Tries for spatial data range search

by

Linke Bu

TR03-160, February 2003 (this is an unaltered version of the author's MCS Thesis)

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

Abstract

Orthogonal range search finds and reports all objects falling in a specified query window. An algorithm to solve orthogonal range search problem on k dimensions using tries is developed and analyzed. Two hyper-rectangles intersect if and only if their sides on every dimension in the data space intersect. The algorithm reports the set HR ($HR \subseteq$ input data set D, $|HR| = A$, $|D| = n$) of k-dimensional (k-d) hyper-rectangles intersecting a k-d axis-aligned query hyper-rectangle W, and supports dynamic operations. We assume that the input data set D and query hyper-rectangle W drawn from a uniform, random distribution. The storage $S(n,k) = \Theta(kBn)$ and the expected preprocessing time $P(n,k) = \Theta(kBn)$ for a trie containing n k-d hyper-rectangles where B is the number of bits for representing a coordinate value. The expected orthogonal range search time $Q(n,k) = O(n^{\alpha})$ for $0.5 \leq \alpha < 1$ and α a complicated function of n and k. Experimental research with randomly generated data and query hyper-rectangles (and various values of k and n up to 10 and 100,000, respectively) is used to empirically validate the expected range search time. Our algorithm compares favorably to the existing dynamic orthogonal range search algorithm when k is large.

Acknowledgements

First and foremost I would like to express my appreciation to my supervisor, Dr. Bradford G. Nickerson for his suggestion of this specific topic, his indispensable support and encouragement. He contributed a great deal of his time, effort, and thought to the work presented in this dissertation. During my master's program, I also received generous financial support from him. Without his support and invaluable guidance, I could not have finished my thesis.

I am grateful to the Faculty of Computer Science for their financial support. Thanks go to my thesis committee members, Dr. Joe Horton, Dr. Patricia Evans, Dr. Y.C. Lee for their serious examination of this thesis and many constructive suggestions. Thanks must also go to Mrs. Linda Sales, Mrs. Dianne Muir, Mrs. Jennifer Bishop and all the secretaries for their patience and readiness to provide administrative help, as well as Mr. Sean Seeley and all system support staff for their technical assistance.

A special thank you is extended to Dr. Jane M. Fritz for her kindly help and encouragement.

A thank you is extended to Dr. Weichang Du whose knowledge and technical skills help me a lot in my study, to Dr. Roman Mureika and to Oleg D. Goloubitski for their helpful discussions about probability and set theory, and to my friends in our Faculty of Computer Science for the encouragement and the joy and fun we shared.

Special thanks are extended to Dr. Virendra C. Bhavsar, Chris MacPhee of Advanced Computational Research Laboratory in UNB, Darryl Reid and other staff in the Memorial University Advanced Computation and Visualization Centre for their kindly help in making my code run on herzberg.physics.mun.ca.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

k = number of dimensions for the stated problem.

n = number of data objects to be searched.

B = number of bits for representing a coordinate value in binary.

$Q(n, k)$ = time for range search (including reporting).

$S(n, k)$ = space for data storage.

$P(n, k)$ = time for preprocessing data.

A = number of data objects in range.

D = set of axis-aligned orthogonal data objects.

HR = the set of data in range.

Chapter 1

Introduction

1.1 Overview

Orthogonal range search [Knut73-2] [BeFr79] finds and reports all objects intersecting a specified query hyper-rectangle W. This thesis considers the case where the objects to be searched for are k-dimensional (k-d) hyper-rectangles. We investigate the use of k-d tries (e.g. Flajolet and Puech [FlPu86], Shang [Shan01]) as a data structure to support orthogonal range search.

Given a collection F of records, each containing several attributes or keys, an orthogonal range query asks for all records with key values each inside specified ranges. Range search is the process of reporting the appropriate records intersecting the query range. The range search problem can be interpreted geometrically by considering the record attributes as coordinates and the k values for each record as a point in a k-dimensional coordinate space [BeFr79]. The definition for orthogonal range search is

as follows:

Definition 1 *For a data space R^k, for k = number of dimensions, orthogonal range search is defined as finding and reporting the set $HR, (|HR| = A, HR \subseteq D, D =$ set of axis-aligned orthogonal data objects represented as hyper-rectangles, $|D| = n)$ of data intersecting the query hyper-rectangle $W = \{[L_1, H_1],\ [L_2, H_2], \cdots, [L_k, H_k]\}$, where $[L_j, H_j]$ represents a range for dimension j of the query hyper-rectangle.*

The classical orthogonal range search of Bentley and Friedman [BeFr79] is generalized to allow each of the n records in the collection F to be defined by a coordinate (or key) range.

In the following discussion, we use the following notation:

$Q(n, k)$ represents the time complexity of orthogonal range search on a data structure containing n k-dimensional objects.

$P(n, k)$ is the cost of processing n k-dimensional objects into a data structure supporting orthogonal range search. In this thesis, $P(n, k)$ refers to the building time for inserting all the n k-dimensional hyper-rectangles into a trie T.

$S(n, k)$ is the amount of storage required by the data structure.

1.2 Previous Work

Data structures supporting orthogonal range search on such records have been constructed, with one of the most popular being the k-dimensional variant of the B-tree known as the R-tree introduced by Guttman [Gutt84]. Edelsbrunner [Edel83]

introduced the d-fold rectangle tree to support orthogonal range search on k-d hyper-rectangles with $S(n,k)=\Theta(n\log^{k-1} n)$, $P(n,k)=O(n\log^{k} n)$, and $Q(n,k)=O(\log^{2k-1} n+A)$.

Bentley et al. [BeFr79] [BeMa80] review several data structures for k-dimensional point range searching including sequential scan, projection, cells, k-d trees, range trees and k-ranges. The d-fold $BB(\alpha)$ tree [Luek78] [LuWi82] has worst-case total time of $O(n\log^{k} n)$ for n operations (where an operation can be to insert or delete a point, or to perform a range search). The k-d tree [Bent75] is a binary tree. The k-d tree requires $O(kn)$ space and a total path length of $O(n\log n)$ for n k-d points inserted in random order. The analysis of range search for balanced k-d trees shows that $Q(n,k)=O(sn^{1-1/k}+A)$ for s of the k coordinates restricted to a subrange [LeWo77], and $(k-s)$ of the coordinates unspecified. Devroye et al. [DJZ00] analyzed range search on squarish k-d trees and random k-d trees [CDZ01]. Fredman constructed a model for complexity analysis of range search [Fred81-a] [Fred81-b]. Lower bounds for range search were studied by Chazelle [Chaz90-a] [Chaz90-b], who showed that a sequence of n operations for insertion, deletion, and reporting points in a given range costs $\Omega(n(\log n)^k)$. Bentley and Maurer [BeMa80] investigated three k-ranges for range search. They showed that one level k-ranges had $Q(n,k)=O(k\log n+A)$, $S(n,k)=P(n,k)=O(n^{2k-1})$, and multi-level k-ranges require linear space, $P(n,k)=O(n\log n)$ and $Q(n,k)=O(n^{\epsilon})$. Merrett et al. [MSZ96] introduced the zoom trie for spatial data display, whose upper levels are used for lower resolutions, with the leaf level used for full resolution. Chazelle [Chaz88] gives a comprehensive

overview of data structures for k-dimensional searching, including the description of a k-dimensional rectangle reporting algorithm (supporting dynamic operations) with $Q(n,k) = O(A(\log(\frac{2n}{A})^2) + \log^{k-1} n)$, which is close to the lower bound.

There are thirteen possible relationships between two intervals [Alle83]. Shang's approach achieves interval containment by superimposing a PR-Trie with a zoom trie [Shan01]. In Shang's thesis, one-dimensional interval relationships are represented as queries in two dimensions. In this paper, we consider only one relationship (intersection) for orthogonal range search.

As pointed out in Flajolet and Puech [FlPu86], 1-d tries tend to be better balanced than 1-d search trees. For k-dimensional search, this improved balance can lead to asymptotically smaller search times. Our research investigate k-d tries for orthogonal range search which, to our knowledge, has not been investigated previously.

1.3 Thesis Objectives

The principle objective of this research is to explore the use of tries for combined text and spatial data range search. As part of this primary objective, the subgoals are followed:

- Review and analyze data structures that permit k-dimensional range search.
- Design an efficient algorithm using tries to support range search on spatial data which also supports dynamic operations.

- Determine the average and worst case storage of tries storing n data objects.
- Determine the preprocessing time of this algorithm.
- Given assumptions about the random distribution of the spatial data, what is the expected range search time?
- Perform empirical verification of the results with large (e.g. $n = 1 \times 10^5$) randomly generated data sets.

In the following pages, each of these targets is solved. The asymptotic running time of orthogonal range search for k-d tries is analyzed via calculating the expected number of nodes visited in tries.

Chapter 2

Search Tries

2.1 Definition of tries

Tries were introduced by Rene de la Briandais [Bria59]. E. Fredkin [Fred60] and Donald E. Knuth ([Knut73-2], §6.3) developed them further. The word "trie" originates from the word "re*trie*val". If we consider a key as a sequence of characters, instead of comparing the whole key, each comparison is based on a single character of information about the key along the tree. The same idea exists in the thumb index for looking for a word in a dictionary. For example, we build a 26-ary tree each node leads to next level's 26 possible letters and so 26 branches. If there are no words beginning with the letters 'dd', 'dz', we remove those branches and nodes from the tree, and what is left is a trie.

Tries uses characters or digital decomposition of keys to direct the branching [Gonn91]. A digital tree or trie is a data structure that does not store any information

on internal nodes. Tries store the information along the paths. The direction to take at a certain level ℓ in a trie is determined by the ℓ^{th} value of the key.

The trie data structure has the following properties:

- Internal nodes in a trie are empty. The path from the root to a leaf represents a key.
- The trie's shape is independent of the keys' input order; the shape is determined only by the input keys.
- Tries do not require rebalancing for dynamic operations (e.g. deletion, insertion).
- A trie is efficient in storing data since the trie stores the keys' common parts only once.

Tries can be represented in different ways. The straightforward method is tabular form ([Fred60] [Morr68] [Knut73-2] [RBK89]). Table 2.1 gives a tabular form example for keys: bifurcate, binary, ordinary, patricia, peano, pillar, porch, tree, and trie. These keys will also be used later for other trie representations. A linked list will overcome the table representation's shortcoming of too many empty entries in the table. René de la Briandais [Bria59] first recommended a linked list for each node vector to save memory space. Every node in the linked list trie is a linked list of pointers which only go to the right side of the node ([Suss63] [Knut73-2]). Figure 2.1 gives an example of a de la Briandais Trie. Compressed tries (C-tries, [Maly76]) and

bitstrings ([Oren82]) are pointerless trie structures which store bit arrays indicating the links to replace the right outgoing pointer of a linked list representation.

Table 2.1: Tabular form of tries for keys bifurcate, binary, ordinary, patricia, peano, pillar, porch, tree, trie.

	1	2	3	4	5	6	7	8	9	10	11	12	13
a					6		peano						
b	2												
c													
d													
e					7						12	tree	trie
f			bifurcate										
g													
h													
i		3			8						13		
j													
k													
l								pillar					
m													
n			binary										
o	4				9								
p	5												
q													
r				ordinary					porch	11			
s													
t	10					patricia							
u													
v													
w													
x													
y													
z													

A full trie is a tree whose nodes, including leaves, will not store information, and the height of the full trie is equal to the maximum length of a key. Every path from the root down to a leaf builds one key. The branch at level d is determined by the d^{th} character of the key. Figure 2.2 gives an example of a full trie. Figures 2.2, 2.3, and 2.4 use the key set: bifurcate, binary, ordinary, patricia, peano, pillar, porch, tree,

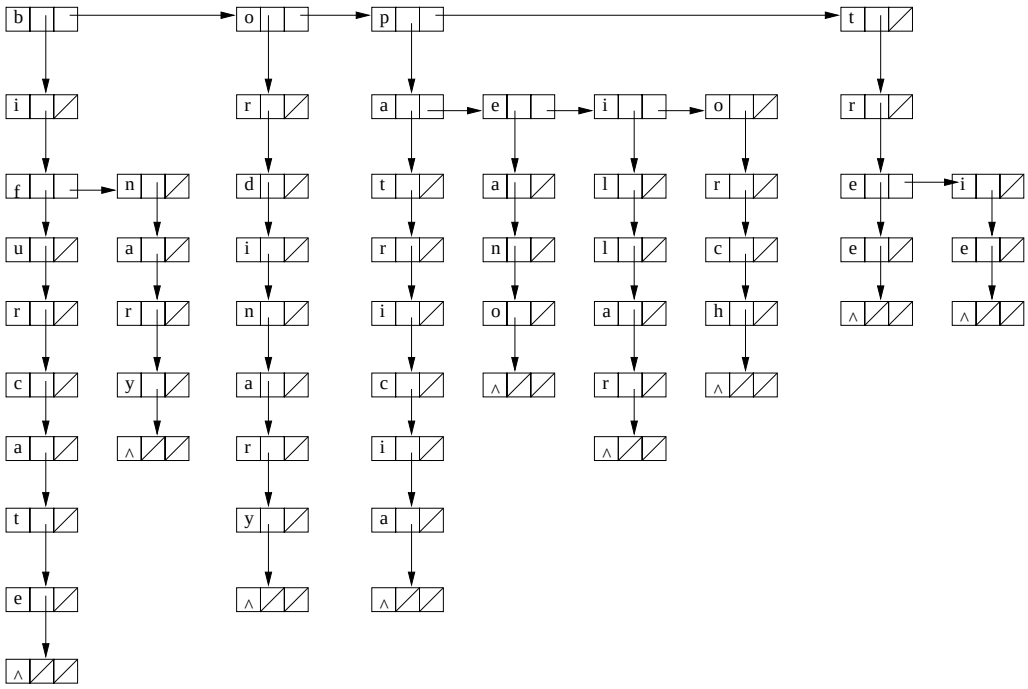


Figure 2.1: De la Briandais trie for keys bifurcate, binary, ordinary, patricia, peano, pillar, porch, tree, trie. $\wedge$ is used to identify the end of a key.

trie.

Binary tries are a special kind of trie which follows the rule of direction determined by the d^{th} bit information of the key: branch left if 0 and branch right if 1. There are three types of binary tries [Shan01]: full binary trie [CoSe77], ordinary binary trie [Fred60], and Patricia binary trie [Morr68].

An ordinary trie is a pruned trie where all the leaf's parents will be the last bifurcating node in the corresponding full trie, and all the nodes between the leaves and the last bifurcating node have been removed. Figure 2.3 gives an example of an ordinary trie.

A Patricia trie and an ordinary trie store eliminated information in their leaves. A Patricia trie removes all the single descendant nodes. These skipped symbols are

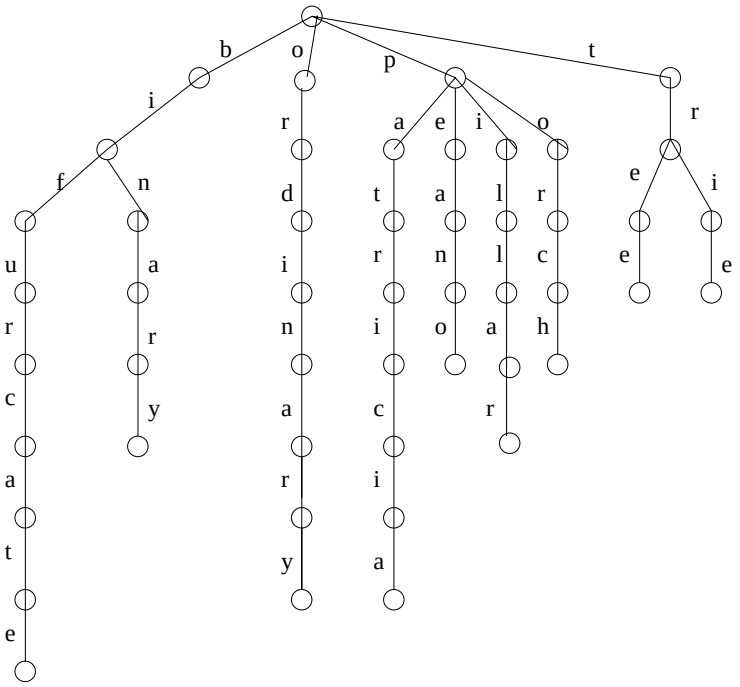


Figure 2.2: A full trie with the same keys as Figure 2.1.

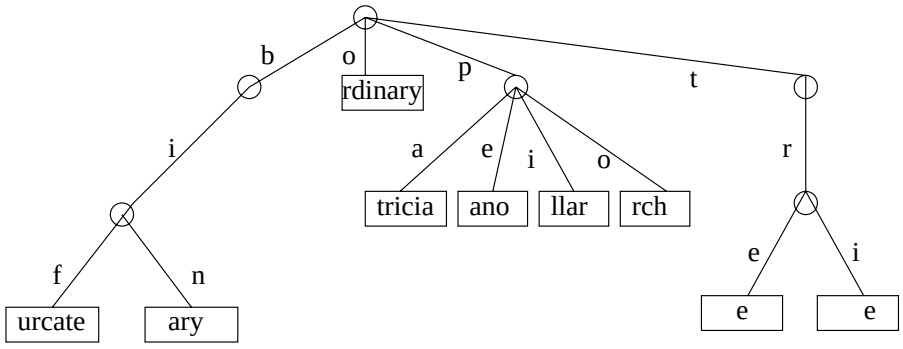


Figure 2.3: Example ordinary trie.

stored in the internal nodes or in the leaves (Figure 2.4).

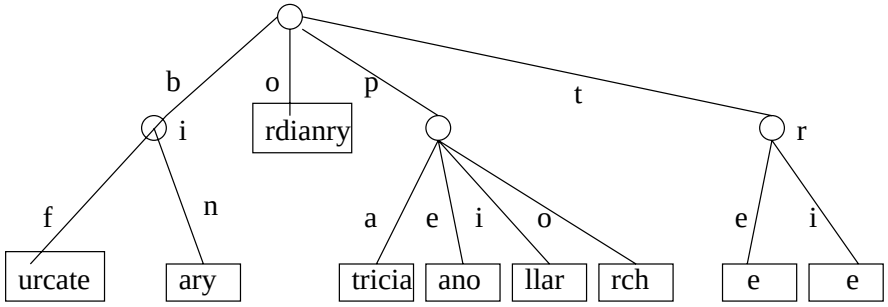


Figure 2.4: Example Patricia trie.

Binary tries store bits strings. Each node has at most two outgoing branches. If a bit is 0, go left; if the bit is 1, go right.

Figure 2.5 illustrates a full binary trie, Figure 2.6 shows an ordinary binary trie, and Figure 2.7 is a Patricia binary trie for the keys *peano, porch, tree, trie, try*. In the ordinary binary trie and the Patricia binary trie, internal nodes and leaves store the bit sequences representing the skipped nodes and branches. The ASCII codes for the keys are:

peano: 01110000 01100101 01100001 01101110 01101111

porch: 01110000 01101111 01110010 01100011 01101000

tree: 01110100 01110010 01100101 01100101

trie: 01110100 01110010 01101001 01100101

try: 01110100 01110010 01111001

For example, the leaf node in Figure 2.6 with 5 and 11001 sealed in brackets means 5 branches having been skipped and the skipped key is 11001.

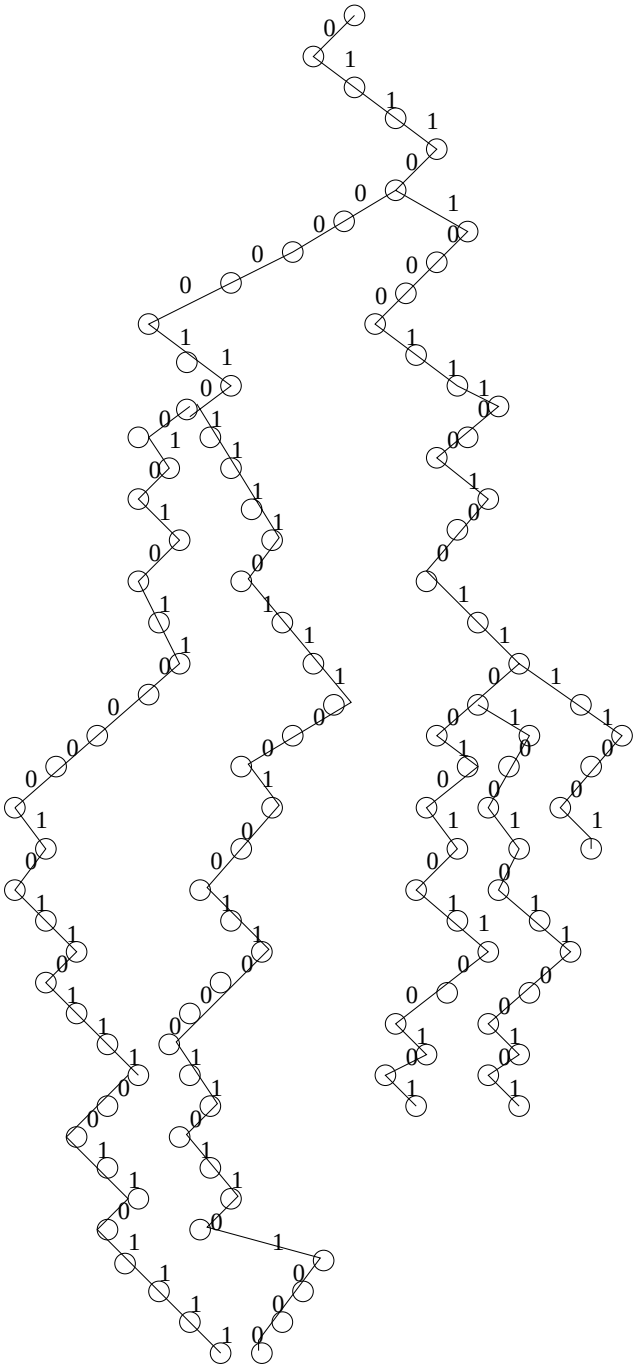


Figure 2.5: Full binary trie for the set of keys {peano, porch, tree, trie, try}.

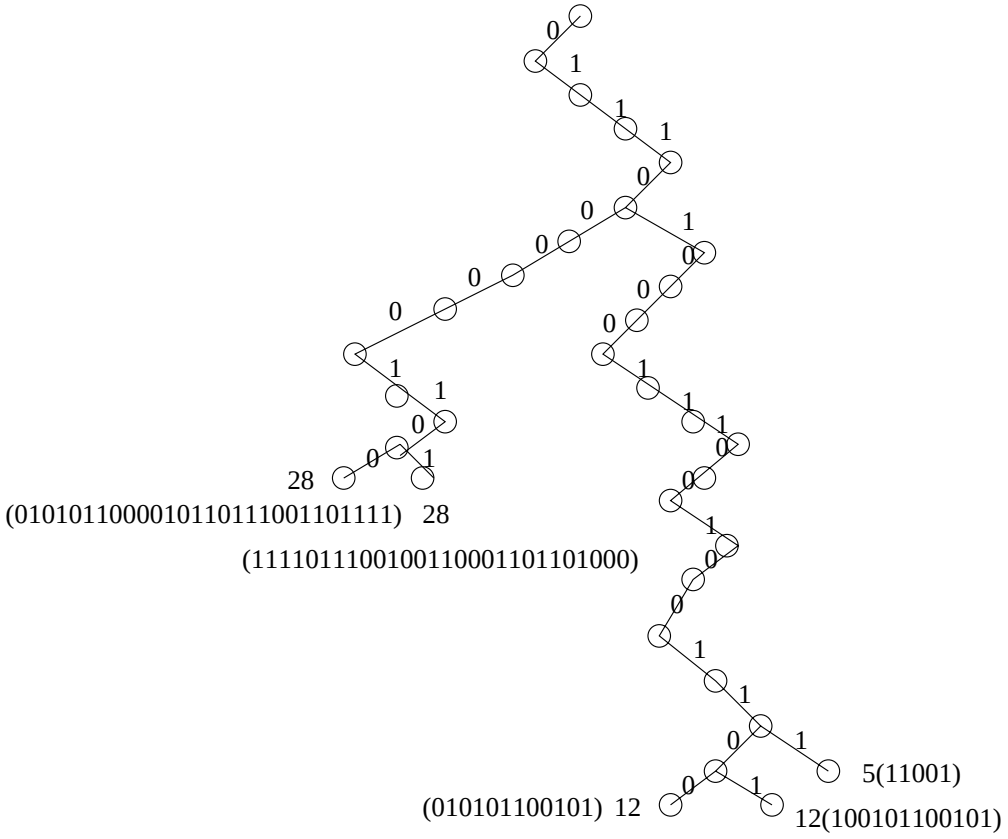


Figure 2.6: Ordinary binary trie for the key set {peano, porch, tree, trie, try }.

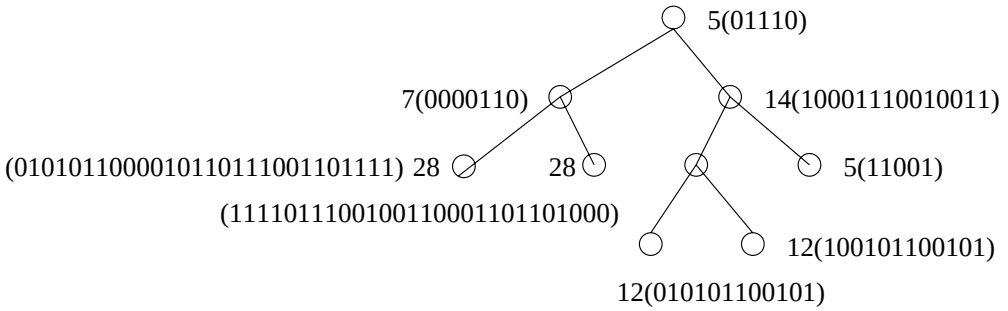


Figure 2.7: Patricia binary trie for the key set {peano, porch, tree, trie, try }.

2.2 Tries for Text

In 1983, the Oxford University Press and the University of Waterloo (UW) became partners in Oxford English Dictionary's electronic form [Tomp94] project. Differing from traditional text-based systems which lack concurrent data accessing and updating as well as a user interface to read text data interactively, the OED project provides the grammar-defined databases with licensed technology, including GOEDEL and PAT for text searching, and LECTOR for the user interface.

The PAT system is a text search engine developed by UW for the OED project [Gonn88] [GBS91] [BaGo96] [SaTo92]. PAT overcomes the shortcomings of traditional models which associate documents with a list of keywords, for example, by using inverted files to store a sorted list of keywords with each keyword linked to the documents containing the keyword. The traditional techniques result in an application that is only suitable for certain kinds of text retrieval but not for other kinds of queries. Keywords must also be extracted from indexed documents which is laborious and error prone, and we can only do queries on those stored keywords [GBS91]. The PAT system uses a different model which does not have any text structure or keywords.

2.2.1 PAT tree

Sistring

The whole text is treated as a single long string or as an array of characters. A suffix or semi-infinite string (sistring) is the sequence of characters taken from a given starting position within the text and continuing to the right [Knut73-2]. The name semi-string originates from the name of semi-infinite lines in geometry which start at one point but shoot out infinitely in one direction. Sistrings are distinguished by the position they start within a certain text. Table 2.2 gives an example of sistrings. If the size of the text is n characters, then the number of sistrings is n.

Table 2.2: An example of sistrings.

Text	*Sistrings are distinguished in the position they start* $\cdots$
sistring 1	Sistrings are distinguished in the position $\cdots$
sistring 2	istrings are distinguished in the position $\cdots$
sistring 3	strings are distinguished in the position $\cdots$
sistring 6	gs are distinguished in the position they $\cdots$
sistring 12	re distinguished in the position they start $\cdots$
sistring 20	nguished in the position they start $\cdots$

PAT Tree

Tries are recursive tree structures using the digital decomposition of strings to direct the branching. A PAT tree is a Patricia trie constructed from all the possible sistrings of a text. A trie built on the sistrings is also called a suffix tree and a Patricia trie is a compact suffix tree [BaGo96] which eliminates internal nodes with one child. As introduced in the previous section, the individual bits of the sistrings are used

to determine the direction of the branching; a zero bit causes branching to the left subtree, a one bit causes branching to the right subtree.

The external nodes of the PAT tree store sistrings or keep a reference to the sistrings. The internal nodes store the position indicating which bit of the sistring is to be used for branching or store the number of bits skipped, or in the following example (Figure 2.8) given by Gonnet in [Gonn88] [GBS91], the internal nodes store the total displacement of the bit to be inspected. The text string used in the example is 01100100010111 $\cdots$. After the first eight bits are inserted in the PAT tree, we obtain the shape shown in Figure 2.8. The internal nodes store the number of bits of displacement. For example, sistrings 1 and 5 have the same first two bits (01), and differ at the third bit. Their parent node, with 3 inside a circle, indicates this. Similarly, the internal node with 4 inside the circle indicates sistrings 6 and 3 have the same first three bits (100) and differ at the fourth bit. External nodes store a reference pointing to the position of the string.

If a text has n characters, the PAT tree built on the text has n external nodes and n-1 internal nodes. So the space required for a Patricia trie is O(n). The preprocessing time for building a Patricia trie is $O(n \times$ *Height of Trie*$)$ or $O(n \log n)$ [Gonn88]. The average height of Patricia trees is $O(\log n)$ [FlOd82]. PAT tree's construction time was improved to be $O(n)$ by E. Ukkonen [Ukko95].

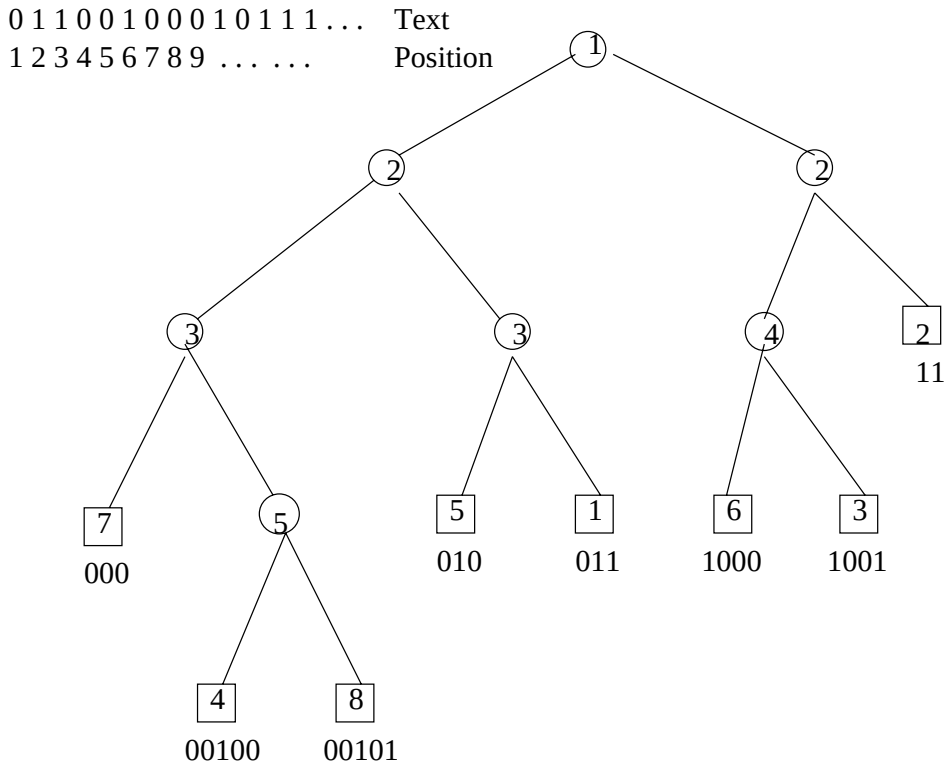


Figure 2.8: PAT tree example from Gonnet et al. [GBS91]. The bit patterns beneath the leaves show the bit pattern necessary to uniquely identify the leaf sistring.

2.2.2 PAT Tree for Searching

A PAT tree is efficient for text searching. PAT trees have been applied to some kinds of text queries in the OED project [GBS91]. These include prefix searching, proximity searching, longest repetition searching, and frequency searching. Prefix searching is to find all the sistrings with a given prefix string and can be performed in time linear to the height of the trie. Proximity searching is to find all the occurences of a string s_1 whose position is at most a fixed number of characters away from another string s_2, and the worst case for it is $O(n \log n)$ for n the number of characters in the text. Longest repetition searching is to find the longest match between two different positions of a text and can be obtained in $O(\textit{Height of Trie}) = O(\log n)$. "Most significant" or "most frequent" searching is to find the most frequently occurring strings. The worst case time to find this string or strings is $O(n/m)$, for $m = \textit{average length of the most frequently occurring strings}$ [GBS91].

2.2.3 Implementation of PAT Tree

Patricia Trie

PAT tree implementation by a Patricia trie is the obvious method. The size of the Patricia trie is too large however, and reading large records (much larger than the internal node size) stored on external physical storage is inefficient. To solve these two problems, OED designers used the following two methods. The first solution is to bucket the external nodes. Assuming a bucket's size is b, then those subtrees whose

size is less than b are put into a bucket and hence save $b-1$ internal nodes. This method has an advantage and a disadvantage. The advantage is that the bucket saves space. Based on a random distribution of keys, and the average bucket size is $b \ln 2$, the number of internal nodes decreases from $n-1$ to $n/(b \ln 2)$. A disadvantage is that the use of buckets increases the search time. There is only one entrance to a bucket, and searching the bucket might require visiting all the nodes inside the bucket one by one giving a worst time of b.

Another solution is mapping the tree onto the disk using super-nodes. This method is also very common. By allocating as many nodes as possible on one disk page and only maintaining an entry point to each disk page, every internal node will address a disk page or another node, and the system can reduce the storage cost of internal nodes along with the time to retrieve them.

PAT Array

For the bucket method in Patricia tries, if the size of the bucket is too big, the sequential search will still be intolerable. Manber and Myers [MaMy90] solved this problem by storing the whole index for the text to be searched into a single array of external nodes ordered lexicographically by sistrings.

Merrett and Shang's Improvement

Previous techniques represent internal nodes as two pointers and skipped information and external nodes (leaves) by a "start" pointer to individual bytes. A "start"

is a pointer to the text contained in leaf nodes. Merrett and Shang [MeSh93] improve the implementation by storing the Patricia trie in preorder, so the left descendants are the neighbors of their parent and hence no pointer is needed. They also make "start" point to text pages instead of individual bytes. The pointerless Patricia trie means no pointer for internal nodes, but external nodes still store "start" pointers for sistrings. Figure 2.9 is an example given by [MeSh93]. The Patricia trie is an example for text string "there!". We mark the six letters in the string as 1, 2, 3, 4, 5, 6 and the ASCII codes for them are 01110100, 01101000, 01100101, 01110010, 01100101, 00100001. In Figure 2.9, the two axes denote the level of the Patricia trie beginning from 1, and the nodes on every level. The internal nodes are represented by 1 with the number of skip bits following inside the braces. The external nodes (leaves) are represented by 0 followed by the "start" pointer.

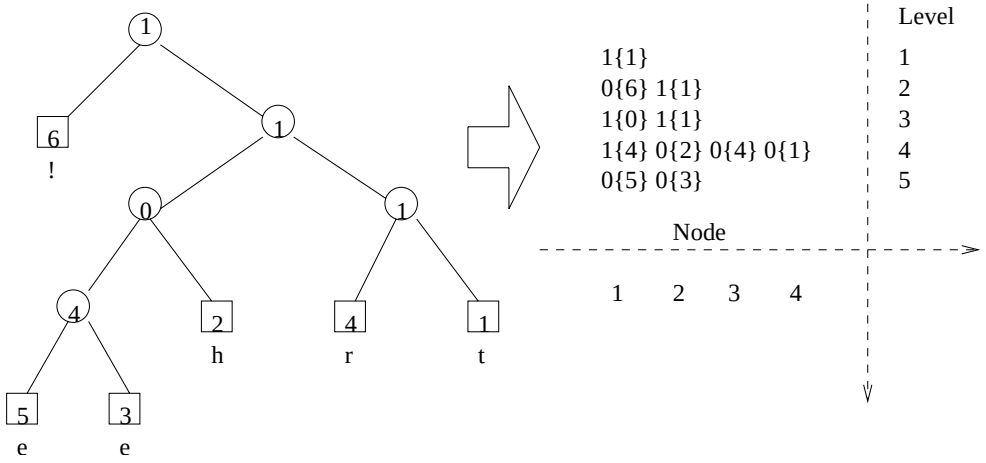


Figure 2.9: Example of a pointerless Patricia trie for text string "there!" (from [MeSh93]). Internal nodes are represented as 1{skip} where skip = number of bits to skip. External nodes (leaf nodes) are represented as 0{start} where start represents the index of the letter in the string (e.g. 0{4} represents the letter "r").

A certain number of levels of the trie are stored in index pages. Higher levels

form higher level index pages, and lower levels of the trie form lower level index pages. Each level is entirely on or entirely off one page. Every page stores the bits of the trie nodes, and two integers: $Tcount$ and $Bcount$. $Tcount$ counts the number of links going down from the upper level page to this level entering the left siblings of this page. $Bcount$ counts the total number of links down from this level to the lower level on the left side of this page. Figure 2.10 gives an example to illustrate the method. The two numbers marked on every page are $Tcount$ and $Bcount$. For example, the page P_3 with $Tcount = 2$ and $Bcount = 4$ indicates there are two links from the upper page P_1 going to the left page P_2 of page P_3, and there are 4 links going down to all lower left pages on the left side of page P_3 (from P_2 to P_5). Page P_4 with $Tcount = 6$ and $Bcount = 18$ indicates there are six links from the upper page P_1 going to the left pages P_2 and P_3 of page P_4, and there are 18 links going down to all lower left pages on the left side of page P_4 (from P_2 and P_3 to P_5 and P_6). With these two counters, the search time can be confined only to the pages visited.

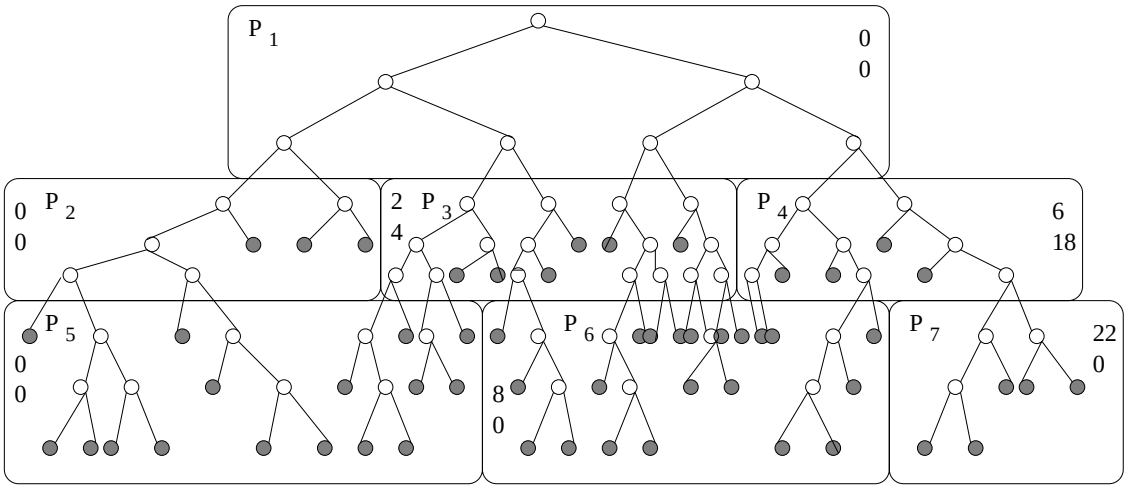


Figure 2.10: Example of pointerless Patricia trie with index paging (from [MeSh93]).

2.3 Tries for Interval Search

Spatial relations can be categorized as the three types [Egen91] listed below:

1. topological relations, which describe how the two objects' boundaries are related;

2. metric relations about distances and directions;

3. partial and total order of spatial objects such as in front of, above, and behind.

J.F.Allen [Alle83] described the possible relationships among time points and time intervals. He represented points and intervals by modelling their endpoints. He defined an interval to be an ordered pair of points with the left edge point less than the right edge point. There are seven relations between two intervals (e.g. interval t and interval s): $t < s$, when the right edge point of t is less than the left edge point of s; $t = s$, when t and s's left edge point and right edge point are equal to each other; t *overlaps* s, when the left edge point of t is less than the left edge point of s, and t's right edge point is greater than the left edge point of s, and t's right edge point is less than s's right edge point; t *meets* s, when t's right edge point is equal to the left edge point of s; t *during* s, when t's left edge point is greater than s's left edge point and t's right edge point is less than s's right edge point; t *starts* s, when t's left edge point is equal to s's left edge point and t's right edge point is less than s's right edge point; t *finishes* s, when t's left edge point is greater than s's left edge point and t's right edge point is equal to s's right edge point. Reversing the place of t and s, there

are altogether 13 relations between two intervals, as shown in Figure 2.11 and Table 2.3. The equal relation counts as one. The thirteen relations are exclusive and they will not generate any ambiguity.

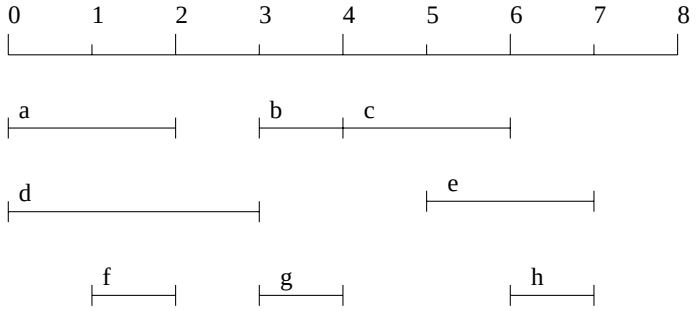
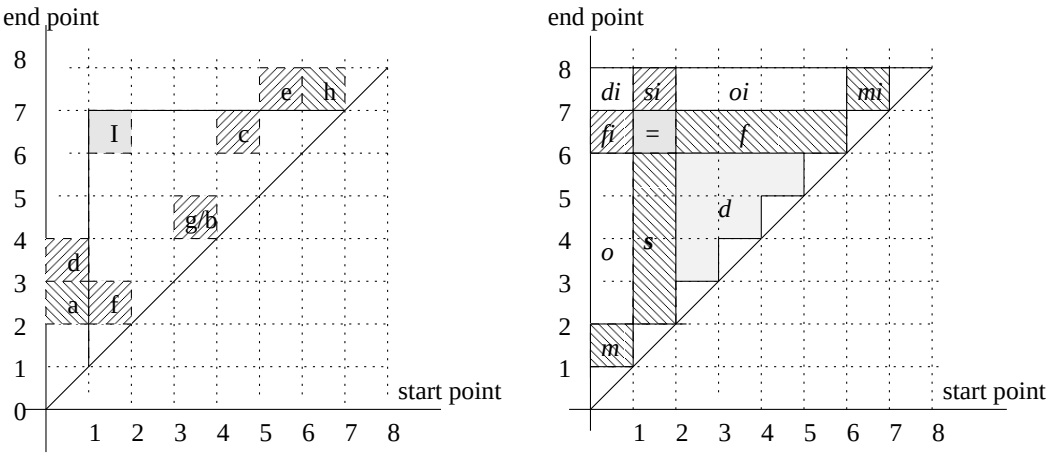


Figure 2.11: Intervals for illustrating interval relations.

