L): deletes the item at which the pointer points.

The is pointer is moved to the item immediately after the deleted item.

Advance(L): moves the pointer to the next item in the list.

IsNull(L): returns true if the pointer is out of the end of the list .

Construct(L, V): constructs the list from an already sorted set V of vertices, and takes $O(|V|)$ time.

b_{head} : an integer representing the right endpoint of the "head",
which is the leftmost connected component in the accumulated set.

v_{new} : a vertex that is to be added to a set, the "tail".

The connectivity of the set is thereby checked by verifying if the "head"
and the "tail" overlaps.

```

Computing NEXT2
/*Initialize */
Construct(L, V)
for i = 0 to n - 1
    bhead ← 0;
    SetBegin(L) ;
    Let Ab(i),0 be vacuously ∅ ;
    j ← 1 ;
    while ( j ≤ Z )
        if (not IsNull(L)) then
            vnew ← Read(L);
            if ( a(vnew) < b(i) ) then /* This is a neighbor of vertex i */
                Delete(L); /* We shall remove it from L */
                continue
            else /* This is not a neighbor of i, so we accumulate it */
                newv ← Read(L);
                Advance(L);
                j ← j + 1;
                Ab(i),j ← Ab(i),j-1 ∪ {vnew};
            /* Now we check the connectivity */
            if ( a(vnew) < bhead ) then
                NEXT2(b(i), j) ← Ab(i),j;
                bhead ← b(vnew); /* Updating the connectivity */
            else
                NEXT2(b(i), j) ← ∅;
        else /* There are no more vertices to select */
            j ← j + 1;
            NEXT2(b(i), j) ← ∅;
    end for
end for

```

Theorem 2.9 *The algorithm NRCI solves the reporting center problem for interval graphs in $O(n^Z)$ time.*

Proof. The correctness follows by the lemmas. The major time consumption of the algorithm comes from the dynamic programming part. During the process we actually expand all possible staircases. Since a staircase can be described as a non-decreasing Z -dimensional vector with each component taking value from $\{1, 2, \dots, 2n\}$, there are at most $n^Z/Z!$ distinct staircases. These are the *nodes* (in the computation graph) that we will visit. Associated to each node (a staircase H), the best solution that is compatible with H ($\Gamma(H)$) is stored, together with its size $\pi(H)$. The storage could be implemented as a linked-list, since each solution can be denoted by some previous solution with a packet appended. So we do not have to worry about the overhead of recording solutions. In order to determine the solution that should be recorded in each node, at most Z nodes (supposedly recursively solved) need to be examined and compared (only the π values are compared), which means that there are Z outgoing edges from each node of the computation graph. Traversing each edge, the *UpdateStaircase* involves calculating Z integers, which takes $O(Z)$ time. So there is a cost of $O(Z^2)$ associated to each node. The time of the traversal of the computation graph is therefore $= O(Z^2 \times n^Z/Z!) = O(n^Z)$, which is the time spent for the dynamic programming. The preprocessing (computing *RMN*'s and *NEXT2*'s) time is overwhelmingly dominated by this term, so the total cost is $O(n^Z)$. \square

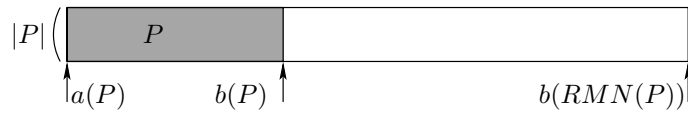


Figure 2.16: The definition of a brick.

2.2 Bricks Transformation Method

In this section we shall introduce an alternative perspective on the problem, which is inspired by noticing that the ideas of staircases and packets are both quite *graphical*. It is natural to take one step further, that is to transform the whole problem into a geometric one. Roughly speaking, packets are transformed into *bricks* and a staircase is just the outline of a pile of bricks. Formally, the transformation takes place in the following way:

Definition Bricks. A packet P is transformed into a brick with two segments, representing the packet itself and its rightmost neighbor, respectively. The first segment has endpoints defined by the original packet, namely $a(P)$ and $b(P)$, while the second segment spans from $b(P)$ to $b(RMN(P))$. The heights are the cardinality of the packet, $|P|$.

See Fig. 2.16 for the definition of a brick. For convenience, we refer to the first segment of a brick the *shaded* part of it, while calling a brick without specifying which part means the whole brick. Since it has been shown in the previous section that the *NEXT2*-packets suffice to solve the problem, we only need those bricks transformed from the *NEXT2*-packets, and are referred to as *useful* bricks. There are at most nZ useful bricks. Fig. 2.17 provides an example of transforming an interval graph into a collection of useful bricks when $Z = 3$. The reporting center problem is thereby transformed

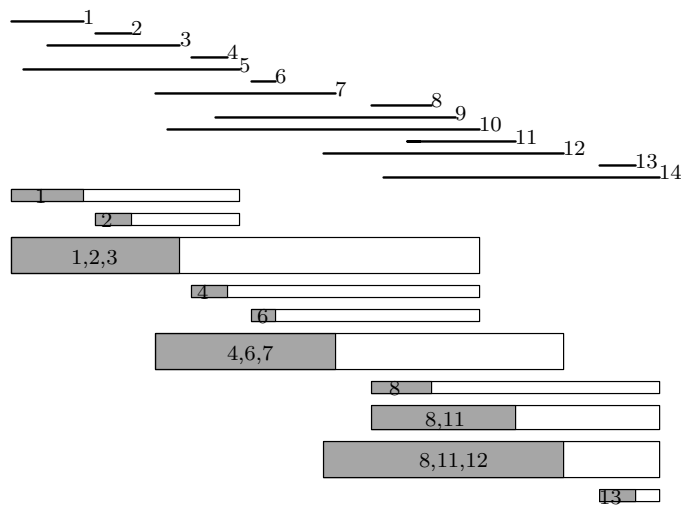


Figure 2.17: Bricks transformation with $Z = 3$.

into the following problem :

Definition Brick Piling Problem. Given a set of useful bricks, select a subset of them such that:

1. no shaded parts overlap,
2. the density (local height piled) for all x coordinates does not exceeds Z , and
3. the total height of the piling is maximized.

