2014

Algorithmic Aspects of Some Problems in Computational Biology

Md. Shafiul Alam
University of Windsor

Algorithmic Aspects of Some Problems in Computational Biology

by

Md. Shafiul Alam

A Dissertation

Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

Algorithmic Aspects of Some Problems in Computational Biology

by

Md. Shafiul Alam

APPROVED BY:

D. Rappaport, External Examiner
Queen's University

R. Barron
Department of Mathematics & Statistics

A. Jaekel
School of Computer Science

A. Ngom
School of Computer Science

A. Mukhopadhyay, Advisor
School of Computer Science

05 December, 2013

Declaration of Co-Authorship / Previous Publication

I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows: This thesis incorporates the outcome of a joint research undertaken in collaboration with Dr. A. Sarker under the supervision of my thesis supervisor Professor A. Mukhopadhyay. The collaboration is covered in Chapter 3 of the thesis. In that investigation the key ideas, primary contributions, etc. were performed by myself working under my thesis superviosr Professor A. Mukhopdhyay. The contribution of co-author Dr. A. Sarker was primarily a theorem for determing the total number of layer graphs for a cycle. But the theorem is not included in this dissertation. It is used to confirm the total number of layer graphs for different cycles.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Declaration of Previous Publication

This thesis includes some materials from 2 original papers that have been previously published /submitted for publication in peer reviewed conferences/journals, as follows:

Thesis Chapter	Publication Title/Full Citation	Publication Status
Chapter 3	Alam, M. S., Mukhopadhyay, A. and Sarker, A. (2009). Generalized jewels and the point placement problem. In *Proceedings of the 21st Canadian Conference on Computational Geometry*, University of British Columbia, Vancouver, Canada, August 17-18, 2009, 45-48.	Published
Chapter 3	Alam, M. S. and Mukhopadhyay, A. (2010) A new algorithm and improved lower bound for point placement on a line in two rounds. In *Proceedings of the 22nd Canadian Conference on Computational Geometry, 2010 (CCCG 2010)*, University of Manitoba, Winnipeg, MB, Canada, August 9 -11, 2010, 229-232.	Published

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s)

in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other university or institution.

Abstract

Given a sequence of pairs of numbers $(a_i, l_i), i = 1, 2, ..., n$, with $l_i > 0$, and another pair of numbers L and U, the length-constrained maximum density segment problem is to find a subsequence $[a_i, a_j]$ whose density $\sum_{s=i}^{j} a_s / \sum_{s=i}^{j} l_s$ is the maximum under the constraint $L \leqslant \sum_{s=i}^{j} l_s \leqslant U$. It has application to DNA sequence analysis in Computational Biology, particularly in the determination of the percentage of CG contents in a DNA sequence. A linear time geometric algorithm is presented that is more powerful than the existing linear time algorithms.

The method is extended to solve the k length-constrained maximum density segments problem in $O(nk), O((n+k)\lg^2(U-L))$ and $O(n(U-L))$ time when $k \in O(\lg^2(U-L))$, $k \in \omega(\lg^2(U-L)) \cap o(n(U-L)/\lg^2(U-L))$ and $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$ respectively. Previously, there was no known algorithm with non-trivial time complexity for this problem. We present a linear time algorithm to solve the length-constrained maximum sum segment problem. It is extended to solve the k length-constrained maximum sum segments problem in $O(n+k)$ time. The algorithms are extended to solve the problem of finding all the length-constrained segments satisfying user specified sum or density lower bound in $O(n+h)$ time, where h is the size of the output.

The point placement problem is to determine the positions of a linear set of points uniquely up to translation and reflection from the fewest possible distance queries between

pairs of points. The motivation comes from a problem known as the restriction site mapping. If the points are necessarily distinct the lower bound and the upper bound for 2 rounds are $17n/16$ and $4n/3 + O(\sqrt{n})$ respectively, where n is the number of points. We present 2-round algorithms with queries $10n/7 + O(1), 4n/3 + O(1)$ and $9n/7 + O(1)$ respectively. The lower bound for 2 rounds is improved from $17n/16$ to $9n/8$.

We also present a modification of a geometric method called MSPocket for detection of ligand binding sites on protein surfaces. Experimentation using 48 benchmark dataset of bound protein structures shows that the success rate of our method is slightly better than that of MSPocket.

Dedication

Dedicated to my parents Md. Khorshed Alam and Rokeya Begum, my son Md. Ashraful Alam, and my daughters Shaima Alam and Tasnim Alam.

Acknowledgements

I would like to express my gratitude and sincere thanks to my supervisor Dr. A. Mukhopadhyay, Professor, School of Computer Science, University of Windsor for his continuous guidance, valuable advices and patience throughout the creation of this dissertation and throughout my doctoral studies at the University of Windsor. Without his help I could not have completed this work. I would also like to express my gratitude to Dr. R. M. Barron, Professor, Department of Mathematics & Statistics, University of Windsor, Dr. D. Rappaport, Professor, School of Computing, Queen's University, Dr. A. Ngom, Associate Professor, School of Computer Science, University of Windsor and Dr. A. Jaekel, Professor, School of Computer Science, University of Windsor for their time and valuable suggestions.

I would like to thank my wife Amena Akter, my son Md. Ashraful Alam, and my daughters Shaima Alam and Tasnim Alam for their love, understanding and cooperation.