Table 2.3: The 13 interval relations in 1-d space (from Allen [Alle83]).

Relation	*Symbol*	*Inverse Symbol*
a *before* b	a < b	b > a
b *equals* g	b = g	g = b
b *meets* c	b *m* c	c *mi* b
c *overlaps* e	c *o* e	e *oi* c
f *during* d	f *d* d	d *di* f
a *starts* d	a *s* d	d *si* a
h *finishes* e	h *f* e	e *fi* h

H.Shang [Shan01] gave an algorithm for interval containment. He represented the 1-dimensional problem in 2-dimensional space. He used marked regions to show the 13 relationships to the given query interval. Figure 2.12 (a) shows the interval relation of intervals in Figure 2.11 and a query interval I = (1, 6). For this query interval I, the 13 relations are shown in Figure 2.12 (b) (except for the two relations of "<" and ">").



(a) Intervals and query interval I

(b) Interval relations for I

Figure 2.12: Interval example for a query interval I = (1, 6) and the intervals of Figure 2.11 (adapted from Shang [Shan01]).

The shadow in Figure 2.13 is the containment area of query interval I, and Figure 2.14 shows the PR-Trie for this area. A PR-Trie is a data structure to represent points and regions [Same90]. It has black leaves which represent the corresponding subspace inside the query region, and white leaves which represent the corresponding subspace outside the query region. Containment includes the interval's four relations: equals, starts, finishes, and during. The full binary trie for the input intervals in Figure 2.12 is shown in Figure 2.15.

After building the PR-Trie for containment area and the full binary trie for the input data set, superimposing the PR-Trie onto the full binary trie means traversing both tries simultaneously. When we meet with black nodes in the PR-Trie, all the leaves inside the corresponding nodes in the full binary trie satisfy the containment relation. The subtree of nodes corresponding to white nodes in the PR-Trie can be

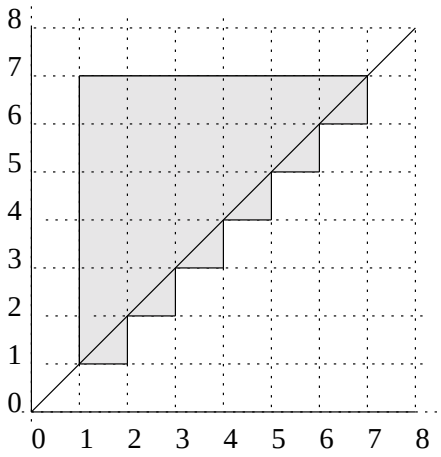


Figure 2.13: Containment area of I = (1, 6).

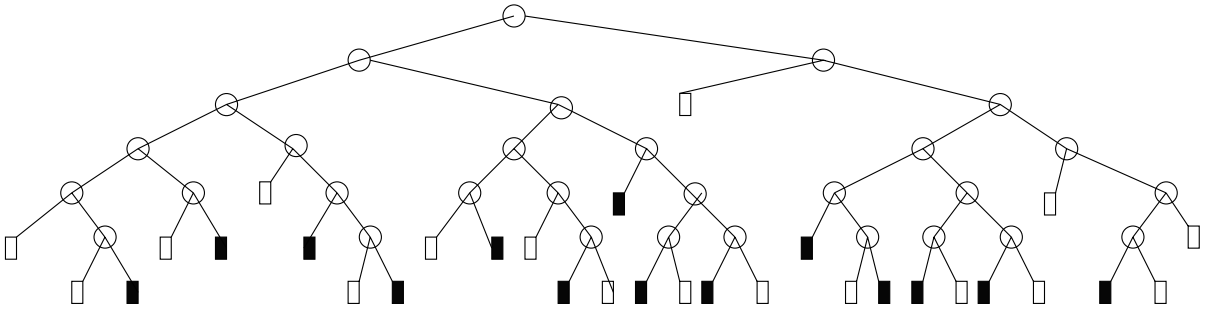


Figure 2.14: PR-Trie for containment area of Figure 2.13.

bit interleaved values

a (0, 2) 000 010		000100
b (3, 4) 011 100	⇒	011010
c (4, 6) 100 110	⇒	110100
d (0, 3) 000 011		000101
e (5, 7) 101 111		110111
f (1, 2) 001 010		000110
g (3, 4) 011 100		011010
h (6, 7) 110 111		111101

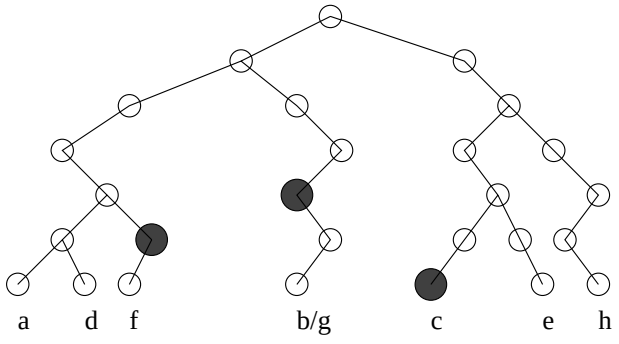


Figure 2.15: The full binary trie for the example in Figure 2.12 (adapted from Shang [Shan01]).

pruned during the search. The black filled circles in Figure 2.15 are nodes corresponding to the black nodes in the PR-Trie for a containment search using interval I. This method's disadvantages are: (1) the need to build two tries, and (2) the time required to traverse both tries in parallel.

2.4 Zoom Tries

A significant issue in displaying maps is how to change resolution, or level of abstraction, which is called zooming. Using full binary tries to store large spatial data which are represented as sequences of coordinate vectors, we can get the desired level of detail without duplicate storage. The zoom effect is to display all the data as a point when we set the level to 0 (root), and to display more detail as we go down the trie. The highest resolution is displayed when we reach the leaf level. To get this continuous zoom capability, a full trie is required.

2.4.1 Previous methods and disadvantages

To get different levels of abstraction, in general, people stores several versions of map data in a hierarchy. The top level stores the least detailed map, suitable for display at small scales. As the map scale increases, more details are shown of the same objects and so demand more storage space.

The disadvantages of this approach are

1. the data are stored redundantly,

2. The zoom is discontinuous which permits only those chosen scales of map to be shown, and

3. At every new scale, the data needs to be read in and displayed which can waste time.

2.4.2 Tries for zooming

T.H.Merrett and H.Shang [MeSh94] give a method for zoom using tries. For implementation of zoom tries, see [Bu00]. Using tries, zooming occurs in the following way. For a polygon represented by a sequence of coordinate vector data, after bit interleaving, we store the bits strings into the trie. The trie level controls the displayed resolution. Displaying at a specified resolution is controlled by reading only the appropriate levels of the trie. The user can select a certain level (which can be from 1 to 32 for vector data whose type is long), and we collect all paths from the tries' root down to the user selected level and decode these paths (bits strings) back into vector data and redraw them. Figure 2.16 gives an example of zoom tries.

In the triangle example of Figure 2.16 (a), the data is a two dimensional diagram shown to two bits resolution for each field. If we want only one bit resolution, we would truncate the full trie to retain only the top four levels (one bit from each field). The result in the lower resolution Figure 2.16 (b) is displayed as a point at the origin, a line from (0, 0) to (1, 0) and back again. Thus the triangle at a coarser resolution is shown to be a line, with its apex invisible at the lower resolution.

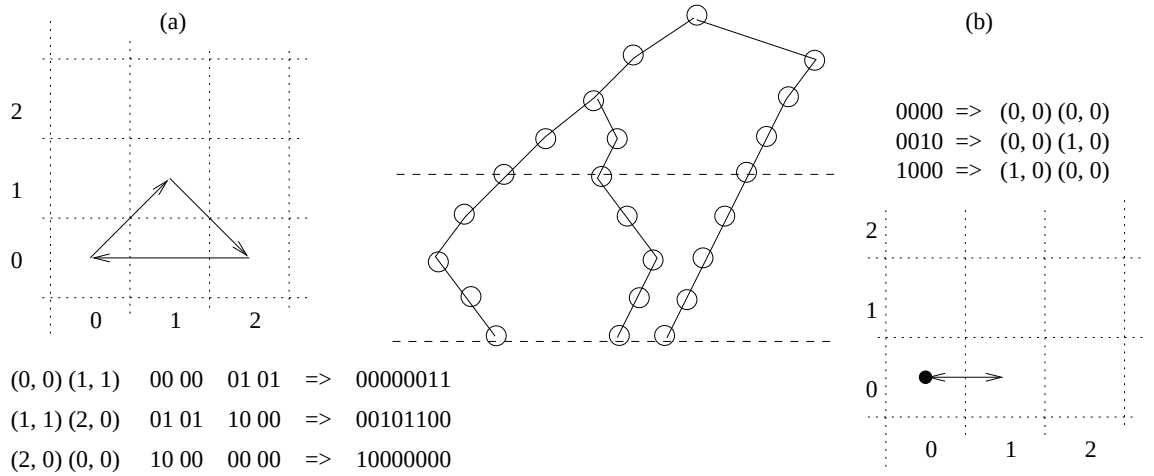


Figure 2.16: Example of a zoom trie with 8 levels displayed at level 4 (at the right) and level 8 (on the left).

2.5 Tries for Partial Match Query

2.5.1 Partial match query

To solve the query problem on k dimensions, we assume the domain is defined as

$$D = D_1 \times D_2 \times \cdots \times D_k$$

Given the input data set F containing n k-d records with each record $r = (r_1, r_2, \cdots, r_k)$, which is a subset of D, the size of F is denoted as n. For a given query $q = (q_1, q_2, \cdots, q_k)$,

$$q \in (D_1 \cup \{*\}) \times (D_2 \cup \{*\}) \times \cdots \times (D_k \cup \{*\}),$$

a partial match query asks for a subset $q(F)$, and * is the wildcard symbol meaning "matching anything". The subset $q(F)$ should match precisely on s dimensions, where $s = |S|$ = number of specified (not wildcard) attributes where r_j must equal to q_j. This is called a partial match [FlPu86]. We can also write it as:

$$\forall j, 1 \leq j \leq k, \text{ if } q_j \neq *,\ r_j = q_j.$$

A specified pattern S of query q means there are $s = |S|$ specified attributes in q. For example, a query $q = (first, *, secAttr, *, *, 6000)$, is a query on 6 dimensions, the first, third, and sixth attributes are specified to match precisely "first", "secAttr", "6000", and the second, fourth, and fifth attributes are wildcards.

2.5.2 Tries under the Bernoulli model

Assume each attribute in the input data set (of size n) and query q is uniformly and independently distributed over the domain. This is called the uniform probabilistic model or the Bernoulli model [FlPu86]. Without loss of generality, we assume that each attribute domain is mapped to the real interval [0, 1]. For those algorithms based on comparison, such an assumption is general enough. So, the sequence of keys are assumed to be independently taken from a uniform distribution model. Bits of arbitrary positions in arbitrary fields of keys are independent uniform {0, 1} random variables.

Flajolet and Puech [FlPu86] use the term shuffle for the sequence of attributes of keys. For a record $r \in F$, where $F \subset D$, and $r = (r_1, r_2, \cdots, r_k)$, every attribute of record r can be represented as a binary sequence:

$$r_j = r_j^1, r_j^2, \cdots, \text{ where } r_j^i \in \{0, 1\}.$$

Then the shuffle of record r is:

$$\textit{shuffle(r)}= r_1^1, r_2^1, \cdots, r_k^1, r_1^2, r_2^2, \cdots, r_k^2, r_1^3, r_2^3, \cdots, r_k^3, \cdots.$$

The record's shuffle is obtained by taking alternatively the first bit of attribute 1, the

first bit of attribute 2, $\cdots$, the first bit of attribute k, and then cyclically starting again from the first attribute to the k^{th} attribute with the second bit, and so on.

2.5.3 Theorem of Flajolet and Puech

Flajolet and Puech [FlPu86] determined the expected partial match query time cost in [FlPu86] as follows:

Theorem 1 *The average cost, measured by the number of internal nodes traversed, of a partial match query of specification pattern u with s specified attributes in a k-d-trie constructed from a file of either size n (under the Bernoulli model) or expected size n (under the Poisson model) satisfies*

$$C_{u,n} = \gamma(\tfrac{1}{k}\log_2 n)n^{1-s/k} + O(1),$$

where $\gamma(u)$ is a periodic function of u with period 1, small amplitude, and mean value

$$\gamma_0 = -\tfrac{s}{k^2 \log 2}\Gamma(\tfrac{s}{k} - 1)\,\Sigma_{\ell=0}^{k-1}(\delta_1\,\delta_2\,\cdots\,\delta_\ell)2^{-\ell(1-s/k)}$$

with $\delta_\ell = 1$, if the ℓ^{th} attribute of the query is specified, and $\delta_\ell = 2$ if it is unspecified.

This theorem is the basis for my analysis of orthogonal range search time complexity using tries. Theorem 1 is investigated more fully in Chapter 4.

$\gamma(u)$ function is obtained using a Mellin transform ([FlPu86] p394-398). The exact definition for γ function is:

$$\begin{aligned}&\gamma(\log_2 n/k)\\ &= \tfrac{(1+\sigma_0)\Gamma(\sigma_0)}{k\log 2}\Sigma_{\ell=0}^{k-1}\delta_1\delta_2\cdots\delta_\ell 2^{\ell\sigma_0}\\ &\quad +\Sigma_{j=1}^{\infty}\exp(\tfrac{-2ij\pi}{k}\log_2 n)(1+\alpha_j)\Gamma(\alpha_j)\Sigma_{\ell=0}^{k-1}\delta_1\delta_2\cdots\delta_\ell 2^{\ell\alpha_j} \qquad (2.1)\end{aligned}$$

where $\sigma_0 = -1 + s/k$ and $\alpha_j = \sigma_0 + \frac{2ij\pi}{k \log 2} = -1 + s/k + \frac{2ij\pi}{k \log 2}$, so we get:

$$\gamma(\log_2 n/k)$$

$$= \frac{s\Gamma(-1+s/k)}{k^2 \log 2} \Sigma_{\ell=0}^{k-1} \delta_1 \delta_2 \cdots \delta_\ell 2^{\ell(-1+s/k)}$$

$$+\Sigma_{j=1}^{\infty} \exp(\frac{-2ij\pi}{k} \log_2 n)(s/k + \frac{2ij\pi}{k \log 2})\Gamma(-1 + s/k + \frac{2ij\pi}{k \log 2})\Sigma_{\ell=0}^{k-1} \delta_1 \delta_2 \ldots \delta_\ell 2^{\ell(-1+s/k+\frac{2ij\pi}{k \log 2})} \tag{2.2}$$

The second item shows that γ function is a periodic function and the mean value is the first item:

$$\gamma_0 = -\frac{s}{k^2 \log 2} \Gamma(\frac{s}{k} - 1)\, \Sigma_{\ell=0}^{k-1} (\delta_1\, \delta_2\, \cdots\, \delta_\ell) 2^{-\ell(1-s/k)} \tag{2.3}$$

with $\delta_\ell = 1$, if the ℓ^{th} attribute of the query is specified, and $\delta_\ell = 2$ if it is unspecified. Calculating the mean value of γ function as given in equation (2.3) (see Appendix D), we obtain the results shown in Table 2.4.

Table 2.4: Average value of γ function.

k \ s	1	2	3	4	5	6	7	8	9	average
2	2.63									2.63
3	2.00	4.05								3.03
4	1.81	2.69	5.51							3.24
5	1.72	2.26	3.41	6.98						3.34
6	1.67	2.06	2.74	4.16	8.46					3.37
7	1.64	1.94	2.42	3.24	4.91	9.95				3.36
8	1.62	1.87	2.23	2.80	3.75	5.67	11.44			3.33
9	1.60	1.81	2.11	2.54	3.19	4.27	6.44	12.94		3.31
10	1.59	1.78	2.03	2.37	2.86	3.58	4.79	7.21	14.43	3.28

As to $k = 15$, the mean value is 3.26 and when $k = 20$, it is 3.34.

Besides Theorem 2 for the average cost of a partial match query pattern u with s specified attributes, Flajolet and Puech also give their Lemma 8 for the cost of a specified pattern u.

Lemma 8. *The expected cost of a partial match retrieval has for $n \geq 2$ the explicit form*

$$C_{u,n,s} = \Sigma_{\ell=0}^{k-1} \delta_1 \delta_2 \cdots \delta_\ell \Sigma_{j \geq 0} 2^{j(k-s)} \tau_{j,\ell}(n),$$

where for j and ℓ not both zero

$$\tau_{j,\ell}(x) = 1 - (1 - 2^{-kj-\ell})^x - x 2^{-kj-\ell} (1 - 2^{-kj-\ell})^{x-1}$$

and $\tau_{0,0}(x) = 1.$

Lemma 8 will be used in calculating the mean value of a partial match query in chapter 4.

Chapter 3

Tries for Spatial Range Search

Without loss of generality, we assume our search space is defined on the set of positive integers in k-dimensional space. We assume the space is finite, limited by the number of bits B used to represent an integer. B is the number of bits used for representing a coordinate value in binary, $B = \log_2(MAXIMUM - MINIMUM + 1)$, where MINIMUM and MAXIMUM are the whole search space's upper and lower bounds. From the application point of view, if on every dimension the number of maximum distinct values that can be represented is 2^B, then the maximum number of distinct hyper-rectangles is $\chi = \left(\begin{array}{c} 2^B \\ \\ 2 \end{array} \right)^k = (2^{2B-1} - 2^{B-1})^k$. The input data size $n \leq \chi = (2^{2B-1} - 2^{B-1})^k$.

3.1 Tries for spatial data

Binary tries are data structures which use a binary representation of the key to store keys as a path in the tree. Binary k-d tries use the principle of bit interleaving (also called a shuffle operation by Flajolet and Puech [FlPu86]). Child nodes in a k-d trie cover half of the search space volume of their parent.

For a point $p(x, y)$ in two dimensions, each coordinate value has B bits, and the bit interleaved value is $b_0^x\, b_0^y\, b_1^x\, b_1^y \cdots b_{B-1}^x\, b_{B-1}^y$, where b_i^x is the i^{th} bit value for x, and b_i^y is the i^{th} bit value for y. For a line segment with a start point s and end point e, bit interleaving treats the line segment as a 4-dimensional point $p = (x_s, y_s, x_e, y_e)$. For this so-called 4-d trie, the bit interleaving results in a bit string:$b_0^{x_s}\, b_0^{y_s}\, b_0^{x_e}\, b_0^{y_e} \cdots b_{B-1}^{x_s}\, b_{B-1}^{y_s}\, b_{B-1}^{x_e}\, b_{B-1}^{y_e}$. For a triangle which has three points on the plane, we treat it as a 6-d point $p = (x_0, y_0, x_1, y_1, x_2, y_2)$ or as three segments which construct the triangle. We represent a 2-dimensional rectangle as four coordinate values$(x^{min}, x^{max}, y^{min}, y^{max})$, which, after bit interleaving gives the bit string: $b_0^{x^{min}}\, b_0^{x^{max}}\, b_0^{y^{min}}\, b_0^{y^{max}}\, b_1^{x^{min}}\, b_1^{x^{max}}\, b_1^{y^{min}}\, b_1^{y^{max}} \cdots b_{B-1}^{x^{min}}\, b_{B-1}^{x^{max}}\, b_{B-1}^{y^{min}}\, b_{B-1}^{y^{max}}$.

Extending the bit interleaving principle to k dimensions, on every dimension j, $\forall j \in \{1 \cdots k\}$, we represent the k-d hyper-rectangle as $(x_j^{min}, x_j^{max})^k$, so the resultant bit string will be

$$b_0^{x_1^{min}}\, b_0^{x_1^{max}}\, b_0^{x_2^{min}}\, b_0^{x_2^{max}}\, b_0^{x_3^{min}}\, b_0^{x_3^{max}} \cdots b_0^{x_k^{min}}\, b_0^{x_k^{max}}$$

$$\ddots$$

$$b_{B-1}^{x_1^{min}}\, b_{B-1}^{x_1^{max}}\, b_{B-1}^{x_2^{min}}\, b_{B-1}^{x_2^{max}}\, b_{B-1}^{x_3^{min}}\, b_{B-1}^{x_3^{max}} \cdots b_{B-1}^{x_k^{min}}\, b_{B-1}^{x_k^{max}}.$$

Thus, a k-d hyper-rectangle can be represented as a $2k$-dimensional point in a binary trie of height $2kB$. Figure 3.1 below is an algorithm for performing bit interleaving of a k-dimensional hyper-rectangle to give a single key.

```
struct HyperRectangle
    array<integer> min    //the left value on each dimension
    array<integer> max    //the right value on each dimension
```

```
INTERBIT(R : HyperRectangle)
1  number ← 2kB − 1
2  for i ← B − 1 to 0
3  do for j ← 0 to k − 1
4       do key[number] = i^th bit of x_j^min
5            key[number − 1] = i^th bit of x_j^max
6       number ← number − 2
7  return key
```

Figure 3.1: Pseudo-code for converting a k-dimensional hyper-rectangle R to a single bit-interleaved key and the data structure for HyperRectangle.

3.2 Building a k-d trie

Given a set D of hyper-rectangles on k-dimensional space, the collection of these hyper-rectangles is denoted by $D = \{R_1, R_2, \cdots, R_n\}$, where n is the number of hyper-rectangles in the set. For the i^{th} hyper-rectangle $R_i \in D$, let $(x_{ij}^{min}, x_{ij}^{max})$ denote the j^{th} side of hyper-rectangle R_i, $1 \leq j \leq k$ and $1 \leq i \leq n$. We denote by T the $2k$-d trie constructed by inserting all the hyper-rectangles in D into an initially empty trie. Given a node u in T, we denote by T_u the subtree of T rooted at u. There are altogether n leaves in T. Every leaf is associated with one hyper-rectangle. Figure 3.2 is an algorithm to insert one k-d hyper-rectangle into trie T. The height

of the trie (i.e. the length of the key) is $2kB$. In the insertion loop from $2kB-1$ to 0, when a bit value equals 0, we go to the left branch and when the bit value equals 1, we go to the right branch of the trie. After preprocessing all n hyper-rectangles in D, we obtain the trie T, which allows us to carry out an orthogonal range search.

```
struct TrieNode
    TrieNode* Left        //left child pointer
    TrieNode* Right       //right child pointer
    TrieNode* Parent      //parent pointer

INSERT(R : HyperRectangle, T : TrieNode)
 1  if T = NIL
 2     then T ← new TRIENODE(k)
 3   Key keyPoint ← INTERBIT(R)// get the key from hyper-rectangle R
 4   TrieNode P ← T// begin traversing
 5  for level ← (2kB − 1) to 0
 6  do if keyPoint→getKey(level)
 7        then // if the keyPoint bit on this level=1, go right
 8             if P→right = NIL
 9                then p ← new TRIENODE(P, K)
10                     P→Right ← p
11                     // create a trie node and insert into right side of Trie T
12             P = P→Right// continuing traversing on the right side
13        else // go left
14             if P→Left = NIL
15                then p ← new TRIENODE(P, K)
16                     P→Left ← p
17                     // create a trie node and insert into left side of Trie T
18             P ← P→Left// continuing traversing on the left side
```

Figure 3.2: Pseudo-code for inserting a k-d hyper-rectangle R into a $2k$-d trie T.

Figures 3.3 and 3.4 give one example of building a binary trie from 15 2-d rectangles with number of data bits $B = 5$.

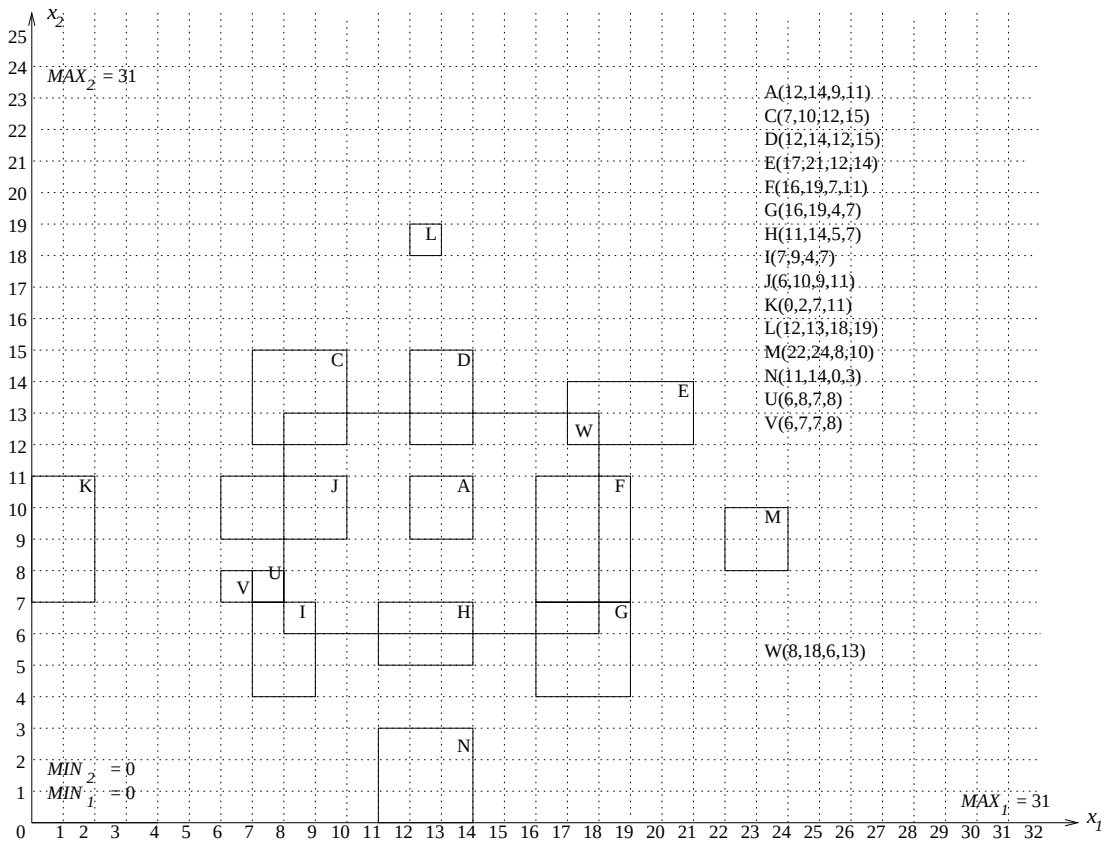


Figure 3.3: Example of 15 rectangles $A, C, D, E, F, G, H, I, J, K, L, M, N, U$, and V with a query hyper-rectangle W and number of data bits $B = 5$.

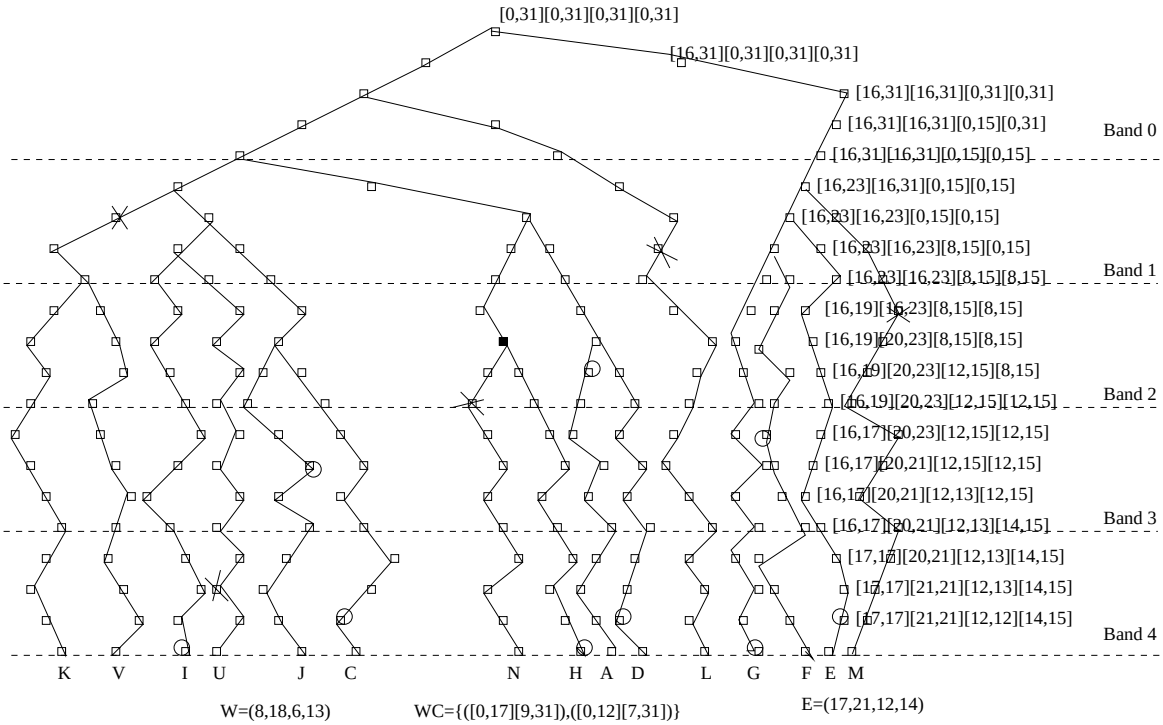


Figure 3.4: Example of a binary 4-d trie for the 2-d data of Figure 3.3. The list of 8-tuples near the right hand side is the cover space NC of each node on the trie path representing rectangle E.

Theorem 2 *$P(n,k) = \Theta(kBn)$ for a k-dimensional trie containing n hyper-rectangles.*

Proof. The proof is straightforward as each of the n inserted hyper-rectangles visits $2kB$ nodes, $P(n,k) = \Theta(kBn)$. ∎

3.3 Spatial range search

3.3.1 Range search

The query hyper-rectangle $W = [L_1, H_1] \times [L_2, H_2] \times \cdots \times [L_k, H_k]$, which we abbreviate as $[L_j, H_j]^k$. For a hyper-rectangle $R_i \in D$, the set of k hyper-rectangle sides is defined by $\{(x_{ij}^{min}, x_{ij}^{max}), \forall j \in \{1, \cdots, k\}\}$. We define $[MIN_j, MAX_j]$, $\forall j \in \{1, \cdots, k\}$, as the minimum and maximum possible data coordinate values for dimension j. On every dimension, $MIN_j \leq x_{ij}^{min} \leq x_{ij}^{max} \leq MAX_j, \forall j \in \{1 \cdots k\}, i \in \{1 \cdots n\}$.

Fact 1 *Two hyper-rectangles intersect if and only if their sides on every dimension in the data space intersect, i.e. $R_1 \cap R_2$ is true, iff $\forall j \in \{1, \cdots, k\}$, $(x_{1j}^{min}, x_{1j}^{max}) \cap (x_{2j}^{min}, x_{2j}^{max})$ is true; $x_{1j}^{min} \in [MIN_j, x_{2j}^{max})$ and $x_{1j}^{max} \in (x_{2j}^{min}, MAX_j]$.*

This defines intersection strictly as an overlap in the sense of Allen [Alle83] and Egenhofer [Egen94]. Based on Fact 1, the hyper-rectangle R_i intersects W iff $x_{ij}^{min} \in [MIN_j, H_j)$ and $x_{ij}^{max} \in (L_j, MAX_j]$, $\forall j \in \{1 \cdots k\}$.

The k-dimensional orthogonal range search is performed using our $2k$-d trie for a query W. We use j as the index of the data space, $j \in \{1 \cdots k\}$, and we use p as

the index for our problem space, $p \in \{1 \cdots 2k\}$. They are related as $j = \lceil p/2 \rceil$.

Definition 2 *Each node in the trie T is a $2k$ range state; that is, every node has a cover space defined as $NC^{2k} = [L_p, H_p]^{2k}$, $1 \leq p \leq 2k$. For a given query hyper-rectangle $W = [L_j, H_j]^k$, we obtain the query hyper-rectangle's cover space WC^{2k} and define it to be $WC^{2k} = \{([MIN_j, H_j - 1], [L_j + 1, MAX_j])\}^k$.*

Definition 2 is based on Fact 1. Each left and right edge point on every dimension has a range of intersection. Thus, each node in the trie T represents a $2k$ dimensional hyper-rectangle. At the root of T, the hyper-rectangle occupies the whole search space. Each node along a path splits the range in half, alternating the split dimension at each level. Going left, the lower half is used; going right, the higher half is used. At the leaves, the lower and upper bound of the $2k$ ranges on every dimension is exactly the coordinate values of the hyper-rectangle.

There are three types of relations of WC^{2k} with NC^{2k}, which we call BLACK, GREY, and WHITE. Figure 3.5 below illustrates the three cases. Dashed lines are used for WC and solid lines for NC. After projecting the edge points to the problem space, we can distinguish the relation of a node's cover space NC with the query hyper-rectangle W's cover space WC on the p^{th} side.

For example, for the data of Figure 3.3 and Figure 3.4,
$WC^4 = \{([0, 17], [9, 31]), ([0, 12], [7, 31])\}$ for query hyper-rectangle W, nodes marked with a cross sign in Figure 3.4 are white, and nodes marked with a circle sign are black.

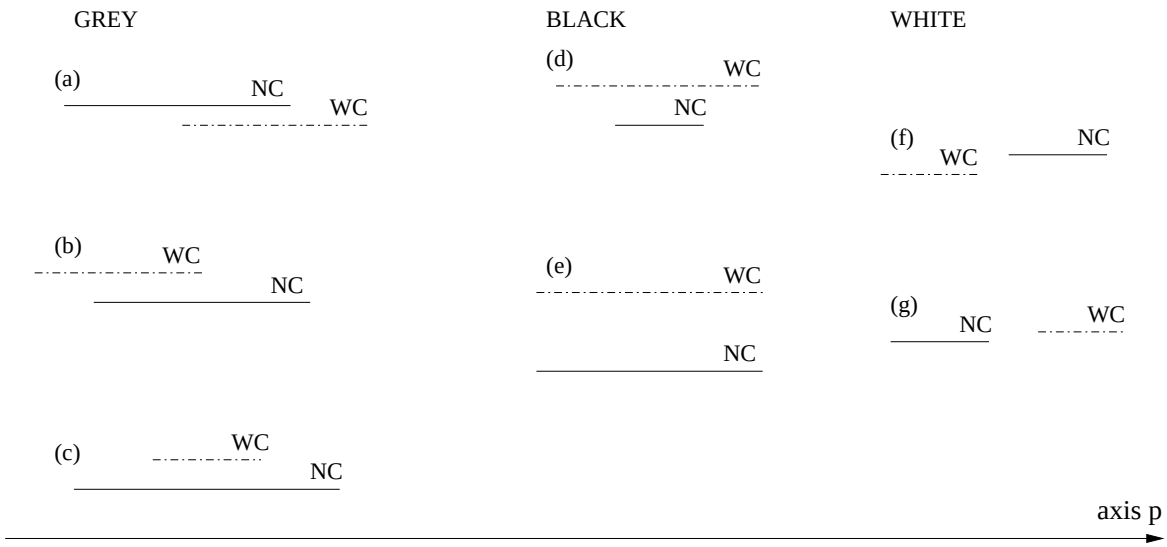


Figure 3.5: GREY ((a), (b), and (c)), BLACK ((d) and (e)), WHITE ((f) and (g)) relationships of a trie node cover space NC to a query hyper-rectangle cover space WC in dimension p of the $2k$-d problem space.

Definition 3 *If on all $2k$ dimensions, the NC to WC relationship satisfies $WC^p \cap NC^p = BLACK$, $\forall p \in \{1, 2, \cdots, 2k\}$, the node in the trie is a black node. If the NC to WC relationship satisfies $\exists p \in \{1, 2, \cdots, 2k\}$, such that $WC^p \cap NC^p = WHITE$, the node in the trie is a white node. All the other nodes are grey nodes, defined as follows: if $S \subseteq \{1, 2, \cdots, 2k\}$, $p \in S$, $WC^p \cap NC^p = GREY$, and $\forall p \notin S, p \in \{1, 2, \cdots, 2k\}$, $WC^p \cap NC^p = BLACK$, the node will be grey.*

We use GN to denote the set of grey nodes in the trie, BN to denote the set of black nodes in the trie, and WN to denote the set of white nodes in the trie. Based on this definition, we can now define our k-d orthogonal range search algorithm (see Figure 3.6).

The range search algorithm traverses from the root of trie T down to its leaves. We do a depth first traversal. At the root , level $\ell = 0$. For the root, the cover space

```
RANGESEARCH(T : TrieNode, ℓ, L, H, RI, W : HyperRectangle, List)
 1  if T = NIL or ℓ > 2kB
 2     then return
 3  p ← (ℓ − 1) mod (2k)
 4  RI[p] ← INRANGE(L[p], H[p], ℓ, W)
 5  if RI[p] is grey
 6     then
 7          ℓ ← ℓ + 1
 8          p ← ℓ mod (2k)
 9          if T→Left ≠ NIL
10             then
11                  H[p] ← (L[p] + H[p])/2
12                  RANGESEARCH(T→Left, ℓ,
13                       L, H, RI, W, List)
14          if T→Right ≠ NIL
15             then
16                  L[p] ← (L[p] + H[p])/2 + 1
17                  RANGESEARCH(T→Right, ℓ,
18                       L, H, RI, W, List)
19     else  if RI[p] is black
20             then COLLECT(T, List)
```

Figure 3.6: Pseudo-code for the k-d orthogonal range search algorithm. InRange(L[p],H[p],level,W,S) and NodeColor(RIT) are functions to decide the color of NC and WC relationships for node T.

NC^{2k} has $L_p = MIN_j$ and $H_p = MAX_j$, $\forall p \in \{1, 2, \cdots, 2k\}$. The cover space is split on the p^{th} coordinate as we move down, $p = \ell \, mod \, 2k$, $\forall \, \ell \in \{0, 1, \cdots, (2kB - 1)\}$. If on the p^{th} dimension, a parent node T has cover space $[L, H]$, then T's left child's cover space is $[L, (L + H)/2]$ and T's right child's cover space is $[(L + H)/2 + 1, H]$. Comparing a node's cover space NC (stored in L and H) with query hyper-rectangle W's cover space $[MIN_j, H_j)$ and $(L_j, MAX_j]$, if one of the $2k$ ranges falls outside (as determined by the InRange function), we encounter a white node and the search need not check any subtrees of T. If all the $2k$ ranges fall within NC, we encounter a black node. The nodes in the subtree of a black node are all black nodes. The nodes in the subtree of a white node are all white nodes. When we meet a white

node or a black node, we stop traversing down; otherwise we continue splitting and traversing the subtrees. When we reach a black node u, all the leaves associated with a hyper-rectangle inside subtree T_u intersect the query hyper-rectangle W, and we get the range search's results and append them into the reporting List. All the leaves contained in the subtree of a white node do not intersect W. Integer arrays L and H store the lower and upper bounds of node T's cover space on $2k$ dimensions.

Figure 3.4 is the trie for the data shown in Figure 3.3. In Figure 3.4, nodes with a cross mark ($\times$) on them are white nodes. We know no children of the white node can intersect the query hyper-rectangle W, so the subtree attached to the white node is pruned from the search space. The node with a circle sign ($\bigcirc$) on it stands for a black node, which means all hyper-rectangles represented by leaves inside the subtree attached to the black node intersect W. For query hyper-rectangle $W = [8, 18] \times [6, 13]$, $k = 2$, the query hyper-rectangle's cover space $WC^4 = [0, 17] \times [9, 31] \times [0, 12] \times [7, 31]$. The hyper-rectangle denoted as $E = [17, 21] \times [12, 14]$ has its cover space NC^4, the four ranges, listed along the right side of Figure 3.4. We do half splitting continuously in $2k$ space as we move down the trie. The trie is divided into B bands, each band of height $2k$ (see Figure 3.4). When the first band $2k$ half-splits are finished, we begin the second band, till the $(B-1)th$ band. For hyper-rectangle E's case, traversal of T during the 2-d range search for rectangles intersecting W stopped at the last $(B-1)th$ band, which is a black node. If we continue one extra step to the leaf level, we will get $NC^4 = [17, 17] \times [21, 21] \times [12, 12] \times [14, 14]$, and the lower bound and upper bound of every $2k$ range is equal to the coordinate value of

hyper-rectangle E in the data space.

3.3.2 Collection

After we find a black node, we collect the hyper-rectangles stored in the leaves inside the subtree of the black node. The following algorithm (see Figure 3.7) first traces back from the black node to the root and stores the key's bit string in keyPoint. From the black node, we call the recursive function PreOrder (see Figure 3.8) which traverses the subtree of the black node using pre-order traversal. When a leaf node is reached, we decode the key into a hyper-rectangle's coordinate values and append the hyper-rectangle to the resultant hyper-rectangle list *List*.

```
COLLECT(T : TrieNode, List : HyperRectangleList)
 1   Key keyPoint ← new KEY(K)
 2   TrieNodeP ← new TRIENODE()
 3   P ← T// trace back from solid node T to the root and store the path in keyPoint
 4   level ← GETDEPTH(P)
 5   while P→Parent ≠ NIL
 6   do
 7       num ← 2KB − (level − i)
 8       bool isRightChild ← P is P's parent's right child
 9       keyPoint→putKey(num, isRightChild)
10       i ← i + 1
11       P ← P→Parent
12   // traverse down from solid node T to leaves in the subtree of T
13   PREORDER(P, level, keyPoint, List)
```

Figure 3.7: Pseudo-code for collecting intersected hyper-rectangles from leaf nodes of the trie.

```
PreOrder(T : TrieNode, level, keyPoint : Key, List)
 1  // PreOrder is to collect subtree of black node T
 2  if level = 2KB
 3     then // decode keyPoint which stores a path from root via solid node T
 4          // to a leaf and add it to a hyper rectangle list
 5          List→addHypRec(DeCode(keyPoint))
 6  level ← level + 1
 7  num ← 2KB − level// traversing begins from T's level
 8  if T→Left
 9     then // if T's left child exists
10          keyPoint→putKey(num, false)// store the bit into KeyPoint
11          PreOrder(T→Left, level, keyPoint, List)
12          // continue traversing left subtree
13     else  if T→Right
14              then // if T's right child exists
15                   keyPoint→putKey(num, true)// store the bit into KeyPoint
16                   PreOrder(T→Right, level, keyPoint, List)
17                   // continue traversing right subtree
```

Figure 3.8: Pseudo-code for pre-order collection of leaf node hyper-rectangles.