The second condition is just the avatar of the vicinity constraint. A solution satisfying the first and the second conditions is called *feasible*, and in addition satisfying condition 3 is called *optimal*. For the example shown in Fig. 2.17, Fig .2.18 shows both feasible and non-feasible solutions.

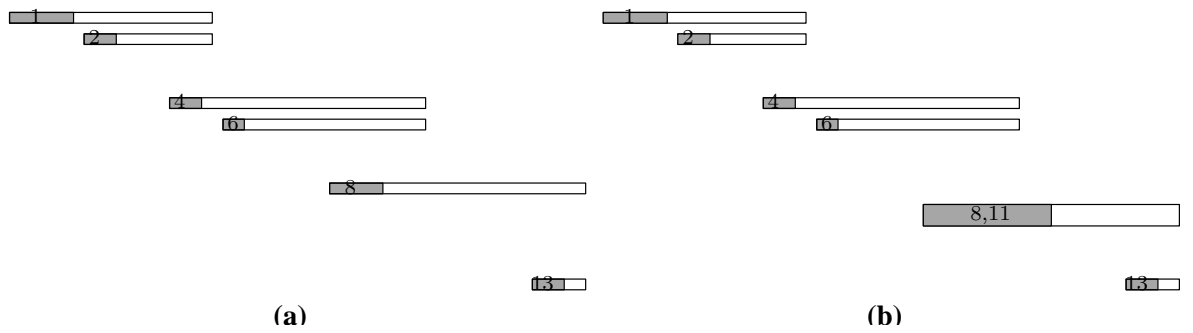


Figure 2.18: (a) Feasible and (b) non-feasible solutions.

So far we have not found any theoretical advantage of the bricks transformation over the original reporting center problem. Nevertheless, it provides us with a more intuitive and pictorial way of dealing with the problem, which is valuable especially for the computational-geometers.

Chapter 3

Reporting Center Problem for Trees

In this chapter we study the reporting center problem for trees and present an algorithm that runs in $O(n)$ time, improving the $O(nZ^3)$ result of Bar-Noy and Kessler [3]. A graph with no cycle is *acyclic*. A *forest* is an acyclic graph. A *tree* is a connected acyclic graph, and a *leaf* is a vertex with degree 1 [11]. The idea of packets introduced in the previous chapter is still useful. Fig. 3.1 gives an illustration of how packets shall look like in a tree. The filled nodes represent reporting centers, the hollow ones the non-reporting centers, and packets are shown as shaded regions. For instance, node v_{12} is a reporting center. The vicinity of v_{12} includes itself, and vertices from 3 packets: P_4 , P_5 and P_6 . The vicinity constraint is thus $|P_4| + |P_5| + |P_6| + 1 \leq Z'$. If the vertices are weighted, namely, vertex x has weight $a(x)$, then we replace $|P|$ by the total weight $a(P) \equiv \sum_{v \in P} a(v)$. We know that the vicinity constraint shall be checked for every reporting center. But what can we say about a non-reporting center? What "rule" should it obey? For a non-reporting center x , we want to know if every reporting center y for which $x \in \Lambda_R(y)$ satisfies the vicinity constraint. It is obvious that the whole packet $P(x)$

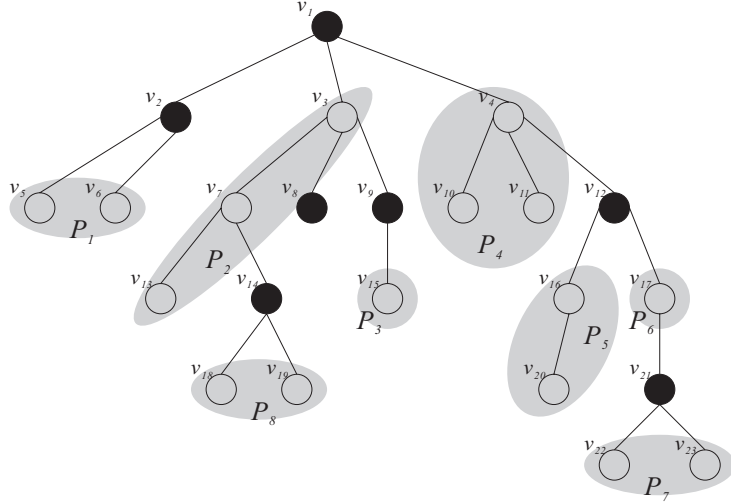


Figure 3.1: Packets in a tree.

is contained in $\Lambda_R(y)$, i.e., $P(x) \subseteq \Lambda_R(y)$, where $P(x)$ denotes the packet containing x . Unless y is the root, there are other (external) packets that are contained in $\Lambda_R(y)$. The total size of those packets must satisfy the vicinity constraint. Among those reporting centers r , whose $\Lambda_R(r)$ contains x the one that is most likely to violate the vicinity constraint is the y that has the largest total size of external packets (packets other than $P(x)$). We denote this external load by a value $b(x)$, which is attached to x (in fact to each vertex in $P(x)$). In other words, $b(x)$ is the heaviest external load of all reporting centers whose reachability sets contains x . Fig. 3.2 shows the idea. So the constraint that x must satisfy is that $b(x) + a(P(x)) \leq Z'$. Notice that this is not an extra constraint to the original problem. It is just another version of the vicinity constraint, a solution satisfying one constraint automatically satisfies the other. Consider a slightly modified version of the original problem. Assume that there are a certain number of *pre-selected* reporting centers on the tree. If these pre-selected reporting centers satisfy the vicinity constraint, we do nothing; if on the contrary some of them have vicinity greater than Z' , then we must choose additional vertices as reporting