Contents

Chapter 1

Introduction

1.1 The Problems of This Study

Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA) and protein are the three essential macromolecules of all living cells. DNA is made of nucleotides (Figure 1.1). It does not usually exist as a single molecule in living organisms. Instead, it exists in pair, which are held together tightly in the form of a double helix.

A nucleotide consists of a sugar, a phosphate group and one of a set of 4 nucleobases. The nucleobase is attached to the sugar. The nucleobases are adenine, cytosine, guanine and thymine (or uracil). They are represented by the four letters A, C, G and T (or U). The nucleotides are linked in a chain to form the DNA. Consequently, DNA can be specified as a sequence of nucleobases. In fact, it is written as a sequence of letters representing the nucleobases. It is called the primary structure of DNA. DNA has secondary and tertiary structures as well. DNA contains information which is used by a living cell to manufacture proteins. A triplet of consecutive nucleobases corresponds to a specific amino acid. The triplet is called a codon. More than one codon can correspond to an amino acid; but one

codon can correspond to exactly one amino acid. The sequence of nucleobases of a DNA is translated into a sequence of amino acids.

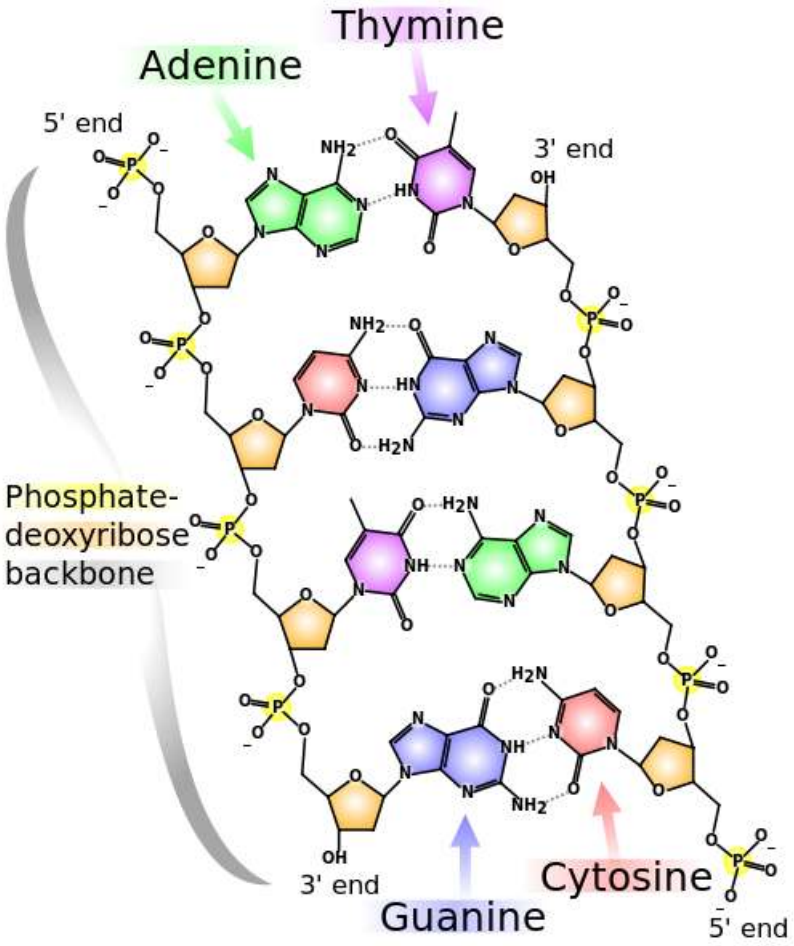


Figure 1.1: DNA molecular structure [Figure from Wikipedia: http://upload.wikimedia.org/wikipedia/commons/e/e4/DNA˙chemical˙structure.svg].

Different composition of DNA sequence is associated with different properties of DNA. For example, genes are found in most of the cases in GC-richest regions of a DNA sequence. DNA sequences are analyzed to find biologically significant regions. In this dissertation we propose some algorithms for sequence analysis.

Molecules are three-dimensional entities. Their functions often depend on their three-dimensional structures. The energy associated with each set of possible atomic positions create intramolecular motions in large and complicated molecules such as proteins. The set of all conformations generated by the intramolecular motion is called the conformational space of the molecule. The difference between the energies of two conformations is due to various nonbonded interactions. The relation between those interactions and the conformational state is very important in molecular biology.

It is essential to determine the conformational space of a molecule from experimental data about its conformational state. The data are obtained by x-ray crystallography, nuclear magnetic resonance (NMR) spectroscopy, etc. While x-ray crystallography can produce the conformational space effectively, there are problems with this technique. For example, many macromolecules does not form good crystals. Much of the experimental data, about the conformational state of molecules, obtained by other methods, and the majority of energy functions can be expressed in terms of intermolecular distances. The conformational space can be described in terms of interatomic distances. The approach that determines the conformational space from the intermolecular distance data and chirality constraints is called distance geometry approach [25, 42]. Distance geometry can also be used to evaluate the effectiveness of other methods for determining molecular conformation space.

When the distances lie in between prescribed bounds the problem is called a bound embedding problem, otherwise it is called a distance embedding problem which is a spe-

cial case of the bound embedding problem. The one-dimensional version of the distance embedding problem without chirality constraint is the point placement problem on a line.