3.4 Containment

Definition 4 *For two hyper-rectangles R_1 and R_2, R_1 contains R_2 if and only if R_2's sides on every dimension in the data space are within R_1's sides, i.e. $x_{2j}^{min} \geq x_{1j}^{min}$ and $x_{2j}^{max} \leq x_{1j}^{max}$.*

This definition strictly defines the containment relationship as defined by Allen [Alle83] and Egenhofer [Egen94]. Based on this definition, the hyper-rectangle $R_i = (x_{ij}^{min}, x_{ij}^{max})$, $\forall j \in \{1, \cdots, k\}$ is contained by query hyper-rectangle $W = (L_j, H_j)$ if and only if $x_{ij}^{min} \in [MIN_j, L_j]$ and $x_{ij}^{max} \in [H_j, MAX_j]$.

Similarly, if we want to get those hyper-rectangles that wrap W inside, on all the dimensions, W's side should be contained by hyper-rectangle R_i's side. Hyper-rectangle R_i contains query hyper-rectangle W if and only if $x_{ij}^{min} \in [L_j, H_j)$ and $x_{ij}^{max} \in (L_j, H_j]$.

The relation also includes $W = R_i$. This definition is stricter than the definition for intersection. If we want to get hyper-rectangles with more precise relations, the RangeSearch algorithm shown in Figure 3.6 still can be used for containment relations. Only the sub-function InRange needs to be changed; instead of satisfying Fact 1 in section 3.3.1, it will satisfy Definition 4 in section 3.4. The query hyper-rectangle W's cover space will change from $[MIN_j, x_{2j}^{max}], [x_{2j}^{min}, MAX_j]$ to $[MIN_j, x_{2j}^{min}], [x_{2j}^{max}, MAX_j]$. The GREY, BLACK, WHITE color relations of WC with NC are still the same as Figure 3.5. Applying the splitting along the trie and comparing the nodes' cover space with the query hyper-rectangle's cover space, we can get the grey and black nodes, and then collect the subtree of black nodes, decoding them to get the hyper-rectangles satisfying a containment relation.

Chapter 4

Analysis

4.1 Worst Case Analysis

Theorem 3 *The space $S(n,k)$ for a k-d trie containing n k-d hyper-rectangles is $O(kBn)$.*

Proof. The worst case space requirement occurs if every input hyper-rectangle, after bit interleaving, occupy as many different paths as possible; that is, their maximum bit distance value would be as large as possible. In this case, the trie occupies the largest number of nodes. From the root to level $r = \lceil \log_2 n \rceil - 1$, the binary trie is complete. At level $\lceil \log_2 n \rceil$, the full binary tree stops and the remaining paths are branches dangling from the last full binary tree's level, as shown in Figure 4.1. From root level to level $\lceil \log_2 n \rceil - 1$, the worst case trie storage is the same as a full binary tree. From level $\lceil \log_2 n \rceil$ to the leaf level, the trie is slimmer than the triangle of a full binary tree (drawn in dashed lines in Figure 4.1). The number of nodes in the

worst case is thus

$$S(n,k) = 1 + 2 + 2^2 + \cdots + 2^{\lceil \log_2 n \rceil - 1} + (2kB - \lceil \log_2 n \rceil)n$$

$$= 2^{\lceil \log_2 n \rceil} - 1 + n(2kB - \lceil \log_2 n \rceil)$$

$$= (2kB + 1)n - n \log_2 n - 1$$

$$= O(kBn). \quad \blacksquare$$

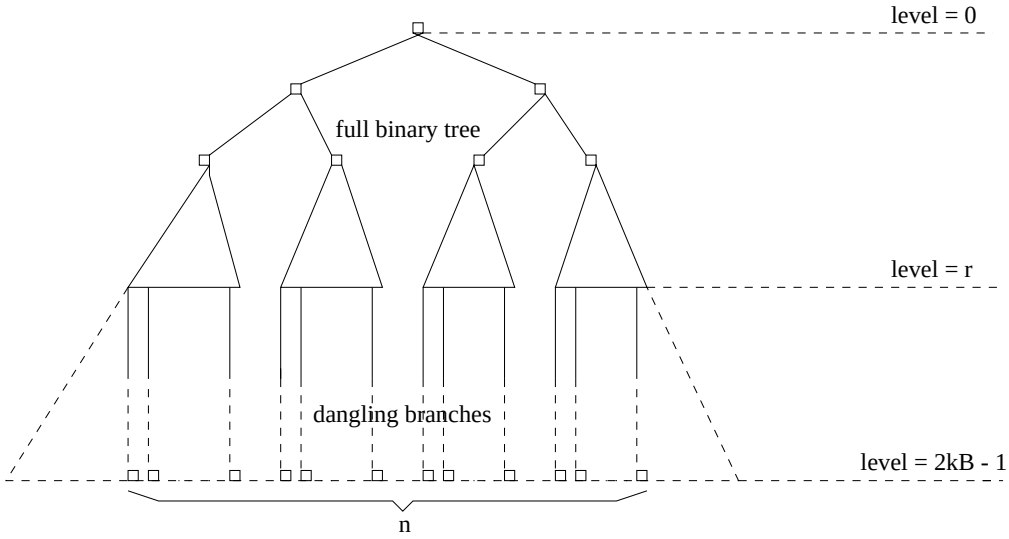


Figure 4.1: Worst case for storage in a $2k$-d trie.

4.2 Average Case Analysis

4.2.1 Space Analysis

The full trie T's depth is a constant value if the number of dimensions k and coordinate range on every dimension are considered constant. If we assume the coordinate ranges on every dimension are the same, that is $[MIN, MAX]$, B is the number of bits for representing the coordinate value inside the coordinate range. We have T's

height $= 2kB = 2k\log_2(MAX - MIN + 1)$, and level $\ell \in \{0, \cdots, 2kB - 1\}$. At the root level $\ell = 0$, and at the leaf level $\ell = 2kB - 1$. n is the size of input data set D. We divide the trie's levels into B bands, each of height $2k$.

Given any coordinate value $u \in [MIN, MAX]$, the bit distance between u and another value $v \in [MIN, MAX]$ is denoted as $bdis(u, v)$. $bdis(u, v)$ is defined as the path length from P (the common ancestor of u and v) down to the leaf level of u or v in the bit distance tree V. For example, $bdis(12, 12) = 0$, $bdis(12, 13) = 1$, $bdis(12, 11) = 3$, $bdis(31, 32) = 6$, $bdis(31, 19) = 4$, $bdis(7, 19) = 5$ (see Figure 4.2).

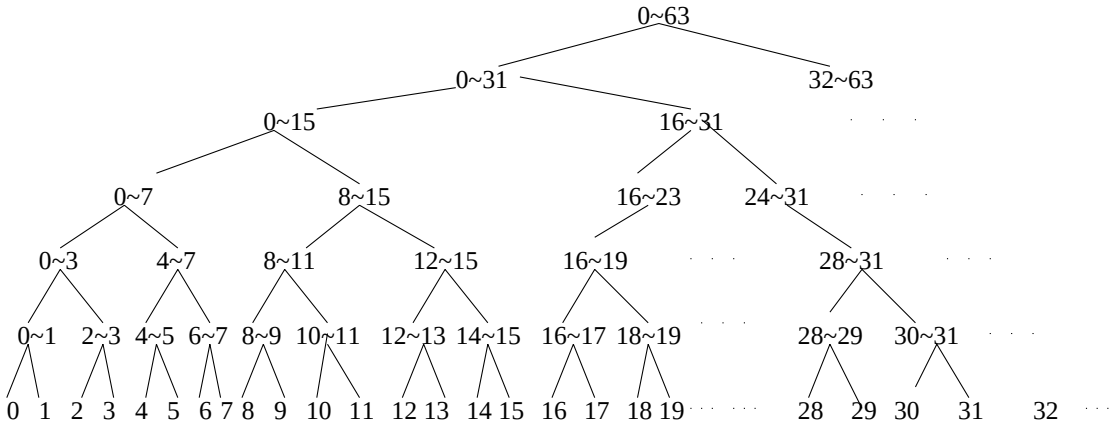


Figure 4.2: Example of bit distance tree V for $B = 6$.

We use $max_{j\in\{1,\cdots,k\}}\{\}$ to denote the maximum value of the set inside braces on k dimensions, and we have the following lemma:

Lemma 1 *For any two hyper-rectangles* $R_1 = (x_{1j}^{min}, x_{1j}^{max})^k$ *and* $R_2 = (x_{2j}^{min}, x_{2j}^{max})^k$, *their nearest common ancestor lies in the band*

$B - max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}$,

bands are numbered from 0.

Proof. Bit distance tree V with maximum bit distance B stores coordinate values on one dimension in $2k$ dimensions. The height of the bit distance tree V is B. Each node's cover space is one-half of it's parents' cover space in V. From the root whose domain is [0,2^B-1] down to u and v's nearest common ancestor node, u and v share the same path. The path length from leaf level to the nearest common ancestor node P is $bdis(u, v)$. The level of P is $B - bdis(u, v)$. From root level 0 to level $B - bdis(u, v)$, u and v's first $B - bdis(u, v)$ bits are identical. From level $B - bdis(u, v)$ down to leaf level B, the bit sequence for u and v is different. Our trie is a $2k$-d trie, and each of the $2k$ ranges will half split alternately. From the root, whose cover is $[0, 2^B - 1]^{2k}$, u_i will separate with v_i on the band of $B - bdis(u_i, v_i)$, for each of the $2k$ dimensions. That is, u and v's nearest common ancestor node lies in the band of $B - max_{p \in \{1,\cdots,2k\}}\{bdis(u_p, v_p)\}$. ∎

For example, as shown in Figure 3.3 and Figure 3.4 where $k = 2$, rectangle $N(11, 14, 0, 3)$ and rectangle $H(11, 14, 5, 7)$, we compare $2k$ pairs of bit distance: $bdis(11, 11) = 0$, $bdis(14, 14) = 0$, $bdis(0, 5) = 3$, $bdis(3, 7) = 3$. The maximum bit distance among N and H is 3, so N and H's nearest common ancestor lies in the band $B - 3 = 2$. This common ancestor node is indicated in Figure 3.4 by a black filled in square in Band 2.

We use lemma 1 to compute the expected storage space for our trie.

Theorem 4 *The expected space $S(n, k)$ for a binary trie containing n random k-dimensional hyper-rectangles is $\Theta(kBn)$.*

Proof. After the first two hyper-rectangles R_1 and R_2 are inserted, the number of nodes they share will be $2k(B - E[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}])$. The separate branches for both will occupy

$2kE[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}]$ nodes. The expected total number of nodes occupied by two hyper-rectangles will be:

$$2k(B - E[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}])$$
$$+2kE[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}]2$$
$$= 2kB + 2kE[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}].$$

Adding a third hyper-rectangle R_3, the nearest common ancestor of the three of them would be in the band:

$$B - E[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{min}, x_{3j}^{min}), bdis(x_{2j}^{min}, x_{3j}^{min}),$$
$$bdis(x_{1j}^{max}, x_{2j}^{max}), bdis(x_{1j}^{max}, x_{3j}^{max}), bdis(x_{2j}^{max}, x_{3j}^{max})\}]$$

The expected number of nodes in the trie is now

$$2kB + 2kE[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{max}, x_{2j}^{max})\}]$$
$$+2kE[max_{j\in\{1,\cdots,k\}}\{bdis(x_{1j}^{min}, x_{2j}^{min}), bdis(x_{1j}^{min}, x_{3j}^{min}), bdis(x_{2j}^{min}, x_{i3j}^{min}),$$
$$bdis(x_{1j}^{max}, x_{2j}^{max}), bdis(x_{1j}^{max}, x_{3j}^{max}), bdis(x_{2j}^{max}, x_{3j}^{max})\}].$$

No extra storage is needed for common nodes. We use $max_{i,j\in\{1,\cdots,k\}}\{\}$ to denote the maximum bit distance of hyper-rectangles, where i is the number of hyper-rectangles. There are $2k\begin{pmatrix} i \\ \\ 2 \end{pmatrix}$ bit distances from which to determine $max_{i,j\in\{1,\cdots,k\}}\{\}$. Using induction on i, we obtain the total number of nodes in the trie for n input hyper-rectangles as

$$S(n,k) = 2kB + 2k\Sigma_{i=2}^{n}E[max_{i,j\in\{1,\cdots,k\}}\{\}] \tag{4.1}$$

For any given value $x_{1p} \in \{0, 1, 2, \cdots, 2^B - 1\}$, the value $x_{2p} \in \{0, 1, 2, \cdots, 2^B - 1\}$ has a bit distance relation with x_{1p} as follows:

if $bdis(x_{1p}, x_{2p}) = 0$, the number of possible x_{2p} values $= 1$, except this, the number of possible x_{2p} values is $2^{bdis(x_{1p}, x_{2p})-1}$, the bit distance between any two x_{ij} values is one of $\{0, 1, \cdots, B\}$.

The probability that the bit distance is any specific value a would be

$Pr[bdis(x_{1p}, x_{2p}) = a] = \frac{2^{a-1}}{2^B}$, for $a \in \{1, \cdots, B\}$

with $Pr[bdis(x_{1p}, x_{2p}) = 0] = \frac{1}{2^B}$ and $Pr[bdis(x_{1p}, x_{2p}) = B] = \frac{2^{B-1}}{2^B} = \frac{1}{2}$.

The probability for two hyper-rectangles' maximum bit distance value to be 0 in the $2k$ problem space would be

$Pr[max_{2,p\in\{1,\cdots,2k\}}\{bdis(x_{1p}, x_{2p})\} = 0] = (\frac{1}{2^B})^{2k}$,

because the hyper-rectangles are independently distributed in $2k$ dimensions. If we want the maximum bit distance to be 0, then the bit distances on all $2k$ dimensions should be zero. On one dimension p, the probability that $bdis(x_{1p}, x_{2p}) \leq a$ is

$\frac{1}{2^B} + \Sigma_{q=1}^{a} \frac{2^{q-1}}{2^B}$. For any $a \in \{1, 2, \cdots, B\}$:

$$Pr[max_{2,p\in\{1,\cdots,2k\}}\{bdis(x_{1p}, x_{2p})\} = a] = \Sigma_{p=1}^{2k} \begin{pmatrix} 2k \\ p \end{pmatrix} (\frac{2^{a-1}}{2^B})^j (\frac{1}{2^B} + \Sigma_{q=1}^{a-1} \frac{2^{q-1}}{2^B})^{2k-p}.$$

When the maximum bit distance equals to a, we want one bit distance among 2k values to be a, and all the others should be equal or less than a. This problem can be calculated as follows: there exist p pairs in $2k$ pairs with bit distance equals to a, $(2k - p)$ pairs would have bit distance values $< a$, or $\leq (a - 1)$. This leads to

$Pr[max_{2,p\in\{1,\cdots,2k\}}\{bdis(x_{1p}, x_{2p})\} = a]$

$$= \Sigma_{j=1}^{2k} \begin{pmatrix} 2k \\ j \end{pmatrix} (\frac{2^{a-1}}{2^B})^j (\frac{1}{2^B} + \Sigma_{q=1}^{a-1} \frac{2^{q-1}}{2^B})^{2k-j}$$

$$= \Sigma_{j=1}^{2k} \begin{pmatrix} 2k \\ j \end{pmatrix} (\frac{2^{a-1}}{2^B})^{2k}$$

because $\Sigma_{j=1}^{2k} \begin{pmatrix} 2k \\ j \end{pmatrix} = 2^{2k} - 1$,

$Pr[max_{2,p\in\{1,\cdots,2k\}}\{bdis(x_{1p}, x_{2p})\} = a] = (\frac{2^{a-1}}{2^B})^{2k}(2^{2k} - 1)$.

For three hyper-rectangles' maximum bit distance value in $2k$ problem space, we compare $\begin{pmatrix} 3 \\ 2 \end{pmatrix} 2k$ pairs of coordinate values. The function for the probability of three hyper-rectangles with maximum bit distance value equal to a is:

$$Pr[max_{3,p\in\{1,\cdots,2k\}}\{bdis(x_{1p}, x_{2p}), bdis(x_{1p}, x_{3p}), bdis(x_{2p}, x_{3p})\} = a]$$

$$= \Sigma_{j=1}^{3\cdot 2k} \begin{pmatrix} 3 \cdot 2k \\ j \end{pmatrix} (\frac{2^{a-1}}{2^B})^j (\frac{1}{2^B} + \Sigma_{q=1}^{a-1} \frac{2^{q-1}}{2^B})^{3\cdot 2k-j}$$

$$= \Sigma_{j=1}^{3\cdot 2k} \begin{pmatrix} 3 \cdot 2k \\ j \end{pmatrix} (\frac{2^{a-1}}{2^B})^{3\cdot 2k}$$

$$= (\frac{2^{a-1}}{2^B})^{3\cdot 2k}(2^{3\cdot 2k} - 1)$$

The general function for the probability of i random hyper-rectangles in $2k$ space is as follows:

$Pr[max_{i,p\in\{1,\cdots,2k\}}\{\} = a]$

$$= \Sigma_{j=1}^{\binom{i}{2}2k} \binom{\binom{i}{2}2k}{j} (\tfrac{2^{a-1}}{2^B})^j [(\tfrac{1}{2^B}) + \Sigma_{q=1}^{a-1}(\tfrac{2^{q-1}}{2^B})] \binom{i}{2}^{2k-j}$$

$$= \Sigma_{j=1}^{i(i-1)k} \binom{i(i-1)k}{j} (\tfrac{2^{a-1}}{2^B})^{i(i-1)k}$$

$$= (\tfrac{2^{a-1}}{2^B})^{i(i-1)k}(2^{i(i-1)k}-1) \tag{4.2}$$

Using equation (4.2) in equation (4.1) gives the expected number of nodes in the trie as

$S(n,k) = 2k\mathrm{B}+2k\Sigma_{i=2}^{n}(\Sigma_{a=1}^{B} a(\tfrac{2^{a-1}}{2^B})^{i(i-1)k}(2^{i(i-1)k}-1))$

Letting $u = 2^{i(i-1)k}$, we have

$S(n,k) = 2k\mathrm{B}+2k\Sigma_{i=2}^{n}\Sigma_{a=1}^{B} a u^a \, \frac{u-1}{u \cdot u^B}$

Letting $I = \Sigma_{a=1}^{B} a u^a$:

$I = 1u + 2u^2 + 3u^3 + \cdots + Bu^B$

$\frac{I}{u} = 1 + 2u + 3u^2 + \cdots + Bu^{B-1}$

$\frac{I}{u} - I = 1 + u + u^2 + \cdots + u^{B-1} - B\,u^B$

$I(\frac{1}{u} - 1) = \frac{1-u^B}{1-u} - \mathrm{B}u^B$

$I = (\frac{1-u^B}{1-u} - Bu^B)\frac{u}{1-u}$, and

$S(n,k) = 2kB + 2k\Sigma_{i=2}^{n}\frac{u-1}{u}\frac{1}{u^B}\frac{u}{1-u}(\frac{1-u^B}{1-u} - Bu^B)$

$= 2kB + 2k\Sigma_{i=2}^{n}\frac{1}{u^B}(Bu^B - \frac{1-u^B}{1-u})$

$= 2kB + 2k\Sigma_{i=2}^{n}(B - \frac{\frac{1}{u^B}-1}{1-u})$

$= 2kB + 2k\Sigma_{i=2}^{n}B - 2k\Sigma_{i=2}^{n}\frac{1}{u-1} + 2k\Sigma_{i=2}^{n}\frac{1}{u^B(u-1)}$

$= 2kB + 2k\Sigma_{i=2}^{n} B - o(1) + 2ko(1)$

$= \Theta(kBn).$ ∎

4.2.2 Range Search Cost

Without loss of generality, we consider our analysis in real $[0,1]^k$ space. Using a mapping function $f = \frac{coordinate\ values}{2^B - 1}$, we can map any integer coordinate values into the unit interval on each dimension.

Partial Match Retrieval Using Tries

The analysis of partial match retrieval using k-d tries was eloquently addressed by Flajolet and Puech [FlPu86]. Adapting this analysis to k-d hyper-rectangles in $2k$-d tries, we ask for all hyper-rectangles in data set D satisfying query hyper-rectangle q, where $q = (q_1, q_2, \cdots, q_{2k})$, $S \subset \{1, 2, \cdots, 2k\}$, $s = |S|$, s of the $2k$ query key values are specified, and $2k - s$ query values are left unspecified; we denote the unspecified queries as $q_p = *$, $p \notin S$ as wild cards. Hyper-rectangle R_i satisfies a partial match query q if and only if $q_p = R_{ip}, \forall p \in S$.

Our analysis is based on the uniform probabilistic model where we assume the input data D and the query hyper-rectangle W are drawn from a uniform random distribution, and that the keys of D are independent. Here, we assume that the keys of D are $2k$-dimensional points representing a k-dimensional hyper-rectangle. A k-d random variable $\mathbf{x}$ exists for the left side x_{ij}^{min} of each hyper-rectangle R_i, and a second k-d random variable $\mathbf{y}$ exists for the right side x_{ij}^{max} of each R_i. The $2k$-d keys of D are thus formed from two independent random variables, and the joint density function $\mathbf{t}$ of each key is the product of the density functions of $\mathbf{x}$ and $\mathbf{y}$. The product of two uniform random density functions gives a uniform random density function $\mathbf{t}$,

which allows us to use the following theorem:

Theorem 5 *Given a binary $2k$-d trie T containing a set of k-dimensional input hyper-rectangles $D = \{R_1, \cdots, R_n\}$, assuming data set D and query hyper-rectangle q satisfy the uniform probabilistic model, $q = (q_1, q_2, \cdots, q_{2k})$, $S \subset \{1, 2, \cdots, 2k\}$, $2k > s = |S| \geq 0$, the average cost of partial match retrieval $Q_S(n,k)$ measured by the number of nodes traversed in trie T is*

$$Q_S(n,k) = \gamma(\tfrac{1}{2k}\log_2 n) n^{1-\frac{s}{2k}} + O(1),$$

where γ_u is a periodic function of u with period 1, small amplitude, and mean value

$$\gamma_0 = -\tfrac{s}{4k^2 \log 2}\Gamma(\tfrac{s}{2k} - 1)\, \Sigma_{\ell=0}^{2k-1}(\delta_1\, \delta_2\, \cdots\, \delta_\ell) 2^{-\ell(1-s/2k)}$$

with $\delta_\ell = 1$, if the ℓ^{th} attribute of the query is specified, and $\delta_\ell = 2$ if it is unspecified.

Theorem 5 is a restatement of Flajolet and Puech's theorem 2 in [FlPu86] for our $2k$-d trie.

Proposition 1 *Given a binary trie T containing a set of k-dimensional input hyper-rectangles $D = \{R_1, \cdots, R_n\}$, assuming input data set D and query hyper-rectangle q satisfy the uniform probabilistic model, $q = (q_1, q_2, \cdots, q_{2k})$, $S \subset \{1, 2, \cdots, 2k\}$, the expected cost of partial match retrieval $Q_S(n,k)$ measured by the number of nodes traversed in trie T is*

$$Q_S(n,k) = E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p \in S} |NC_t^p|].$$

Proof. If a node is visited, $q_p \in NC^p = [L_p, H_p], \forall p \in S$. The probability that a node in trie T will be visited is determined by the volume of every node's cover space in the space $[0,1]$. ∎

Analysis of Orthogonal Range Search Using Tries

To get the color types for a node in the trie, we compare all the $2k$ ranges of WC^{2k} with NC^{2k}. In our algorithm, on each level, we do one comparison of the $2k$ ranges and store the color as the node's state. If all $2k$ ranges are black, the node is black; if one range is white, the node is white, and all the other conditions indicate the node is a grey node. Traversing from the root down through the first $2k$ levels, we finish the comparison of $2k$ ranges of WC^{2k} and NC^{2k} on $2k$ dimensions and get the first batch of white and black nodes. On a certain level ℓ in the trie T, after half splitting the cover space from the root (which is the full search space), there are altogether $2^{\frac{\ell}{2k}+1}$ possibilities of the ranges for NC^p. They are $[MIN, MIN + 2^{B-\frac{\ell}{2k}-1}], [MIN + 2^{B-\frac{\ell}{2k}-1}, MIN + 2 \cdot 2^{B-\frac{\ell}{2k}-1}], \cdots, [MIN + (2^{\frac{\ell}{2k}+1} - 1)2^{B-\frac{\ell}{2k}-1}, MAX]$. Each range's length equals to $2^{B-\frac{\ell}{2k}-1}$, $|NC^p| = 2^{B-\frac{\ell}{2k}-1}$, and the exact NC^p we want to compare with WC^p on this node is determined by previous paths. On level ℓ, the query cover space WC^p we want to compare with is on the j^{th} dimension in data space, $j = (\ell \, mod \, (2k))$. If $j \, mod \, 2 = 0$, we select $WC^p = [MIN, H_j)$; if $j \, mod \, 2 = 1$, $WC^p = (L_j, MAX]$.

Traversing stops on paths when we meet with black or white nodes and continues when grey nodes are encountered and continues collecting black nodes in the subtree of the black nodes we first met. The time complexity can be determined by computing the number of grey and black nodes in the trie built from input data D. We have the following equation:

$$Q(n,k) = \Sigma_{t=1}^{\textit{the number of nodes in trie}} 1_{[node_t \in GN \cup BN]}$$

where we use $1_{[A]}$ as the characteristic function of the event A. The formula counts the number of grey nodes, which, apart from the black nodes traversed to report the in-range hyper-rectangles, represents the time complexity of the orthogonal range search algorithm (see Figure 4.3).

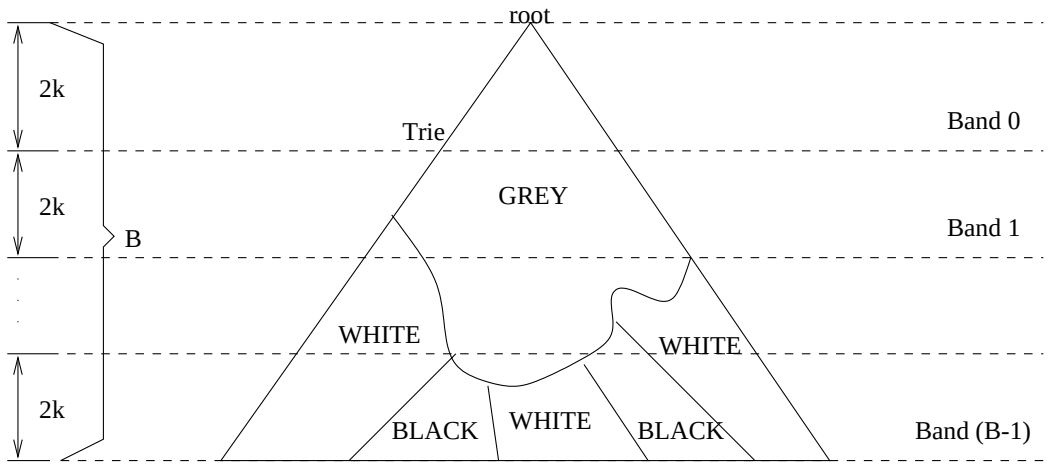


Figure 4.3: GREY nodes for computing the time complexity.

Lemma 2 $E[\Sigma_{t=1}^{number\ of\ nodes\ in\ trie} \prod_{p=1}^{2k} |NC_t^p|] = O(\log_2 n)$.

Proof. Every band has height $2k$. In the b^{th} band of height $2k$ ($h = 1..2k$), we have

$$|NC_{bh}^j| = \frac{1}{2^{b+1}}, 1 \leq j \leq h;\ |NC_{bh}^j| = \frac{1}{2^b}, h+1 \leq j \leq 2k$$

where we have used the notation $|NC_{bh}^p|$ to represent the range for one dimension p at a node of height h within band b. This leads to the volume of one node's data cover $|NC_t|$ on $2k$ dimensions as

$$|NC_t| = \prod_{p=1}^{2k} |NC_t^p| = (\frac{1}{2^{b+1}})^h (\frac{1}{2^b})^{2k-h} = 2^{-2kb-h} \tag{4.3}$$

for node t lying at height h in band b.

The volume $|NC_t|$ is the same for all nodes on the same level in the trie; as the level ℓ increases, the value of $|NC_t|$ decreases.

We calculate the number of nodes on a certain height h in one band using the worst case for storage (shown in Figure 4.1). The level $r = \lfloor \log_2 n \rfloor$ (the last level in the trie that coincides with a complete binary tree, see Figure 4.1) lies in the band $d = \lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor$, which is at height $h = \lfloor \log_2 n \rfloor + 1 - 2kd$ in the d^{th} band. As shown in Figure 4.4, we divide B bands into three parts: (a) from band 0 to band $d-1$, (b) band d, and (c) from band $d+1$ to band B-1. Within part(a), the number of nodes on height h of band b is 2^ℓ, $\ell = 2kb + h - 1$. We divide band d into two: one is from height 1 to height $\lceil \log_2 n \rceil + 1 - 2kd$, the other is from height $\lceil \log_2 n \rceil + 2 - 2kd$ to $2k$. Within part(b)'s first part, the number of nodes on height h of band d is $2^{2kd+h-1}$. Within band d's second part, the number of nodes on height h of band d is n. The number of nodes on each level within part(c) is n. These three sections are illustrated in Figure 4.4. Thus, we have

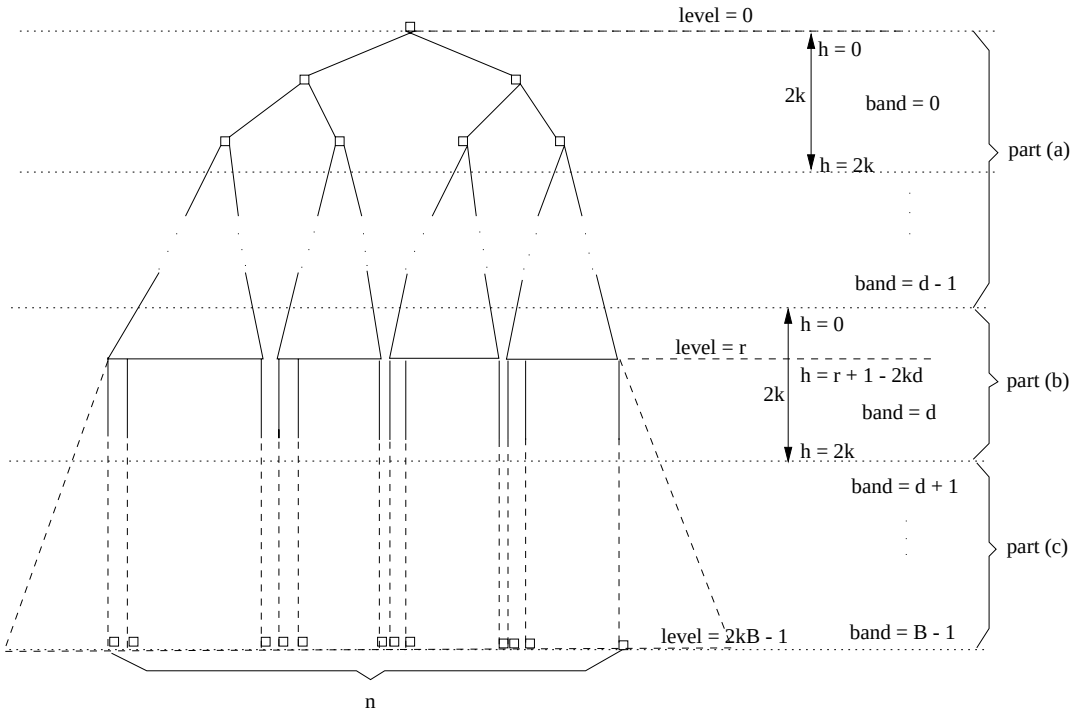


Figure 4.4: Three sections of the $2k$-d trie for average case space analysis.

$$
\begin{aligned}
&E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \textstyle\prod_{p=1}^{2k} |NC_t^p|] \\
&\quad = E[\Sigma_{b=0}^{B-1} \Sigma_{h=1}^{2k} \Sigma_{t=1}^{\textit{number of nodes on height } h \textit{ in band } b} \textstyle\prod_{p=1}^{2k} |NC_{bh}^p|] \\
&\quad = \Sigma_{b=0}^{\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor - 1} \Sigma_{h=1}^{2k} \Sigma_{t=1}^{2^{2kb+h}} \textstyle\prod_{p=1}^{2k} |NC_{bh}^p| && \text{part(a)} \\
&\qquad + \Sigma_{b=\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor}^{\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor} \Sigma_{h=1}^{\lfloor \log_2 n \rfloor + 1 - 2kb} \Sigma_{t=1}^{2^{2kb+h}} \textstyle\prod_{p=1}^{2k} |NC_{bh}^p| && \text{part(b)1} \\
&\qquad\quad + \Sigma_{b=\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor}^{\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor} \Sigma_{h=\lfloor \log_2 n \rfloor + 2 - 2kb}^{2k} \Sigma_{t=1}^{n} \textstyle\prod_{p=1}^{2k} |NC_{bh}^p| && \text{part(b)2} \\
&\qquad + \Sigma_{b=\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor + 1}^{B-1} \Sigma_{h=1}^{2k} \Sigma_{t=1}^{n} \textstyle\prod_{p=1}^{2k} |NC_{bh}^p| && \text{part(c)}
\end{aligned}
$$

Because $\Sigma_{t=1}^{2^{2kb+h}} \prod_{p=1}^{2k} |NC_{bh}^p| = \Sigma_{t=1}^{2^{2kb+h}} 2^{-2kb-h} = \frac{1}{2}$, we have

$$
\begin{aligned}
&E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \textstyle\prod_{p=1}^{2k} |NC_t^p|] \\
&= \Sigma_{b=0}^{\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor - 1} \Sigma_{h=1}^{2k} \frac{1}{2}
\end{aligned}
$$

$$+\Sigma_{b=\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor}^{\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor} \Sigma_{h=1}^{\lfloor \log_2 n \rfloor + 1 - 2kb} \frac{1}{2}$$

$$+\Sigma_{b=\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor}^{\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor} \Sigma_{h=\lfloor \log_2 n \rfloor + 2 - 2kb}^{2k} n 2^{-2kb-h}$$

$$+\Sigma_{b=\lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor + 1}^{B-1} \Sigma_{h=1}^{2k} n 2^{-2kb-h}$$

$$= \lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor \frac{2k-1}{2}$$

$$+ (\frac{\lfloor \log_2 n \rfloor}{2} - \frac{2k}{2} \lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor)$$

$$+ (\frac{n}{2^{\lfloor \log_2 n \rfloor}} - \frac{2n}{2^{2k} 2^{2k \lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor}})$$

$$+ (\frac{n}{2^{2k} 2^{2k \lfloor \frac{\lfloor \log_2 n \rfloor}{2k} \rfloor}} - \frac{n}{2^{2kB+4k}})$$

As $n < (2^{2B-1} - 2^{B-1})^k$, $\frac{n}{2^{2kB+4k}} < 1$, and we obtain the following result:

$E[\Sigma_{t=1}^{\text{number of nodes in trie}} \prod_{p=1}^{2k} |NC_t^p|] = O(\log_2 n)$. ∎

Theorem 6 *Given a binary trie T containing a set of k-dimensional input rectangles $D = \{R_1, R_2, \cdots, R_n\}$, R_i with i.i.d. random variable center c_i on $[0,1]^k$, and with i.i.d. random variable side length d_i distributed on $[0,1]^k$, consider a random orthogonal range search with query hyper-rectangle W with center at Z which is uniformly distributed on $[0,1]^k$, and independent of the centers of D, and with size $\Delta_1 \times \Delta_2 \times \cdots \times \Delta_k$ which are also i.i.d. random variables on $[0,1]^k$. The expected orthogonal range search time*

$$E[Q(n,k)] \leq \Sigma_{S \subset \{1,\cdots,2k\}} (\prod_{p \notin S} |WC^p|)(\gamma(\frac{1}{2k} \log_2 n) n^{1 - \frac{s}{2k}} + O(1)) + O(\log_2 n),$$

where γ_u is a periodic function of u with period 1, small amplitude, and mean value

$$\gamma_0 = -\frac{s}{4k^2 \log 2} \Gamma(\frac{s}{2k} - 1) \Sigma_{\ell=0}^{2k-1} (\delta_1 \, \delta_2 \, \cdots \, \delta_\ell) 2^{-\ell(1 - s/2k)}$$

with $\delta_\ell = 1$, if the ℓ^{th} attribute of the query is specified, and $\delta_\ell = 2$ if it is unspecified.

Proof. $E[Q(n,k)] = E[\, \Sigma_{t=1}^{\text{number of nodes in trie}} \, 1_{[node_t \in GN \cup BN]}]$

This calculation includes the reporting time for collection of the subtree of black nodes which arises during the traversal. The probability that a node is black or grey is given as: $Pr[node_i \in GN \cup BN] \leq \prod_{p=1}^{2k}(|NC^p| + |WC^p|)$. The probability for query hyper-rectangle W's cover space WC to intersect a node's cover space NC is the probability that Z_j, the center of W, is within distance $\frac{\Delta_j}{2}$ of NC^j. This probability is bounded by the volume of NC expanded by Δ_j in the j^{th} dimension, $\forall j \in \{1, \cdots, k\}$. There are two cases. On the left side of the j^{th} dimension, $|WC_j^{min}| = |WC^p| = |\,[0, H_j)\,| = H_j = Z_j + \frac{\Delta_j}{2}$, and $p \bmod 2 = 1$. On the right side of the j^{th} dimension, $|WC_j^{max}| = |WC^p| = |\,(L_j, 1]\,| = 1 - L_j = 1 - (Z_j - \frac{\Delta_j}{2}) = 1 - Z_j + \frac{\Delta_j}{2}$, and $p \bmod 2 = 0$. We have

$$
\begin{aligned}
E[Q(n,k)] &\leq E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p=1}^{2k}(|WC^p| + |NC_t^p|)] \\
&= \Sigma_{S \subseteq \{1,\cdots,2k\}}(\prod_{p \notin S} |WC^p|) E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p \in S} |NC_t^p|] \\
&= \Sigma_{S \subset \{1,\cdots,2k\}}(\prod_{p \notin S} |WC^p|) E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p \in S} |NC_t^p|] \\
&\quad + \Sigma_{S = \{1,\cdots,2k\}}(\prod_{p \notin S} |WC^p|) E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p=1}^{2k} |NC_t^p|] \\
&= \Sigma_{S \subset \{1,\cdots,2k\}}(\prod_{p \notin S} |WC^p|) E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p \in S} |NC_t^p|] \\
&\quad + E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p=1}^{2k} |NC_t^p|]
\end{aligned}
$$

From Theorem 5 and Proposition 1, we obtain

$$
\begin{aligned}
E[Q(n,k)] &\leq \Sigma_{S \subset \{1,\cdots,2k\}}(\prod_{p \notin S} |WC^p|)(\gamma(\tfrac{1}{2k} \log_2 n) n^{1 - \frac{s}{2k}} + O(1)\,) \\
&\quad + E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p=1}^{2k} |NC_t^p|],
\end{aligned}
$$

and by Lemma 2, we obtain

$$E[Q(n,k)] \leq \Sigma_{S \subset \{1,\cdots,2k\}}(\prod_{p \notin S} |WC^p|)(\gamma(\tfrac{1}{2k} \log_2 n) n^{1 - \frac{s}{2k}} + O(1)\,) + O(\log_2 n). \quad \blacksquare$$

Lower Bounds

Using the approach demonstrated by [CDZ01], we arrive at the following proposition:

Proposition 2 $E[\Sigma_{t=1}^{t=number\ of\ nodes\ in\ trie} \prod_{p\in S} 1_{[\exists p\in\{1,\cdots,2k\}:|NC_t^p|\geq\frac{1}{2}]}] \leq C$,

where C is a constant value, $C > 0$, depending on k only.

Proof. Within the first two bands of the trie T, the node's cover space's size on one dimension will be greater or equal to a half.

Table 4.1: The node's cover space on $2k$ dimensions in the trie T.