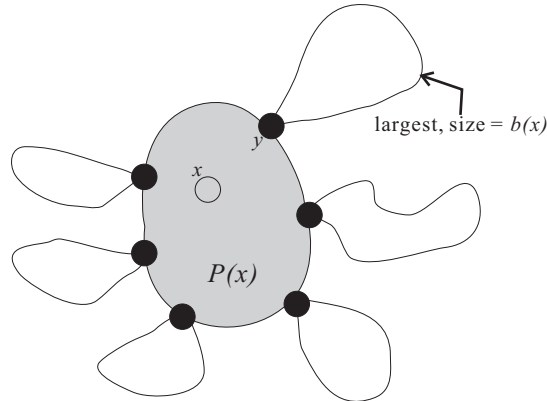


Figure 3.2: The $b(x)$ value of a non-reporting center x .

centers to reduce the vicinity. The problem then becomes that of choosing a minimum number of extra reporting centers so that the vicinity constraint is satisfied. We shall realize the above idea by associating a variable $c(x)$ to each vertex x , such that for pre-selected vertices x , we have $c(x) = true$, while for others we have $c(x) = false$.

Consider a tree $T = (V, E)$. Let v be the root of T . The vertices connected to the root v are said to be *children* of v , and v is said to be their *parent*. A child w of v in turn may have its children. The number of children of a vertex w is called the (out-) *degree* of w . (The in-degree of each vertex, except the root, is always one.) The vertex of degree 0 (or with no children) is called a *leaf*. Now we are ready to define the weighted version of the tree reporting center problem.

Definition Tree Reporting Centers. Consider a rooted tree $T = (V, E)$, with $v = \text{root}(T)$. Associated with each vertex $x \in V$ we have a label $L(x) = (a(x), b(x), c(x))$, where $a(x)$ denotes the *weight* and $b(x)$ the *external load* of x respectively, and $c(x)$ is a boolean variable. Given a nonnegative integer

$Z' > 0$, a (T, L, Z') -reporting center is a subset $R \subseteq V$, which is a collection of tree reporting centers for T , if it satisfies the following conditions:

default-center: If $c(x) = true$, then x must be in R ;

weight: $\sum_{y \in \Lambda_R(x)} a(y) \leq Z', \forall x \in R$;

load: $b(x) + \sum_{y \in \Lambda_R(x)} a(y) \leq Z', \forall x \notin R$.

In other words, given $a(x)$, $b(x)$ and $c(x)$, $\forall x \in V$ and an integer $Z' > 0$, we want to find a subset $R \subseteq V$ of minimum cardinality such that the above conditions are satisfied. The unweighted tree reporting center problem is one where $\forall x \in V, a(x) = 1, b(x) = 0$, and $c(x) = false$.

A solution satisfying the items **default-center**, **weight** and **load** in Definition Tree Reporting center above is called *feasible*. A solution minimizing the size of (T, L, Z') -reporting center is said to be *optimal for (T, L, Z')* , and the size is denoted by $\Gamma_L(T)$. To solve the unweighted tree reporting center problem we shall reduce it to solving instances of the above weighted tree reporting center problem. Initially $c(x)$ is set to *false* for all $x \in V$. We shall gradually turn some non-reporting centers x with $c(x) = false$ into reporting center by creating another instance with x marked as "default-center", i.e. $c'(x) = true$, $a'(x)$ and $b'(x)$ modified, and a new tree T' obtained from T possibly with some leaves deleted. The optimal R is therefore obtained recursively by calling the algorithm for the new instance (T', L', Z') , and merging the resulting R' with some others to obtain R .

Now we shall establish several lemmas to complete the induction. In each iteration we will work with a vertex $x \in T$ whose children are leaves,

say $x_1, x_2, \dots, x_r, r > 0$. Each time when we modify the tree and the labels, we have to check the conditions **default-center**, **weight** and **load** for each vertex. (Namely x, x_i 's with $c(x_i) = true$, x_i 's with $c(x_i) = false$, and other vertices in T'). The modification of labels (from L to L' and vice versa) is done by specifying only the *changed part*, while others remain the same. In the following discussion it should be kept in mind that these invariants hold:

i.1 $a(x) \leq Z'$, if $c(x) = true$.

i.2 $b(x) + a(x) \leq Z'$, if $c(x) = false$.

We distinguish five cases, considering the status of x and of its children x_i 's and the situation whether the vicinity constraint is satisfied. They are summarized in Lemma 3.1 through Lemma 3.5.

Lemma 3.1 *Suppose $c(x) = true$. If there exists a child x_i with $c(x_i)$ also true, we let $T' = T \setminus \{x_i\}$ and $L' = L$, then we have $\Gamma_{L'}(T') = \Gamma_L(T) - 1$.*

Proof. As shown in Fig. 3.3.

1. Assume R' is optimal for (T', L', Z') , i.e. $|R'| = \Gamma_{L'}(T')$. Let $R = R' \cup \{x_i\}$, For x_i , since $x_i \in R$ the condition **default-center** is satisfied; also $\Lambda_R(x_i) = \{x_i\}$ and $a(x_i) \leq Z'$ (**i.1**), the condition **weight** is satisfied. For any vertex $w \in T, w \neq x$, we have $\Lambda_R(w) = \Lambda_{R'}(w)$ and $L(w) = L'(w)$. All the conditions are satisfied in T if they are satisfied in T' . So R is a feasible solution for (T, L, Z') , and $\Gamma_L(T) \leq |R| = |R'| + 1 = \Gamma_{L'}(T') + 1$.
2. Assume R is optimal for (T, L, Z') , i.e. $|R| = \Gamma_L(T)$. Since $c(x_i) = true, x_i \in R$. Let $R' = R \setminus \{x_i\}$. For all vertices $w \in T', \Lambda_{R'}(w) =$

$\Lambda_R(w)$ and $L'(w) = L(w)$. So R' is a feasible solution for (T', L', Z') , and $\Gamma_{L'}(T') \leq |R'| = |R| - 1 = \Gamma_L(T) - 1$.