A polypeptide is a linked chain of amino acids. A protein consists of one or more polypeptides. Usually a protein folds into globular or fibrous form that facilitates a biological function 1.2. Proteins participate in almost all functions of a cell. Many proteins are enzymes and work as a catalytic agent in metabolism. Proteins also perform mechanical functions, cell signaling, immune responses, etc. Proteins can also work together to perform a task. Functions of proteins are the results of their interactions with other molecules. The interactions usually occur in the concave regions on the surface of a protein.

Like other molecular surfaces, protein surface is the solvent excluded surface, called Connolly surface. It is a purely geometric feature. Connolly [24] proposed a purely geometric method to compute it. Sanner *et al.* [60] proposed an improved method to calculate it. His method is also based on geometry. Consequently, geometry plays a major role in making a pocket suitable for ligand binding site. Among the best performing methods are those methods that uses purely geometric features of the pocket to predict the ligand binding site. We study one such method called MSPocket [75].

Geometry is involved in all the problems, directly or indirectly (by transformation into a geometric problem), that we study in this dissertation. Techniques of computational geometry can play a big role in solving these problems and other problems in structural biology involving geometry. The problems are described in the following subsections.

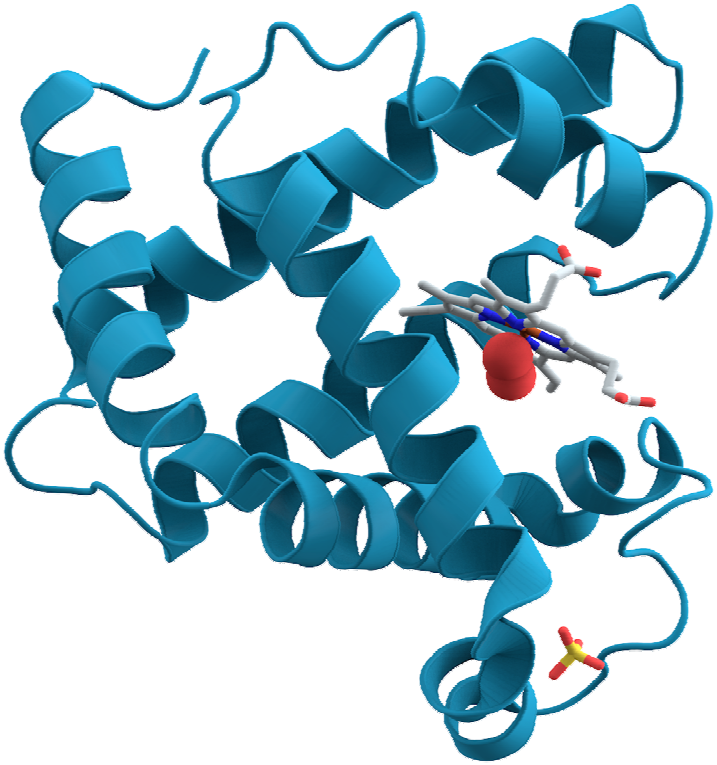


Figure 1.2: Myoglobin protein [Figure from Wikipedia: http://en.wikipedia.org/wiki/Protein].

1.1.1 Sequence Analysis

Let A be a sequence (a_i, l_i) $(i = 1, ..., n)$ of n ordered pairs of real numbers a_i $(i = 1, ..., n)$ called values and positive real numbers $l_i > 0$ $(i = 1, ..., n)$ called lengths, and L and U be two positive real parameters $0 < L \leq U$. A segment, denoted by $A[i, j], 1 \leq i \leq j \leq n$, of A is a consecutive subsequence of A between the indices i and j. The length $l[i, j]$, sum $s[i, j]$ and density $d[i, j]$ of a segment $A[i, j]$ are $l[i, j] = \sum_{t=i}^{j} l_t$, $s[i, j] = \sum_{t=i}^{j} a_t$ and

$d[i,j] = \frac{\sum_{t=i}^{j} a_t}{\sum_{t=i}^{j} l_t}$ respectively. $A[i,j]$ is feasible if $L \leq l[i,j] \leq U$. The length-constrained maximum sum segment problem is to find a feasible segment of maximum sum. The k length-constrained maximum sum segments problem is to find k feasible segments such that their sums are the k largest. The length-constrained maximum density segment problem is to find a feasible segment of maximum density. The k length-constrained maximum density segments problem is to find k feasible segments such that their densities are the k largest.

We present algorithms for length-constrained maximum sum segment and maximum density segment problems, in particular, and the problems of finding length-constrained heaviest segments, in general, for a sequence of real numbers. The length-constrained maximum density segment problem and the k length-constrained maximum density segments problem have been transformed into geometric slope selection problems.

The algorithms have potential applications in different areas of biomolecular sequence analysis, including finding CG-rich regions, TA and CG-deficient regions, CpG islands and regions rich in periodical three-base patterns, post processing sequence alignment, annotating multiple sequence alignments, and computing length constrained ungapped local alignment. They have applications in other areas also, such as pattern recognition, digital image processing and data mining [1, 34, 35].

1.1.2 Point Placement Problem

The point placement problem is to determine the positions of a set of n distinct points, $P = \{p_1, p_2, p_3, \ldots, p_n\}$, on a line uniquely, up to translation and reflection, from the fewest

possible distance queries between pairs of points. This problem is closely related to the restriction site mapping problem in DNA sequence and turnpike problem in computational geometry. In the latter problems, the points are deduced from a set of interpoint distances between unlabeled points. While the distances in point placement problem are between labeled points. The point placement problem has application in the graph embedding problem.