BAND	*LEVEL*	NC^1	NC^2	NC^3	NC^4	$\cdots$	NC^{2k}
	0	1	1	1	1		1
0	1	1/2	1	1	1		1
	2	1/2	1/2	1	1		1
	⋮						
	$2k$	1/2	1/2	1/2	1/2		1/2
1	$2k$+1	1/4	1/2	1/2	1/2		1/2
	$2k$+2	1/4	1/4	1/2	1/2		1/2
	⋮						
	$2k$+$2k$	1/4	1/4	1/4	1/4		1/4
2	$4k$+1	1/8	1/4	1/4	1/4		1/4
	$4k$+2	1/8	1/8	1/4	1/4		1/4
	⋮						
	$6k$	1/8	1/8	1/8	1/8		1/8
⋮							
B-1	$2k$(B-2)+1	$1/2^B$	$1/2^{B-1}$	$1/2^{B-1}$	$1/2^{B-1}$		$1/2^{B-1}$
	$2k$(B-2)+2	$1/2^B$	$1/2^B$	$1/2^{B-1}$	$1/2^{B-1}$		$1/2^{B-1}$
	⋮						
	$2k$(B-1)	$1/2^B$	$1/2^B$	$1/2^B$	$1/2^B$		$1/2^B$

From Table 4.1, we find that from level 0 to level $4k$-1, $\exists p \in \{1,\cdots,2k\}, |NC_t^p| \geq \frac{1}{2}$. Even if our trie coincides with a full binary tree which has the maximum number

of nodes in these levels, the number of nodes with a cover space's size $\geq \frac{1}{2}$ is still below a limited number, i.e.

$$2^0 + 2^1 + 2^2 + \cdots + 2^{4k-1} = 2^{4k} - 1.$$

$E[\Sigma_{t=1}^{t=number of nodes in the trie} \prod_{p \in S} 1_{[\exists p \in \{1,\cdots,2k\}:|NC_t^p| \geq \frac{1}{2}]}] \leq 2^{4k} - 1$, which is a constant value only related to k. ∎

Theorem 7 *Given a binary trie T containing a set of k-dimensional input rectangles $D = \{R_1, R_2, \cdots, R_n\}$, R_i with i.i.d. random variable center c_i on $[0,1]^k$, and with i.i.d. random variable side length d_i distributed on $[0,1]^k$, consider a random orthogonal range search with query hyper-rectangle W with center at Z which is uniformly distributed on $[0,1]^k$, and independent of the centers of D, and with size $\Delta_1 \times \Delta_2 \times \cdots \times \Delta_k$ which are also i.i.d. random variables on $[0,1]^k$. The expected orthogonal range search time*

$$E[Q(n,k)] \geq \Sigma_{S \subset \{1,\cdots,2k\}} (\prod_{p \notin S} \frac{|WC^p|}{2}) (\gamma(\frac{1}{2k} \log_2 n) n^{1-\frac{s}{2k}} + O(1)) - C,$$

where C is a constant value determined by k, and γ_u is a periodic function of u with period 1, small amplitude, and mean value

$$\gamma_0 = -\frac{s}{4k^2 \log 2} \Gamma(\frac{s}{2k} - 1) \, \Sigma_{\ell=0}^{2k-1} (\delta_1 \, \delta_2 \, \cdots \, \delta_\ell) 2^{-\ell(1-s/2k)}$$

with $\delta_\ell = 1$, if the ℓ^{th} attribute of the query is specified, and $\delta_\ell = 2$ if it is unspecified.

Proof. $E[Q(n,k)] \geq E[\Sigma_{t=1}^{number of nodes in trie} 1_{[node_t \in GN \cup BN]} 1_{[\forall p \in \{1,\cdots,2k\}:|NC_t^p| < \frac{1}{2}]}]$

$\geq E[\Sigma_{t=1}^{number of nodes in trie} \prod_{p=1}^{2k} (|NC_t^p| + \frac{|WC^p|}{2}) 1_{[\forall p \in \{1,\cdots,2k\}:|NC_t^p| < \frac{1}{2}]}]$

$= E[\Sigma_{t=1}^{number of nodes in trie} \prod_{p=1}^{2k} (|NC_t^p| + \frac{|WC^p|}{2})]$

$-E[\Sigma_{t=1}^{number of nodes in trie} \prod_{p=1}^{2k} (|NC_t^p| + \frac{|WC^p|}{2}) 1_{[\exists p \in \{1,\cdots,2k\}:|NC_t^p| \geq \frac{1}{2}]}]$

$$= \Sigma_{S\subseteq\{1,\cdots,2k\}} \prod_{p\notin S} \frac{WC^p}{2} E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p\in S} |NC_t^p|]$$

$$-\Sigma_{S\subseteq\{1,\cdots,2k\}} \prod_{p\notin S} \frac{WC^p}{2} E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p\in S} |NC_t^p| 1_{[\exists p\in\{1,\cdots,2k\}:|NC_t^p|\geq\frac{1}{2}]}]$$

From Proposition 2, we can bound the second item as:

$$\Sigma_{S\subseteq\{1,\cdots,2k\}} \prod_{p\notin S} \frac{WC^p}{2} E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p\in S} |NC_t^p| 1_{[\exists p\in\{1,\cdots,2k\}:|NC_t^p|\geq\frac{1}{2}]}]$$

$$\leq E[\Sigma_{t=1}^{\textit{number of nodes in trie}} 1_{[\exists p\in\{1,\cdots,2k\}:|NC_t^p|\geq\frac{1}{2}]}]$$

$$\leq C$$

Based on Theorem 5,

$$E[Q(n,k)] \geq \Sigma_{S\subseteq\{1,\cdots,2k\}} \prod_{p\notin S} \frac{WC^p}{2} E[\Sigma_{t=1}^{\textit{number of nodes in trie}} \prod_{p\in S} |NC_t^p|] - C$$

$$= \Sigma_{S\subseteq\{1,\cdots,2k\}} \prod_{p\notin S} \frac{WC^p}{2} (\gamma(\tfrac{1}{2k}\log_2 n) n^{1-\frac{s}{2k}} + O(1))) - C. \quad \blacksquare$$

Further Analysis

Based on Theorem 6 and Theorem 7, we see that

$$\Sigma_{S\subset\{1,\cdots,2k\}} (\prod_{p\notin S} \frac{|WC^p|}{2}) (\gamma(\tfrac{1}{2k}\log_2 n) n^{1-\frac{s}{2k}} + O(1)) - C$$

$$\leq E[Q(n,k)] \leq$$

$$\Sigma_{S\subset\{1,\cdots,2k\}} (\prod_{p\notin S} |WC^p|) (\gamma(\tfrac{1}{2k}\log_2 n) n^{1-\frac{s}{2k}} + O(1)) + O(\log_2 n)$$

and we can write the expected range search time as

$$E[Q(n,k)] = c_1 \prod_{p=1}^{2k} |WC^p| n + c_2 \Sigma_{S\subset\{1,\cdots,2k\},0<|S|<2k} (\prod_{p\notin S} |WC^p|) \gamma(\tfrac{1}{2k}\log_2 n) n^{1-\frac{s}{2k}}$$

$$+O(\log_2 n) \tag{4.4}$$

where c_1 and c_2 are constant values related to the specified S. The first term accounts for the number of hyper-rectangles returned by the orthogonal range search. If $S = \emptyset$, all queries are wildcards, and everything is in range (just report), so the first term dominates. When $0 < s < 2k$, the second term dominates. When

$S = \{1, \cdots, 2k\}$, (i.e. s = 2k), $O(\log_2 n)$ dominates. The third item arises from the height of the trie which is unavoidable.

One-dimensional special case

When $k = 1$,

$$E[Q(n,1)] = c_1 \prod_{p=1}^{2} |WC^p| n + c_2 \Sigma_{S \subset \{1,2\}, 1<|S|<2} (\prod_{p \notin S} |WC^p|) \gamma(\frac{\log_2 n}{2}) n^{1-\frac{s}{2}} + O(\log_2 n)$$

$$= c_1 |WC^1||WC^2| n + c_2 \gamma(\frac{\log_2 n}{2}) n^{\frac{1}{2}} (|WC^1| + |WC^2|) + O(\log_2 n)$$

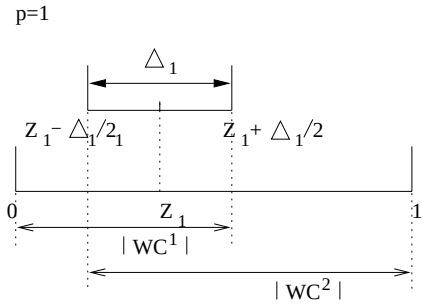


Figure 4.5: Query hyper-rectangle's cover space $|WC^1|$ and $|WC^2|$ on the first dimension ($p = 1$).

As shown in Figure 4.5, $|WC^1| = Z_1 + \frac{\Delta_1}{2}$, $|WC^2| = 1 - Z_1 + \frac{\Delta_1}{2}$, and

$$E[Q(n,1)] = c_1(Z_1 + \tfrac{\Delta_1}{2})(1 - Z_1 + \tfrac{\Delta_1}{2}) n + c_2 \gamma(\tfrac{\log_2 n}{2}) n^{\frac{1}{2}} (Z_1 + \tfrac{\Delta_1}{2} + 1 - Z_1 + \tfrac{\Delta_1}{2}) +$$

$O(\log_2 n)$

$$= c_1(Z_1 + \tfrac{\Delta_1}{2})(1 - Z_1 + \tfrac{\Delta_1}{2}) n + c_2 \gamma(\tfrac{\log_2 n}{2}) n^{\frac{1}{2}} (1 + \Delta_1) + O(\log_2 n)$$

Either the first term or the second term dominates, depending on Z_1 and Δ_1. If the second term dominates, then $max\{Z_1, \Delta_1\} < O(n^{-\frac{1}{2}})$, otherwise the first term dominates. If we write $Z_1 = \frac{1}{n^{a_1}}$, $\Delta_1 = \frac{1}{n^{b_1}}$, $a_1, b_1 \geq 0$, then:

$$E[Q(n,1)] = \begin{cases} O(n^{\frac{1}{2}}), & min\{a_1, b_1\} \geq \frac{1}{2} \\ O(n^{1-min\{a_1,b_1\}}), & min\{a_1, b_1\} < \frac{1}{2} \end{cases} \quad (4.5)$$

Two-dimensional special case

When $k = 2$, $E[Q(n,2)]$

$= c_1 \prod_{p=1}^{4} |WC^p| n + c_2 \Sigma_{S \subset \{1,2,3,4\}, 0<|S|<4} (\prod_{p \notin S} |WC^p|) \gamma(\frac{\log_2 n}{4}) n^{1-\frac{s}{4}} + O(\log_2 n)$

$= c_1 n(|WC^1||WC^2||WC^3||WC^4|)$

$+c_2\gamma(\frac{\log_2 n}{4})n^{1-\frac{1}{4}}(|WC^2||WC^3||WC^4| + |WC^1||WC^3||WC^4|$

$+ |WC^1||WC^2||WC^4| + |WC^1||WC^2||WC^3|)$

$+c_2\gamma(\frac{\log_2 n}{4})n^{1-\frac{2}{4}}(|WC^3||WC^4| + |WC^2||WC^4| + |WC^2||WC^3|$

$+ |WC^1||WC^4| + |WC^1||WC^3| + |WC^1||WC^2|)$

$+c_2\gamma(\frac{\log_2 n}{4})n^{1-\frac{3}{4}}(|WC^1| + |WC^2| + |WC^3| + |WC^4|) + O(\log_2 n)$

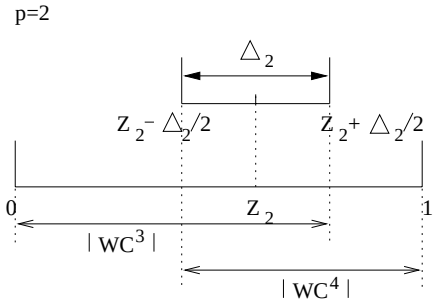


Figure 4.6: Query hyper-rectangle's cover space $|WC^3|$ and $|WC^4|$ on the second dimension ($p = 2$).

As shown in Figure 4.5 and 4.6, $|WC^1| = Z_1 + \frac{\Delta_1}{2}$, $|WC^2| = 1 - Z_1 + \frac{\Delta_1}{2}$, $|WC^3| = Z_2 + \frac{\Delta_2}{2}$, $|WC^4| = 1 - Z_2 + \frac{\Delta_2}{2}$

based on the assumptions of Theorems 6 and 7. Substituting these query window cover values gives:

$E[Q(n,2)] = c_1 n(Z_1 + \frac{\Delta_1}{2} - Z_1^2 + \frac{\Delta_1^2}{4})(Z_2 + \frac{\Delta_2}{2} - Z_2^2 + \frac{\Delta_2^2}{4})$

$+c_2\gamma(\frac{\log_2 n}{4})n^{\frac{3}{4}}(Z_1 + Z_2 + \frac{\Delta_1}{2} + \frac{\Delta_2}{2} - Z_1^2 - Z_2^2 + \frac{\Delta_1^2}{4} + \frac{\Delta_2^2}{4} + Z_1\Delta_2 + Z_2\Delta_1 -$

$Z_1^2\Delta_2 - Z_2^2\Delta_1 + \frac{1}{4}\Delta_1\Delta_2(\Delta_1 + \Delta_2))$

$+c_2(\gamma(\frac{\log_2 n}{4})n^{\frac{1}{2}}(1 + Z_1 + Z_2 + \frac{3}{2}\Delta_1 + \frac{3}{2}\Delta_2 - Z_1^2 - Z_2^2 + \frac{1}{4}\Delta_1^2 + \frac{1}{4}\Delta_2^2 + \Delta_1\Delta_2)$

$+c_2\gamma(\frac{\log_2 n}{4})n^{\frac{1}{4}}(2 + \Delta_1 + \Delta_2) + O(\log_2 n)$

If $Z_1 = \frac{1}{n^{a_1}}$, $Z_2 = \frac{1}{n^{a_2}}$, $\Delta_1 = \frac{1}{n^{b_1}}$, $\Delta_2 = \frac{1}{n^{b_2}}$, $a_1, a_2, b_1, b_2 \geq 0$, then:

$$E[Q(n,2)] = \begin{cases} O(n^{\frac{1}{2}}), & min\{a_1, a_2, b_1, b_2\} \geq \frac{1}{4} \\ O(n^{\frac{3}{4} - min\{a_1, a_2, b_1, b_2\}}), & min\{a_1, a_2, b_1, b_2\} < \frac{1}{4}\ AND \\ & min\{a_1 + a_2, a_1 + b_2, a_2 + b_1, b_1 + b_2\} > \frac{1}{4} + min\{a_1, a_2, b_1, b_2\} \\ O(n^{1 - min\{a_1 + a_2, a_1 + b_2, a_2 + b_1, b_1 + b_2\}}), & otherwise. \end{cases} \quad (4.6)$$

For example, assuming $Z_1 = Z_2 = 0.5$, $\Delta_1 = \Delta_2 = 0.9$,

$min\{a_1, a_2, b_1, b_2\} = 0.0115$, $min\{a_1 + a_2, a_1 + b_2, a_2 + b_1, b_1 + b_2\} = 0.023$,

$E[Q(n,2)] = O(n^{1-0.023}) = O(n^{0.977})$.

If we assume $\Delta_1 = \Delta_2 = 0.1$,

$min\{a_1, a_2, b_1, b_2\} = 0.075$, $min\{a_1 + a_2, a_1 + b_2, a_2 + b_1, b_1 + b_2\} = 0.15$,

$E[Q(n,2)] = O(n^{1-0.15}) = O(n^{0.85})$.

The expected search time increases with the size of the query hyper-rectangle.

Three-dimensional special case

When $k = 3$, $E[Q(n,k)]$

$= c_1 \prod_{p=1}^{6} |WC^p| n + c_2 \Sigma_{S \subset \{1,2,3,4,5,6\}} \prod_{p \notin S} |WC^p| \gamma(\frac{\log_2 n}{6}) n^{1 - \frac{s}{6}} + O(\log_2 n)$

$= c_1 \prod_{p=1}^{6} |WC^p| n + c_2 \gamma(\frac{\log_2 n}{6})(n^{1-\frac{1}{6}} I_1 + n^{1-\frac{2}{6}} I_2 + n^{1-\frac{3}{6}} I_3 + n^{1-\frac{4}{6}} I_4 + n^{1-\frac{5}{6}} I_5) +$

$O(\log_2 n)$

I_1 has $C_6^1 = 6$ items, and each item consists of five WC^p terms; I_2 has $C_6^2 = 15$

items, and each item consists of four WC^p terms; I_3 has $C_6^3 = 20$ items, and each item consists of three WC^p terms; I_4 has $C_6^4 = 15$ items, and each item consists of two WC^p terms; and $I_5 = (|WC^1| + |WC^2| + |WC^3| + |WC^4| + |WC^5| + |WC^6|)$. In the case of $|WC^1| = Z_1 + \frac{\Delta_1}{2}$, $|WC^2| = 1 - Z_1 + \frac{\Delta 1}{2}$, $|WC_3| = Z_2 + \frac{\Delta_2}{2}$, $|WC^4| = 1 - Z_2 + \frac{\Delta 2}{2}$, $|WC^5| = Z_3 + \frac{\Delta_3}{2}$, $|WC^6| = 1 - Z_3 + \frac{\Delta_3}{2}$, we put $|WC^p|$ into the formula and we can get the expected time complexity.

$n^{\frac{5}{6}} I_1 =$

$$n^{\frac{5}{6} - min\{a_1+b_1, a_1+c_1, a_1+a_2, a_1+b_2, a_1+c_2, b_1+c_1, b_1+a_2, b_1+b_2, b_1+c_2, c_1+a_2, c_1+b_2, c_1+c_2, a_2+b_2, a_2+c_2, b_2+c_2\}}$$

$n^{\frac{2}{3}} I_2 = n^{\frac{2}{3} - min\{a_1, a_2, a_3, b_1, b_2, b_3\}}$

$n^{\frac{1}{2}} I_3 = O(n^{\frac{1}{2}})$

$n^{\frac{1}{3}} I_4 = O(n^{\frac{1}{3}})$

$n^{\frac{1}{6}} I_5 = O(n^{\frac{1}{6}})$

Let $\Theta_1 = \{a_1 + b_1, a_1 + c_1, a_1 + a_2, a_1 + b_2, a_1 + c_2, b_1 + c_1, b_1 + a_2, b_1 + b_2, b_1 + c_2, c_1 + a_2, c_1 + b_2, c_1 + c_2, a_2 + b_2, a_2 + c_2, b_2 + c_2\}$,

$\Theta_2 = \{a_1, a_2, a_3, b_1, b_2, b_3\}$,

$\Theta_3 = \{a_1 + a_2 + a_3, b_1 + a_2 + a_3, a_1 + b_2 + a_3, b_1 + b_2 + a_3, a_1 + a_2 + b_3, b_1 + a_2 + b_3, a_1 + b_2 + b_3, b_1 + b_2 + b_3\}$

and we use $Lmin\{\Theta\}$ to represent the value in the set Θ closest to the minimum value in the set.

$$E[Q(n,3)] = \begin{cases} O(n^{\frac{1}{2}}), min\{\Theta_2\} \geq \frac{1}{6} \\ O(n^{\frac{2}{3}-min\{\Theta_2\}}),\ min\{\Theta_2\} < \frac{1}{6}\, AND\, Lmin\{\Theta_2\} \geq \frac{1}{6} \\ O(n^{\frac{5}{6}-min\{\Theta_1\}}), min\{\Theta_2\} < \frac{1}{6}\, AND\, Lmin\{\Theta_2\} < \frac{1}{6} \\ \qquad\qquad AND\ min\{\Theta_3\} - min\{\Theta_1\} \geq \frac{1}{6} \\ O(n^{1-min\{\Theta_3\}}),\ otherwise. \end{cases} \tag{4.7}$$

k dimensions

Due to the long polynomial function of $\Sigma_{S\in\{1,2,\cdots,2k\}}(\prod_{p\notin S}|WC^p|)$, we omit the details of the expansion, but we do tabulate the exact values for specific values of k, n, and $\prod_{p=1}^{2k}|WC^p|$. First let us have a look about the cost of partial match query. Table 4.2 is calculated from Lemma 8 [FlPu86] (also see section 2.5.3). When n is set to $500, 1000, 10000, \cdots$, we get the exact value of every specified pattern, and the "Asymptote" column is obtained from those values. Then we calculate the mean value $= \frac{\Sigma C_{u,n,s}}{C_{2k}^s}$.

Table 4.2: Exact mean value of the cost of partial-match query in a $2k$-d trie for all specified patterns with $k = 5$.

s/2k	Asymptote	n							
		500	1000	10000	50000	100000	500000	1000000	100000000
1/10	$1.58n^{0.9}$	425	788	6299	27131	50663	213439	395633	25393099
2/10	$1.76n^{0.8}$	255	440	2797	10360	18076	64441	110865	4541039
3/10	$2.01n^{0.7}$	156	251	1266	4021	6557	19864	31782	825774
4/10	$2.33n^{0.6}$	97	146	586	1591	2426	6266	9345	153181
5/10	$2.77n^{0.5}$	62	87	279	646	920	2033	2833	29174
6/10	$3.33n^{0.4}$	40	53	137	271	361	684	893	5774
7/10	$4.03n^{0.3}$	26	33	70	119	149	243	297	1213
8/10	$5.19n^{0.2}$	18	21	38	56	65	93	107	281
9/10	$6.45n^{0.1}$	12	14	22	28	31	39	43	77

Tables 4.3 and 4.4 illustrate the time complexity of range search. In Tables 4.3 and 4.4, we give the values of range search time cost for different k, n, and query hyper-rectangle's cover space $\prod_{p=1}^{2k} |WC^p|$. To make the formulation simpler, we select $|WC^p|$ on every dimension equal to the $2k$ root of $\prod_{p=1}^{2k} |WC^p|$. From equation (4.4), Tables 4.3 and 4.4 give the expected cost for orthogonal range search in our trie. Term1 $= \prod_{p=1}^{2k} |WC^p| n$ without constant factor c_1,

Term2 $= \Sigma_{S \subset \{1,\cdots,2k\}} (\prod_{p \notin S} |WC^p|) \gamma(\frac{\log_2 n}{2k}) n^{1-\frac{s}{2k}}$ without constant factor c_2, and

Term3 $= \log_2 n$. Omitting the constant coefficients, we obtain the results shown in Table 4.3 (for $k = 2$) and Table 4.4 (for $k = 5$). Appendices A and B give the C++ code for generating the three terms.

Table 4.3: Expected cost of orthogonal range search (equation (4.4)) for $k = 2$.

2k	$\prod_{p=1}^{2k} \lvert WC^p \rvert$	Terms	n				
			10	100	1000	10000	100000
4	0.05	Term1	0.5	5	50	500	5000
		Term2	20.15	78.85	293.59	1174.86	3414.70
		Term3	3.32	6.64	9.97	13.29	16.61
	0.15	Term1	1.5	15	150	1500	15000
		Term2	33.47	140.10	562.43	2396.27	7188.71
		Term3	3.32	6.64	9.97	13.29	16.61
	0.25	Term1	2.5	25	250	2500	25000
		Term2	43.01	185.74	770.43	3367.57	10228.87
		Term3	3.32	6.64	9.97	13.29	16.61
	0.35	Term1	3.5	35	350	3500	35000
		Term2	50.99	224.74	951.60	4225.01	12929.34
		Term3	3.32	6.64	9.97	13.29	16.61
	0.45	Term1	4.5	45	450	4500	45000
		Term2	58.05	259.76	1116.34	5011.51	15416.31
		Term3	3.32	6.64	9.97	13.29	16.61
	0.55	Term1	5.5	55	550	5500	55000
		Term2	64.50	292.03	1269.56	5747.70	17751.00
		Term3	3.32	6.64	9.97	13.29	16.61
	0.65	Term1	6.5	65	650	6500	65000
		Term2	70.48	322.25	1414.07	6445.53	19969.05
		Term3	3.32	6.64	9.97	13.29	16.61
	0.75	Term1	7.5	75	750	7500	75000
		Term2	76.10	350.86	1551.68	7112.70	22093.54
		Term3	3.32	6.64	9.97	13.29	16.61
	0.85	Term1	8.5	85	850	8500	85000
		Term2	81.44	378.15	1683.61	7754.53	24140.50
		Term3	3.32	6.64	9.97	13.29	16.61
	0.95	Term1	9.5	95	950	9500	95000
		Term2	86.53	404.33	1810.77	8374.92	26121.65
		Term3	3.32	6.64	9.97	13.29	16.61

4.2.3 Counting Grey Nodes Only

If we change the analysis to count only grey nodes, i.e.

$$Q(n,k) = \Sigma_{t=1}^{\textit{number of nodes in trie}} 1_{[node_t \in GN]},$$

our analysis becomes more complex due to the lack of simplification provided by the fact that $Pr[node_t \in GN \cup BN] = \prod_{p=1}^{2k}(|WC^p| + |NC^p|)$. The following is the exact probability calculation for BLACK and GREY color conditions in Figure 3.5.

Table 4.4: Expected cost of orthogonal range search (equation (4.4)) for $k = 5$.

2k	$\prod_{p=1}^{2k} \lvert WC^p \rvert$	Terms	n				
			10	100	1000	10000	100000
10	0.05	Term1	0.5	5	50	500	5000
		Term2	1498.25	6146.01	21629.74	83719.28	373124.93
		Term3	3.32	6.64	9.97	13.29	16.61
	0.15	Term1	1.5	15	150	1500	15000
		Term2	2522.22	10838.65	40298.76	165560.22	779595.17
		Term3	3.32	6.64	9.97	13.29	16.61
	0.25	Term1	2.5	25	250	2500	25000
		Term2	3246.15	14256.20	54373.89	229513.337	1107490.47
		Term3	3.32	6.64	9.97	13.29	16.61
	0.35	Term1	3.5	35	350	3500	35000
		Term2	3846.66	17137.41	6465.70	285522.87	1399556.70
		Term3	3.32	6.64	9.97	13.29	16.61
	0.45	Term1	4.5	45	450	4500	45000
		Term2	4374.57	19699.15	77358.55	336653.25	1669286.57
		Term3	3.32	6.64	9.97	13.29	16.61
	0.55	Term1	5.5	55	550	5500	55000
		Term2	4853.10	22041.57	87419.53	384359.60	1923175.23
		Term3	3.32	6.64	9.97	13.29	16.61
	0.65	Term1	6.5	65	650	6500	65000
		Term2	5295.17	24220.92	96856.79	429475.78	2164981.74
		Term3	3.32	6.64	9.97	13.29	16.61
	0.75	Term1	7.5	75	750	7500	75000
		Term2	5708.86	26272.59	105802.34	472535.10	2397129.10
		Term3	3.32	6.64	9.97	13.29	16.61
	0.85	Term1	8.5	85	850	8500	85000
		Term2	6099.64	28220.61	114346.39	513904.45	2621294.93
		Term3	3.32	6.64	9.97	13.29	16.61
	0.95	Term1	9.5	95	950	9500	95000
		Term2	6471.38	30082.19	122553.93	553849.71	2838701.74
		Term3	3.32	6.64	9.97	13.29	16.61

Lemma 3 *For the point set* $\{0, 1, \cdots, N\}$*, the number of distinct intervals on* $[0, N]$ *is* $\frac{(N+1)N}{2}$.

Proof. To get interval [x,y] on [0,N], the number of intervals

$$= \begin{pmatrix} N+1 \\ \\ 2 \end{pmatrix} = \frac{(N+1)N}{2} \qquad \blacksquare$$

In our calculation, $N = 2^B - 1$. The possible intervals on $[0, 2^B - 1]$ is $\frac{2^B(2^B-1)}{2}$.

As displayed in Figure 4.5, a BLACK nodes' cover space NC should satisfy conditions (d) and (e) of Figure 3.5. We assume the left coordinate value for WC is a and the right coordinate value for WC is b as shown in Figure 4.5 and Figure 4.6. Applying Lemma 3 on conditions (d) and (e), it is easy to see that we want to get intervals in [a,b], so the number of possible NC for BLACK on one dimension is $(b-a)(b-a+1)/2$. $Pr[node_t \in BN] = \prod_{p=1}^{2k} \frac{(b-a)(b-a+1)/2}{2^B(2^B-1)/2}$.

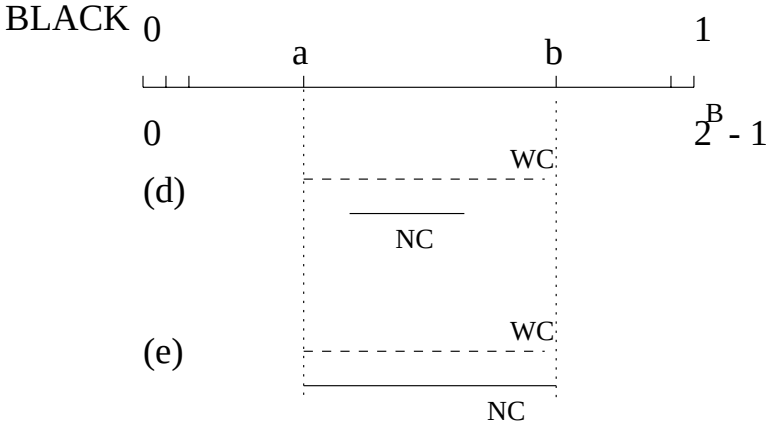


Figure 4.7: BLACK color on one dimension.

As displayed in Figure 4.8, a GREY nodes's cover space NC should satisfy condition (a), (b), and (c). For condition (a), NC's left coordinate value $\in [0, a-1]$, NC's right coordinate value $\in [a, b-1]$, so the number of nodes for case (a) is $a(b-a)$; for condition (b), NC's left coordinate value $\in [a+1, b]$, NC's right coordinate value $\in [b+1, 2^B-1]$, so the number of nodes for case (b) is $(b-a)(2^B-1-b)$; for condition (c), we obtain the following: $NC \supseteq [a,b]$ excluding $NC = [a,b]$, NC's left coordinate value $\in [0, a]$, NC's right coordinate value $\in [b, 2^B-1]$, and the number of nodes for case (c) is $(a+1)(2^B-1-b+1)-1$. So the total number of GREY nodes on one dimension is $(a)+(b)+(c) = (2^B+a-b-1)(b+1)-a^2$. This gives

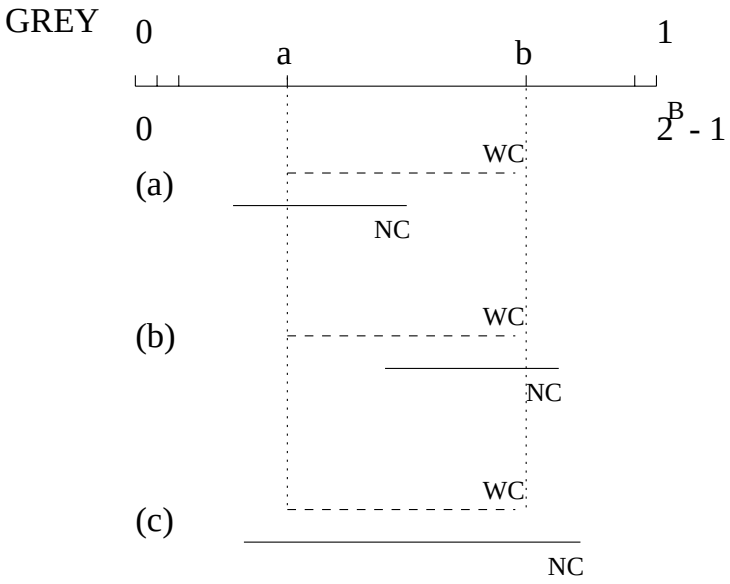


Figure 4.8: GREY color on one dimension.

$Pr[node_t \in GN] = \prod_{p=1}^{2k} \frac{(2^B+a-b-1)(b+1)-a^2}{2^B(2^B-1)/2}$. As we have mentioned before, as to a and b for WC, there are two cases. $a = 0$, $b = (Z_j + \frac{\Delta_j}{2})(2^B - 1)$, when $p \bmod 2 = 1$; $a = (Z_j - \frac{\Delta_j}{2})(2^B - 1)$, $b = 2^B - 1$, when $p \bmod 2 = 0$. Thus, we can see that determining $Pr[node_t \in GN]$ is challenging.

Chapter 5

Experimental Results

After we finished building the trie from a randomly generated data set which satisfied the uniform distribution, we randomly generated several query hyper-rectangles with different sizes, then searched the trie to get the range search results. The following sections contain the algorithm used for random number generation, a description of how we obtained the query hyper-rectangle and data set, and the main procedure for testing.

5.1 Algorithms

The main testing process is shown in Figure 5.1.

```
MAINTEST()
 1  for k ← 2 to 10
 2  do n ← 10
 3     while n < 100000
 4     do  call generateRandData function // create k-d trie using Trie_Insert
 5         // function to insert n randomly generated hyper-rectangles
 6         for outeri ← 0 to NUM_STEPS
 7         do for i ← 0 to NQUERY − 1
 8            do // we set NQUERY to get average performance
 9               call getQhR function to get a randomly generated query
10               hyper-rectangle
11               call Rangesearch function to get the range search results
12          free the trie space
13         n ← n × 10
```

Figure 5.1: Main test process.

5.1.1 Random number generator

My random number generator uses Knuth's algorithm in section 3.6 [Knut73-1] which implements the best linear congruential random number generator proposed by D. H. Lehmer in 1951 and illustrated in [PaMi88] [Weis00]. As shown in Figure 5.2, function myrand will return a random number between the two parameters min and max.

```
MYRAND(min, max)
1  if  first time call myrand
2    then SRANDOM(time(NIL))// set a random seed for random function
3         RAN_START(random())
4         // call Knuth's function to start [Knut02]
5  randomNumber ← RAN_ARR_NEXT() mod (max − min + 1)
6  // call Knuth's function to get next random number [Knut02]
7  return randomNumber + min
```

Figure 5.2: Pseudo-code for random number generator.

Routine ran_start and ran_arr_next are from Knuth's algorithm [Knut02]. setseed sets the seed for the random number generators using the ran_start function. The continuous sequence of random numbers is obtained from ran_arr_next. The seed for

ran_start is a randomly generated integers which is got from the current machine time. random() and srandom are BSD random number functions [GNU01].

5.1.2 Random data set

To generate the set of n uniform randomly distributed hyper-rectangles, we use the algorithm shown in Figure 5.3.

```
GETRANDDATA(space : SearchSpace)
 1  for i ← 0 to n − 1
 2  do // we will generate n random data
 3      for j ← 0 to k − 1
 4      do // on every dimension
 5          center ← MYRAND(space.GetMinSpace(j), space.GetMaxSpace(j))
 6          width ← MYRAND(space.GetMinSpace(j), space.GetMaxSpace(j))
 7          if center + width/2 > space.GetMaxSpace(j)
 8            then max ← space.GetMaxSpace(j)
 9            else  max ← center + width/2
10          if center − width/2 < space.GetMinSpace(j)
11            then min ← space.GetMinSpace(j)
12            else  min ← center − width/2
13           HyperRectangle hR ← new HYPERRECTANGLE(k)
14          hR→SetMin(min,j)
15          // set the min value to the hyper-rectangle on k^th dimension
16          hR→SetMax(max,j)
17          // set the max value to the hyper-rectangle on k^th dimension
18      // end of one hR
19      TRIE_INSERT(hR, T)// insert hR into trie T
```

Figure 5.3: Pseudo-code for generating n k-d uniformly distributed random hyper-rectangles into a $2k$-d trie T.

5.1.3 Random query hyper-rectangle generator

The query hyper-rectangle W is also generated randomly. QhRatio is an array which stores the fraction of the search space occupied by W on one dimension.

QhRatio has 40 elements starting from 0.01, and proceeding to 0.99 with a step size of 0.025. Each time getQhR (see Figure 5.4) is called, the fraction will increase by one step. The k-d trie is built for one set of n random hyper-rectangles, and the range search is performed 30 times for each fractional query window size. In Figure 5.1. $NUM\_STEPS = 40$ and $NQUERY = 100$. The number of hyper-rectangles in range increases as the volume of the query hyper-rectangle increases.

```
GETQHR(stepNum : integer)
 1  HyperRectangle hR ← new HYPERRECTANGLE(k)
 2  for i ← 0 to k − 1
 3  do width ← space.GetMaxSpace(i) − space.GetMinSpace(i)
 4     qwidth ← width × QhRatio[stepNum][i])
 5     hR→SetMin(MYRAND(0, width − qwidth) + space.GetMinSpace(i), i)
 6     hR→SetMax(hR→GetMin(i) + qwidth, i)
 7  return hR
```

Figure 5.4: Pseudo-code for generating a random query hyper-rectangle.

5.2 Experimental Results

Our experiments ran on herzberg.physics.mun.ca, a sgi Onyx 3400 with 28 MIPS R12000 processors and 14 Gigs of main memory, provided by the Memorial University of Newfoundland Computation and Visualization Centre, part of the C3.ca Association. Times were obtained by using the clock() function which returns the amount of CPU time, accurate to 10^{-6} of a second, used since the first call to clock() in the calling process. The time reported is the sum of the user and system times of the calling process and its terminated child process [Sun96]. Tables 5.1 to 5.9 are the average range search time for our trie (B=32). The item NA inside a table means there is no experimental results falling in this range. This is due to the fact that our hyper-rectangles are generated randomly. As shown in the maintest procedure (see Figure 5.1), for each fractional query hyper-rectangle size, we perform *NQUERY* range queries. We can decrease the number of NA entries in these tables by increasing *NQUERY*.

Table 5.1: The average range search time (ms) for a 4-d ($k = 2$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
2	[0%, 10%]	0.04	0.18	0.92	9	96.29
	[10%, 20%]	0.11	0.88	4.05	37.79	388.11
	[20%, 30%]	0.16	0.78	6.41	57.97	593.76
	[30%, 40%]	0.25	1.16	8.5	89.71	858.07
	[40%, 50%]	0.22	1.45	11.86	111.84	1107.42
	[50%, 60%]	0.35	1.46	13.01	144.27	1394.12
	[60%, 70%]	0.32	1.92	14.9	167.16	1595.97
	[70%, 80%]	0.24	2.27	16.98	191.27	1900.76
	[80%, 90%]	0.42	2.31	20.06	219.76	2150.95
	[90%, 100%]	NA	2.6	21.81	244.89	2408.28

Table 5.2: The average range search time (ms) for a 6-d ($k = 3$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
3	$[0\%, 10\%]$	0.06	0.36	1.95	15.51	187.19
	$[10\%, 20\%]$	0.17	0.83	6.8	57.28	503.72
	$[20\%, 30\%]$	0.28	1.02	10.84	97.27	900.91
	$[30\%, 40\%]$	0.23	1.81	13.82	138.55	1281.69
	$[40\%, 50\%]$	0.27	2.03	16.5	168.45	1627.14
	$[50\%, 60\%]$	0.37	2.43	21.38	204.66	1992.71
	$[60\%, 70\%]$	0.39	2.89	23.56	242.36	2278.64
	$[70\%, 80\%]$	0.43	3.4	27.2	276.79	2665.85
	$[80\%, 90\%]$	NA	NA	30.38	311.81	3036.11
	$[90\%, 100\%]$	NA	NA	33.1	347.29	3413.75

Table 5.3: The average range search time (ms) for a 8-d ($k = 4$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
4	$[0\%, 10\%]$	0.13	0.67	3.41	25.76	222.71
	$[10\%, 20\%]$	0.25	1.65	9.44	83.65	766.75
	$[20\%, 30\%]$	0.28	2.51	15.44	138.14	1243.93
	$[30\%, 40\%]$	0.39	2.75	19.58	186.39	1723.38
	$[40\%, 50\%]$	0.41	3.39	24	240.23	2189.05
	$[50\%, 60\%]$	NA	4.03	28.27	290.82	2655.54
	$[60\%, 70\%]$	NA	4.43	33.73	329.67	3084.36
	$[70\%, 80\%]$	NA	NA	39.59	382.62	3577.59
	$[80\%, 90\%]$	NA	NA	42.74	439.16	4093.18
	$[90\%, 100\%]$	NA	NA	NA	NA	4378.26

Table 5.4: The average range search time (ms) for a 10-d ($k = 5$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
5	$[0\%, 10\%]$	0.19	1.36	8.09	41.79	419.43
	$[10\%, 20\%]$	0.44	2.59	17.53	124.9	1056.89
	$[20\%, 30\%]$	0.43	3.34	23.96	202.42	1672.12
	$[30\%, 40\%]$	0.5	4.19	32.36	257.47	2266.51
	$[40\%, 50\%]$	0.63	4.8	40.6	324.56	2936.88
	$[50\%, 60\%]$	0.56	NA	48.96	377.27	3427.31
	$[60\%, 70\%]$	NA	NA	NA	468.28	4102.77
	$[70\%, 80\%]$	NA	NA	NA	537.53	4858.89
	$[80\%, 90\%]$	NA	NA	NA	NA	5288.15
	$[90\%, 100\%]$	NA	NA	NA	NA	NA

Table 5.5: The average range search time (ms) for a 12-d ($k = 6$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
6	$[0\%, 10\%]$	0.29	2.22	15.37	93.01	668.42
	$[10\%, 20\%]$	0.34	3.62	28.21	189.54	1396.95
	$[20\%, 30\%]$	0.49	4.67	38.25	281.13	2265.26
	$[30\%, 40\%]$	0.66	5.54	49.59	380.73	2934.63
	$[40\%, 50\%]$	0.59	NA	56.42	464.43	3826.77
	$[50\%, 60\%]$	0.74	NA	NA	587.44	4686.71
	$[60\%, 70\%]$	0.78	NA	NA	632.77	5542.19
	$[70\%, 80\%]$	NA	NA	NA	NA	6234.03
	$[80\%, 90\%]$	NA	NA	NA	NA	NA
	$[90\%, 100\%]$	NA	NA	NA	NA	NA

Table 5.6: The average range search time (ms) for a 14-d ($k = 7$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
7	$[0\%, 10\%]$	0.33	2.33	23.51	172.9	NA
	$[10\%, 20\%]$	0.7	4.11	38.94	324.83	2102.5
	$[20\%, 30\%]$	0.73	5.27	52.11	458.69	2938.08
	$[30\%, 40\%]$	0.71	6.14	59.19	593.94	3647.47
	$[40\%, 50\%]$	NA	NA	NA	NA	4945.12
	$[50\%, 60\%]$	NA	NA	NA	NA	6466.5
	$[60\%, 70\%]$	NA	NA	NA	NA	6722.86
	$[70\%, 80\%]$	NA	NA	NA	NA	NA
	$[80\%, 90\%]$	NA	NA	NA	NA	NA
	$[90\%, 100\%]$	NA	NA	NA	NA	NA

Table 5.7: The average range search time (ms) for a 16-d ($k = 8$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
8	$[0\%, 10\%]$	0.48	3.58	31.15	288.25	1896.74
	$[10\%, 20\%]$	0.64	5.57	53.19	509.03	2834.83
	$[20\%, 30\%]$	NA	6.64	63.17	640.88	4722.63
	$[30\%, 40\%]$	NA	NA	NA	NA	6535.78
	$[40\%, 50\%]$	NA	NA	NA	NA	NA
	$[50\%, 60\%]$	NA	NA	NA	NA	NA
	$[60\%, 70\%]$	NA	NA	NA	NA	NA
	$[70\%, 80\%]$	NA	NA	NA	NA	NA
	$[80\%, 90\%]$	NA	NA	NA	NA	NA
	$[90\%, 100\%]$	NA	NA	NA	NA	NA

Table 5.8: The average range search time (ms) for a 18-d ($k = 9$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
9	$[0\%, 10\%]$	0.47	4.3	39.4	373.91	3263.33
	$[10\%, 20\%]$	0.73	6.7	64.59	644.84	5820.74
	$[20\%, 30\%]$	NA	7.57	70.88	727.13	6954.41
	$[30\%, 40\%]$	NA	NA	NA	NA	NA
	$[40\%, 50\%]$	NA	NA	NA	NA	NA
	$[50\%, 60\%]$	NA	NA	NA	NA	NA
	$[60\%, 70\%]$	NA	NA	NA	NA	NA
	$[70\%, 80\%]$	NA	NA	NA	NA	NA
	$[80\%, 90\%]$	NA	NA	NA	NA	NA
	$[90\%, 100\%]$	NA	NA	NA	NA	NA

Table 5.9: The average range search time (ms) for a 20-d ($k = 10$) trie on herzberg.