□

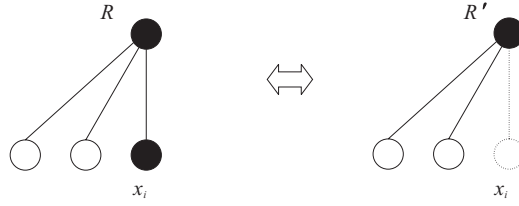


Figure 3.3: Lemma 3.1

Lemma 3.2 Suppose $c(x) = true$ and $\forall x_i, c(x_i) = false$. If

$$a(x) + \sum_{c(x_i)=false} a(x_i) \leq Z',$$

we let $T' = T \setminus \{x_1, \dots, x_r\}$, $a'(x) = a(x) + \sum_{c(x_i)=false} a(x_i)$, and we have $\Gamma_{L'}(T') = \Gamma_L(T)$.

Proof. See Fig. 3.4

1. Assume R' is optimal for (T', L', Z') , i.e. $|R'| = \Gamma_{L'}(T')$. Let $R = R'$, since $c(x) = c'(x) = true$, $x \in R$ and $x \in R'$. For x , it must satisfy condition **weight** in T' . Notice that $\Lambda_R(x) = \Lambda_{R'}(x) \cup \{x_i \mid c(x_i) =$

$false\}$, so in T ,

$$\begin{aligned}
\sum_{y \in \Lambda_R(x)} a(y) &= a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\
&= a'(x) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\
&= \sum_{y \in \Lambda_{R'}(x)} a(y) \\
&\leq Z'.
\end{aligned}$$

For any vertex $w \in T'$, $w \neq x$, we have $\Lambda_R(w) = \Lambda_{R'}(w)$ and $L(w) = L'(w)$, so all conditions are satisfied. For all x_i , $\Lambda_R(x_i) = \{x_i\}$, therefore condition **load** (now becomes $a(x_i) + b(x_i) \leq Z'$) is satisfied, by **(i.2)**. Thus R is a feasible solution for (T, L, Z) , and $\Gamma_L(T) \leq |R| = |R'| = \Gamma_{L'}(T')$.

2. Assume R is optimal for (T, L, Z') , i.e. $|R| = \Gamma_L(T)$. We claim that $\{x_1, \dots, x_r\} \cap R = \emptyset$. First of all there is no x_i with $c(x_i) = true$, so there is no default-center. Secondly if there exists some $x_j \in R$, then we can replace it by the unique parent of x . (If there is no such parent or the parent is already in R , then $a(x) = \sum a(x_i)$ is already no more than Z' , so we do not need x_j anyway, which contradicts the optimality of R .) After the replacement, $a(x) = \sum a(x_i) \leq Z'$ so the resulting solution is still feasible. Now $\{x_1, \dots, x_r\} \cap R = \emptyset$, and we let

$R' = R$. Notice that $\Lambda_{R'}(x) = \Lambda_R(x) \setminus \{x_i \mid c(x_i) = false\}$, so in T' ,

$$\begin{aligned}
\sum_{y \in \Lambda_{R'}(x)} a'(y) &= a'(x) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a'(y) \\
&= a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\
&= \sum_{y \in \Lambda_R(x)} a(y) \\
&\leq Z'.
\end{aligned}$$

For any vertex $w \in T'$, $w \neq x$, we have $\Lambda_{R'}(w) = \Lambda_R(w)$ and $L'(w) = L(w)$, so it satisfies all the conditions. Thus R' is a feasible solution for (T', L', Z') , and $\Gamma_{L'}(T') \leq |R'| = |R| = \Gamma_L(T)$.

□

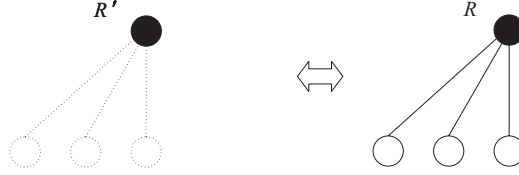


Figure 3.4: Lemma 3.2

Lemma 3.3 Suppose $c(x) = true$ and $\forall x_i, c(x_i) = false$. If

$$a(x) + \sum_{c(x_i)=false} a(x_i) > Z',$$

we let $T' = T$ and $c'(x_j) = true$, where $a(x_j) = \max_{1 \leq k \leq r} \{a(x_k)\}$, and we have $\Gamma_{L'}(T') = \Gamma_L(T)$.

Proof.

1. Assume R' is optimal for (T', L', Z') , i.e. $|R'| = \Gamma_{L'}(T')$. See Fig. 3.5. Then it can be easily checked that $R = R'$ is also a feasible solution for (T, L, Z') , since all the conditions are identical in both cases. Therefore $\Gamma_L(T) \leq |R| = |R'| = \Gamma_{L'}(T')$.
2. Assume R is optimal for (T, L, Z') , i.e. $|R| = \Gamma_L(T)$. See Fig. 3.6. First we have to claim that $\{x_1, \dots, x_r\} \cap R \neq \emptyset$. Otherwise x violates condition **weight**. So there exists some $x_k \in R$. Let $R' = R \setminus \{x_k\} \cup \{x_j\}$. For x ,

$$\begin{aligned} a'(x) &= a(x) + a(x_k) - a(x_j) \\ &\leq a(x) \leq Z'. \end{aligned}$$

For all the other vertices the conditions remain unchanged. So R' is a feasible solution for (T', L', Z') , and $\Gamma_{L'}(T') \leq |R'| = |R| = \Gamma_L(T)$.