The higher dimensional version of the point placement problem has application in the area of molecular conformation. In both the cases the interpoint distances of labeled points are specified. The molecular configuration of a rigid molecule is unique upto translation. It is not unique upto reflection. The same interatomic distances of a molecule can also occur in its mirror images. So, in order to uniquely determine the configuration of a rigid molecule, one needs to specify the interatomic distances and the chirality of a single asymmetric centre. This problem is called a distance embedding problem. On the other hand, solution to the three-dimensional version of the point placement problem is unique upto translation and reflection. So, the distances can uniquely determine the positions. If the distances are the interatomic distances of a rigid molecule, then the solution to the point placement problem will be the configuration of either the molecule or its mirror image.

If the molecule is mobile then the distances lie in between prescribed bounds. For this case the inputs are the lower and upper bounds on the interatomic distances and the chirality of quadruples of atoms. The problem is to determine the conformation space of

the molecule. This problem is called a bound embedding problem.

1.1.3 Ligand Binding

Molecular surface is a 3D Euclidean surface. Ligand binding sites are situated at dents on this surface. Consequently, geometry plays a major role in making a pocket suitable for ligand binding site. Among the best performing methods for predicting ligand binding site are those methods that uses purely geometric features of the pocket.

1.2 Contributions

The main contributions of this dissertation are described below:

- Sequence analysis problems: For the maximum sum segment problem with non-uniform length there is an algorithm with time and space complexities in $O(n)$. An algorithm with time complexity in $O(n)$ and space complexity in $O(\frac{U-L}{l_{min}})$ is presented in this dissertation. For the maximum density segment problem with non-uniform length there is a combinatorial solution with time complexity in $O(n)$ and space complexity in $O(\frac{U}{l_{min}})$. We present a simple geometric algorithm with the same time complexity and $O(\min\{\frac{U-L}{l_{min}}, \frac{L}{l_{min}}\})$ space complexity.

  The algorithms are extended to respectively solve the k length-constrained maximum sum segments problem in $O(n+k)$ time, and the k length-constrained maximum density segments problem in $O(nk)$, $O((n+k)(\lg(U-L))^2)$ and $O(n(U-L))$ time when $k \in O(\lg^2(U-L))$, $k \in \omega(\lg^2(U-L)) \cap o(n(U-L)/\lg^2(U-L))$ and $k \in$

$\Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$ respectively. They are extended to find all the length-constrained segments satisfying user specified sum or density lower bound in $O(n+h)$ time, where h is the size of the output. Previously, there was no known linear time algorithm for these problems. We indicate the extensions of our algorithms to higher dimensions.

- Point placement problem: A 2-round algorithm is presented which solves the point placement problem with $9n/7 + O(1)$ queries, where n is the number of points. The lower bound on 2-round algorithms is improved from $17n/16$ to $9n/8$. This improves the current best results for 2-round algorithm reported in [20].

- Ligand binding: We study a geometric method called MSPocket [75] for the detection of ligand binding site. It is one of the few best performing method for predicting ligand binding site. In this dissertation MSPocket is modified by replacing one constraint. Experiment on a set of 48 benchmark dataset of bound proteins shows that our method has a slightly better success rate than that of MSPocket.

1.3 Structure of the Thesis

Chapter 2 deals with the problems of maximum sum/density segment problems, k-maximum sum/density segment problems and segments satisfying sum/density requirement problems. In Chapters 3 and 4 we present some improved algorithms and an improved lower bound for the point placement problem. Chapter 5 presents the study of ligand binding. Results

are summarized in Chapter 6.

Chapter 2

Maximum Density Segment

2.1 Introduction

Let A be a sequence (a_i, l_i) $(i = 1, ..., n)$ of n ordered pairs of real numbers a_i $(i = 1, ..., n)$ called values and positive real numbers $l_i > 0$ $(i = 1, ..., n)$ called lengths, and L and U be two positive real parameters $0 < L \leq U$. A *segment*, denoted by $A[i, j], 1 \leq i \leq j \leq n$, of A is a consecutive subsequence of A between the indices i and j. The *length* $l[i, j]$, *sum* $s[i, j]$ and *density* $d[i, j]$ of a segment $A[i, j]$ are $l[i, j] = \sum_{t=i}^{j} l_t$, $s[i, j] = \sum_{t=i}^{j} a_t$ and $d[i, j] = \frac{\sum_{t=i}^{j} a_t}{\sum_{t=i}^{j} l_t}$ respectively. A *feasible segment* of A is a segment $A[i, j]$ such that $L \leq l[i, j] \leq U$. The prefix sums of the sequence are defined as $s_0 = 0$ and $s_i = a_1 + a_2 + \ldots + a_i$ for $i = 1, ..., n$. s_i $(i = 1, ..., n)$ can be computed in linear time by noting that $s_i = s_{i-1} + a_i$. Once s_i's are known, $s[i, j]$ $(1 \leq i \leq j \leq n)$ can be computed in constant time since $s[i, j] = s_j - s_{i-1}$.