Dimension	*% of data*	*Average range search time (milliseconds)*				
k	*in range*	n = 10	100	1000	10000	100000
10	$[0\%, 10\%]$	0.76	6.07	50.26	486.42	4570.18
	$[10\%, 20\%]$	NA	NA	72.91	737.29	7135.06
	$[20\%, 30\%]$	NA	NA	NA	NA	NA
	$[30\%, 40\%]$	NA	NA	NA	NA	NA
	$[40\%, 50\%]$	NA	NA	NA	NA	NA
	$[50\%, 60\%]$	NA	NA	NA	NA	NA
	$[60\%, 70\%]$	NA	NA	NA	NA	NA
	$[70\%, 80\%]$	NA	NA	NA	NA	NA
	$[80\%, 90\%]$	NA	NA	NA	NA	NA
	$[90\%, 100\%]$	NA	NA	NA	NA	NA

From Tables 5.1 to 5.9, we can see that the range search time increases as the dimension and query hyper-rectangle size increase. As the time used for range search is proportional to the number of nodes traversed, we use the time in the above tables for our analysis. From equation (4.4) in section 4.2.2,

$$E[Q(n,k)] = c_1 \prod_{p=1}^{2k} |WC^p| n + c_2 \Sigma_{S \subset \{1,\cdots,2k\}, 0<|S|<2k} (\prod_{p \notin S} |WC^p|) \gamma(\tfrac{1}{2k} \log_2 n) n^{1-\frac{s}{2k}} + c_3 \log_2 n + c_4 \quad (5.1)$$

There are three terms dependent on n in the equation. We rewrite equation (5.1) as

$$E[Q(n,k)] = c_1 d_1 + c_2 d_2 + c_3 d_3 + c_4 \quad (5.2)$$

Variables d_1, d_2, d_3 are called Term1, Term2, Term3 in Tables 4.3 and 4.4.

To determine an empirical formula for expected time $E[Q(n,k)]$, we formulate a least squares estimation to solve for $\underline{x} = (c_1, c_2, c_3, c_4)^T$. Our formulation is

$$\underline{v} = A\underline{x} - \underline{\ell} \quad (5.3)$$

where $A = m \times 4$ is the coefficient matrix, and $m =$ the number of observed experimental values (237 in this case, i.e. all observed range search times in Tables 5.1 to 5.9). The vector $\underline{\ell}$ $(m \times 1)$ contains the m observed values in milliseconds.

The least squares principle [MiAc76] obtains a solution for $\underline{x}$ such that:

$$\phi = \underline{v}^T \underline{v} \rightarrow minimum \quad (5.4)$$

where $\underline{v}$ are called the residuals. We used the method of least squares to obtain $\underline{x} = (c_1, c_2, c_3, c_4)^T$. Maple 6 [Mapl00] was used to find the following solution:

$$E[Q(n,k)] = .04540639823 d_1 + .00009165109674 d_2 + 4.079024506 d_3 - 35.97801026. \quad (5.5)$$

The process used to determine equation (5.5) is documented in Appendix C. Counting

$m = 237$ values, we observed that the root-mean-square (RMS) error (the square root of the mean squared deviation [HoNi85]) $\sqrt{\frac{1}{m}\underline{v}^T\underline{v}} = 452.67$ where $\underline{v}$ is the vector of residuals. The average absolute deviation $\frac{1}{m}\sum_{i=1}^{m}|v_i| = 201.64$. The average signed deviation $\frac{1}{m}\sum_{i=1}^{m} v_i = 0.46 \times 10^{-7}$. The linear correlation between $A\underline{x}$ and $\underline{\ell}$ is 0.946, close to 1, which shows that the fitted times have high positive linear correlation with the observed experimental times.

Figure 5.5 shows the experimental time ratio $Q(n_1, k)/Q(n_2, k)$ for $n_1 = 100000$, $n_2 = 10000$. Figures 5.6 and 5.7 show the experimental time $Q(n, k)$ and expected time $E[Q(n, k)]$ (using equation (5.5)) for percent of data in range $= [0\%, 10\%]$. Appendix E shows plots of experimental time (data from tables 5.1 to 5.9) compared to plots of expected time obtained from equation (5.5). Figure 5.8 shows a logarithmic plot for the relationship of all 237 experimental results with all 237 theoretical expected results (see Appendix C). The straight line represents a perfect match. We observe that the experimental results fit the theoretical analysis quite well.

The noise comes mainly from three sources. First, we use the same $|WC^p|$ for $2k$ dimensions and the determined $\prod_{p=1}^{2k}|WC^p|$ to compute d_1, d_2 and d_3 for simplicity, but the experimental query hyper-rectangle is generated randomly. For example, to compute (d_1, d_2, d_3) in equation (5.2), a data space occupancy proportion of 0.05 was used for $[0\%, 10\%]$ of data in range, 0.15 was used for $[10\%, 20\%]$ of data in range, $\cdots$, 0.95 data space occupancy was used for $[90\%, 100\%]$ of data in range. Secondly, when n is small (e.g. 10), the experimental data generated has high variance. Thirdly, the system function clock() used to calculate the time used for range search is not perfect;

when more processes are running on the machine, clock() will count more time due to higher context switching among active processes.

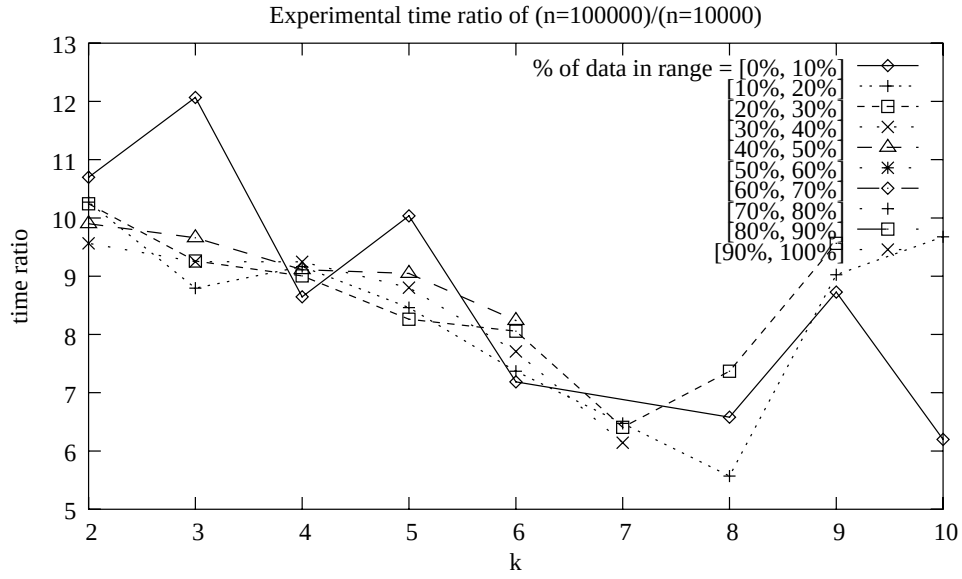


Figure 5.5: Experimental time ratio for Time(n=100000)/Time(n=10000).

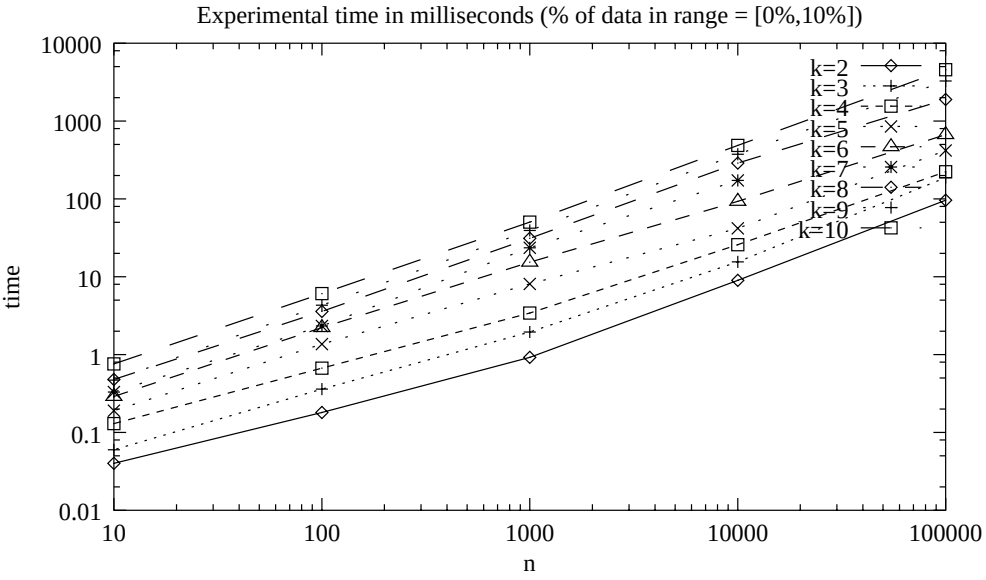


Figure 5.6: Experimental time for percent of data in range = $[0\%, 10\%]$.

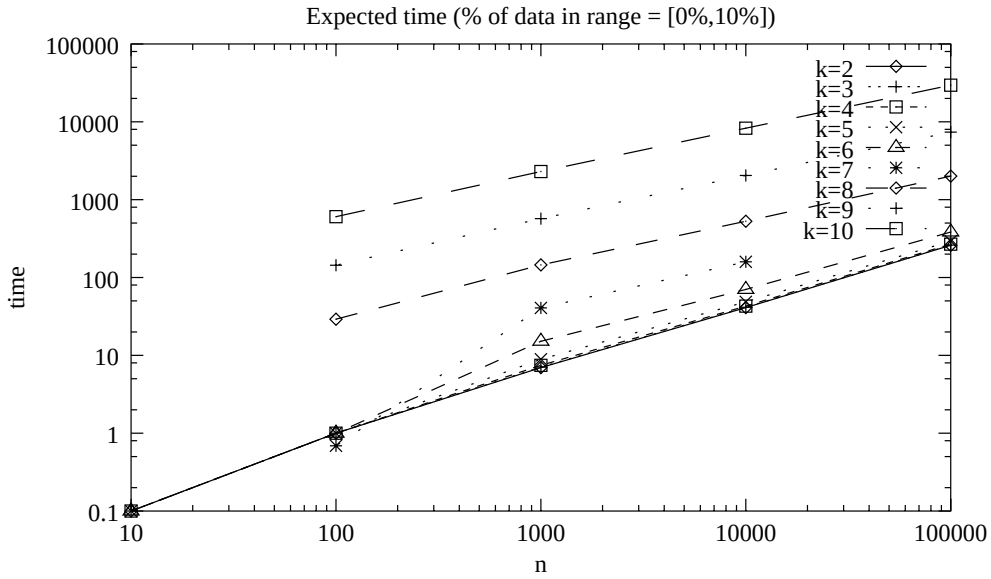


Figure 5.7: Expected time for percent of data in range = $[0\%, 10\%]$.

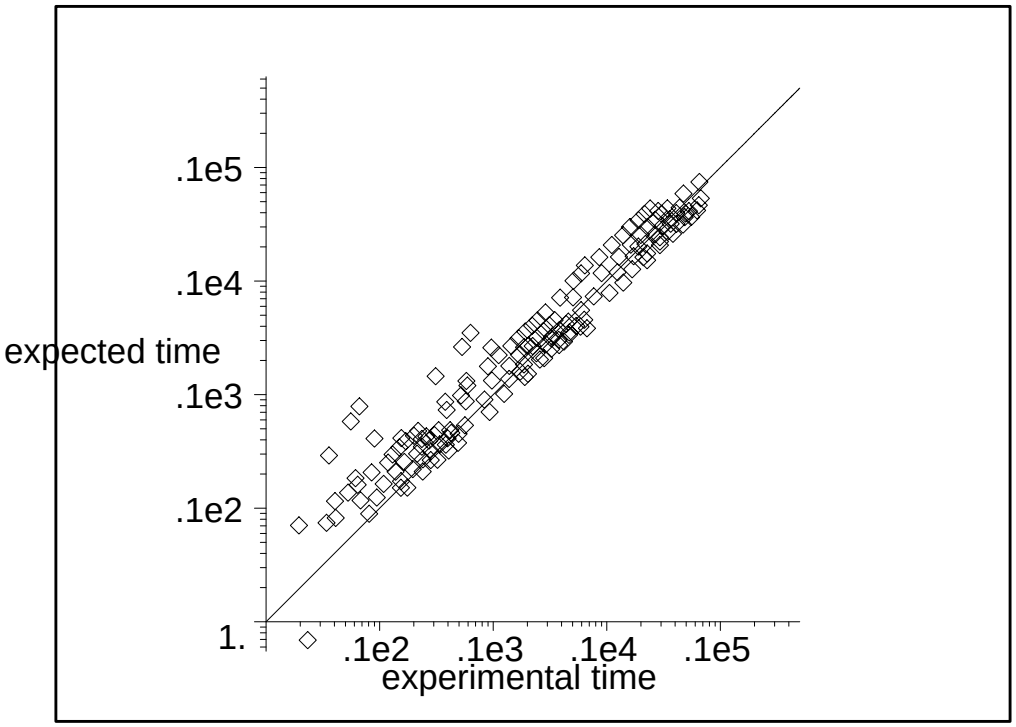


Figure 5.8: Logarithmic plot for the relationship of experimental time to expected time (from equation 5.5).

Chapter 6

Conclusions and Future Research

Using tries, we have shown that preprocessing time for storing n k-d hyper-rectangles is $\Theta(kBn)$. The space requirement is $S(n,k) = \Theta(kBn)$. The expected orthogonal range search time $E[Q(n,k)] = c_1 \prod_{p=1}^{2k} |WC^p| n +$

$c_2 \Sigma_{S \subset \{1,\cdots,2k\}, 0<|S|<2k} (\prod_{p \notin s} |WC^p|) \gamma(\frac{1}{2k} \log_2 n) n^{1-\frac{s}{2k}} + O(\log_2 n)$, where c_1 and c_2 are constants. The first term accounts the report time, the third term arises from the height of the trie, and the other terms represent contributions from lower-dimensional searches. For $k = 2$ and $k = 3$, we have determined that $E\{Q(n,k)\}$ behaves as $O(A + n^{\alpha})$ for $\alpha = f(n,k,r)$ with r = proportion of data space occupied by the query hyper-rectangle W. Section 4.2.2 gives special case analyses. Our algorithm for k-d range search using $2k$-d tries takes expected time proportional to k, n, and the relative size of the query hyper-rectangle W. Our analysis indicates this approach is competitive with other k-d hyper-rectangle range search algorithms, particularly for large k (i.e. for k approaching $\log_2 n$ or greater). The algorithm presented supports

dynamic operations, and is straightforward to implement.

Improvements in expected storage cost $S(n,k)$ and expected time for orthogonal range search $Q(n,k)$ can be obtained if we compress our binary trie to avoid storing nodes with only one child as is done for Patricia tries. The improvement can be planned in the following ways:

1. Using pointerless representation, we can avoid using a Parent pointer in each trie node to save space.

2. If we adapt a pointerless Patricia trie to implement our orthogonal range search algorithm, the search speed can be improved by a constant factor F which equals to the number of bits of the machine. For example, if we run our program on a 32-bit machine, the maximum improvement can be 32 times. Figure 6.1 gives a best case for illustration.

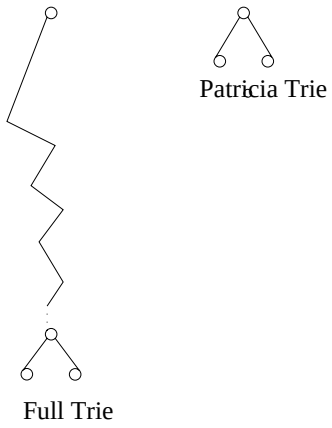


Figure 6.1: Example of full binary trie with Patricia trie.

Assuming that the full trie's height is $32 \times 2 \times 10 = 640$ (for k = 10), our range search algorithm searches to the leaf node, so we count 640 nodes per

hyper-rectangle in range (in the worst case). In the Patricia trie, we store the bit string in the internal node, and we need only do one step for comparison for each of $2k$ dimensions, so the expected cost improvement factor F $= 2k$ in the best case.

3. Instead of storing all the unshared bits, these bits are replaced by pointers only in Patricia trie's leaves pointing to hyper-rectangles. This method will improve our search speed because we do not need to traverse all the black nodes in the subtree to get the results; we need only count grey nodes as our query cost.

Further investigation of the expected cost for orthogonal range search $Q(n, k)$ is warranted to determine the relationship with partial match query expected cost. Can the upper bound we determined for $Q(n, k)$ be reduced by determining $Pr[node_t \in GN]$ instead of $Pr[node_t \in GN \cup BN]$? Under different assumptions about the nature of query window W distribution (e.g. random Gaussian in size), does the expected range search cost improve? Can the $2k$-d trie be adapted for other geometric problems such as half-space range search?

References

[Alle83] J.F. Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, 26(11):832-843, 1983.

[BaGo96] R.A. Baeza-Yates, G.H. Gonnet, "Fast Text Searching for Regular Expressions or Automaton Searching on Tries", *Jounal of the ACM*, 43(6), pp.915-936, 1996.

[BaNa98] R. Baeza-Yates and G. Navarro, "Fast Approximate String Matching in a Dictionary", Proc. 5th South American Symposium on String Processing and Information Retrieval (SPIRE'98), pp.14-22, IEEE CS Press, 1998.

[Bent75] J.L. Bentley, "Multidimensional binary search trees used for associative searching", *Communications of the ACM vol. 18*, no. 9, pp.509-517, September 1975.

[BeFr79] J.L. Bentley and J.H. Friedman, "Data Structures for Range Searching", *Computing Surveys*, vol. 11, no. 4, pp.397-409, 1979.

[BeMa80] J.L. Bentley and H.A. Maurer, "Efficient Worst-Case Data Structures for Range Searching",*Acta Informatica*, 13, pp.155-168, 1980.

[Bria59] R.de la Briandais, "File searching using variable length keys", *Proc. Western Joint Computer Conference*, vol.15, San Francisco, pp.295-298, 1959.

[Bu00] L. Bu, "Project Report: Zoom Tries", CS6345 course project report, December 2000.

[CDZ01] P. Chanzy, L. Devroye and C. Zamora-Cura,"Analysis of range search for random k-d trees", *Acta Informatica*, Vol.37, Fasc. 4/5, pp.355-383, 2001.

[Chaz88] B. Chazelle, "A Functional Approach for Data Structure and its use in multidimensional searching", *SIAM Journal of Computing*, vol. 17, no. 3, pp.427-462, June 1988.

[Chaz90-a] B. Chazelle, "Lower bounds for orthogonal range searching: I. The reporting case", *Journal of the ACM*, vol. 37, no. 2, pp.200-212, April 1990.

[Chaz90-b] B. Chazelle, "Lower bounds for orthogonal range search: II. The Arithmatic Model", *Journal of the ACM*, vol. 37, no. 3, pp. 439-463, July 1990.

[CoSe77] D. Comer, and R. Sethi, "The complexity of trie index construction", *Journal of the ACM*, vol.24(3), pp.428-440, 1977.

[DJZ00] L. Devroye, J. Jabbour and C. Zamora-Cura, "Squarish k-d trees", *SIAM Journal of Computing*, vol.30, no.5, pp.678-700, 2000.

[Edel83] H. Edelsbrunner, "A new approach to rectangle intersection Part I", *Int. J. Computer Mathematics*, vol. 13, pp.209-219, 1983.

[Egen91] M. Egenhofer, "Point-set topological spatial relations", *INT. J. Geographical Information Systems*, vol.5, no.2, pp.161-174, 1991.

[Egen94] M. Egenhofer, "Spatial SQL: A query and presentation language", *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 1, pp. 86-95, 1994.

[FlOd82] P. Flajolet and A. M. Odlyzko, "The average height of binary trees and other simple trees", *Journal of Computer System Science*, vol.25, pp. 171-213, 1982.

[FlPu86] P. Flajolet and C. Puech, "Partial Match Retrieval on Multidimensional Data", *Journal of the Association or Computing Machinery*, vol. 33, no. 2, pp.371-407, April 1986.

[Fred60] E. Fredkin, "Trie memory", *Communiations of the ACM*, vol.3, pp.490-500, 1960.

[Fred81-a] M.L. Fredman, "Lower Bounds On The Complexity Of Some Optimal Data Structures", *SIAM J. Comput.* Vol.10, No.1, February 1981.

[Fred81-b] M.L. Fredman, "A Lower Bound on the Complexity of Orthogonal Range Queries", *Jounal of the Association for Computing Machinery*, Vol.28, No.4, pp.696-705, October 1981.

[GBS91] G.H. Gonnet, R.A. Baeza-Yates, T. Snider, "Lexicographical indices for text: inverted files vs. Pat trees", Technical Report OED-91-01, University of Waterloo Centre for the New OED and Text Research, University of Waterloo.

[GNU01] http://www.gnu.org/manual/glibc-2.2.3/html_node/libc_387.html #SEC396, May, 2001.

[Gonn88] G.H. Gonnet, "Efficient Searching of Text and Pictures", Technical Report OED-88-02, UW Centre for the New Oxford English Dictionary, University of Waterloo, June, 1988.

[Gonn91] G.H. Gonnet, Handbook of Algorithms and Data Structures, Addison Wesley, Reading, MA, 1991.

[Gutt84] A. Guttman, "R-trees: a dynamic index structure for spatial searching", Proc. of the SIGMOD Conference, Boston, pp.47-57, June 1984.

[HoNi85] L. Hoehn and I. Niven, "Averages on the move", *The Mathematics Magazine*, vol. 58, pp. 151-156, 1985.

[Knut73-1] D.E. Knuth, *The art of computer programming*, Volume 2, *Seminumerical Algorithms*, 2^{nd} edition, Addison-Wesley, Boston, Section 3.6, pp.170-173, 1973.

[Knut73-2] D.E. Knuth, *The art of computer programming*, Volume 3, *Sorting and Searching*, 2^{nd} edition, Addison-Wesley, Boston, pp.564-566, 1973.

[Knut02] D.E. Knuth, http://www-cs-faculty.stanford.edu/~knuth/programs/ rng.c, Sep. 2002.

[LeWo77] D.T. Lee and C.K. Wong, "Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees", *Acta Informatica*, 9, 23-29, 1977.

[Luek78] G.S. Lueker, "A Data Structure For Orthogonal Range Queries", 19^{th} Symposium on Foundations of Computer Science, pp.28-34, IEEE Computer Society Press, Oct. 1978.

[LuWi82] G.S. Lueker and D.E. Willard, "A Data Structure For Dynamic Range Queries", *Information Processing Letters*, Volume 15, Number 5, pp.209-213, 1982.

[Maly76] K. Maly, "Compressed tries", *Communications of the ACM*, vol.19(7), pp.409-415, 1976.

[MaMy90] U. Manber and G. Myers, "Suffix arrays: A new method for online string searches", *1st ACM-SIAM Symposium on Discrete Algorithms*, pp. 319-327, San Francisco, January, 1990.

[Mapl00] http://www.maplesoft.com, Maple 6, Copyright(c) 1981-2000 by Waterloo Maple Inc.

[MeSh93] T.H. Merrett, H. Shang, "Trie methods for representing text", *Foundations of Data Organization and Algorithms*, Chicago, Illinois, pp.130-145, Chicago Illinois, October 13-15, 1993.

[MeSh94] T.H. Merrett, H. Shang, "Zoom tries: a file structure to support spatial zooming", In T.C.Waugh and R.Healey, Eds., *SDH 94 Advances in GIS Research, Proceedings of Spatial Data Handling Symposium*, International Geographical Union, Edinburgh, pp.792-804, 1994.

[MiAc76] Edward M. Mikhail, F. Ackermann, *Observations and Least Squares*, IEP-A Dun-Donnelley Publisher, New York, 1976.

[MSZ96] T.H. Merrett, H. Shang, X. Zhao, "Database Structures, Based on Tries, for Text, Spatial, and General Data", International Symposium on Cooperative Database Systems for Advanced Applications, Kyoto, Japan, v. 2, pp.316-424, 1996.

[Morr68] D.R. Morrison, "Patricia - practical algorithm to retrieve information coded in alphanumeric", *Journal of the ACM*, vol.14(4), pp.514-534, October 1968.

[Oren82] J.A. Orenstein, "Multidimensional tries used for associative searching", *Information Processing Letters*, vol.14(14), pp.150-156, June 1982.

[PaMi88] S. K. Park, and K. W. Miller, "Random Number Generators: Good ones are hard to find", *Communications of the ACM*, vol.31, pp.1192-1201, October 1988.

[RBK89] R. Ramesh, A.J.G. Babu, and J.P. Kincaid, "Variable-depth trie index optimization: Theory and experimental results", *ACM Transactions on Database Systems*, vol.14(1), pp.41-74, March 1989.

[SaTo92] A. Salminen and F.W. Tompa, "PAT Expressions: An Algebra for Text Search", *Acta Linguistica Hungarica*, 41(1-4), pp.277-306, 1992-93.

[Same90] H. Samet, *The Design and Analysis of Spatial Data Strutures*, Addison-Wesley, Reading, Mass., 1990.

[Shan01] H. Shang, "Trie Methods for Text and Spatial Data on Secondary Storage", Ph.D. thesis, McGill University, ch6, pp.90-100, January 2001.

[Sun96] manual of SunOS5.7, Dec., 1996.

[Suss63] Jr.E.H. Sussenguth, "Use of tree structures for processing files", *Communications of the ACM*, vol.6(5), pp.272-279, 1963.

[Tomp94] F.W. Tompa, "Not Just Another Database Project: Development at UW", Technical Report, UW Centre for the New OED, University of Waterloo, ON, Canada, 1994.

[Ukko95] E. Ukkonen, "On-line Construction of Suffix Trees", *Algorithmica*, vol.14(3), pp249-260, 1995.

[Weis00] M.A. Weiss, *Data Structures and Problem Solving Using C++*, Chapter 10, *Randomization*, 2^{nd} edition, Addison-Wesley, Massachusetts, pp.365-386, 2000.

Appendix A

C++ code for Term1 and Term3 (equation (4.4)) generation

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
/*k in this code means 2k in my thesis's theorem */
void main()
{
      ofstream fout1("term1Output.txt");
      ofstream fout3("term3Output.txt");
      ofstream fout11("term11.txt");
      ofstream fout33("term33.txt");
      for(int k=2; k<=10; k++){
         for(double t=0.05; t<1; t+ =0.1){ //for setting WC
            for(int n=10; n<=100000; n* =410){
               cout << "k = " << k << "; t = " << t << "; n = "<< n << endl;
               fout1 << t*n << ", ";
               fout3 << log(n)/log(2) << ", ";
               fout11 << "k = " << k << "; t = " << t << "; n = " << n << "; t*n="
<< t*n << endl;
               fout33 <<"k = " << k << "; t = " << t << "; n = " << n << "
log(n)/log(2)=" << log(n)/log(2) << endl;
            }
         }
      }
      fout1.close();
      fout3.close();
      fout11.close();
      fout33.close();
}
```

Appendix B

C++ code for Term2 (equation (4.4)) generation

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
double Tjl(int k, int j, int l, int n)
{
    // for Lemma8 of Flajolet and Puech's paper
    int p = -k * j - l;
    double pow2p = pow(2.0, p);
    double pow12pn = pow(1.0 - pow2p, n - 1);
    double value = 1.0 - pow12pn * ((1.0 - pow2p) + n * pow2p);
    if (value < 0) return 0.0;
    return value;
}
double Product(char const * g, int start, int end)
{
    double p = 1.0;
    for (int i = start; i <= end; i ++) p * = g[i];
    return p;
}
double Lemma8Alter(int kmax, int nmax, int s, char const * g)
{
    // calculate Lemma 8 of Flajelot and Puech's paper
    double * dt = new double[kmax + 1];
    dt[0] = 1.0;
    for (int i = 1; i <= kmax - 1; i ++) dt[i] = Product(g, 1, i);
    double sum = 0.0;
    for (int l = 0; l <= kmax - 1; l ++)
    {
        double jsum = 0.0;
        for (int j = 0; j <= kmax-1; j ++)
            jsum + = pow(2.0, j * (kmax - s)) * Tjl(kmax, j, l, nmax);
        sum + = dt[l] * jsum;
    }
    delete[] dt;
    return sum;
}
int CountOnes(char const * g, int kmax)
{
    // determine how many specified characters in pattern u
    int n = 0;
    for (int i = 1; i <= kmax; i ++) if (g[i] == 1) n ++;
```

```
        return n;
    }
    double fact(int x)
    {
        double f = 1.0;
        for (int i = 2; i <= x; i ++) f * = i;
        return f;
    }
    void CallLemma8(double * average, int kmax, int nmax)
    {
      // this procedure gets the average patial match retrieval cost for same s
        // initialization
        char * g = new char[kmax + 1];
        int i;
        for (i = 1; i <= kmax; i ++){
            // unspecified character
            g[i] = 2;
            average[i] = 0.0;
        }
        // partial match pattern u all the possibilities except all empty or full
        int max = 1 << kmax;
        int skip = max / 100;
        if (skip < 1) skip = 1;
        int scale = 20 - kmax;
        if (scale > 1) skip * = scale;
        for (int current = 1; ; current ++){
            int num = CountOnes(g, kmax);
            if (num = 0 && num = kmax) average[num] + = Lemma8Alter(kmax, nmax, num,
g);
            for (i = 1; i <= kmax; i ++){
                if (g[i] = 1){
                    g[i] = 1;
                    break;
                }
                g[i] = 2;
            }
            if (num == kmax) break;
        }
        for (i = 1; i <= kmax - 1; i ++)
            average[i] * = fact(kmax - i) * fact(i) / fact(kmax);
        delete[] g;
    }
    double CompProd(char const * s, double const * WC, int kmax)
    {
        double result = 1.0;
        for (int i = 1; i <= kmax; i ++) if (s[i] == 1) result * = WC[i];
            return result;
    }
    double theorem(double const * WC, int kmax, int nmax)
    {
        char * s = new char[kmax + 1];
        double * average = new double[kmax + 1];
```

```
    CallLemma8(average, kmax, nmax);
    int i;
    for (i = 1; i <= kmax; i ++) s[i] = 0;
    double * finalPart = new double[kmax];
    for (i = 1; i <= kmax - 1; i ++) finalPart[i] = 0.0;
    double final = 0.0;
    int max = 1 << kmax;
    int skip = max / 100;
    if (skip < 1) skip = 1;
    int scale = 20 - kmax;
    if (scale > 1) skip * = scale;
    for (int current = 1; ; current ++){
        int numb = CountOnes(s, kmax);
        if (numb = 0 && numb = kmax){
            double part = average[kmax - numb] * CompProd(s, WC, kmax);
            finalPart[kmax - numb] + = part;
            final + = part;
    }
    for (i = 1; i <= kmax; i ++){
        if (s[i] == 0){
            s[i] = 1;
            break;
        }
        s[i] = 0;
        }
        if (numb == kmax) break;
    }
    delete[] finalPart;
    delete[] average;
    delete[] s;
    return final;
}
void test(int kmax, int nmax, double const * WC)
{
    ofstream fout("term2Sep16.txt", ios::app);
    if(!fout){
        cerr << "Couldn't open file";
        abort();
    }
    fout.precision(10);
    fout << "test(" << kmax << "," << nmax << "," << WC[1] << "):";
    fout << theorem(WC, kmax, nmax) << endl << endl;
    fout.close();
}
/*k in this code means 2k in my thesis's theorem */
void main()
{
    ofstream fout("term2Sep16.txt", ios::app);
    if(!fout){
        cerr << "Couldn't open file";
        abort();
    }
```

```
    for(int k=2; k<=10; k++){
        for(double t=0.05; t<=1; t+ =0.1){ //for setting WC
            double * WC = new double[2*k + 1];
            for(int wci = 1; wci <= 2*k; wci++) WC[wci] = pow(t, 1/double(2*k));
            for(int n=10; n<=100000; n∗ =10){
                cout << "k = " << k << "; t = " << t << "; n = " << n << endl;
                fout << theorem(WC, 2*k, n) << ", ";
            }
            delete [] WC;
        }
    }
    fout.close();
}
```

Appendix C Maple Worksheet

This appendix contains the maple worksheet to generate equation (5.2).

```
> termt := [0.04, 0.18, 0.92, 9, 96.29, 0.11, 0.88, 4.05, 37.79,
> 388.11, 0.16, 0.78, 6.41, 57.97, 593.76, 0.25, 1.16, 8.5, 89.71,
> 858.07, 0.22, 1.45, 11.86, 111.84, 1107.42, 0.35, 1.46, 13.01, 144.27,
> 1394.12, 0.32, 1.92, 14.9, 167.16, 1595.97, 0.24, 2.27, 16.98, 191.27,
> 1900.76, 0.42, 2.31, 20.06, 219.76, 2150.95, 2.6, 21.81, 244.89,
> 2408.28, 0.06, 0.36, 1.95, 15.51, 187.19, 0.17, 0.83, 6.8, 57.28,
> 503.72, 0.28, 1.02, 10.84, 97.27, 900.91, 0.23, 1.81, 13.82, 138.55,
> 1281.69, 0.27, 2.03, 16.5, 168.45, 1627.14, 0.37, 2.43, 21.38, 204.66,
> 1992.71, 0.39, 2.89, 23.56, 242.36, 2278.64, 0.43, 3.4, 27.2, 276.79,
> 2665.85, 30.38, 311.81, 3036.11, 33.1, 347.29, 3413.75, 0.13, 0.67,
> 3.41, 25.76, 222.71, 0.25, 1.65, 9.44, 83.65, 766.75, 0.28, 2.51,
> 15.44, 138.14, 1243.93, 0.39, 2.75, 19.58, 186.39, 1723.38, 0.41,
> 3.39, 24, 240.23, 2189.05, 4.03, 28.27, 290.82, 2655.54, 4.43, 33.73,
> 329.67, 3084.36, 39.59, 382.62, 3577.59, 42.74, 439.16, 4093.18,
> 4378.26, 0.19, 1.36, 8.09, 41.79, 419.43, 0.44, 2.59, 17.53, 124.9,
> 1056.89, 0.43, 3.34, 23.96, 202.42, 1672.12, 0.5, 4.19, 32.36, 257.47,
> 2266.51, 0.63, 4.8, 40.6, 324.56, 2936.88, 0.56, 48.96, 377.27,
> 3427.31, 468.28, 4102.77, 537.53, 4858.89, 5288.15, 0.29, 2.22, 15.37,
> 93.01, 668.42, 0.34, 3.62, 28.21, 189.54, 1396.95, 0.49, 4.67, 38.25,
> 281.13, 2265.26, 0.66, 5.54, 49.59, 380.73, 2934.63, 0.59, 56.42,
> 464.43, 3826.77, 0.74, 587.44, 4686.71, 0.78, 632.77, 5542.19,
> 6234.03, 0.33, 2.33, 23.51, 172.9, 0.7, 4.11, 38.94, 324.83, 2102.5,
> 0.73, 5.27, 52.11, 458.69, 2938.08, 0.71, 6.14, 59.19, 593.94,
> 3647.47, 4945.12, 6466.5, 6722.86, 0.48, 3.58, 31.15, 288.25, 1896.74,
> 0.64, 5.57, 53.19, 509.03, 2834.83, 6.64, 63.17, 640.88, 4722.63,
> 6535.78]:
> termn := [10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000,
> 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100,
> 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000,
> 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000,
> 100000, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 10,
> 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000,
> 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000,
> 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000,
> 1000, 10000, 100000, 1000, 10000, 100000, 10, 100, 1000, 10000,
> 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000,
> 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 100, 1000,
> 10000, 100000, 100, 1000, 10000, 100000, 1000, 10000, 100000, 1000,
> 10000, 100000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000,
> 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000,
> 100000, 10, 100, 1000, 10000, 100000, 10, 1000, 10000, 100000, 10000,
> 100000, 10000, 100000, 100000, 10, 100, 1000, 10000, 100000, 10, 100,
> 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000,
> 10000, 100000, 10, 1000, 10000, 100000, 10, 10000, 100000, 10, 10000,
> 100000, 100000, 10, 100, 1000, 10000, 10, 100, 1000, 10000, 100000,
> 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000, 100000, 100000,
> 100000, 100000, 10, 100, 1000, 10000, 100000, 10, 100, 1000, 10000,
> 100000, 100, 1000, 10000, 100000, 100000]:
```

```
> termk := [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
> 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
> 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
> 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7,
> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8,
> 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]:
> term1 := [0.5, 5, 50, 500, 5000, 1.5, 15, 150, 1500, 15000, 2.5, 25,
> 250, 2500, 25000, 3.5, 35, 350, 3500, 35000, 4.5, 45, 450, 4500,
> 45000, 5.5, 55, 550, 5500, 55000, 6.5, 65, 650, 6500, 65000, 7.5, 75,
> 750, 7500, 75000, 8.5, 85, 850, 8500, 85000, 95, 950, 9500, 95000,
> 0.5, 5, 50, 500, 5000, 1.5, 15, 150, 1500, 15000, 2.5, 25, 250, 2500,
> 25000, 3.5, 35, 350, 3500, 35000, 4.5, 45, 450, 4500, 45000, 5.5, 55,
> 550, 5500, 55000, 6.5, 65, 650, 6500, 65000, 7.5, 75, 750, 7500,
> 75000, 850, 8500, 85000, 950, 9500, 95000, 0.5, 5, 50, 500, 5000, 1.5,
> 15, 150, 1500, 15000, 2.5, 25, 250, 2500, 25000, 3.5, 35, 350, 3500,
> 35000, 4.5, 45, 450, 4500, 45000, 55, 550, 5500, 55000, 65, 650, 6500,
> 65000, 750, 7500, 75000, 850, 8500, 85000, 95000, 0.5, 5, 50, 500,
> 5000, 1.5, 15, 150, 1500, 15000, 2.5, 25, 250, 2500, 25000, 3.5, 35,
> 350, 3500, 35000, 4.5, 45, 450, 4500, 45000, 5.5, 550, 5500, 55000,
> 6500, 65000, 7500, 75000, 85000, 0.5, 5, 50, 500, 5000, 1.5, 15, 150,
> 1500, 15000, 2.5, 25, 250, 2500, 25000, 3.5, 35, 350, 3500, 35000,
> 4.5, 450, 4500, 45000, 5.5, 5500, 55000, 6.5, 6500, 65000, 75000, 0.5,
> 5, 50, 500, 1.5, 15, 150, 1500, 15000, 2.5, 25, 250, 2500, 25000, 3.5,
> 35, 350, 3500, 35000, 45000, 55000, 65000, 0.5, 5, 50, 500, 5000, 1.5,
> 15, 150, 1500, 15000, 25, 250, 2500, 25000, 35000]:
```

```
> term2:=[20.1498, 78.8501, 293.594, 1174.86, 3414.7, 33.4734, 140.104,
> 562.43, 2396.27, 7188.71, 43.0069, 185.736, 770.426, 3367.57, 10228.9,
> 50.9869, 224.737, 951.597, 4225.01, 12929.3, 58.0528, 259.759,
> 1116.34, 5011.51, 15416.3, 64.4962, 292.034, 1269.56, 5747.7, 17751,
> 70.4794, 322.254, 1414.07, 6445.53, 19969.1, 76.1037, 350.859,
> 1551.68, 7112.7, 22093.5, 81.4378, 378.147, 1683.61, 7754.53, 24140.5,
> 404.333, 1810.77, 8374.92, 26121.7, 90.6764, 353.747, 1331.21,
> 5485.75, 26348.1, 150.773, 628.303, 2555.59, 11331.7, 57708, 193.76,
> 832.814, 3507.57, 16049.6, 83741.7, 229.726, 1007.67, 4339.99,
> 20254.5, 107279, 261.562, 1164.78, 5099.33, 24139.8, 129234, 290.585,
> 1309.64, 5807.57, 27798.3, 150054, 317.529, 1445.36, 6477.22, 31283.9,
> 170001, 342.855, 1573.9, 7116.28, 34631.1, 189244, 7730.23, 37864,
> 207903, 8323.05, 41000, 226062, 372.51, 1488, 5351.97, 22045.2,
> 98359.4, 623.418, 2632.83, 10124.4, 44617.1, 211417, 801.803, 3475.76,
> 13782.2, 62582.5, 304394, 950.386, 4191.34, 16955.1, 78479.4, 388082,
> 1081.44, 4830.9, 19833.4, 93096.9, 465931, 5418.18, 22506.4, 106812,
> 539613, 5966.5, 25025.1, 119842, 610101, 27421.7, 132325, 678026,
> 29718.5, 144359, 743825, 807818, 1498.25, 6146.01, 21629.7, 83719.3,
> 373125, 2522.22, 10838.6, 40298.8, 165560, 779595, 3246.15, 14256.2,
> 54373.9, 229513, 1.10749e+06, 3846.66, 17137.4, 66465.7, 285523,
> 1.39956e+06, 4374.57, 19699.1, 77358.6, 336653, 1.66929e+06, 4853.1,
> 87419.5, 384360, 1.92318e+06, 429476, 2.16498e+06, 472535,
> 2.39713e+06, 2.62129e+06, 5996.3, 25107.2, 89380.6, 319948,
> 1.36569e+06, 10140.2, 44161.8, 164619, 619818, 2.7819e+06, 13057.3,
> 57920.7, 220578, 850090, 3.903e+06, 15469.3, 69456.7, 268261,
> 1.04975e+06, 4.89107e+06, 17584.3, 310960, 1.23071e+06, 5.7968e+06,
> 19497.4, 1.39861e+06, 6.64442e+06, 21261.4, 1.55666e+06, 7.4479e+06,
> 8.21622e+06, 23974, 101920, 368656, 1.29052e+06, 40679.4, 178907,
> 673242, 2.45967e+06, 9.9327e+06, 52401.5, 234137, 897440, 3.34435e+06,
> 1.37868e+07, 62071, 280249, 1.08728e+06, 4.10485e+06, 1.71514e+07,
> 2.02147e+07, 2.30665e+07, 2.5758e+07, 95841, 412169, 1.51073e+06,
> 5.32251e+06, 1.91032e+07, 163044, 722363, 2.74107e+06, 1.00227e+07,
> 3.74034e+07, 943781, 3.63947e+06, 1.35391e+07, 5.14446e+07,
> 6.36006e+07]:
```

```
> term3 := [3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193,
> 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578,
> 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096,
> 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386,
> 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877,
> 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193,
> 6.64386, 9.96578, 13.2877, 16.6096, 6.64386, 9.96578, 13.2877,
> 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193,
> 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578,
> 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096,
> 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386,
> 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877,
> 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 9.96578,
> 13.2877, 16.6096, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386,
> 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877,
> 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193,
> 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578,
> 13.2877, 16.6096, 6.64386, 9.96578, 13.2877, 16.6096, 6.64386,
> 9.96578, 13.2877, 16.6096, 9.96578, 13.2877, 16.6096, 9.96578,
> 13.2877, 16.6096, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877,
> 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193,
> 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578,
> 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096,
> 3.32193, 9.96578, 13.2877, 16.6096, 13.2877, 16.6096, 13.2877,
> 16.6096, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096,
> 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386,
> 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578, 13.2877,
> 16.6096, 3.32193, 9.96578, 13.2877, 16.6096, 3.32193, 13.2877,
> 16.6096, 3.32193, 13.2877, 16.6096, 16.6096, 3.32193, 6.64386,
> 9.96578, 13.2877, 3.32193, 6.64386, 9.96578, 13.2877, 16.6096,
> 3.32193, 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386,
> 9.96578, 13.2877, 16.6096, 16.6096, 16.6096, 16.6096, 3.32193,
> 6.64386, 9.96578, 13.2877, 16.6096, 3.32193, 6.64386, 9.96578,
> 13.2877, 16.6096, 6.64386, 9.96578, 13.2877, 16.6096, 16.6096]:
> count := 237;
```

$$count := 237$$

```
> with(plots):
```

```
Warning, the name changecoords has been redefined
```

```
> with(stats):
> fitteq := fit [leastsquare[[t1, t2, t3, tt], tt = a*t1 + b*t2 + c*t3
> + d, {a, b, c, d}]]([term1, term2, term3, termt]);
```