□

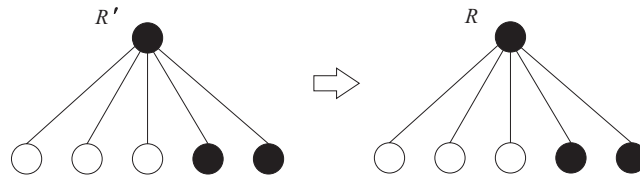


Figure 3.5: Case 1 in Lemma 3.3

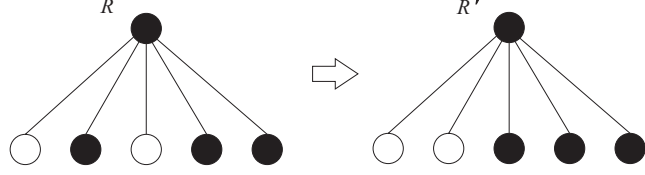


Figure 3.6: Case 2 in Lemma 3.3

Lemma 3.4 Suppose $c(x) = false$, if

$$\begin{aligned}
 a(x) + \sum_{c(x_i)=false} a(x_i) \\
 + \max\{a(x_i) \text{ for } c(x_i) = true; b(x_i) \text{ for } c(x_i) = false; b(x)\} > Z',
 \end{aligned} \tag{3.1}$$

we let $c'(x) = true$, $T' = T$, and we have that $\Gamma_{L'}(T') = \Gamma_L(T)$.

Proof.

1. Assume R' is optimal for (T', L', Z') , i.e. $|R'| = \Gamma_{L'}(T')$. Then $R = R'$ is also feasible for (T, L, Z') . Therefore $\Gamma_L(T) \leq |R| = |R'| = \Gamma_{L'}(T')$. See Fig. 3.7
2. Assume R is optimal for (T, L, Z') , i.e. $|R| = \Gamma_L(T)$. There are two cases to consider (as shown in Fig. 3.8):
 - (a) If $x \in R$, then since $c'(x) = true$, x must also be in R' . Let $R' = R$, then R' is feasible for (T', L', Z') . So R is a feasible solution for (T, L, Z') , and $\Gamma_L(T) \leq |R| = |R'| = \Gamma_{L'}(T')$.
 - (b) If $x \notin R$. Suppose $R \cap \{x_i \mid c(x_i) = false\} = K$, where $K = \{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$ and the vertices in K are sorted in ascending order according to their $a(x_{i_j})$'s. Let $R' = R \setminus \{x_{i_1}\} \cup$

$\{x\}$ (Notice that $|K| \geq 1$. Otherwise (3.1) becomes $a(x) + \max\{a(x_i) \text{ for } c(x_i) = \text{true}; b(x)\} > Z'$, which means either x violates condition **load** or some x_i violates condition **weight**. For x , consider the x_{i_1} as described above in T , x_{i_1} satisfies condition **weight**:

$$\sum_{y \in \Lambda_R(x_{i_1})} a(y) \leq Z', \quad (3.2)$$

since $x_{i_1} \in R$. In T' , $\Lambda_{R'}(x) = \Lambda_R(x) \cup \{x_{i_1}\}$ in T , which is in turn identical to $\Lambda_R(x_{i_1})$. Therefore

$$\begin{aligned} \sum_{y \in \Lambda_{R'}(x)} a'(y) &= \sum_{y \in \Lambda_R(x)} a(y) + a(x_{i_1}) \\ &\leq \sum_{y \in \Lambda_R(x_{i_1})} a(y) \\ &\leq Z'. \end{aligned} \quad \text{by (3.2)}$$

For $x_i \in R'$ (i.e. x_{i_2}, \dots, x_{i_k}) $\Lambda_{R'}(x_i) = \{x_i\}$ and it satisfies condition **weight**. For $x_i \notin R'$, $\Lambda_{R'}(x_i) = \{x_i\}$ and $b'(x_i) + \sum_{y \in \Lambda_{R'}(x_i)} a'(y) = b(x_i) + a(x_i) \leq Z'$. For $w \in T'$, $\Lambda_{R'}(w)$ is $\Lambda_R(w)$ subtracted by possibly some vertices, and $L'(w) = L(w)$. So all the conditions are satisfied. So R is a feasible solution for (T, L, Z') , and $\Gamma_L(T) \leq |R| = |R'| = \Gamma_{L'}(T')$.

Therefore we have $\Gamma_{L'}(T') = \Gamma_L(T)$. □

Lemma 3.5 *Suppose $c(x) = \text{false}$. If*

$$\begin{aligned} a(x) + \sum_{c(x_i)=\text{false}} a(x_i) \\ + \max\{a(x_i) \text{ for } c(x_i) = \text{true}; b(x_i) \text{ for } c(x_i) = \text{false}; b(x)\} \leq Z', \end{aligned} \quad (3.3)$$

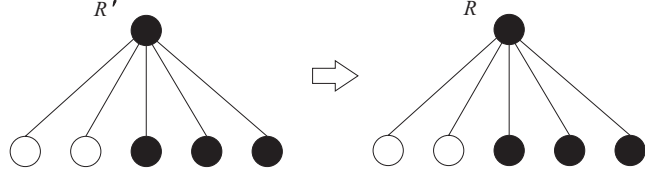


Figure 3.7: Case 1 in Lemma 3.4

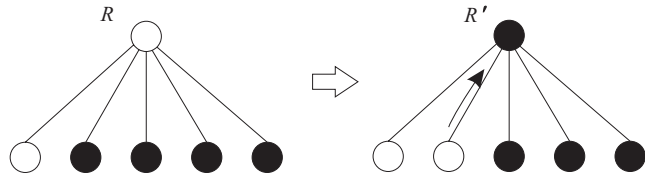


Figure 3.8: Case 2 in Lemma 3.4

we let

$$h = |\{x_i \mid c(x_i) = true\}|,$$

$$T' = T \setminus \{x_1, \dots, x_r\},$$

$$a'(x) = a(x) + \sum_{c(x_i)=false} a(x_i) \quad , \text{ and}$$

$$b'(x) = \max\{a(x_i) \text{ for } c(x_i) = true; b(x_i) \text{ for } c(x_i) = false; b(x)\},$$

and we have $\Gamma_{L'}(T') = \Gamma_L(T) - h$.

Proof.