In this chapter we study some problems concerning the determination of length-constrained heaviest segments in a sequence of real numbers. The problems are formally described below:

Definition 2.1.1 *Let A be a sequence of pairs of real numbers (a_i, l_i), $i = 1, 2, \ldots, n$, with $l_i > 0$, and L and U be a pair of real numbers with $L \leq U$. (a) The length-constrained maximum sum segment problem is to find a segment $A[i, j]$ whose sum $s[i, j]$ is the maximum under the constraint $L \leqslant l[i, j] \leqslant U$. (b) The length-constrained maximum density segment problem is to find a segment $A[i, j]$ whose density $d[i, j]$ is the maximum under the constraint $L \leqslant l[i, j] \leqslant U$.*

The maximum sum segment problem, with uniform length ($l_i = 1$ for all i) and no restriction on segment length, was first studied by Grenander [39]. This problem arose while researching in the area of pattern recognition in digitized images. The original problem, as proposed by Grenander, was in 2-dimensions. In that setting the maximum sum subarray was an estimator for the maximum likelihood of a pattern in a digital image. He also simplified the problem to 1-dimension. The problem also has applications in other areas such as graphics, data mining [1, 34, 35] and bioinformatics [7]. An optimal linear time algorithm for the problem proposed by Kadane is described by Bentley [9] and Gries [40]. Its space complexity is $O(1)$. The two-dimensional version of the problem is to find a connected rectangular submatrix of maximum sum from a two-dimensional rectangular input matrix of real numbers [9]. Here the lengths are uniform, i.e., $l_i = 1$ for all i, and there is no restriction on the size of the submatrix. The problem has been extended to higher dimensions [70]. In higher dimensions the problem is called the maximum sum subarray problem. The higher dimensional problem has applications in the area of data mining (for dimensions less than

4) and Monte-Carlo simulation (dimensions being high) [70]. It can be solved by reducing it to 1-dimensional problems [10, 70]. For a 2-dimensional $m \times n$ matrix there are $O(m^2)$ column intervals. Each of them is solved using Kadane's linear time algorithm for maximum sum segment problem. Hence, its time complexity is $O(m^2 n)$ [10, 70]. For this case, i.e., 2-dimensions with uniform length and no restriction on length, there is a better algorithm based on a distance matrix multiplication technique [70, 69]. Its running time is subcubic.

Huang [44] introduced the restriction of length cutoff L in the setting of biomolecular sequence analysis to avoid reporting extremely short segments. He gave a linear time algorithm for computing the maximum sum segment of length at least L, but no restriction on the upper bound of its length, i.e., $U = n$. He had observed that the segments reported by the algorithm are usually much larger than L. From this observation Lin *et al.* [50] argued that the segments reported by the method may contain some poor and irrelevant segments. To avoid this they introduced the restriction of upper bound U on the length of the segment. They proposed a linear time algorithm for the problem when there is only the upper bound U on the length of the segment, but no lower bound, i.e., $L = 0$. They combined that algorithm with Huang's [44] technique to develop a linear time algorithm for arbitrary L and U. Its space requirement is also linear. Fan *et al.* [31] gave an $O(n)$ time and $O(U)$ space algorithm for the problem for the case of uniform length. In this chapter, we present an algorithm for this general problem with time complexity in $O(n)$ and space complexity in $O(U - L)$. It can be modified in a straightforward way to solve the problem

with non-uniform length in $O(n)$ time and $O(\frac{U-L}{l_{min}})$ space. We indicate the extension of this algorithm to solve the problem in higher dimensions by using the technique of reducing the problem to 1-dimension [10, 70].

The k maximum sum segments problem was introduced by Bae and Takaoka [8]. There was no restriction on the segment length. A natural extension of this problem is the k length-constrained maximal sum segments problem. The problem is defined as follows:

Definition 2.1.2 *Given a sequence A of real numbers a_i, $i = 1, 2, \ldots, n$, a pair of real numbers $L \leqslant U$ and an integer k such that $1 \leqslant k \leqslant (n-U+1)(U-L+1)+\frac{1}{2}(U-L)(U-L+1)$, the k length-constrained maximum sum segments problem is to find k segments of A of length at least L and at most U such that their sums are the k largest among all the possible segments of A of length at least L and at most U.*

When there is no restriction on segment length, i.e., $L = 0$ and $U = n$, Cheng *et al.* [18, 19] gave an $O(n + k \log(\min\{n, k\}))$ time algorithm, and Brodal and Jorgensen [13] gave an optimal $O(n+k)$ time algorithm for this. The latter algorithm constructs a partially persistent [29] binary maximum heap that implicitly contains all the $\binom{n}{2}+n$ number of sums for all possible segments in $O(n)$ time. The heap is a modified version of the self-adjusting heap of Sleator and Tarjan [64]. The k maximum sums are selected from the heap using the linear time heap selection algorithm of Frederickson [33]. Brodal and Jorgensen [13] extended their algorithm to higher dimension by using the technique of reducing the problem to 1-dimension [10, 70]. Liu and Chao [51] gave an $O(n + k)$ time and $O(n)$ space

algorithm for the k length-constrained maximum sum segments problem. Combining with the technique of Brodal and Jorgensen [13] we extend our algorithm for the maximum sum segment problem to solve the k length-constrained (i.e., arbitrary L and U) maximum sum segments problem. Its time and space complexities are in $O(n + k)$ and $O(U - L + k)$ respectively.