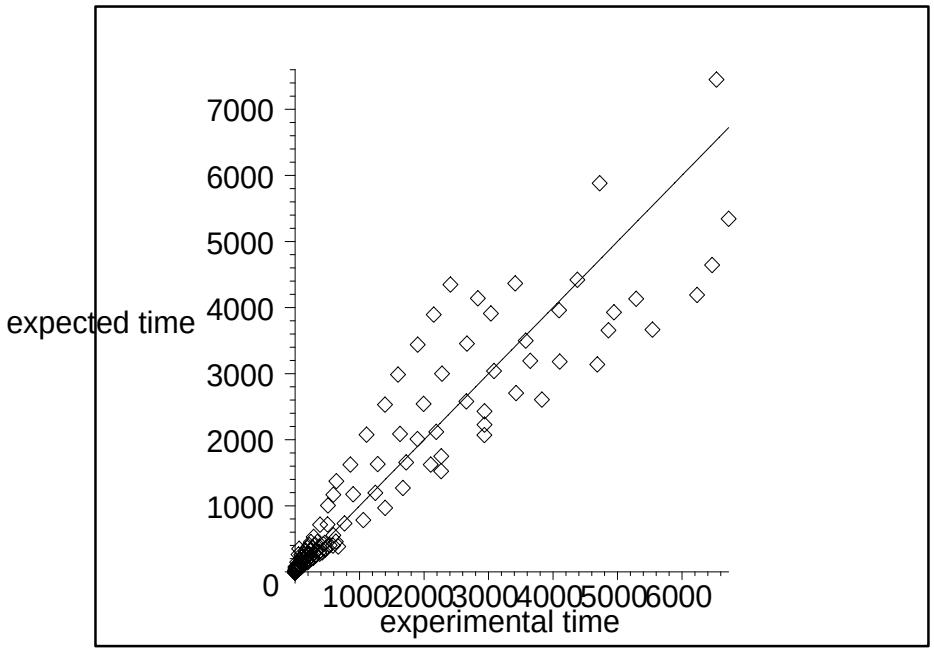
$$fitteq := tt = .04540639823\,t1 + .00009165109674\,t2 + 4.079024506\,t3 - 35.97801026$$

```
> fitt := unapply(rhs(fitteq),(t1, t2, t3));
```

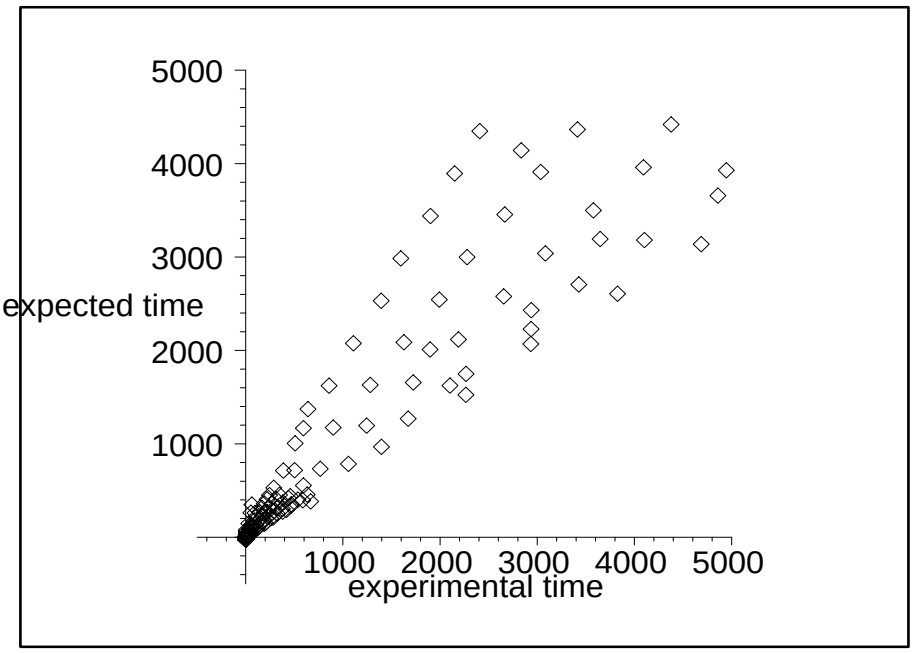
$$fitt := (t1,\ t2,\ t3) \rightarrow .04540639823\,t1 + .00009165109674\,t2 + 4.079024506\,t3 - 35.97801026$$

```
> termtfit := [seq(fitt(term1[i], term2[i], term3[i]),i =
> 1..count)]:
> timeplot := pointplot3d({seq([termn[i], termk[i], termt[i]],i =
> 1..count)}, orientation=[-60, 60], symbol=(BOX,10), axes=NORMAL,
> scaling=UNCONSTRAINED, labels=["n", "k", "experimental time"],
> color=BLACK): display(timeplot);
> fitplott := pointplot3d({seq([termn[i], termk[i], termtfit[i]],i =
> 1..count)}, orientation=[-60, 60], symbol=(CROSS,20), axes=NORMAL,
> scaling=UNCONSTRAINED, labels=["n", "k", "expected time"],
> color=BLACK): display(fitplott);
> display(timeplot, fitplott);
```

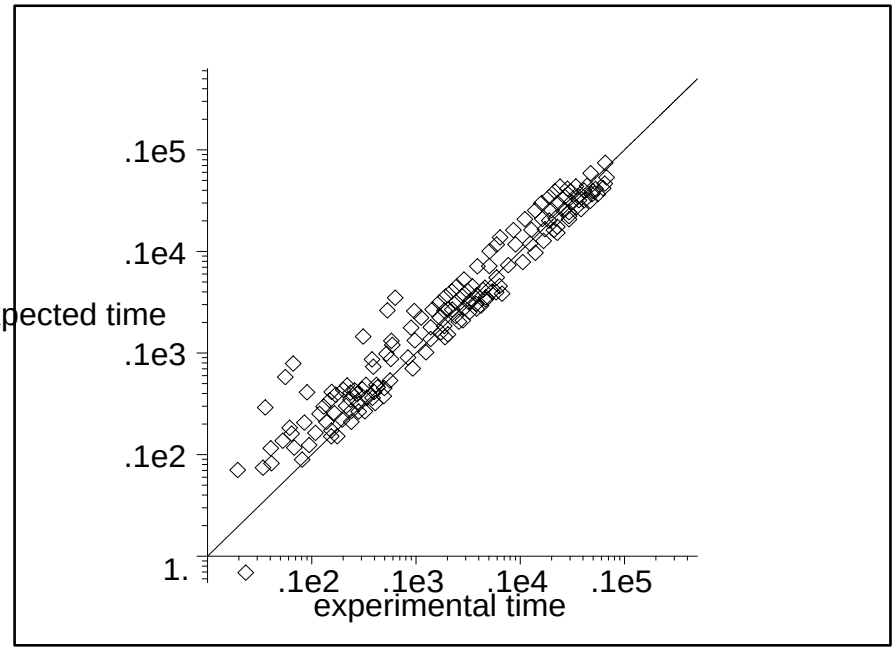
```
> compareplott := pointplot({seq([termt[i], fitt(term1[i], term2[i],
> term3[i])],i = 1..count)}, symbol=(DIAMOND, 20), axes=NORMAL,
> scaling=CONSTRAINED, labels=["experimental time", "expected time"],
> color=BLACK):
> lineplot := plot(x, x = 1..max(seq(termt[i], i = 1..count)),
> thickness = 2, axes=NORMAL, scaling=CONSTRAINED, labels=["experimental
> time", "expected time"], color=BLACK):
> display(compareplott, lineplot);
```




```
> pointplot({seq([termt[i], fitt(term1[i], term2[i], term3[i])],i =
> 1..count)}, symbol=(DIAMOND, 20), axes=NORMAL, scaling=CONSTRAINED,
> labels=["experimental time", "expected time"], color=BLACK,
> view=[-500..5000,-500..5000]);
```




```
> comparelog := loglogplot([seq([termt[i], fitt(term1[i], term2[i],
> term3[i])],i = 1..count)], symbol=(DIAMOND, 20), axes=NORMAL,
> scaling=CONSTRAINED, labels=["experimental time", "expected time"],
> color=BLACK, style=POINT):
> linelog := loglogplot(x, x = 1..50000, thickness = 2, axes=NORMAL,
> scaling=CONSTRAINED, labels=["experimental time", "expected time"],
> color=BLACK):
> display(comparelog, linelog);
```




```
> residuals := termt - termtfit:
```

```
> with(linalg):
> root_mean_square_error = (evalm(transpose(residuals) &*
> residuals)/count)^0.5;
```

$$root\_mean\_square\_error = 452.6666380$$

```
> with(stats):
> describe[linearcorrelation](termt, termtfit);
```

$$.9461393772$$

```
> average_absolute_deviation := sum(abs(termt[i] - termtfit[i]),
> i=1..count)/count;
```

$$average\_absolute\_deviation := 201.6421538$$

```
> average_signed_deviation := sum((termt[i] - termtfit[i])/count,
> i=1..count);
```

$$average\_signed\_deviation := .46\,10^{-7}$$

Appendix D
γ function (equation (2.3))

```
# include <iostream>
# include <vector>
# include <math.h>
using namespace std;

 double gammmaKS( int k,  int s)
{
     //values are got from maple
     if (k == 2)  return -3.544907701811032;
     if (k == 3)
    {
         double  const g[] =
        {
            -4.018407802061622, -4.062353818279200
        };
         return g[s - 1];
    }
     if (k == 4)
    {
         double  const g[] =
        {
            -4.834146544295875, -3.544907701811032, -4.901666809860712
        };
         return g[s - 1];
    }
     if (k == 5)
    {
         double  const g[] =
        {
            -5.738554639998505, -3.696932572929480, -3.722980622032043, -5.821148568626515
        };
         return g[s - 1];
    }
     if (k == 6)
    {
         double  const g[] =
        {
            -6.679579202136281, -4.018407802061622, -3.544907701811032, -4.062353818279200,
            -6.772722179448756
        };
         return g[s - 1];
    }
     if (k == 7)
    {
         double  const g[] =
        {
            -7.639406763622460, -4.408761164863911, -3.618145521480401, -3.636689076772442,
            -4.465974364227054, -7.740369506306969
        };
```

```
         return g[s - 1];
    }
     if (k == 8)
    {
         double  const g[] =
        {
            -8.610218970054416, -4.834146544295875, -3.792697895066562, -3.544907701811032,
            -3.825383594908152, -4.901666809860712, -8.717218859383176
        };
         return g[s - 1];
    }
     if (k == 9)
    {
         double  const g[] =
        {
            -9.588024156621911, -5.279888014349222, -4.018407802061622, -3.587208340962461,
            -3.601611266431462, -4.062353818279200, -5.355680341107928, -9.699829498201463
        };
         return g[s - 1];
    }
     if (k == 10)
    {
         double  const g[] =
        {
            -10.57056410963193, -5.738554639998505, -4.273669982410846, -3.696932572929480,
            -3.544907701811032, -3.722980622032043, -4.326851108825193, -5.821148568626515,
            -10.68628702119319
        };
         return g[s - 1];
    }
     if (k == 15)
    {
         double  const g[] =
        {
            -15.51960093424780, -8.123745140089901, -5.738554639998505, -4.618238335543039,
            -4.018407802061622, -3.696932572929480, -3.558270690149276, -3.566900802267063,
            -3.722980622032043, -4.062353818279200, -4.680905938427318, -5.821148568626515,
            -8.227908487366704, -15.64749765297494
        };
         return g[s - 1];
    }
     if (k == 20)
    {
         double  const g[] =
        {
            -20.49482664342686, -10.57056410963193, -7.317968087117503, -5.738554639998505,
            -4.834146544295875, -4.273669982410846, -3.917149195711212, -3.696932572929480,
            -3.578429819277059, -3.544907701811032, -3.591387263852389, -3.722980622032043,
            -3.956557434361457, -4.326851108825193, -4.901666809860712, -5.821148568626515,
            -7.416558246323102, -10.68628702119319, -20.62906634258064
        };
         return g[s - 1];
    }
     return 0;
}

 double gamma( int k, vector<int> & sigma, vector<double> & valueS, vector<int> & numberS)
```

```
{
     int s = 0;
     for ( int i = 0; i < sigma.size(); i ++)  if (sigma[i] == 1) s ++;
    numberS[s] ++;

     double  const term1 = -s / (k*k * log(2.0));
     double  const term2 = gammmaKS(k, s);
     double term3 = 1.0;
     double  const ex = 1.0 - (double)s / k;
     int prod = 1;
     for ( int l = 1; l < k; l ++)
    {
        prod *= sigma[l];
        term3 += prod * pow(2.0, -l*ex);
    }
     double result = term1 * term2 * term3;
    valueS[s] += result;
     return result;
}

 double averageGamma( int k)
{
    vector<int> sigma(k, 2);
    vector<int> ones(k, 1);
    vector<double> valueS(k, 0.0);
    vector<int> numberS(k, 0);
     int count = 0;
     double sum = 0.0;
     while (1)
    {
         int i = 0;
         while (-- sigma[i] == 0)  //loop for move carrier
        {
            sigma[i] = 2;
            i ++;
        }
         if (sigma == ones) break;
         double g = gamma(k, sigma, valueS, numberS);
        sum += g;
        count ++;
        }
     return sum / count;
}

 void main()
{
    for (int k = 10; k <= 10; k ++)
        cout << "total average" << k << ": " << averageGamma(k) << endl << endl;
    cout << 15 << ": " << averageGamma(15) << endl;
    cout << 20 << ": " << averageGamma(20) << endl;
}
```

Appendix E
Scripts and plots of experimental results.

GNUPLOT scripts and generated figures for expected time due to equation (5.5) and experimental data (Tables 5.1 to 5.9).

nstablefig.gnu

```
set autoscale
set nologscale y
set nologscale x
set xlabel 'k'
set ylabel 'time'

set terminal fig

set title 'Experimental time (n = 10)'
set output 'experimentn10.fig'
plot 'experimentn10WC1.txt' using 1:3 title '% of data in range = [0%, 10%]' with \
linespoints, 'experimentn10WC2.txt' using 1:3 title '% of data in range = [10%, 20%]'\
with linespoints, 'experimentn10WC3.txt' using 1:3 title '% of data in range = [20%,30%]'\
with linespoints, 'experimentn10WC4.txt' using 1:3 title '% of data in range = [30%,40%]'\
with linespoints, 'experimentn10WC5.txt' using 1:3 title '% of data in range = [40%,50%]'\
with linespoints, 'experimentn10WC6.txt' using 1:3 title '% of data in range = [50%,60%]'\
with linespoints, 'experimentn10WC7.txt' using 1:3 title '% of data in range = [60%,70%]'\
with linespoints, 'experimentn10WC8.txt' using 1:3 title '% of data in range = [70%,80%]'\
with linespoints, 'experimentn10WC9.txt' using 1:3 title '% of data in range = [80%,90%]'\
with linespoints, 'experimentn10WC10.txt' using 1:3 title '% of data in range=[90%,100%]'\
with linespoints

set title 'Experimental time (n = 100)'
set output 'experimentn100.fig'
plot 'experimentn10WC1.txt' using 1:3 title '% of data in range = [0%, 10%]' with \
linespoints, 'experimentn100WC2.txt' using 1:3 title '% of data in range = [10%, 20%]' \
with linespoints,'experimentn100WC3.txt' using 1:3 title '% of data in range = [20%,30%]'\
with linespoints,'experimentn100WC4.txt' using 1:3 title '% of data in range = [30%,40%]'\
with linespoints,'experimentn100WC5.txt' using 1:3 title '% of data in range = [40%,50%]'\
with linespoints,'experimentn100WC6.txt' using 1:3 title '% of data in range = [50%,60%]'\
with linespoints,'experimentn100WC7.txt' using 1:3 title '% of data in range = [60%,70%]'\
with linespoints,'experimentn100WC8.txt' using 1:3 title '% of data in range = [70%,80%]'\
with linespoints,'experimentn100WC9.txt' using 1:3 title '% of data in range = [80%,90%]'\
with linespoints,'experimentn100WC10.txt' using 1:3 title '% of data in range=[90%,100%]'\
with linespoints

set title 'Experimental time (n = 1000)'
set output 'experimentn1000.fig'
plot 'experimentn1000WC1.txt' using 1:3 title '% of data in range = [0%, 10%]' with \
linespoints, 'experimentn1000WC2.txt' using 1:3 title '% of data in range = [10%, 20%]' \
with linespoints,'experimentn1000WC3.txt' using 1:3 title '% of data in range=[20%, 30%]'\
with linespoints,'experimentn1000WC4.txt' using 1:3 title '% of data in range=[30%, 40%]'\
with linespoints,'experimentn1000WC5.txt' using 1:3 title '% of data in range=[40%, 50%]'\
```

```
with linespoints,'experimentn1000WC6.txt' using 1:3 title '% of data in range=[50%, 60%]'\
with linespoints,'experimentn1000WC7.txt' using 1:3 title '% of data in range=[60%, 70%]'\
with linespoints,\

set title 'Experimental time (n = 10000)'
set output 'experimentn10000.fig'
plot 'experimentn10000WC1.txt' using 1:3 title '% of data in range = [0%, 10%]' with \
linespoints, 'experimentn10000WC2.txt' using 1:3 title '% of data in range = [10%, 20%]'\
with linespoints,'experimentn10000WC3.txt' using 1:3 title '% of data in range=[20%,30%]'\
with linespoints,'experimentn10000WC4.txt' using 1:3 title '% of data in range=[30%,40%]'\
with linespoints,'experimentn10000WC5.txt' using 1:3 title '% of data in range=[40%,50%]'\
with linespoints,'experimentn10000WC6.txt' using 1:3 title '% of data in range=[50%,60%]'\
with linespoints,'experimentn10000WC7.txt' using 1:3 title '% of data in range=[60%,70%]'\
with linespoints,'experimentn10000WC8.txt' using 1:3 title '% of data in range=[70%,80%]'\
with linespoints,'experimentn10000WC9.txt' using 1:3 title '% of data in range=[80%,90%]'\
with linespoints,'experimentn10000WC10.txt'using 1:3 title'% of data in range=[90%,100%]'\
with linespoints

set title 'Experimental time (n = 100000)'
set output 'experimentn100000.fig'
plot 'experimentn100000WC1.txt' using 1:3 title '% of data in range = [0%, 10%]' with \
linespoints,'experimentn100000WC2.txt' using 1:3 title '% of data in range = [10%, 20%]'\
with linespoints,'experimentn100000WC3.txt' using 1:3 title'% of data in range=[20%,30%]'\
with linespoints,'experimentn100000WC4.txt' using 1:3 title'% of data in range=[30%,40%]'\
with linespoints,'experimentn100000WC5.txt' using 1:3 title'% of data in range=[40%,50%]'\
with linespoints,'experimentn100000WC6.txt' using 1:3 title'% of data in range=[50%,60%]'\
with linespoints,'experimentn100000WC7.txt' using 1:3 title'% of data in range=[60%,70%]'\
with linespoints,'experimentn100000WC8.txt' using 1:3 title'% of data in range=[70%,80%]'\
with linespoints,'experimentn100000WC9.txt' using 1:3 title'% of data in range=[80%,90%]'\
with linespoints,'experimentn100000WC10.txt'using 1:3 title'% of data in range=[90%,100%]'\
with linespoints

set title 'Expected time (n = 10)'
set output 'expectn10.fig'
plot 'expectn10WC1.txt' using 1:3 title '% of data in range = [0%,10%]' with linespoints,\
'expectn10WC2.txt' using 1:3 title ' [10%, 20%]' with linespoints,\
'expectn10WC3.txt' using 1:3 title '[20%, 30%]' with linespoints, \
'expectn10WC4.txt' using 1:3 title '[30%, 40%]' with linespoints,\
'expectn10WC5.txt' using 1:3 title '[40%, 50%]' with linespoints,\
'expectn10WC6.txt' using 1:3 title '[50%, 60%]' with linespoints, \
'expectn10WC7.txt' using 1:3 title ' [60%, 70%]' with linespoints,\
'expectn10WC8.txt' using 1:3 title ' [70%, 80%]' with linespoints,\
'expectn10WC9.txt' using 1:3 title ' [80%, 90%]' with linespoints, \
'expectn10WC10.txt' using 1:3 title ' [90%, 100%]' with linespoints


set title 'Expected time (n = 100)'
set output 'expectn100.fig'
plot 'expectn100WC1.txt' using 1:3 title '% of data in range=[0%, 10%]' with linespoints,\
'expectn100WC2.txt' using 1:3 title ' [10%, 20%]' with linespoints,\
'expectn100WC3.txt' using 1:3 title '[20%, 30%]' with linespoints, \
'expectn100WC4.txt' using 1:3 title '[30%, 40%]' with linespoints,\
'expectn100WC5.txt' using 1:3 title '[40%, 50%]' with linespoints,\
'expectn100WC6.txt' using 1:3 title '[50%, 60%]' with linespoints, \
'expectn100WC7.txt' using 1:3 title ' [60%, 70%]' with linespoints,\
'expectn100WC8.txt' using 1:3 title ' [70%, 80%]' with linespoints,\
'expectn100WC9.txt' using 1:3 title ' [80%, 90%]' with linespoints, \
```

```
'expectn100WC10.txt' using 1:3 title ' [90%, 100%]' with linespoints


set title 'Expected time (n = 1000)'
set output 'expectn1000.fig'
plot 'expectn1000WC1.txt' using 1:3 title '% of data in range=[0%,10%]' with linespoints,\
'expectn1000WC2.txt' using 1:3 title ' [10%, 20%]' with linespoints,\
'expectn1000WC3.txt' using 1:3 title '[20%, 30%]' with linespoints, \
'expectn1000WC4.txt' using 1:3 title '[30%, 40%]' with linespoints,\
'expectn1000WC5.txt' using 1:3 title '[40%, 50%]' with linespoints,\
'expectn1000WC6.txt' using 1:3 title '[50%, 60%]' with linespoints, \
'expectn1000WC7.txt' using 1:3 title ' [60%, 70%]' with linespoints,\
'expectn1000WC8.txt' using 1:3 title ' [70%, 80%]' with linespoints,\
'expectn1000WC9.txt' using 1:3 title ' [80%, 90%]' with linespoints, \
'expectn1000WC10.txt' using 1:3 title ' [90%, 100%]' with linespoints

set title 'Expected time (n = 10000)'
set output 'expectn10000.fig'
plot 'expectn10000WC1.txt' using 1:3 title'% of data in range=[0%,10%]' with linespoints,\
'expectn10000WC2.txt' using 1:3 title ' [10%, 20%]' with linespoints,\
'expectn10000WC3.txt' using 1:3 title '[20%, 30%]' with linespoints, \
'expectn10000WC4.txt' using 1:3 title '[30%, 40%]' with linespoints,\
'expectn10000WC5.txt' using 1:3 title '[40%, 50%]' with linespoints,\
'expectn10000WC6.txt' using 1:3 title '[50%, 60%]' with linespoints, \
'expectn10000WC7.txt' using 1:3 title ' [60%, 70%]' with linespoints,\
'expectn10000WC8.txt' using 1:3 title ' [70%, 80%]' with linespoints,\
'expectn10000WC9.txt' using 1:3 title ' [80%, 90%]' with linespoints, \
'expectn10000WC10.txt' using 1:3 title ' [90%, 100%]' with linespoints



set title 'Expected time (n = 100000)'
set output 'expectn100000.fig'
plot 'expectn100000WC1.txt'using 1:3 title'% of data in range=[0%,10%]' with linespoints,\
'expectn100000WC2.txt' using 1:3 title ' [10%, 20%]' with linespoints,\
'expectn100000WC3.txt' using 1:3 title '[20%, 30%]' with linespoints, \
'expectn100000WC4.txt' using 1:3 title '[30%, 40%]' with linespoints,\
'expectn100000WC5.txt' using 1:3 title '[40%, 50%]' with linespoints,\
'expectn100000WC6.txt' using 1:3 title '[50%, 60%]' with linespoints, \
'expectn100000WC7.txt' using 1:3 title ' [60%, 70%]' with linespoints,\
'expectn100000WC8.txt' using 1:3 title ' [70%, 80%]' with linespoints,\
'expectn100000WC9.txt' using 1:3 title ' [80%, 90%]' with linespoints, \
'expectn100000WC10.txt' using 1:3 title ' [90%, 100%]' with linespoints

set output
set terminal x11
```

kstablefig.gnu

```
 set autoscale set nologscale y set nologscale x set xlabel
'n' set ylabel 'time'

set terminal fig

set title '% of data in range = [0%, 10%]'
```

```
set output 'expsWC1.txt' plot 'experimentK2WC1.txt' using 2:3
title 'k=2 experimental time' with linespoints, \
'expectK2WC1.txt' using 2:3 title 'k=2 expected time' with
linespoints, \ 'experimentK3WC1.txt' using 2:3 title 'k=3
experimental time' with linespoints, \ 'expectK3WC1.txt'
using 2:3 title 'k=3 expected time' with linespoints, \
'experimentK4WC1.txt' using 2:3 title 'k=4 experimental time' with
linespoints, \ 'expectK4WC1.txt' using 2:3 title 'k=4
expected time' with linespoints, \ 'experimentK5WC1.txt'
using 2:3 title 'k=5 experimental time' with linespoints, \
'expectK5WC1.txt' using 2:3 title 'k=5 expected time' with
linespoints, \ 'experimentK6WC1.txt' using 2:3 title 'k=6
experimental time' with linespoints, \ 'expectK6WC1.txt'
using 2:3 title 'k=6 expected time' with linespoints, \
'experimentK7WC1.txt' using 2:3 title 'k=7 experimental time' with
linespoints, \ 'expectK7WC1.txt' using 2:3 title 'k=7
expected time' with linespoints, \ 'experimentK8WC1.txt'
using 2:3 title 'k=8 experimental time' with linespoints, \
'expectK8WC1.txt' using 2:3 title 'k=8 expected time' with
linespoints, \ 'experimentK9WC1.txt' using 2:3 title 'k=9
experimental time' with linespoints, \ 'expectK9WC1.txt'
using 2:3 title 'k=9 expected time' with linespoints

set title '% of data in range = [10%, 20%]'
set output 'expsWC2.fig' plot 'experimentK2WC2.txt' using 2:3
title 'k=2 experimental time' with linespoints, \
'expectK2WC2.txt' using 2:3 title 'k=2 expected time' with
linespoints, \ 'experimentK3WC2.txt' using 2:3 title 'k=3
experimental time' with linespoints, \ 'expectK3WC2.txt'
using 2:3 title 'k=3 expected time' with linespoints, \
'experimentK4WC2.txt' using 2:3 title 'k=4 experimental time' with
linespoints, \ 'expectK4WC2.txt' using 2:3 title 'k=4
expected time' with linespoints, \ 'experimentK5WC2.txt'
using 2:3 title 'k=5 experimental time' with linespoints, \
'expectK5WC2.txt' using 2:3 title 'k=5 expected time' with
linespoints, \ 'experimentK6WC2.txt' using 2:3 title 'k=6
experimental time' with linespoints, \ 'expectK6WC2.txt'
using 2:3 title 'k=6 expected time' with linespoints,\
'experimentK7WC2.txt' using 2:3 title 'k=7 experimental time' with
linespoints, \ 'expectK7WC2.txt' using 2:3 title 'k=7
expected time' with linespoints, \ 'experimentK8WC2.txt'
using 2:3 title 'k=8 experimental time' with linespoints, \
'expectK8WC2.txt' using 2:3 title 'k=8 expected time' with
linespoints, \ 'experimentK9WC2.txt' using 2:3 title 'k=9
experimental time' with linespoints, \ 'expectK9WC2.txt'
using 2:3 title 'k=9 expected time' with linespoints



set title '% of data in range = [20%, 30%]'
set output 'expsWC3.fig' plot 'experimentK2WC3.txt' using 2:3
title 'k=2 experimental time' with linespoints, \
'expectK2WC3.txt' using 2:3 title 'k=2 expected time' with
linespoints, \ 'experimentK3WC3.txt' using 2:3 title 'k=3
experimental time' with linespoints, \ 'expectK3WC3.txt'
using 2:3 title 'k=3 expected time' with linespoints, \
'experimentK4WC3.txt' using 2:3 title 'k=4 experimental time' with
```

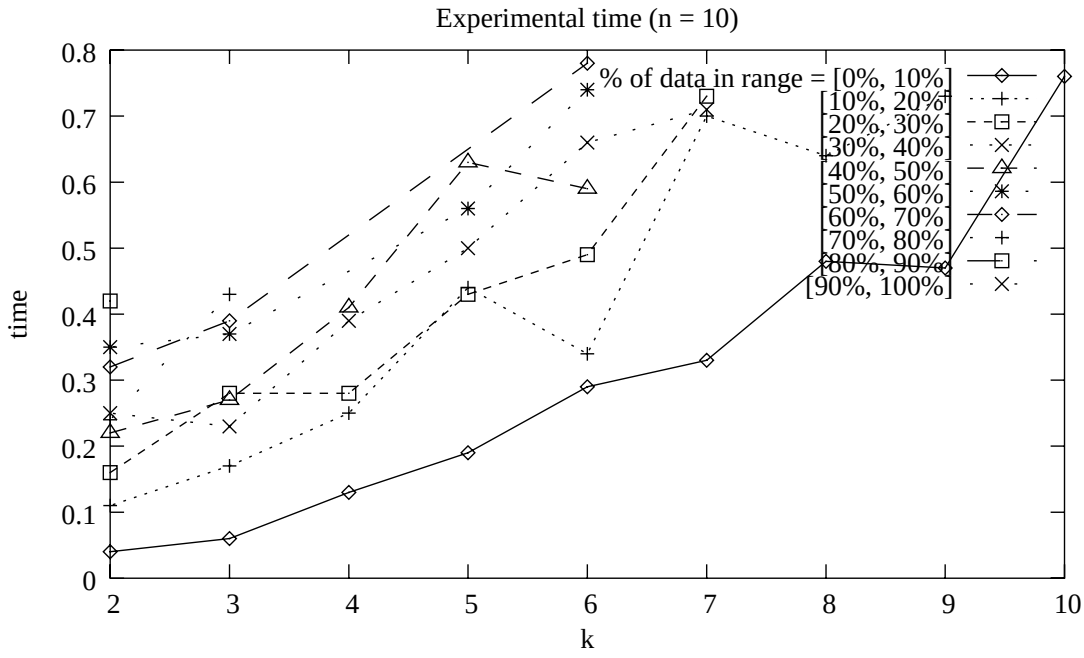
```
linespoints, \ 'expectK4WC3.txt' using 2:3 title 'k=4
expected time' with linespoints, \ 'experimentK5WC3.txt'
using 2:3 title 'k=5 experimental time' with linespoints, \
'expectK5WC3.txt' using 2:3 title 'k=5 expected time' with
linespoints, \ 'experimentK6WC3.txt' using 2:3 title 'k=6
experimental time' with linespoints, \ 'expectK6WC3.txt'
using 2:3 title 'k=6 expected time' with linespoints,\
'experimentK7WC3.txt' using 2:3 title 'k=7 experimental time' with
linespoints, \ 'expectK7WC3.txt' using 2:3 title 'k=7
expected time' with linespoints, \ 'experimentK8WC3.txt'
using 2:3 title 'k=8 experimental time' with linespoints, \
'expectK8WC3.txt' using 2:3 title 'k=8 expected time' with
linespoints, \ 'experimentK9WC3.txt' using 2:3 title 'k=9
experimental time' with linespoints, \ 'expectK9WC3.txt'
using 2:3 title 'k=9 expected time' with linespoints

set title '% of data in range = [30%, 40%]'
set output 'expsWC4.fig' plot 'experimentK2WC4.txt' using 2:3
title 'k=2 experimental time' with linespoints, \
'expectK2WC4.txt' using 2:3 title 'k=2 expected time' with
linespoints, \ 'experimentK3WC4.txt' using 2:3 title 'k=3
experimental time' with linespoints, \ 'expectK3WC4.txt'
using 2:3 title 'k=3 expected time' with linespoints, \
'experimentK4WC4.txt' using 2:3 title 'k=4 experimental time' with
linespoints, \ 'expectK4WC4.txt' using 2:3 title 'k=4
expected time' with linespoints, \ 'experimentK5WC4.txt'
using 2:3 title 'k=5 experimental time' with linespoints, \
'expectK5WC4.txt' using 2:3 title 'k=5 expected time' with
linespoints, \ 'experimentK6WC4.txt' using 2:3 title 'k=6
experimental time' with linespoints, \ 'expectK6WC4.txt'
using 2:3 title 'k=6 expected time' with linespoints,\
'experimentK7WC4.txt' using 2:3 title 'k=7 experimental time' with
linespoints, \ 'expectK7WC4.txt' using 2:3 title 'k=7
expected time' with linespoints, \ 'experimentK8WC4.txt'
using 2:3 title 'k=8 experimental time' with linespoints, \
'expectK8WC4.txt' using 2:3 title 'k=8 expected time' with
linespoints, \ 'experimentK9WC4.txt' using 2:3 title 'k=9
experimental time' with linespoints


set output 'experimentWC1.fig'
set title 'Experimental time (% of data in range = [0%,10%])'
plot 'experimentK2WC1.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC1.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC1.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC1.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC1.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC1.txt' using
2:3 title 'k=7' with linespoints, \ 'experimentK8WC1.txt'
using 2:3 title 'k=8' with linespoints, \
'experimentK9WC1.txt' using 2:3 title 'k=9' with linespoints,
\ 'experimentK10WC1.txt' using 2:3 title 'k=10' with
linespoints

set output 'experimentWC2.fig'
set title 'Experimental time (% of data in range = [10%,20%])'
plot 'experimentK2WC2.txt' using 2:3 title 'k=2' with linespoints,
```

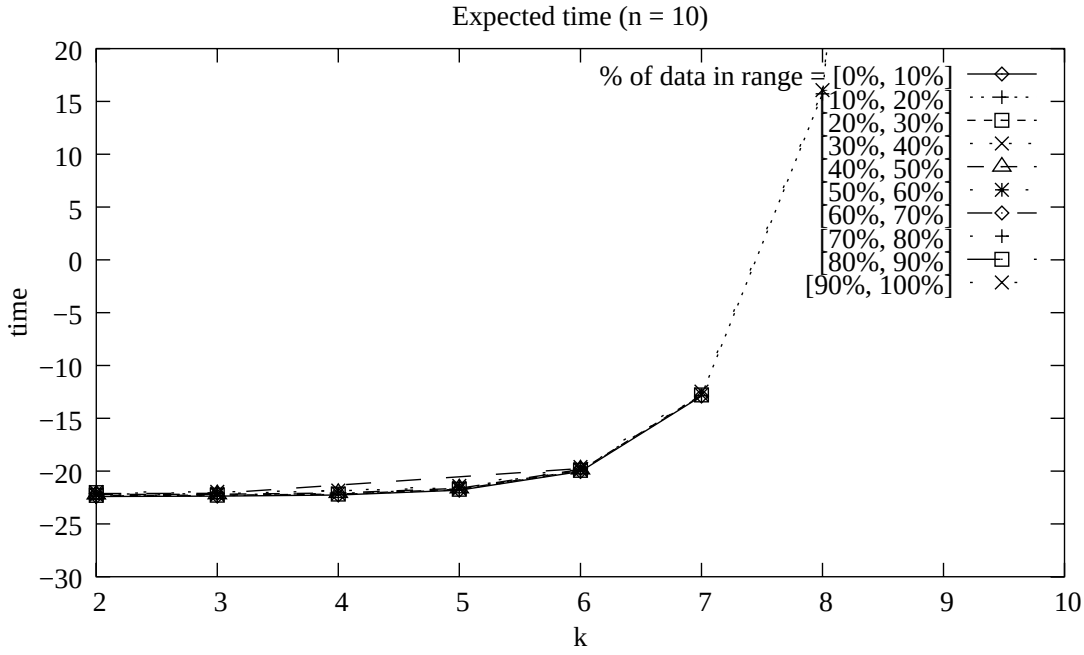
```
\ 'experimentK3WC2.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC2.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC2.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC2.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC2.txt' using
2:3 title 'k=7' with linespoints, \ 'experimentK8WC2.txt'
using 2:3 title 'k=8' with linespoints, \
'experimentK9WC2.txt' using 2:3 title 'k=9' with linespoints,
\ 'experimentK10WC2.txt' using 2:3 title 'k=10' with
linespoints


set output 'experimentWC3.fig'
set title 'Experimental time (% of data in range = [20%,30%])'
plot 'experimentK2WC3.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC3.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC3.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC3.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC3.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC3.txt' using
2:3 title 'k=7' with linespoints, \ 'experimentK8WC3.txt'
using 2:3 title 'k=8' with linespoints, \
'experimentK9WC3.txt' using 2:3 title 'k=9' with linespoints

set output 'experimentWC4.fig'
set title 'Experimental time (% of data in range = [30%,40%])'
plot 'experimentK2WC4.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC4.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC4.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC4.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC4.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC4.txt' using
2:3 title 'k=7' with linespoints

set output 'experimentWC5.fig'
set title 'Experimental time (% of data in range = [40%,50%])'
plot 'experimentK2WC5.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC5.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC5.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC5.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC5.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC5.txt' using
2:3 title 'k=7' with linespoints

set output 'experimentWC6.fig'
set title 'Experimental time (% of data in range = [50%,60%])'
plot 'experimentK2WC6.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC6.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC6.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC6.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC6.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC6.txt' using
2:3 title 'k=7' with linespoints

set output 'experimentWC7.fig'
set title 'Experimental time (% of data in range = [60%,70%])'
plot 'experimentK2WC7.txt' using 2:3 title 'k=2' with linespoints,
```

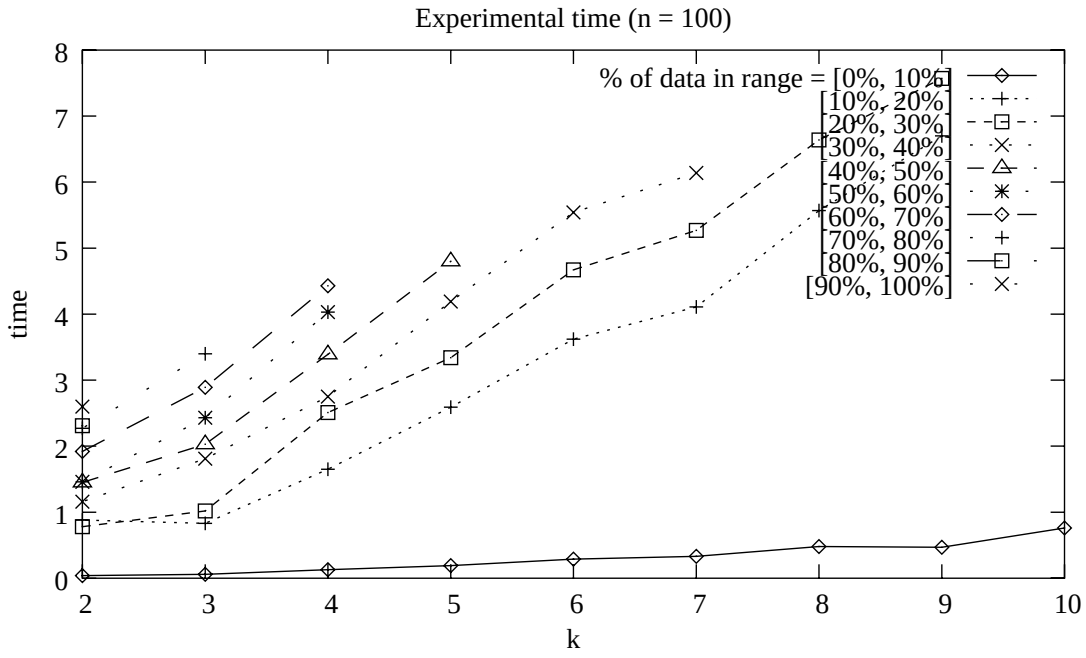
```
\ 'experimentK3WC7.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC7.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC7.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC7.txt' using 2:3
title 'k=6' with linespoints, \ 'experimentK7WC7.txt' using
2:3 title 'k=7' with linespoints

set output 'experimentWC8.fig'
set title 'Experimental time (% of data in range = [70%,80%])'
plot 'experimentK2WC8.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC8.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC8.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC8.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC8.txt' using 2:3
title 'k=6' with linespoints

set output 'experimentWC9.fig'
set title 'Experimental time (% of data in range = [80%,90%])'
plot 'experimentK2WC9.txt' using 2:3 title 'k=2' with linespoints,
\ 'experimentK3WC9.txt' using 2:3 title 'k=3' with
linespoints, \ 'experimentK4WC9.txt' using 2:3 title 'k=4'
with linespoints, \ 'experimentK5WC9.txt' using 2:3 title
'k=5' with linespoints, \ 'experimentK6WC9.txt' using 2:3
title 'k=6' with linespoints

set output 'experimentWC10.fig'
set title 'Experimental time (% of data in range = [90%,10%])'
plot 'experimentK2WC10.txt' using 2:3 title 'k=2' with
linespoints, \ 'experimentK3WC10.txt' using 2:3 title 'k=3'
with linespoints

#claim all the negtive values (noise) to be 0
#for displaying in log scale, change value from 0 to 0.1

set output 'expectWC1.fig'
set title 'Expected time (% of data in range = [0%,10%])'
plot 'expectK2WC1.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC1.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC1.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC1.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC1.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC1.txt' using 2:3 title 'k=7' with linespoints,
\ 'expectK8WC1.txt' using 2:3 title 'k=8' with linespoints,
\ 'expectK9WC1.txt' using 2:3 title 'k=9' with linespoints,
\ 'expectK10WC1.txt' using 2:3 title 'k=10' with
linespoints

set output 'expectWC2.fig'
set title 'Expected time (% of data in range = [10%,20%])'
plot 'expectK2WC2.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC2.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC2.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC2.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC2.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC2.txt' using 2:3 title 'k=7' with linespoints,
\ 'expectK8WC2.txt' using 2:3 title 'k=8' with linespoints,
\ 'expectK9WC2.txt' using 2:3 title 'k=9' with linespoints,
```