1. Assume R' is optimal for (T', L', Z') , i.e. $|R'| = \Gamma_{L'}(T')$. Let $R = R' \cup \{x_i \mid c(x_i) = true\}$. There are two cases to consider:

- (a) If $x \in R'$, then x satisfies $\sum_{y \in \Lambda_{R'}(x)} a'(y) \leq Z'$ and $\Lambda_R(x) =$

$\Lambda_{R'}(x) \cup \{x_i \mid c(x_i) = false\}$, therefore

$$\begin{aligned}
\sum_{y \in \Lambda_R(x)} a(y) &= a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\
&= a'(x) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a'(y) \\
&= \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq Z'.
\end{aligned}$$

For $x_i \in R$, $\Lambda_R(x_i) = \{x_i\}$ and it satisfies condition **weight**. For $x_i \notin R$, $\Lambda_{R'}(x_i) = \{x_i\}$ and $b'(x_i) + \sum_{y \in \Lambda_{R'}(x_i)} a'(y) = b(x_i) + a(x_i) \leq Z'$. For $w \in T'$, $\Lambda_R(w) = \Lambda_{R'}(w)$ and $L(w) = L'(w)$, so all conditions are satisfied and R is feasible for (T, L, Z') . See Fig. 3.9.

(b) If $x \notin R'$, then x satisfies

$$b'(x) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \leq Z' \quad (3.4)$$

and $\Lambda_R(x) = \Lambda_{R'}(x) \cup \{x_i \mid c(x_i) = false\}$. See Fig. 3.10. For x ,

$$\begin{aligned}
b(x) + \sum_{y \in \Lambda_R(x)} a(y) &= b(x) + a(x) + \sum_{c(x_i)=false} a(x_i) \\
&\quad + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\
&= b(x) + a'(x) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\
&\leq b'(x) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq Z'. \quad \text{by (3.4)}
\end{aligned}$$

For $x_i \in R$, $\Lambda_R(x_i) = \{x_i\} \cup \Lambda_{R'}(x) \cup \{x_j \mid c(x_j) = false\}$, and

$$\begin{aligned}
\sum_{y \in \Lambda_R(x_i)} a(y) &= a(x_i) + \sum_{y \in \Lambda_{R'}(x)} a(y) + \sum_{c(x_j)=false} a(x_j) \\
&= a(x_i) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq b'(x) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq Z'. \tag{by (3.4)}
\end{aligned}$$

For $x_i \notin R$, $\Lambda_R(x_i) = \Lambda_{R'}(x) \cup \{x_j \mid c(x_j) = false\}$, and

$$\begin{aligned}
b(x_i) + \sum_{y \in \Lambda_R(x_i)} a(y) &= b(x_i) + \sum_{y \in \Lambda_{R'}(x)} a(y) + \sum_{c(x_j)=false} a(x_j) \\
&= b(x_i) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq b'(x) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq Z'. \tag{by (3.4)}
\end{aligned}$$

For $w \in T'$, the only different part is for those with $\Lambda_R(w) = \Lambda_{R'}(w) \cup \{x_i \mid c(x_i) = false\}$. So if $w \in R'$, then w satisfies $\sum_{y \in \Lambda_{R'}(w)} a'(y) \leq Z'$, and

$$\begin{aligned}
\sum_{y \in \Lambda_R(w)} a(y) &= a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(w) \setminus \{x\}} a(y) \\
&= a'(x) + \sum_{y \in \Lambda_{R'}(w) \setminus \{x\}} a'(y) \\
&= \sum_{y \in \Lambda_{R'}(w)} a'(y) \\
&\leq Z';
\end{aligned}$$

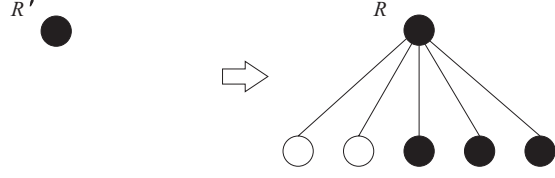


Figure 3.9: Case (a) of case 1 in Lemma 3.5

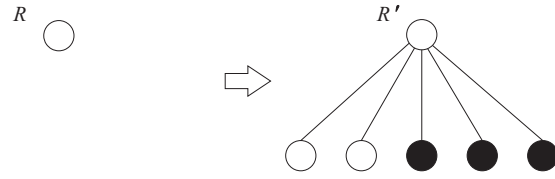


Figure 3.10: Case (b) of case 1 in Lemma 3.5

and if $w \in R'$, then w satisfies $b'(w) + \sum_{y \in \Lambda_{R'}(w)} a'(y) \leq Z'$, and

$$\begin{aligned}
 b(w) + \sum_{y \in \Lambda_R(w)} a(y) &= b(w) + a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(w) \setminus \{x\}} a(y) \\
 &= b'(w) + a'(x) + \sum_{y \in \Lambda_{R'}(w) \setminus \{x\}} a'(y) \\
 &= b'(w) + \sum_{y \in \Lambda_{R'}(w)} a'(y) \\
 &\leq Z'.
 \end{aligned}$$

So R is feasible for (T, L, Z') .

Therefore we have $\Gamma_L(T) \leq |R| = |R'| + h = \Gamma_{L'}(T')$.

2. Assume R is optimal for (T, L, Z') , i.e. $|R| = \Gamma_L(T)$. First we claim that $\{x_1, \dots, x_r\} \cap R = \{x_i \mid c(x_i) = true\}$. This can be assured by replacing the x_j whose $c(x_j) = false$ and $x_j \in R$ with some other

vertex. (By a similar reasoning as in the proof of Lemma 3.2.) Let $R' = R \setminus \{x_i \mid c(x_i) = true\}$. Again there are two cases to consider:

- (a) If $x \in R$, then x satisfies $\sum_{y \in \Lambda_R(x)} a(y) \leq Z'$ in T . See Fig. 3.11. For x , $\Lambda_{R'}(x) = \Lambda_R(x) \setminus \{x_i \mid c(x_i) = false\}$, and

$$\begin{aligned} \sum_{y \in \Lambda_{R'}(x)} a'(y) &= a'(x) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a'(y) \\ &= a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a(y) \\ &= \sum_{y \in \Lambda_R(x)} a(y) \\ &\leq Z'. \end{aligned}$$

For any vertex $w \in T'$, $w \neq x$, we have $\Lambda_{R'}(w) = \Lambda_R(w)$ and $L'(w) = L(w)$, so it satisfies all the conditions. So R' is a feasible solution for (T', L', Z') .