For the maximum density segment problem, when the lengths are uniform and there is no restriction on the segment length, the maximum element in the sequence will be the solution and it can be found in a straight forward way in $n - 1$ comparisons and $O(1)$ space. When $U = L$ the problem is trivially solvable in $O(n)$ time since there are $n - U + 1$ feasible segments. When the lengths are uniform, $U \neq L$ and no upper bound ($U \geq n - L$), Huang [44] showed that the length of the maximum density segment is at most $2L - 1$. So, this case is equivalent to the case when $U = 2L - 1$ and can be solved in $O(nL)$ using brute force method since the number of feasible segments is $O(nL)$. For this case, Lin *et al.* [50] gave an $O(n \log L)$ time algorithm by using a method of right skew decomposition of the sequence. When the lengths are uniform, and U and L are arbitrary, Goldwasser *et al.* [37] gave an $O(n)$ time algorithm. For the general case, where the lengths are not uniform and U and L are arbitrary, Goldwasser *et al.* [38] extended the right skew decomposition method of Lin *et al.* [50] to develop an $O(n)$-time and space algorithm. A combinatorial solution with time-complexity in $O(n)$ and space complexity in $O(U)$ was proposed by [21, 22]. The algorithm works in an online manner. In the same paper it was pointed out that the

linearity claim of a geometric approach by Kim [46] is flawed. Lee *et al.* [48] fixed the flaw of Kim's algorithm by exploiting the property of decomposability of tangent query. Its time and space complexities are in $O(n)$. In this chapter, we present a simple modification to Kim's algorithm to address the flaw, while retaining the simplicity, elegance and linearity of his geometric approach. Our algorithm's time and space complexities are in $O(n)$ and $O(\min\{U-L, L\})$ respectively, and it works in an online manner. [1]

The k maximum sum segments problem was introduced by Bae and Takaoka [8]. A natural extension of this problem is the k length-constrained maximal density segments problem. The problem is defined as follows:

Definition 2.1.3 *Given a sequence A of real numbers a_i, $i = 1, 2, \ldots, n$, a pair of real numbers $L \leqslant U$ and an integer k such that $1 \leqslant k \leqslant (n-U+1)(U-L+1)+\frac{1}{2}(U-L)(U-L+1)$, the k length-constrained maximum density segments problem is to find k segments of A of length at least L and at most U such that their densities are the k largest among all the possible segments of A of length at least L and at most U.*

We extend our algorithm to solve the k length-constrained maximum density segments problem in $O(nk)$, $O((n+k)(\lg(U-L))^2)$ and $O(n(U-L))$ time when $k \in O(\lg^2(U-L))$, $k \in \omega(\lg^2(U-L)) \cap o(n(U-L)/\lg^2(U-L))$ and $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$ respectively.

Huang [44] introduced the problem of finding segments of a sequence satisfying a sum

[1] The algorithm was presented at the 20th Annual Fall Workshop on Computational Geometry 2010 [4].

requirement. The content requirement is expressed as the count of equal length oligomers in biomolecular sequence. We shall call this problem as the required sum segments problem. A natural extension of this is the required density segments problem. The problems are defined as follows:

Definition 2.1.4 *Let A be a sequence of real numbers a_i, $i = 1, 2, \ldots, n$, σ and δ be a pair of real numbers, and L and U be another pair of real numbers with $L \leqslant U$. (a) The length constrained segments satisfying a sum lower bound problem is to find all the segments $A[i, j]$ such that $s[i, j] \geq \sigma$. (b) The length constrained segments satisfying a density lower bound problem is to find all the segments $A[i, j]$ such that $d[i, j] \geq \delta$.*

For the former problem when there is only a lower bound on the length of the sequence and no upper bound on its length, Huang [44] gave a linear time algorithm for a related problem using dynamic programming technique. His algorithm finds all the optimal segments of length at least L satisfying a sum lower bound. Modifying the technique of Liu and Chao [51], we solve both the length-constrained segments satisfying a sum lower bound problem and the length-constrained segments satisfying a density lower bound problem in $O(n + h)$ time and $O(U - L + h)$ space, where h is the size of the output. Previously, there was no known algorithm with non-trivial result for these problems.

All of our algorithms can be used to solve the corresponding higher dimensional problems by reducing them to 1-dimensional problems in the way described in [10, 70]. They can also be extended to solve the problems with non-uniform length. We note that for k maximum

sum segments problem there is another version of the problem where there is no restriction on the segment length (i.e., $L = 0$ and $U = n$) but the segments are not allowed to overlap. For this case there are linear time algorithms for 1-dimension [16, 17, 59]. In this dissertation, we shall not pursue this line. In all of our algorithms in this dissertation, the segments are allowed to overlap.

According to [55, 68], the compositional heterogeneity of a genomic sequence is strongly correlated to its CG content regardless of the size of the genome. It is also found that gene length [30], gene density [76], patterns of codon usage [61], distribution of different types of repetitive elements [30, 66], number of isochores [11], length of isochores [55] and recombination rate within chromosomes [36] are related to CG content. The algorithms can be used directly to find length-constrained CG-rich regions with the maximum sum and average or with some user specified content requirement in a DNA sequence.

The nucleotide composition of a newly determined DNA sequence is analyzed to locate its biologically meaningful segments including finding CG-rich regions [32, 41], TA and CG-deficient regions [56], CpG islands [41], regions rich in periodical three-base pattern [62, 71], post processing sequence alignment [74], annotating multiple sequence alignments [68] and computing length-constrained ungapped local alignment [6]. Our algorithms have potential applications in those areas.