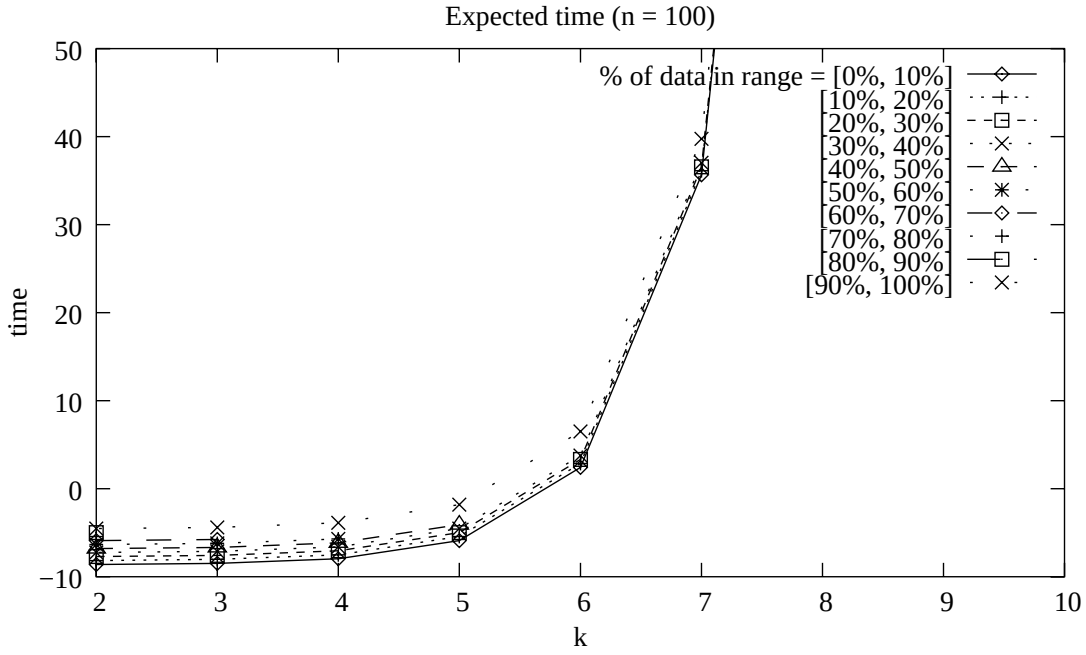
```
\ 'expectK10WC2.txt' using 2:3 title 'k=10' with
linespoints

set output 'expectWC3.fig'
set title 'Expected time (% of data in range = [20%,30%])'
plot 'expectK2WC3.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC3.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC3.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC3.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC3.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC3.txt' using 2:3 title 'k=7' with linespoints,
\ 'expectK8WC3.txt' using 2:3 title 'k=8' with linespoints,
\ 'expectK9WC3.txt' using 2:3 title 'k=9' with linespoints

set output 'expectWC4.fig'
set title 'Expected time (% of data in range = [30%,40%])'
plot 'expectK2WC4.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC4.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC4.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC4.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC4.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC4.txt' using 2:3 title 'k=7' with linespoints,
\ 'expectK8WC4.txt' using 2:3 title 'k=8' with linespoints

set output 'expectWC5.fig'
set title 'Expected time (% of data in range = [40%,50%])'
plot 'expectK2WC5.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC5.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC5.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC5.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC5.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC5.txt' using 2:3 title 'k=7' with linespoints,
\ 'expectK8WC5.txt' using 2:3 title 'k=8' with linespoints


set output 'expectWC6.fig'
set title 'Expected time (% of data in range = [50%,60%])'
plot 'expectK2WC6.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC6.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC6.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC6.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC6.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC6.txt' using 2:3 title 'k=7' with linespoints

set output 'expectWC7.fig'
set title 'Expected time (% of data in range = [60%,70%])'
plot 'expectK2WC7.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC7.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC7.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC7.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC7.txt' using 2:3 title 'k=6' with linespoints,
\ 'expectK7WC7.txt' using 2:3 title 'k=7' with linespoints

set output 'expectWC8.fig'
set title 'Expected time (% of data in range = [70%,80%])'
plot 'expectK2WC8.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC8.txt' using 2:3 title 'k=3' with linespoints,
```

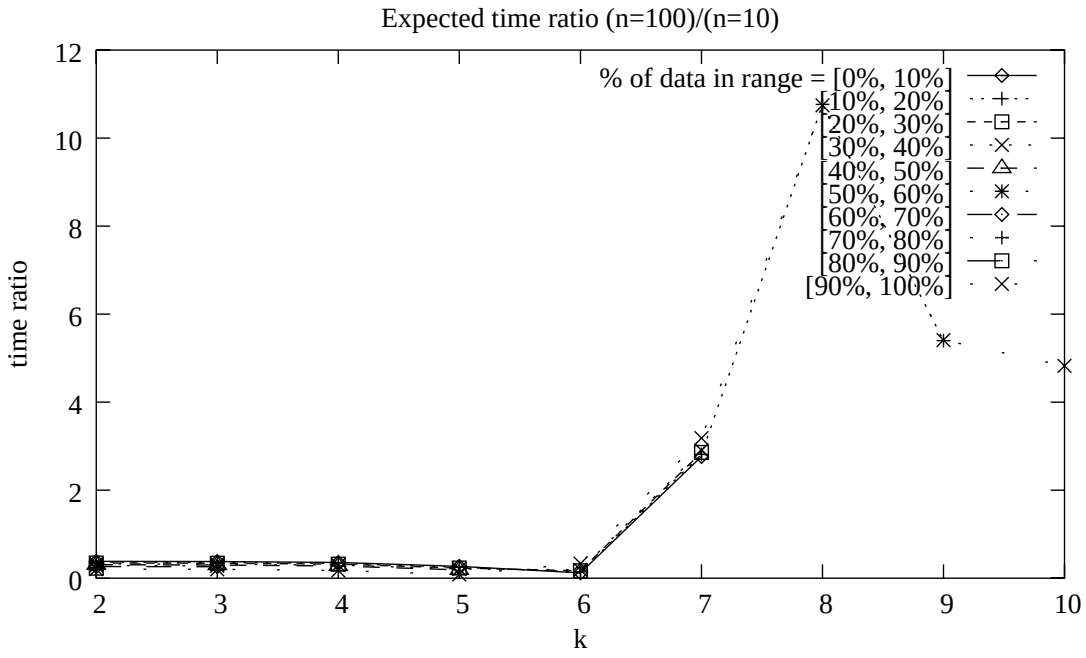
```
\ 'expectK4WC8.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC8.txt' using 2:3 title 'k=5' with linespoints,
\ 'expectK6WC8.txt' using 2:3 title 'k=6' with linespoints

set output 'expectWC9.fig'
set title 'Expected time (% of data in range = [80%,90%])'
plot 'expectK2WC9.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC9.txt' using 2:3 title 'k=3' with linespoints,
\ 'expectK4WC9.txt' using 2:3 title 'k=4' with linespoints,
\ 'expectK5WC9.txt' using 2:3 title 'k=5' with linespoints

set output 'expectWC10.fig'
set title 'Expected time (% of data in range = [90%,100%])'
plot 'expectK2WC10.txt' using 2:3 title 'k=2' with linespoints,
\ 'expectK3WC10.txt' using 2:3 title 'k=3' with
linespoints, \ 'expectK4WC10.txt' using 2:3 title 'k=4'
with linespoints


set output set terminal x11
```

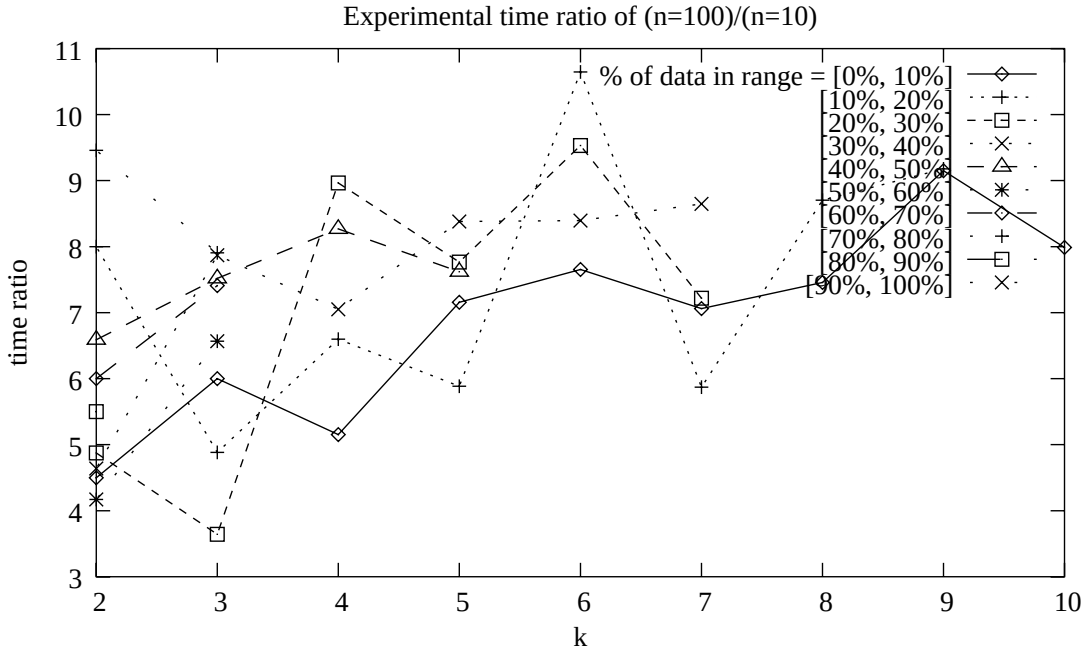

Experimental time (n = 10)
time
k
% of data in range =
[0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

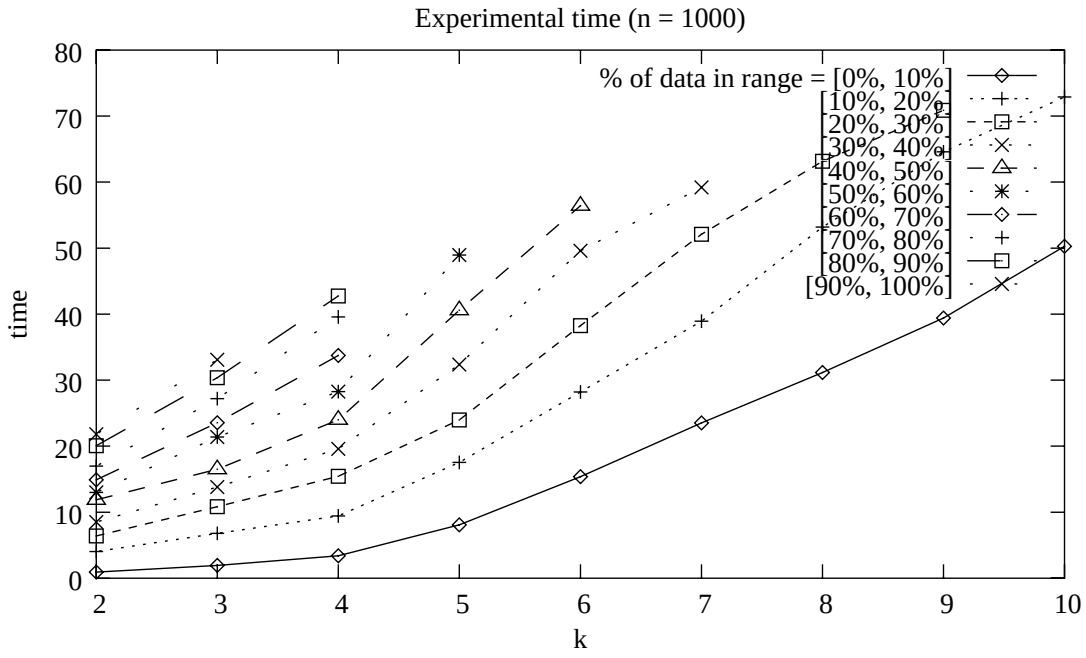

Expected time (n = 10)
time
k
% of data in range =
[0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

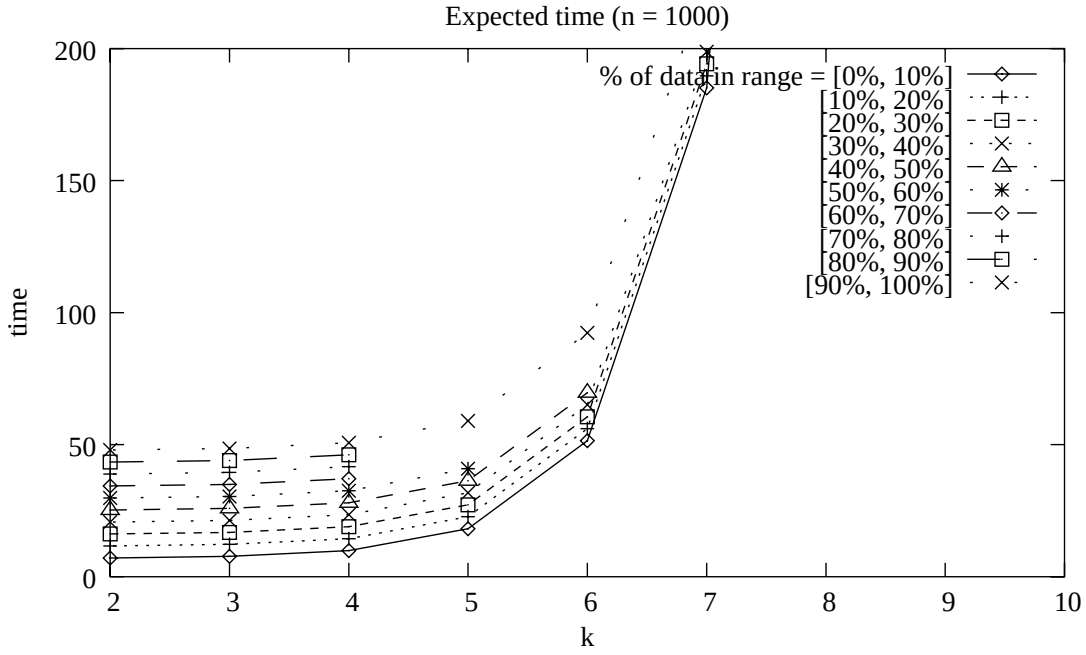

Experimental time (n = 100)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

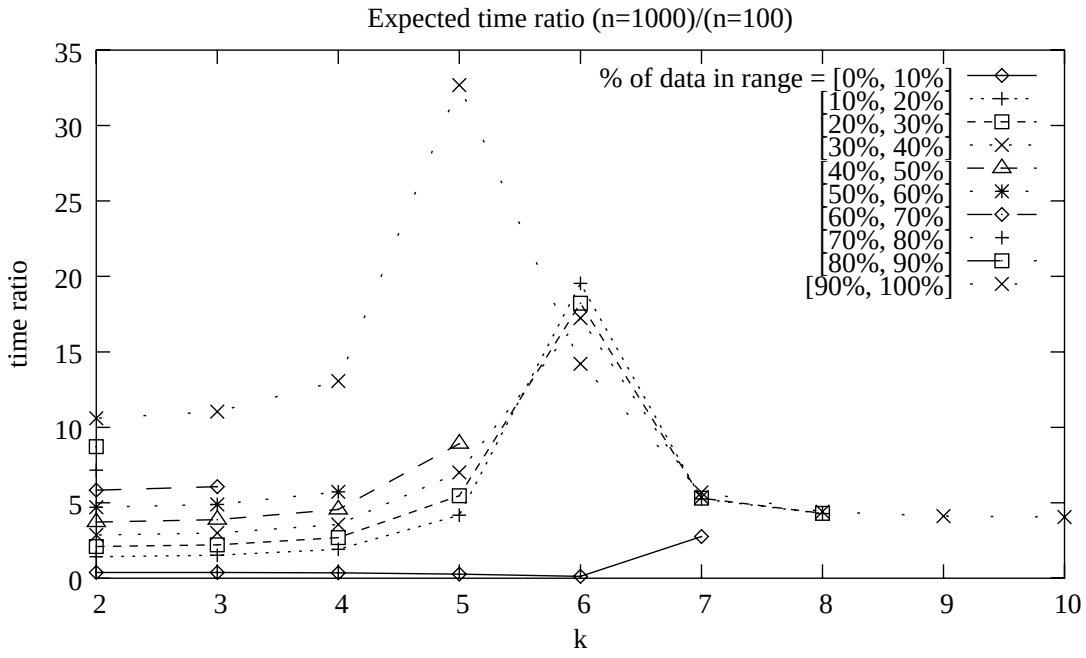

Expected time (n = 100)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

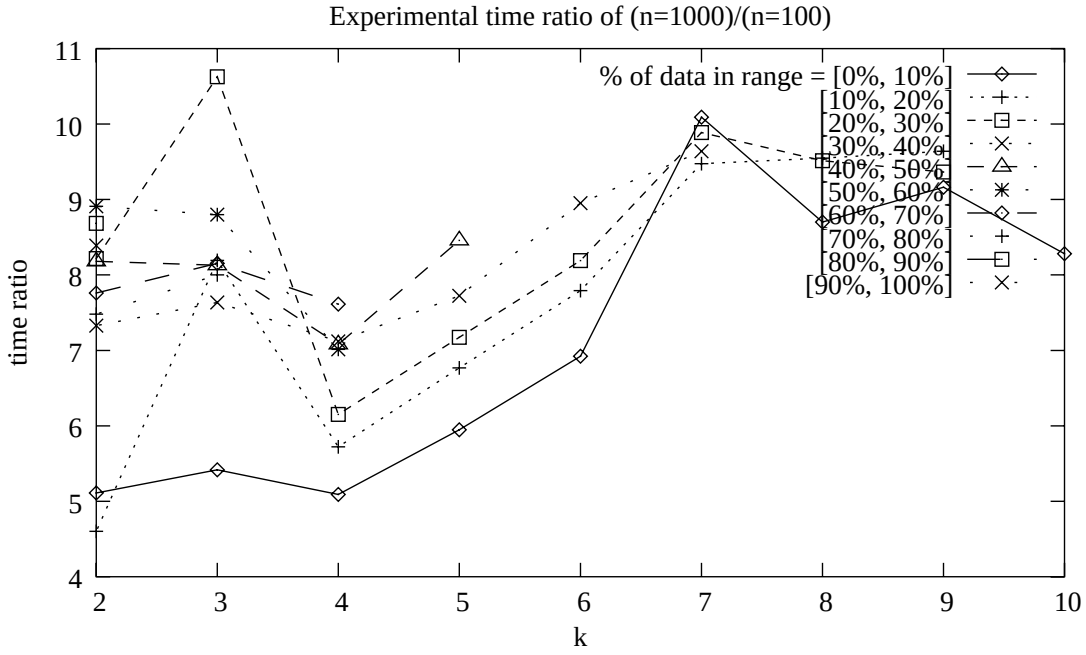

Expected time ratio (n=100)/(n=10)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

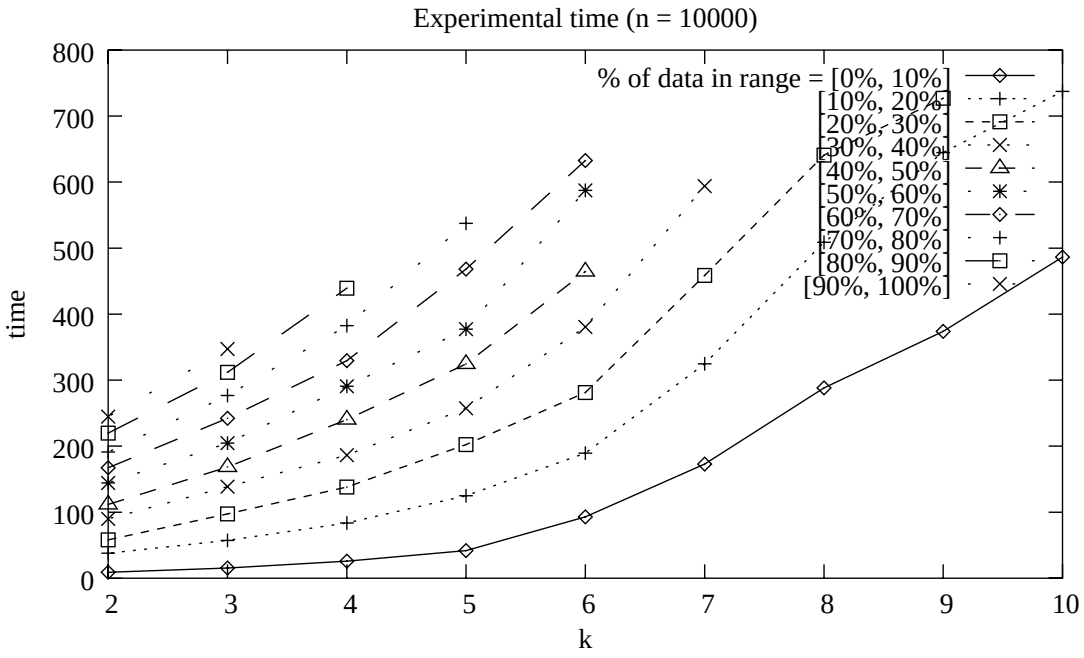

Experimental time ratio of (n=100)/(n=10)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

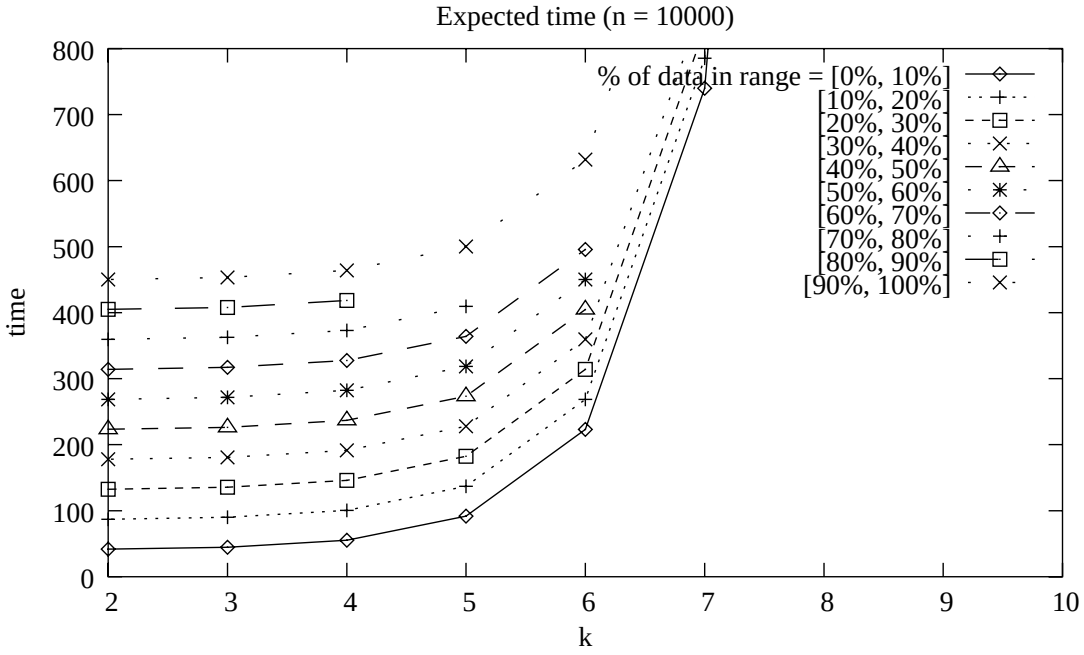

Experimental time (n = 1000)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

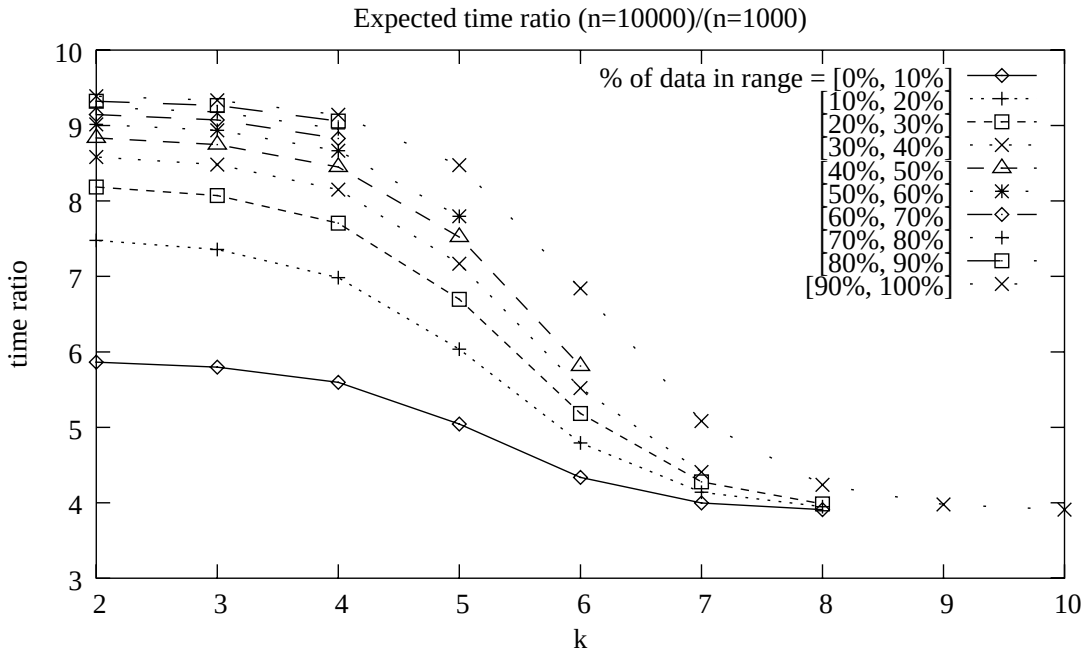

Expected time (n = 1000)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

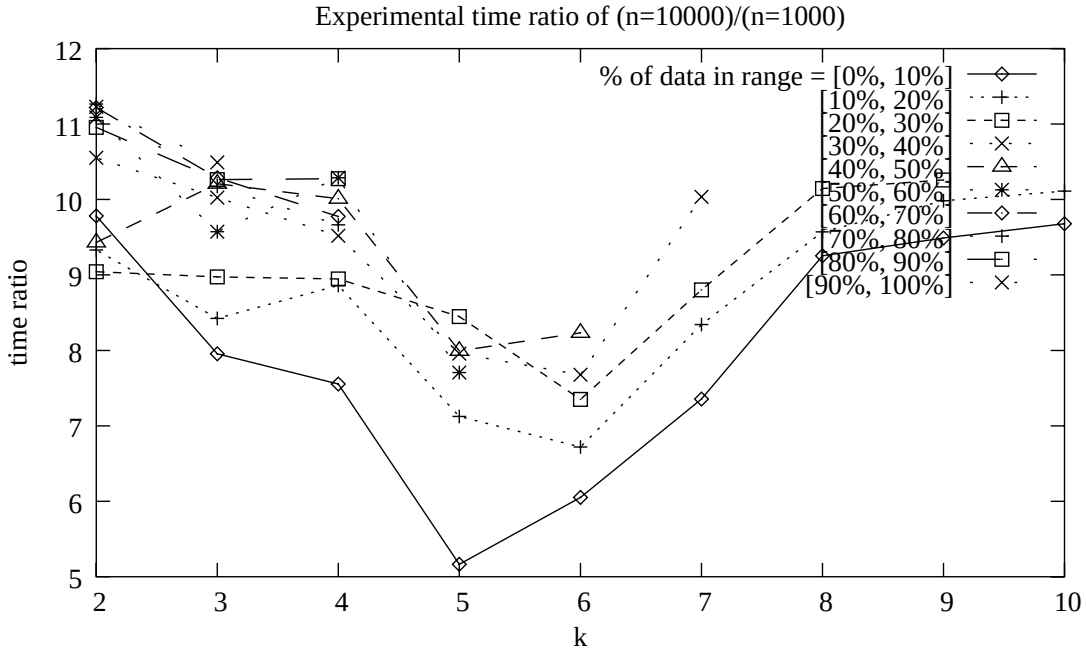

Expected time ratio (n=1000)/(n=100)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

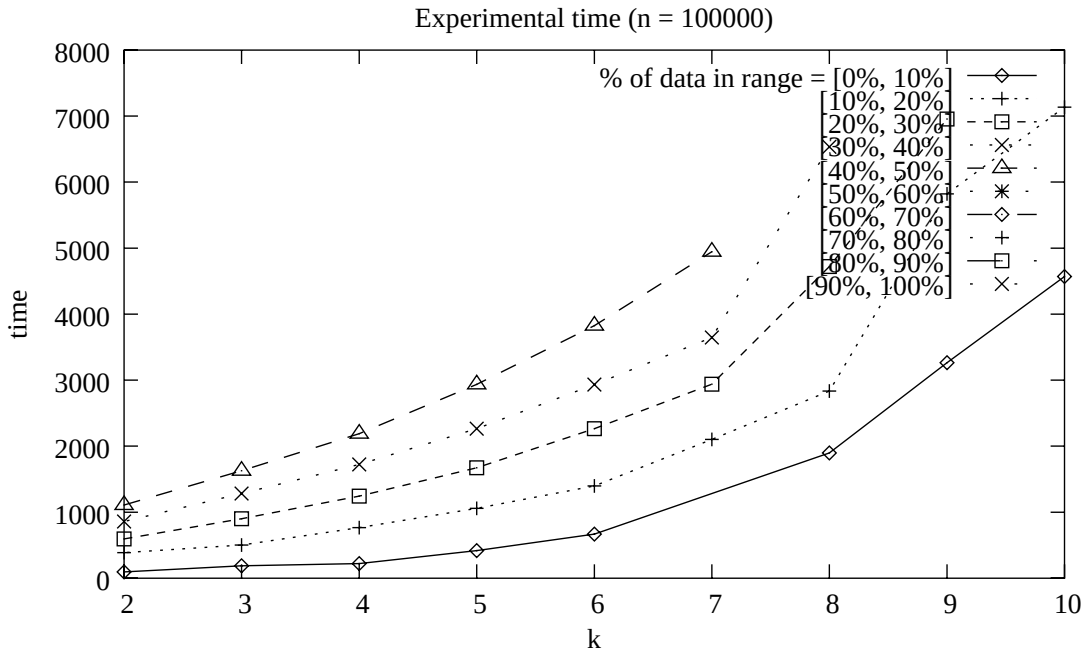

Experimental time ratio of (n=1000)/(n=100)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

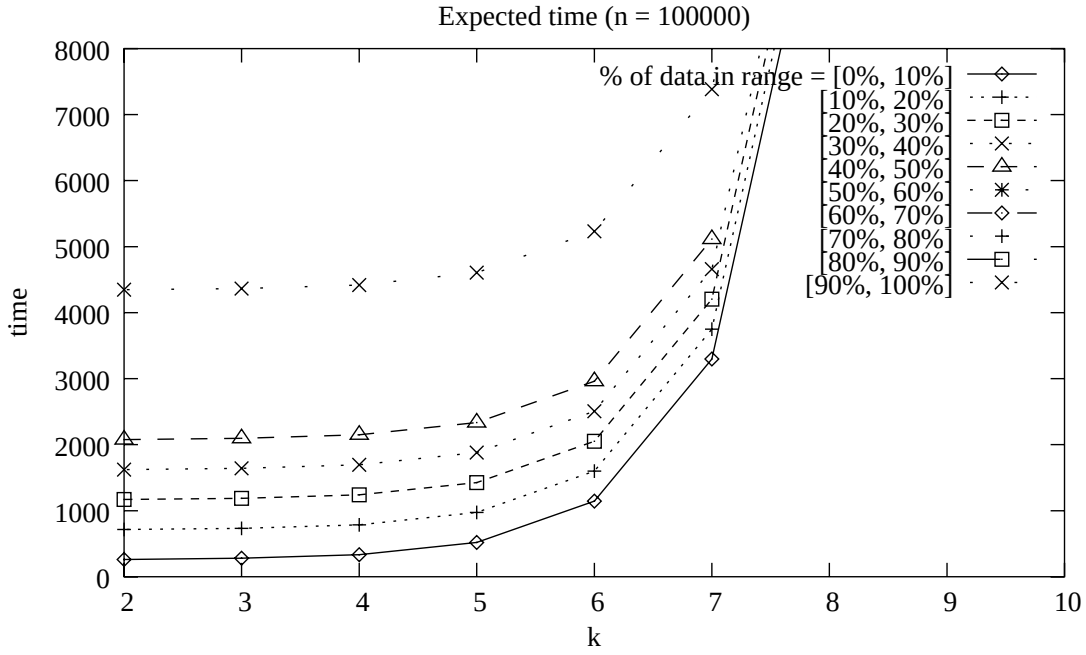

Experimental time (n = 10000)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

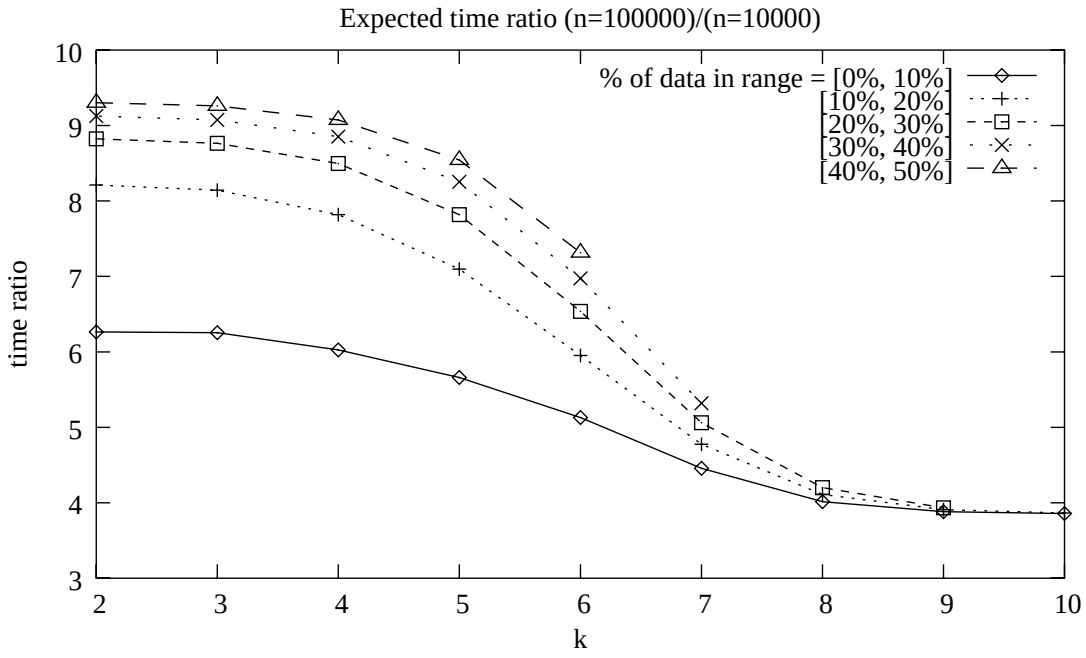

Expected time (n = 10000)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

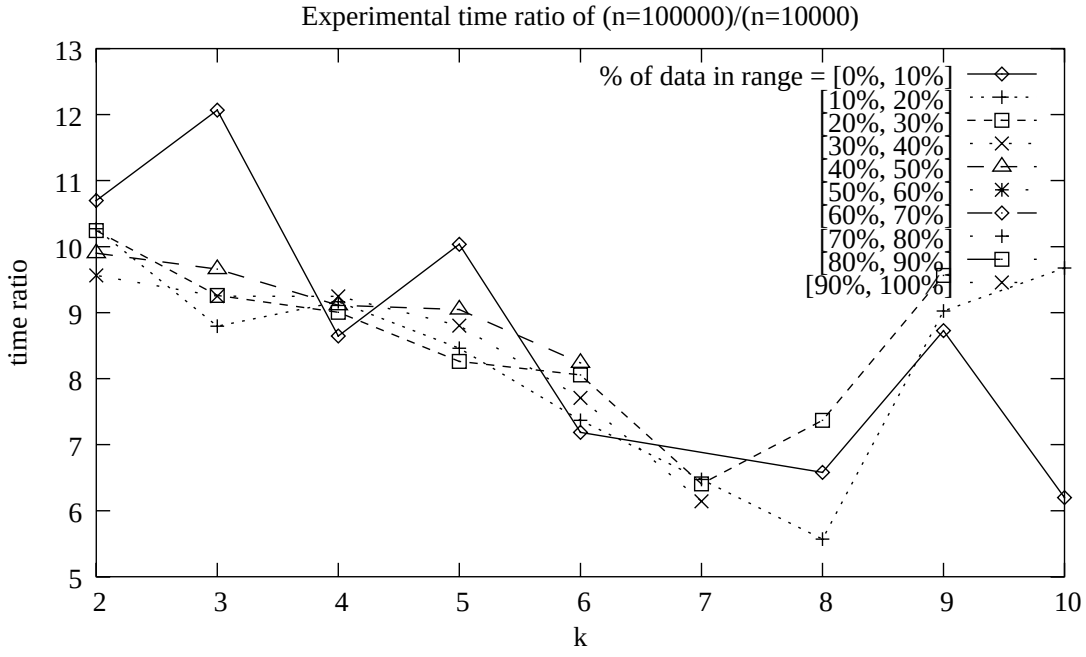

Expected time ratio (n=10000)/(n=1000)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

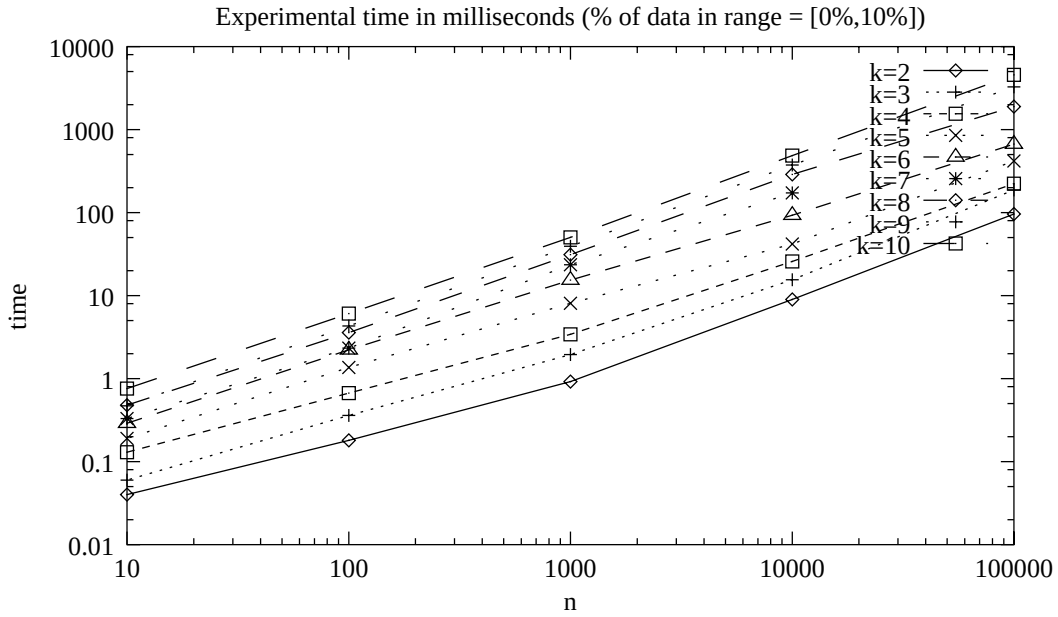

Experimental time ratio of (n=10000)/(n=1000)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

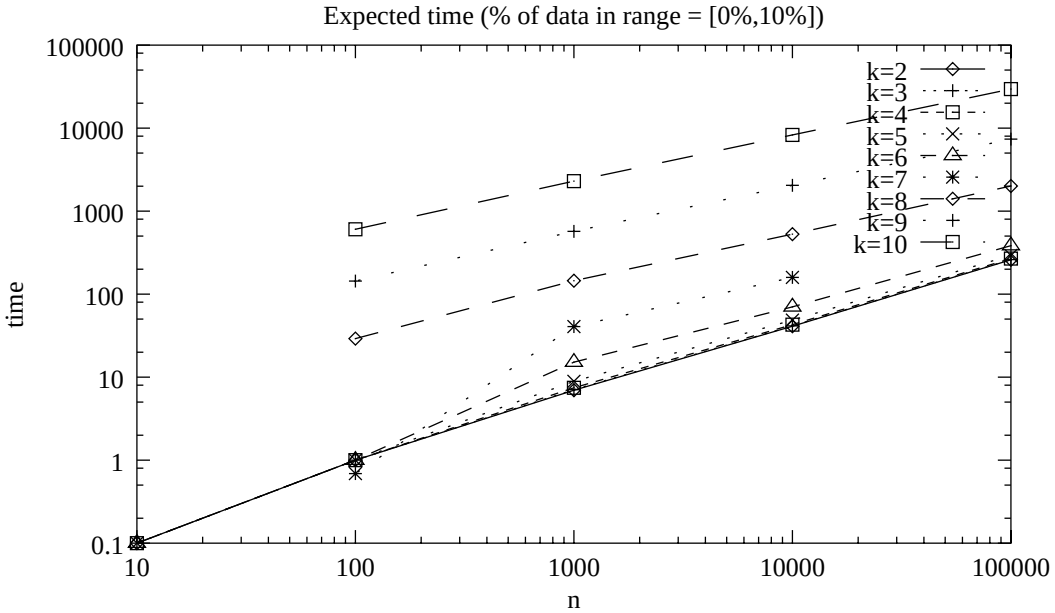

Experimental time (n = 100000)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