- (b) If $x \notin R$, then in T

$$\begin{cases} b(x) + \sum_{y \in \Lambda_R(x)} a(y) \leq Z', \\ b(x_i) + \sum_{y \in \Lambda_R(x_i)} a(y) \leq Z' & \text{for } c(x_i) = false, \text{ and} \\ \sum_{y \in \Lambda_R(x_i)} a(y) \leq Z' & \text{for } c(x_i) = true. \end{cases}$$

See Fig. 3.12. Notice that $\Lambda_R(x_i) = \Lambda_R(x)$ for $c(x_i) = false$, and $\Lambda_R(x_i) = \Lambda_R(x) \cup \{x_i\}$. The above implies

$$\begin{aligned} &\max\{a(x_i) \text{ for } c(x_i) = true; \\ &\quad b(x_i) \text{ for } c(x_i) = false; b(x)\} + \sum_{\Lambda_R(x)} a(y) \leq Z'. \quad (3.5) \end{aligned}$$

For x in T' ,

$$\begin{aligned}
b'(x) + \sum_{y \in \Lambda_{R'}(x)} a'(y) &= b'(x) + a'(x) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a'(y) \\
&= b'(x) + a(x) + \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(x) \setminus \{x\}} a'(y) \\
&= b'(x) + \sum_{y \in \Lambda_{R'}(x)} a'(y) \\
&\leq Z'.
\end{aligned}
\tag{3.5}$$

For $w \in T'$, the only different part is for those with $\Lambda_{R'}(w) = \Lambda_R(w) \setminus \{x_i \mid c(x_i) = false\}$, for which

$$\begin{aligned}
\sum_{y \in \Lambda_{R'}(w)} a'(y) &= a'(x) + \sum_{y \in \Lambda_{R'}(w) \setminus \{x\}} a'(y) \\
&= a(x) \sum_{c(x_i)=false} a(x_i) + \sum_{y \in \Lambda_{R'}(w) \setminus \{x\}} a(y) \\
&= \sum_{y \in \Lambda_R(w)} a(y) \\
&\leq Z',
\end{aligned}$$

so the conditions are satisfied and R' is feasible for (T', L', Z') .

Thus we have $\Gamma_{L'}(T') \leq |R'| = |R| - h = \Gamma_L(T)$.

So the lemma follows. \square

Now we are ready to present the algorithm.

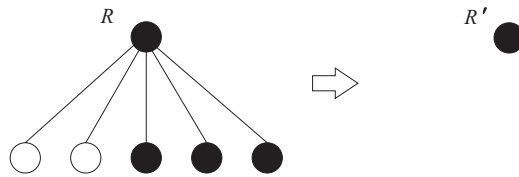


Figure 3.11: Case (a) of case 2 in Lemma 3.5

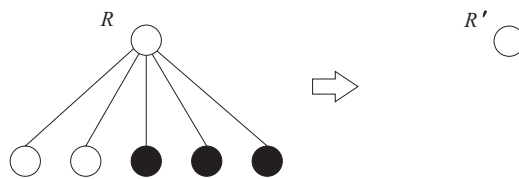


Figure 3.12: Case (b) of case 2 in Lemma 3.5

Algorithm RCT(T, L, Z'). A reporting center problem for a tree.

Input A tree $T = (V, E)$, a label $L(x) = (a(x), b(x), c(x))$ defined on each vertex $x \in T$, and a vicinity constraint $Z' > 1$.

Output A reporting center set for (T, L, Z') of minimum cardinality.

Method

```

Construct the post-order list  $\ell_{PO}$  of all the vertices in  $T$ ;
while(  $(x \leftarrow$  the next vertex in  $\ell_{PO})$  is not  $NULL$ )
  /* We shall do nothing on leaves */
  if ( $x$  is a leaf) continue;
  /* Now that  $x$  is an internal node with all children visited */
  Let  $children(x) = \{x_1, x_2, \dots, x_r\}$  be the children of  $x$ ;
  if ( $c(x) = true$ ) then
     $A \leftarrow a(x) + \sum_{c(x_i)=false} a(x_i)$ ;
    /* case in Lemma 3.3 */
    if( $A > Z'$ ) then  $A \leftarrow Pick(a(x), children(x))$ ;
     $a(x) \leftarrow A$ ; /* reduce to case in Lemma 3.2*/
  else /*  $c(x) = false$  */
     $A \leftarrow a(x) + \sum_{c(x_i)=false} a(x_i)$ ;
     $B \leftarrow \max\{a(x_i) \text{ for } c(x_i) = true; b(x_i) \text{ for } c(x_i) = false; b(x)\}$ ;
    if ( $A + B > Z'$ ) then /* case in Lemma 3.4 */
       $c(x) \leftarrow true$ ;
      /* now it is like the case in Lemma 3.3 */
      if( $A > Z'$ ) then  $A \leftarrow Pick(a(x), children(x))$ ;
       $a(x) \leftarrow A$ ; /* reduce to case in Lemma 3.2*/
    else /*  $A + B \leq Z'$ , case in Lemma 3.5*/
       $a(x) \leftarrow A$ ;
       $b(x) \leftarrow B$ ;
  end while;
Collect all vertices with  $c(x) = true$  into  $R$ ;
return  $R$ ;

```

Algorithm $\text{Pick}(A_{\text{prefix}}, X)$. A subroutine that picks some vertices from the given set X to be reporting centers, until the output integer is no greater than Z' .