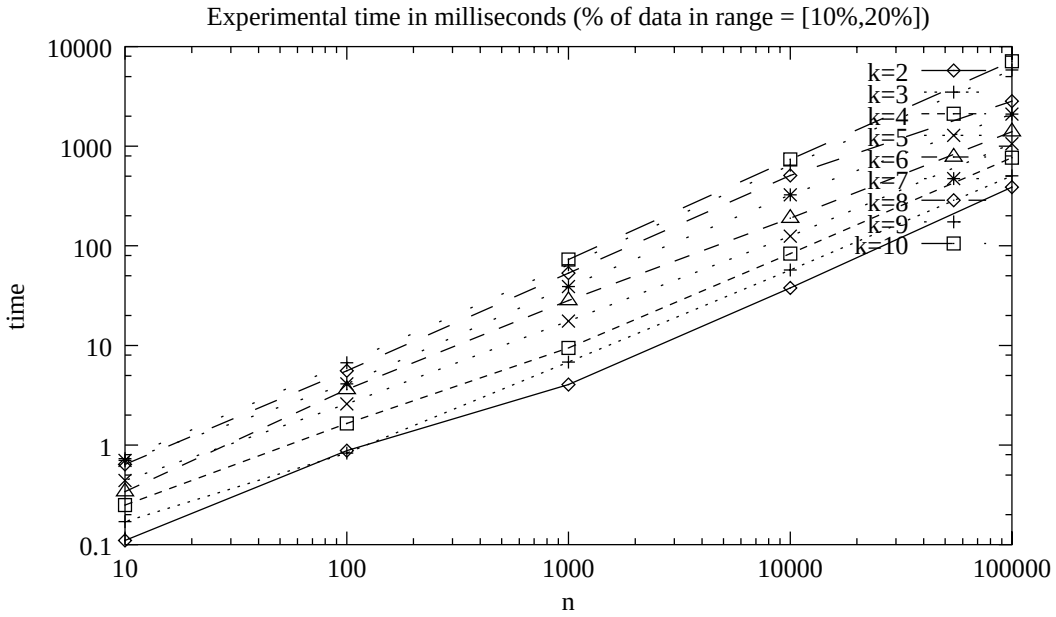

Expected time (n = 100000)
time
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

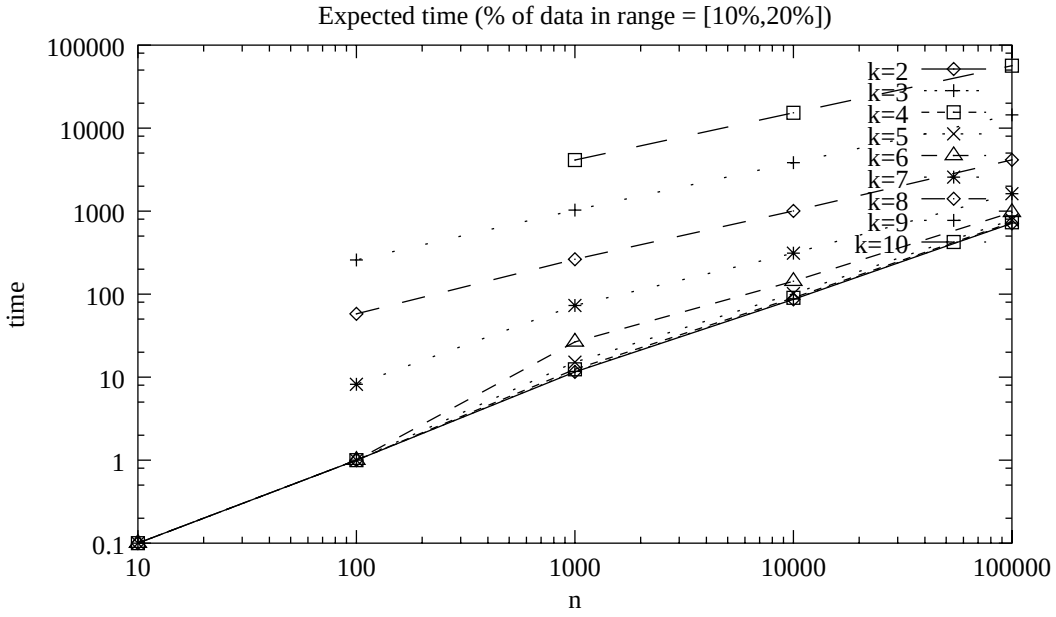

Expected time ratio (n=100000)/(n=10000)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]

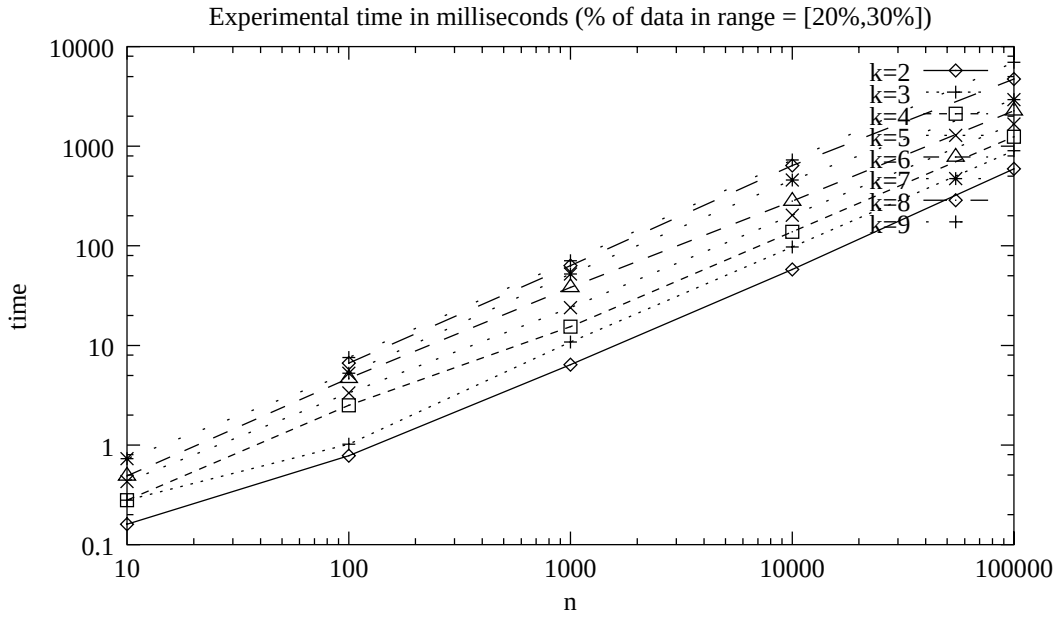

Experimental time ratio of (n=100000)/(n=10000)
time ratio
k
% of data in range = [0%, 10%]
[10%, 20%]
[20%, 30%]
[30%, 40%]
[40%, 50%]
[50%, 60%]
[60%, 70%]
[70%, 80%]
[80%, 90%]
[90%, 100%]

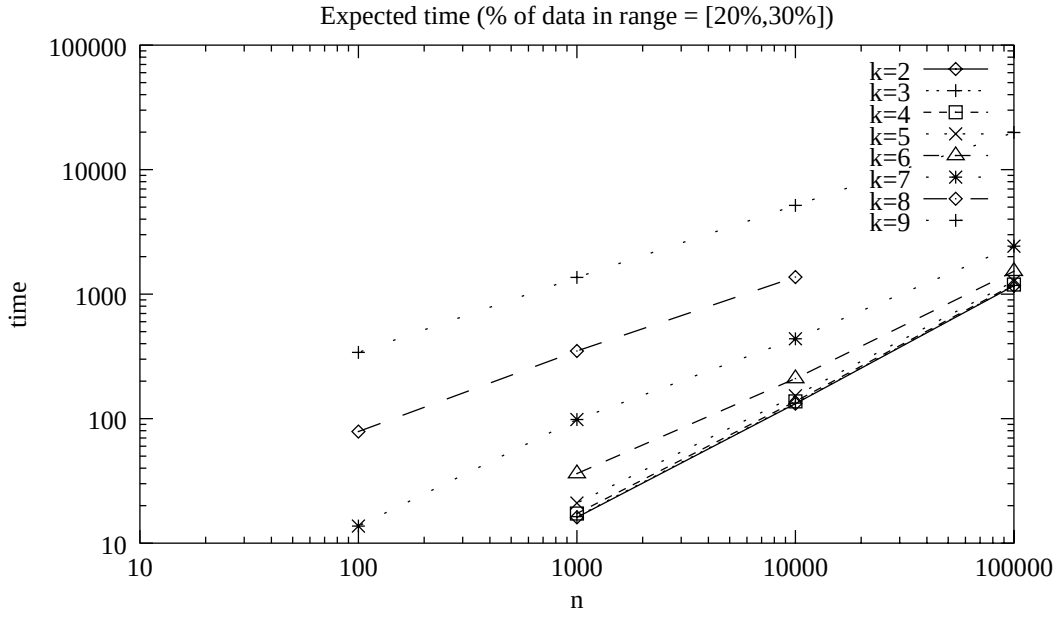

Experimental time in milliseconds (% of data in range = [0%,10%])
time
n
10000
1000
100
10
1
0.1
0.01
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
k=10

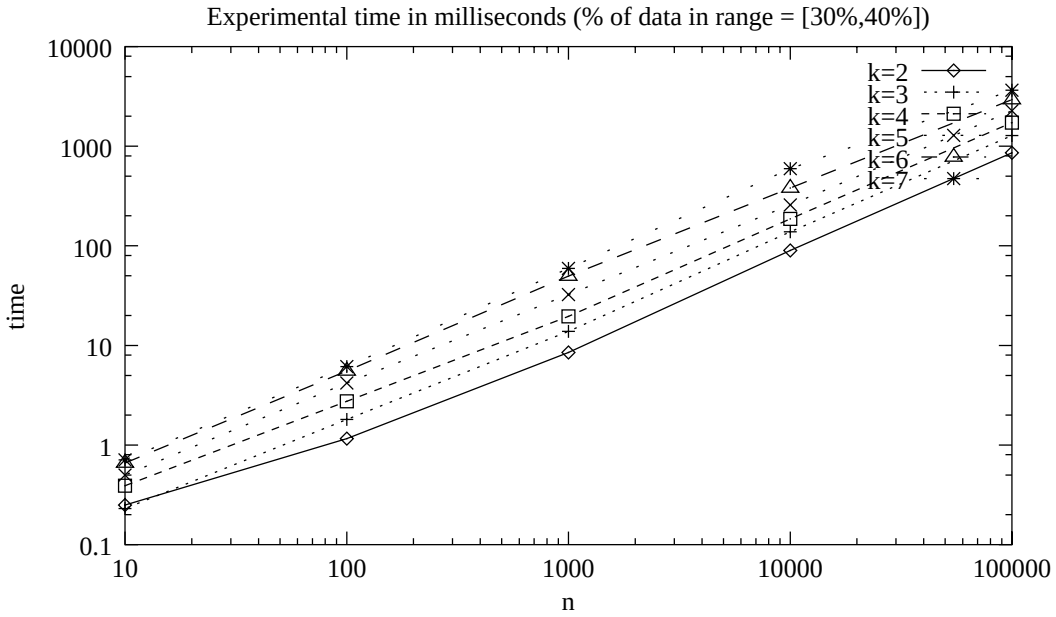

Expected time (% of data in range = [0%,10%])
time
n
100000
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
k=10

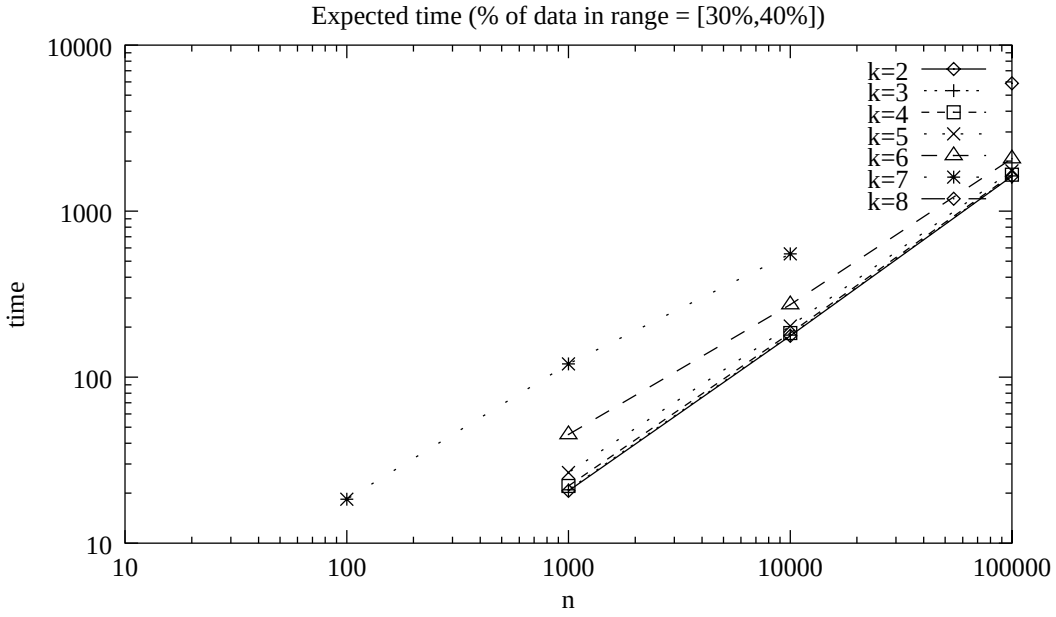

Experimental time in milliseconds (% of data in range = [10%,20%])
time
n
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
k=10

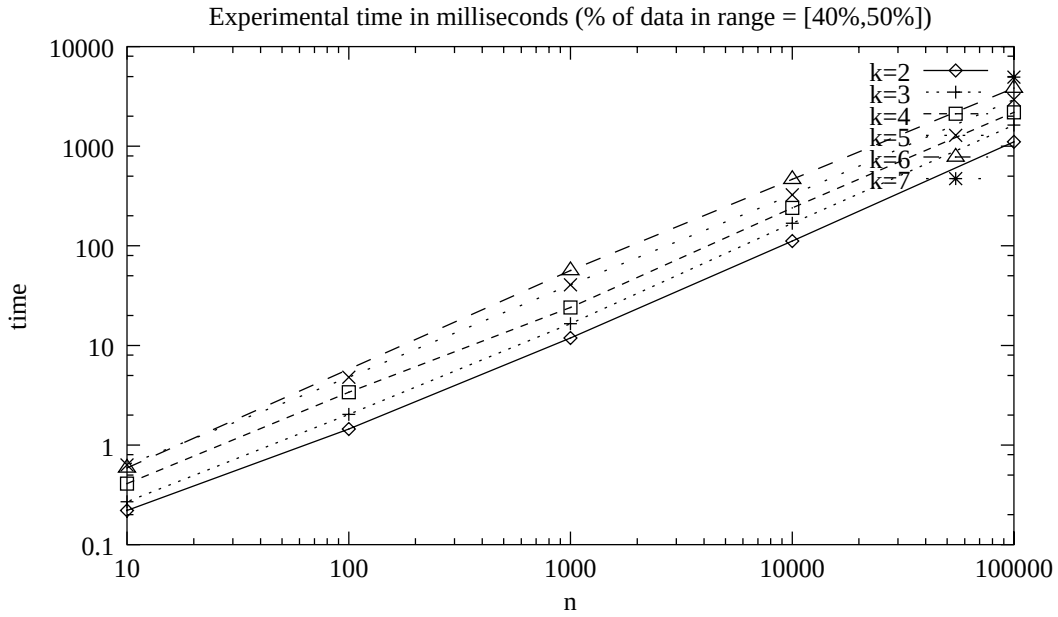

Expected time (% of data in range = [10%,20%])
time
n
100000
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
k=10

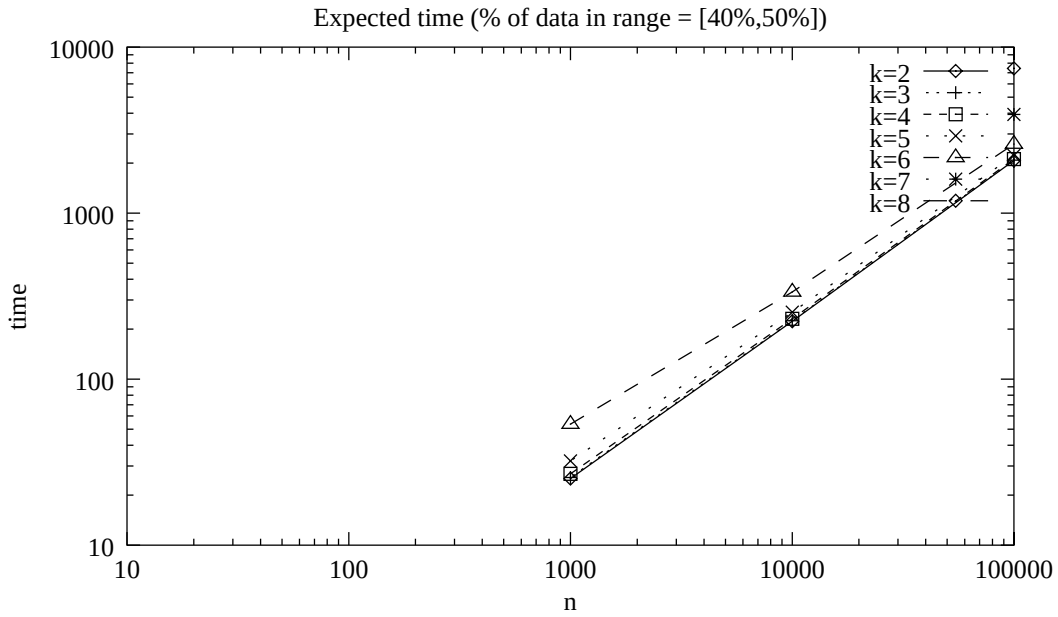

Experimental time in milliseconds (% of data in range = [20%,30%])
time
n
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9

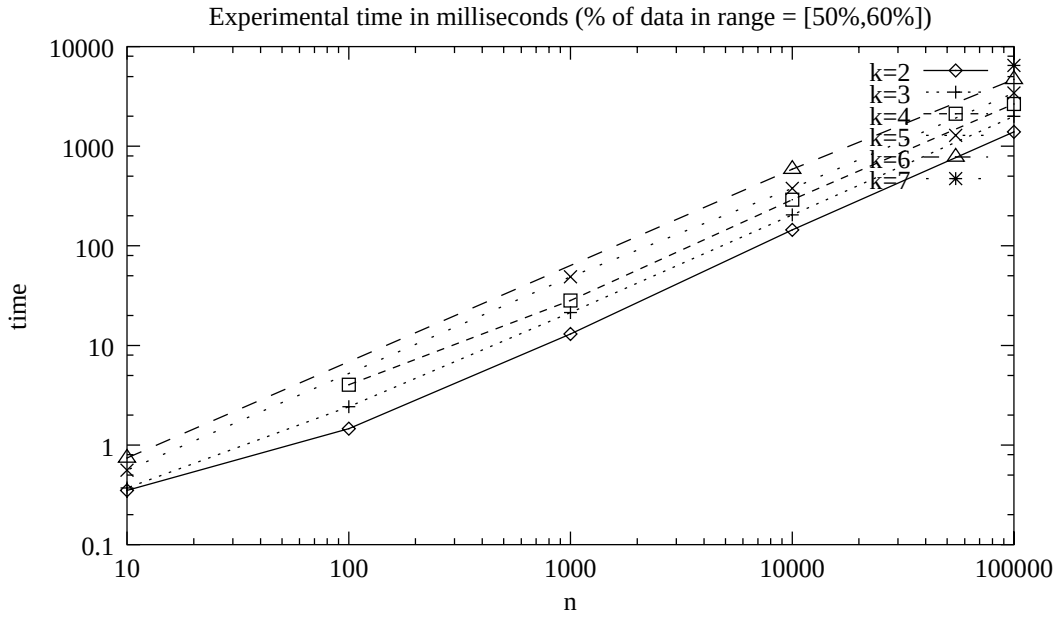

Expected time (% of data in range = [20%,30%])
time
n
100000
10000
1000
100
10
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9

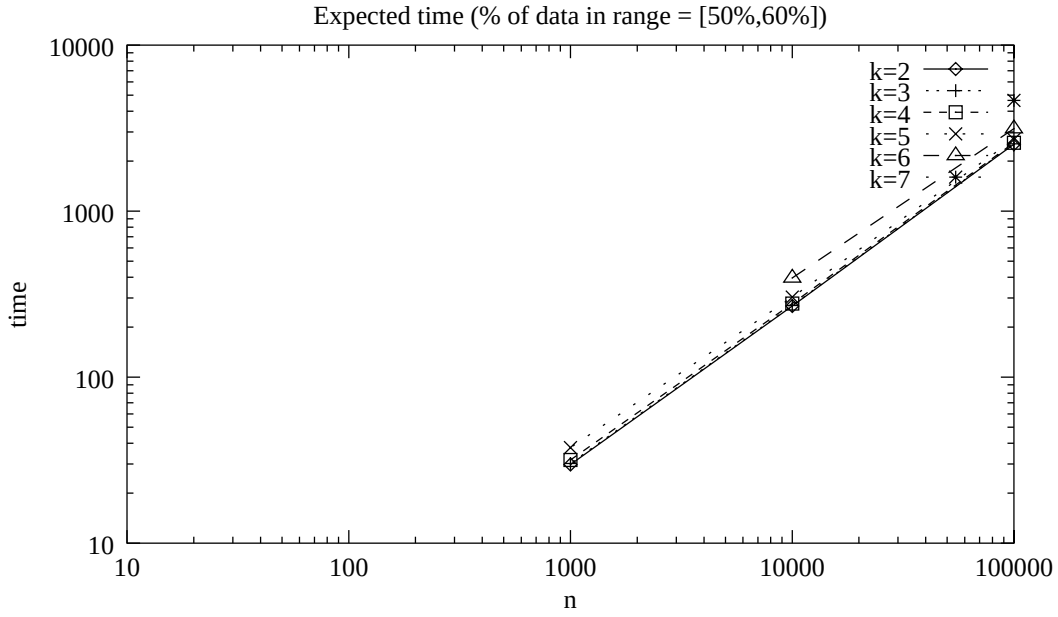

Experimental time in milliseconds (% of data in range = [30%,40%])
time
n
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7

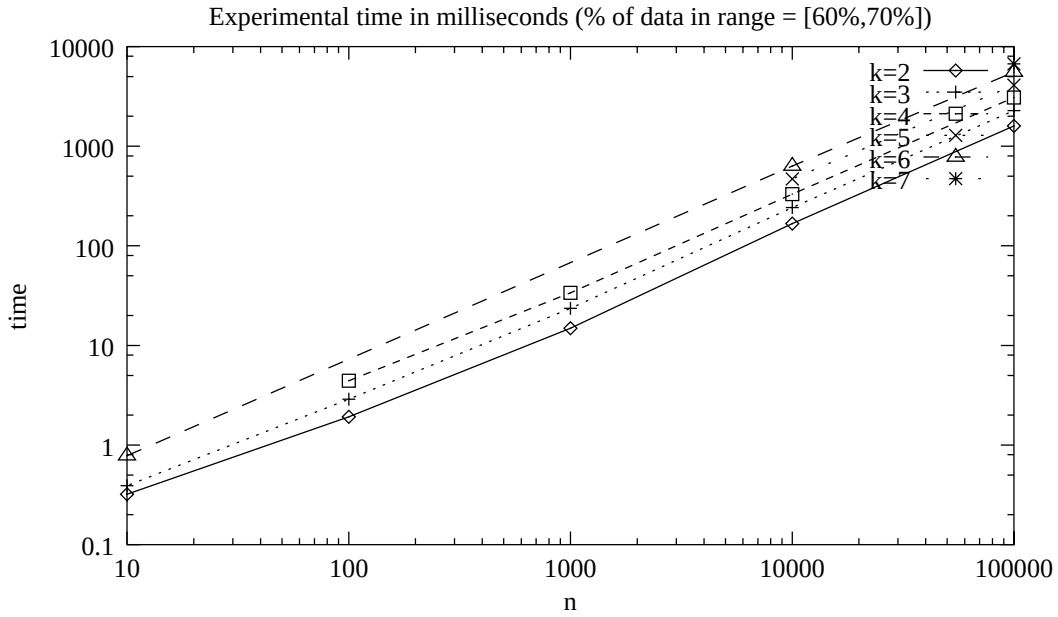

Expected time (% of data in range = [30%,40%])
time
n
10000
1000
100
10
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8

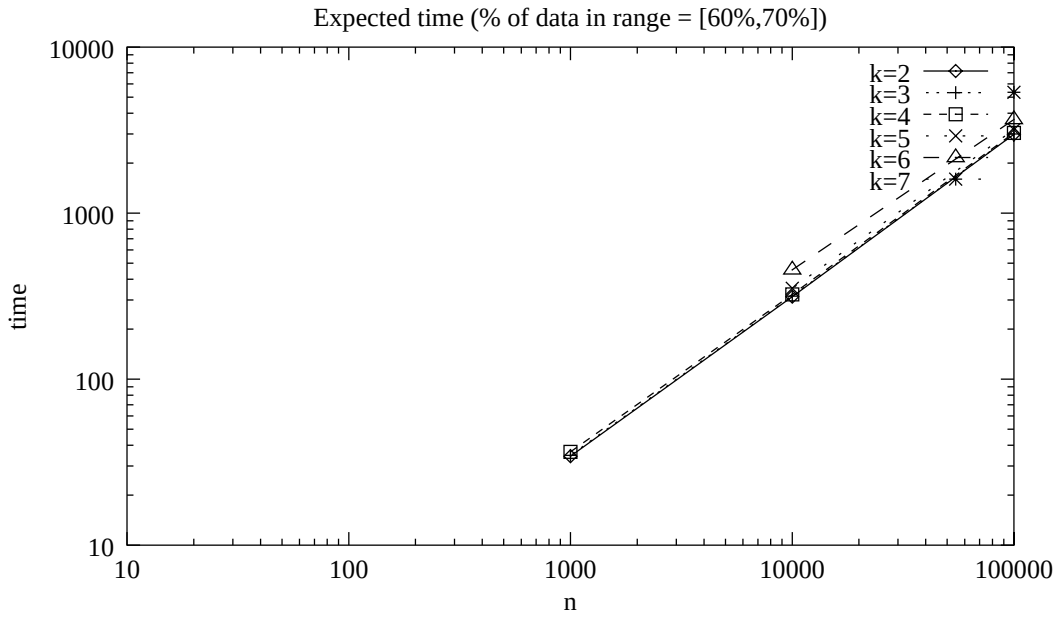

Experimental time in milliseconds (% of data in range = [40%,50%])
time
n
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7

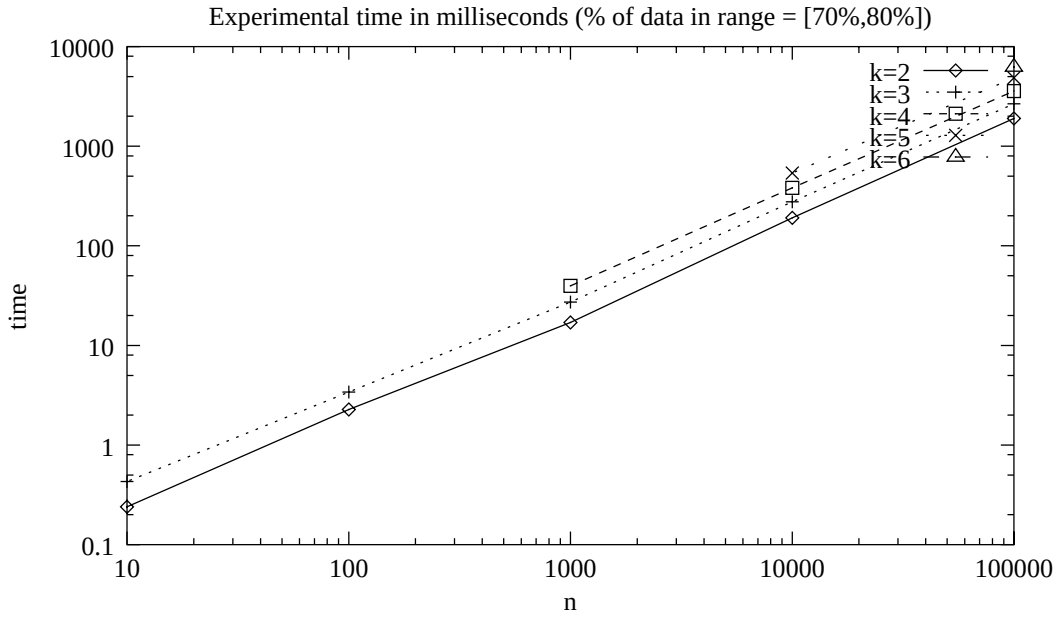

Expected time (% of data in range = [40%,50%])
time
n
10000
1000
100
10
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7
k=8

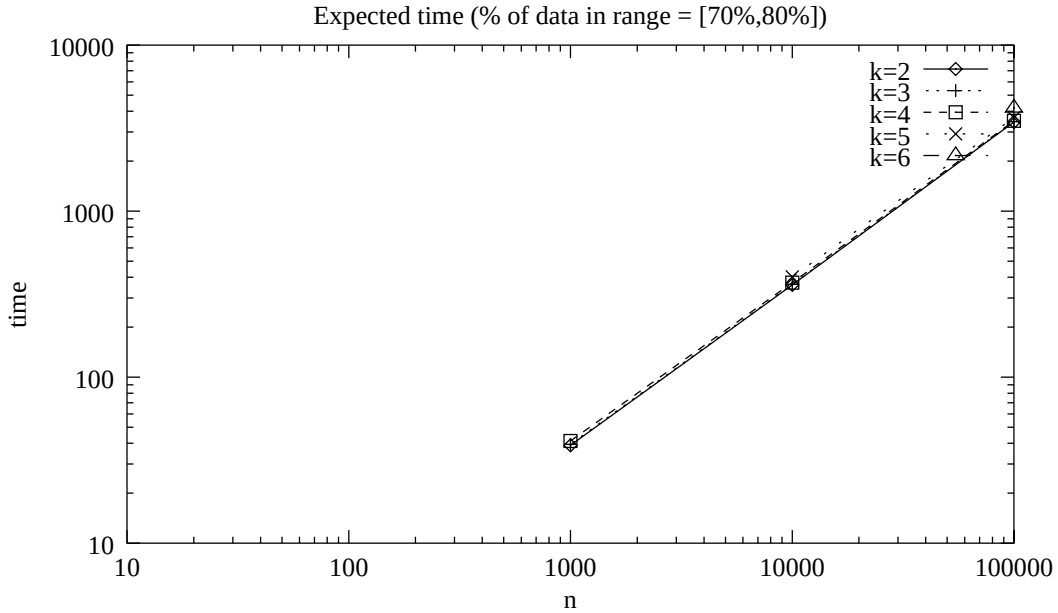

Experimental time in milliseconds (% of data in range = [50%,60%])
time
n
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7

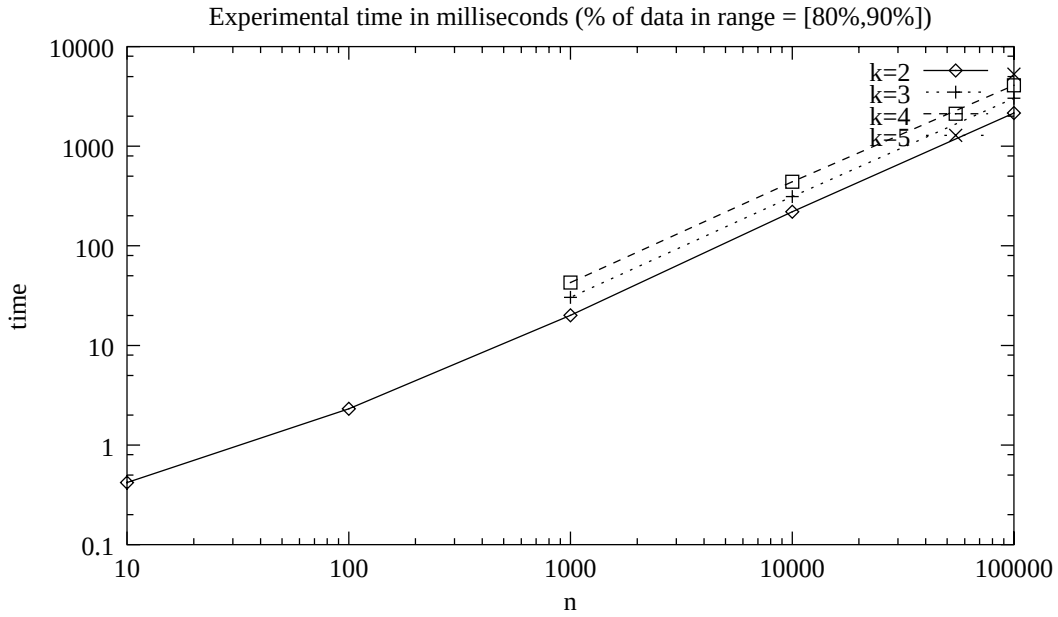

Expected time (% of data in range = [50%,60%])
time
n
10000
1000
100
10
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7

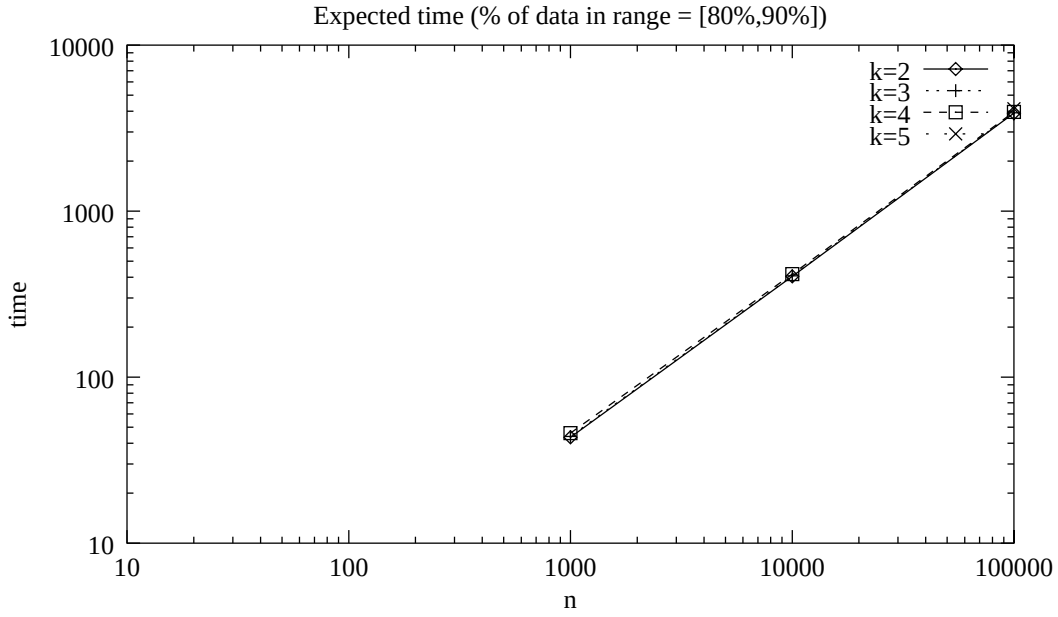

Experimental time in milliseconds (% of data in range = [60%,70%])
time
n
10000
1000
100
10
1
0.1
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7

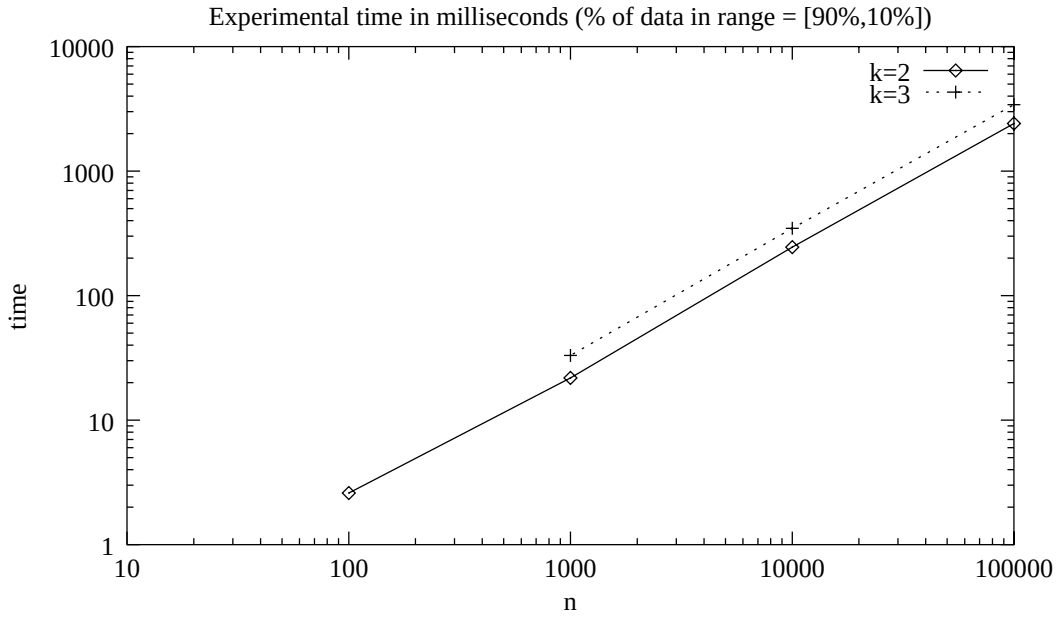

Expected time (% of data in range = [60%,70%])
time
n
10000
1000
100
10
10
100
1000
10000
100000
k=2
k=3
k=4
k=5
k=6
k=7

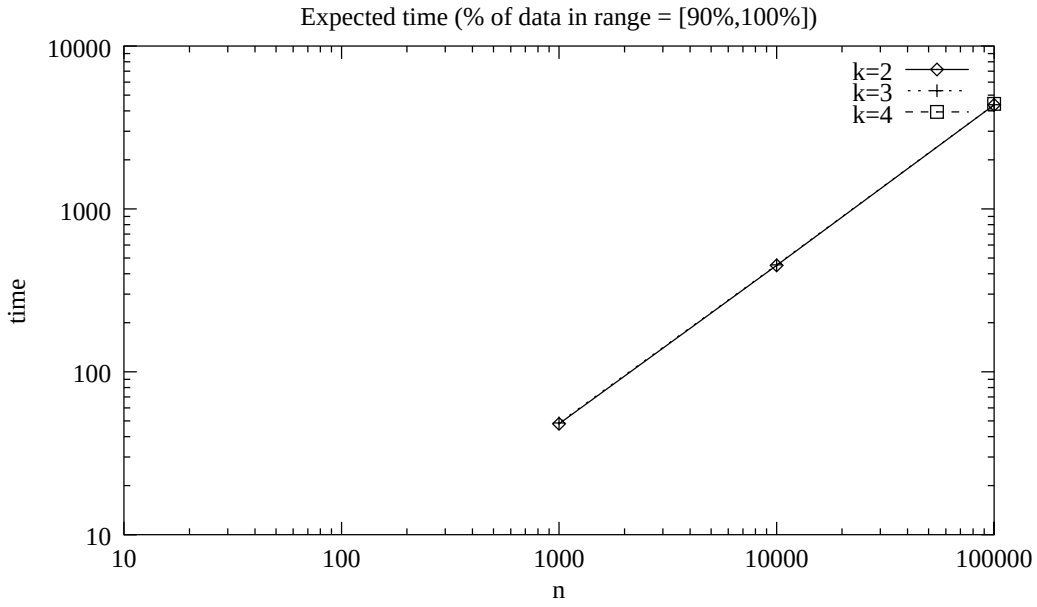

Experimental time in milliseconds (% of data in range = [70%,80%])
time
n
k=2
k=3
k=4
k=5
k=6
10000
1000
100
10
1
0.1
10
100
1000
10000
100000


Expected time (% of data in range = [70%,80%])
time
n
k=2
k=3
k=4
k=5
k=6
10000
1000
100
10
10
100
1000
10000
100000


Experimental time in milliseconds (% of data in range = [80%,90%])
time
n
k=2
k=3
k=4
k=5
10000
1000
100
10
1
0.1
10
100
1000
10000
100000


Expected time (% of data in range = [80%,90%])
time
n
k=2
k=3
k=4
k=5
10000
1000
100
10
10
100
1000
10000
100000


Experimental time in milliseconds (% of data in range = [90%,10%])
time
n
k=2
k=3
10000
1000
100
10
1
10
100
1000
10000
100000


Expected time (% of data in range = [90%,100%])
time
n
k=2
k=3
k=4
10000
1000
100
10
10
100
1000
10000
100000

Vita

Candidate's full name: Lingke Bu

University attended: MEng. (Computer Science and Engineering), 2000
Nanjing University of Aeronautics and Astranautics
Nanjing, P. R. China

BEng. (Computer Science and Engineering), 1995
Nanjing University of Aeronautics and Astronautics
Nanjing, P. R. China

Publications:

1. Bradford G. Nickerson, Lingke Bu, "k-dimensional orthogonal range search with tries", Faculty of Computer Science Technical Report TR 02-154, University of New Brunswick, Fredericton, N.B., Canada, April, 2002.

2. Lingke Bu, *Research and Implementation of a storage-system of spatial databases based on the "Realms" and main memory database techniques*, Master's Degree Thesis, Nanjing University of Aeronautics and Astronautics, Mar. 2000.