Input An integer A_{prefix} , and a set of vertices X .

Output An integer $\leq Z'$.

Method

```

Let  $X_{\text{left}}, X_{\text{right}}$  be subsets of  $X$ ;
 $X_{\text{left}} \leftarrow \emptyset, X_{\text{right}} \leftarrow \emptyset$ ;
 $X \leftarrow \{x \mid x \in X, c(x) = \text{false}\}$ ; /*First collect those unselected vertices.*/
/*Loop invariant:  $A_{\text{prefix}} \leq Z'$ . */
while ( true )
    if (  $|X| = 1$  and  $X = \{x\}$  ) then /*Handle the leaf termination.*/
        if (  $A_{\text{prefix}} + a(x) > Z'$  ) then  $c(x) \leftarrow \text{true}$ ;
        else  $A_{\text{prefix}} \leftarrow A_{\text{prefix}} + a(x)$ ;
        break;
    else
         $a_{\text{med}} \leftarrow \text{Find\_Median\_a}(X)$ ;
         $X_{\text{left}} \leftarrow \{x \mid a(x) \leq a_{\text{med}}\}$ ;
         $X_{\text{right}} \leftarrow \{x \mid a(x) > a_{\text{med}}\}$ ;
         $A_{\text{left}} \leftarrow \sum_{x \in X_{\text{left}}} a(x)$ ;
        if (  $(A_{\text{prefix}} + A_{\text{left}}) > Z'$  ) then
             $c(x) \leftarrow \text{true}$  for all  $x \in X_{\text{right}}$ ;
             $X \leftarrow X_{\text{left}}$ ;
        else
             $A_{\text{prefix}} \leftarrow A_{\text{prefix}} + A_{\text{left}}$ ;
             $X \leftarrow X_{\text{right}}$ ;
    end while;
return  $A_{\text{prefix}}$ ;

```

Theorem 3.6 *The reporting center problem for a tree can be solved by algorithm RCT in $O(n)$ time.*

Proof. The correctness follows from Lemma 3.1 through Lemma 3.5. For the running time, notice that the algorithm traverses the tree in post-order. Besides the *Pick* subroutine, all other operations spend constant time on each child. This sums up as $O(n)$. Now we analyze the time complexity of the *Pick* subroutine. Suppose the set of vertices passed to *Pick* is X . Collecting unselected vertices takes $O(|X|)$ time. The leaf termination takes $O(1)$ time. The *Find_Median_a(x)* is a subroutine that returns the median value of $a(x)$'s in the set X , which is known to be an $O(|X|)$ operation. Partitioning X into X_{left} and X_{right} can be done by comparing the a -value for each item with a_{med} , and takes $O(|X|)$ time totally. Computing A_{left} takes $O(|X|)$ time. So the first iteration of the while-loop takes $O(|X|)$ time, after which the X is set to be either X_{left} or X_{right} and the size is halved. The second iteration therefore takes $O(|X|/2)$ time, and so on. The total time spent in the while-loop is therefore $O(|X|) + O(|X|/2) + O(|X|/4) + \dots = O(|X|)$. Since the input X of the *Pick* subroutine is always the set of children of some vertex, a constant time is spent on each child. The running time of the whole algorithm is thus $O(n)$.

□

Chapter 4

Conclusion

We have studied the reporting center problem for general interval graphs and trees. We have presented an algorithm that solves the reporting center problem for general interval graphs. We have also presented an alternative perspective on the problem, called the bricks transformation, to facilitate (possibly) future work devoted to this problem. Finally an algorithm that solves the reporting center problem for trees is presented, improving the previous results. However, our algorithm for general interval graphs still has an Z -exponent in its running time. It remains open if there is an algorithm that can avoid this exponent, or if it is an NP -complete problem with respect to Z . It shall be noted that this hardness is different from the previously proved NP -hardness for general graphs. For general graphs, even if the vicinity constraint is only 2, it is NP -complete, while our result for interval graphs runs in polynomial time if the vicinity constraint is an *arbitrary constant*. Another problem, i.e given a number r of reporting centers, how to find a placement of these reporting centers in order to minimize the maximum vicinity of each reporting center, is also a worthwhile problem to investigate further.

Bibliography

- [1] I.F. Akyldiz, J.S.M. Ho, Dynamic Mobile User Location Update for Wireless PCS Networks, *Wireless Networks* 1 (1995) 187-196.
- [2] J.S.M. Ho, I.F.Akyldiz, Mobile User Location Update and Paging under Delay Constraints, *Wireless Networks* 1 (1995) 413-425.
- [3] A. Bar-Noy, I. Kessler, Tracking Mobile Users in Wireless Communications Networks, *IEEE Trans. Inform. Theory* 39 (1993) 1877-1886.
- [4] A. Bar-Noy, I Kessler, M. Sidi, Mobile Users: to Update or Not to Update?, *Wireless Networks* 1 (1995) 175-185.
- [5] A. Bhattacharya, S.K. Das, LeZi-Update: An Information Theoretic Approach to Track Mobile Users in PCS Networks, *MobiCom'99*, Seattle, 1999.
- [6] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
- [7] S. Olariu, M.C. Pinotti, L.Wilson, Greedy Algorithms for Tracking Mobile Users in Special Mobility Graphs, *Discrete Applied Math.* 121 (2002) 215-227.

- [8] U. Madhow, M.I. Honig, K. Steiglitz, Optimization of Wireless Resources for Personal Communications mobility Tracking, *IEEE/ACM Trans. Networking* 3 (1995) 688-707.
- [9] C. Rose, Minimizing the Average Cost of Paging and Registration: a Timer-Based Method, *Wireless Networks* 2 (1996) 109-116.
- [10] C. Rose, R. Yates, Minimizing the Average Cost of Paging under Delay Constraints, *Wireless Networks* 1 (1995) 211-219.
- [11] D.B. West, *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, 2001.