In Section 2.2 we briefly describe Kim's [46] algorithm for the maximum density segment problem. Our algorithms for the maximum density, k maximum density segments,

maximum sum segments and k maximum sum segments of a sequence are presented in Sections 2.3, 2.4, 2.5 and 2.6 respectively. Section 2.7 describes our algorithms for finding all the segments of a sequence having sum or density bounded below by some user specified value. Concluding remarks are given in Section 2.8.

2.2 Kim's Algorithm for Maximum Density Segment

We describe Kim's [46] algorithm for the maximum density segment problem using uniform length. He reduced the problem to a geometric one thus. The element indices and corresponding prefix sums give $n+1$ points in the plane $p_0 = (0, s_0), p_1 = (1, s_1), p_2 = (2, s_2), \ldots, p_n = (n, s_n)$, sorted by their x-coordinates. The density of a segment $A[i, j]$ can then be interpreted as the slope of the line segment through the points $(i-1, s_{i-1})$ and (j, s_j). The problem then is to find p_i and p_j such that $\overline{p_i p_j}$ has the largest slope.

Without any restriction on the segment length, the maximum density segment problem is solved by computing the largest slope defined by a pair of the above points. We can use any of a number of $O(n \log n)$ slope selection algorithms for this problem ([23] or [45] for example). The constraints on the segment length add a new dimension to the problem.

For a given *right* endpoint p_j, the set of candidate *left* endpoints p_i has i in the index-window $I_j = [0, j-L]$ when $L \leqslant j < U$ and in $I_j = [j-U, j-L]$ when $j \geqslant U$. If we maintain the lower convex hull of the points in this index-window, then the largest slope is found by drawing a tangent from p_j to a point p_t on this lower hull. The maximum density

segment for a fixed j is then $a_{t+1}, a_{t+2}, \ldots, a_j$. As j goes from L to n the maximum of all slopes found gives the desired maximum density segment.

Based on the above formulation, Kim proposed an algorithm that claimed to be able to perform all the dynamic updates to the lower convex hull as the index-window moves from the left to the right in $O(n)$ amortized time. This claim is inaccurate. Figure 2.1 shows the lower convex hull (*lch*, for short) of the points inside the index-window $I(j)$, where p_x, p_z and p_y are the leftmost, bottommost and rightmost points on the *lch*. Kim maintains the portion of *lch* from p_y to p_z in one array and the portion of the *lch* from p_x to p_z in another array.

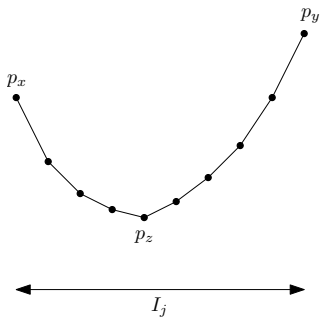


Figure 2.1: *The lower convex hull of the points in the index-window* I_j

Now, it is crucial to the correctness of Kim's algorithm that, as the window I_j slides to the right the algorithm remains updated about the new value of p_z. Kim's algorithm correctly updates p_z, except in the case shown in Figure 2.2.

In this case, as the window slides to the next position the hull update cannot be done in $O(1)$ time as Figure 2.3 shows.

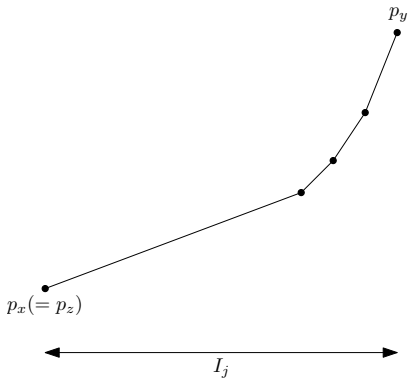


Figure 2.2: *The problem case for Kim's algorithm*

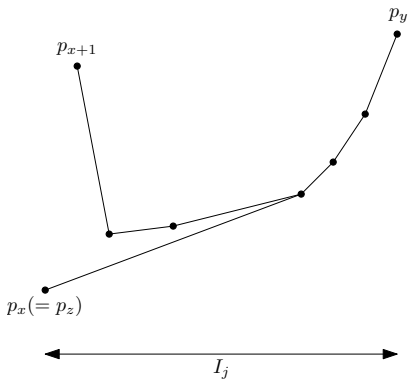


Figure 2.3: p_z *may need to be recomputed*

2.3 Algorithm for Maximum Density Segment

First, we describe our algorithm for the case of uniform length, i.e., $l_i = 1$ for $i = 1, ..., n$. The main idea underlying the new algorithm is to consider the right end point p_j (for $j = L, L+1, \ldots, n$) of all the feasible segments $\overline{p_i p_j}$ in batches of a fixed size. For each p_j, instead of computing a single lower convex hull of the feasible set of left end points p_i, we compute two lower convex hulls - a left one and a right one that start at 2 adjacent points p_{m-1} and p_m, $j - U < m \leqslant j - L$ (Figure 2.4). The right lower hulls are computed incrementally in a left-to-right (LR) pass for a batched set p_j, and the left hulls in a right-to-left (RL) pass for the same batched set. Thus, the problem that arises in Kim's algorithm

from the dynamic convex hull update as a result of deletion on the left is avoided. The correctness of this scheme follows from the following observation for the property of a set:

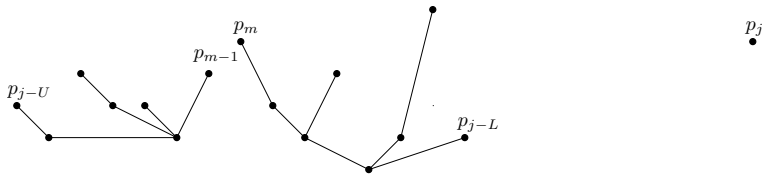


Figure 2.4: *Incremental left and right lower convex hulls*

Observation 1 *For a point $p_j, U \leqslant j \leqslant n$, let G^j be the set of the candidate left end points p_i of all feasible segments. If G_1^j and G_2^j are any 2 subsets of G^j such that $G^j = G_1^j \cup G_2^j$, then*

$$\max_{p_i \in G^j} slope(\overline{p_i p_j}) = \max\{ \max_{p_i \in G_1^j} slope(\overline{p_i p_j}), \max_{p_i \in G_2^j} slope(\overline{p_i p_j})\}.$$

We consider the right end points p_j, $j > U$, in batches of size $U - L + 1$. The details of the LR and RL passes for a batch of points p_j, $j \in [k, k + U - L]$, $k > U$, are described below. For each pass, we first describe the algorithm informally, and then follow it up with a formal description.

2.3.1 The LR pass

In this pass, we consider the right end points p_j, $j \in [k, k+U-L]$, in left-to-right fashion. For each new right end point p_j, $j \in [k, k + U - L]$, we incrementally compute the lower convex hull $H_r = LCH(\{p_{k-L}, p_{k-L+1}, \ldots, p_{j-L}\})$. In other words, for each p_j, H_r is updated by insertion of a new point p_{j-L} on the right end of it. Following Kim [46], we maintain 2

parameters to aid the incremental computation: a tangent line l to the current hull H_r with the maximum slope found so far, and the point of contact α of l with the current hull H_r. We always represent l by a pair of points. The slope of l is the current maximum density for this batch of p_j.

Initially, $H_r = \{p_{k-L}\}$, $l = \overline{p_{k-L}p_k}$ and $\alpha = p_{k-L}$. For the current right end point p_j, $j \in [k, k+U-L-1]$, let H_r, l and α be as shown in Figure 2.5. For the next right end point p_{j+1}, we update H_r, l and α. H_r is updated by inserting the point p_{j+1-L} on the right, i.e., $H_r = LCH(H_r \cup \{p_{j+1-L}\})$. The updated H_r is traversed counterclockwise from α (or from the newly inserted hull point p_{j+1-L} - if α is deleted from H_r) to find the new tangent line l having the maximum slope found so far, and the new point of contact α on H_r with the updated l. There are 4 cases as follows:

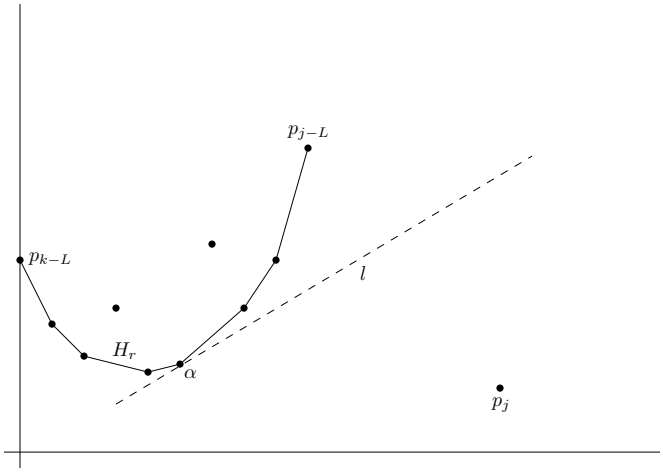


Figure 2.5: *LR pass: The lower hull, l and α for the right end point p_j, $j \in [k, k+U-L-1]$*

Case 1: Both p_{j+1-L} and p_{j+1} are above l (Figure 2.6).

H_r is updated. H_r is traversed counterclockwise from α to the point of contact of the

tangent from p_{j+1} to this new H_r, while these tangent and point of contact are set to be the new l and α respectively.

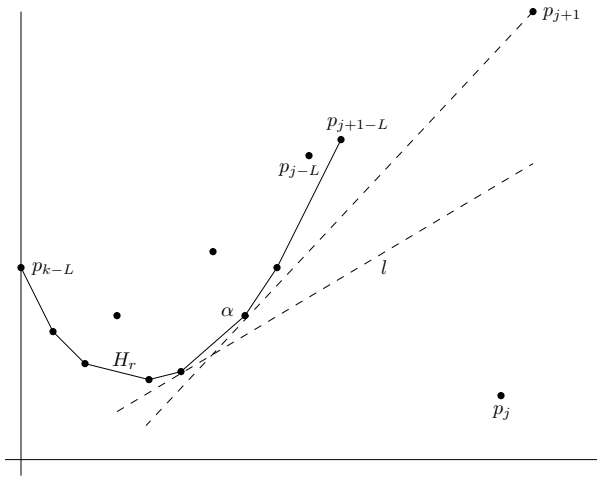


Figure 2.6: *LR pass: Both p_{j+1-L} and p_{j+1} are above l*

Case 2: p_{j+1-L} is above, and p_{j+1} is on or below l (Figure 2.7).

H_r is updated. However, α and l remain unchanged.

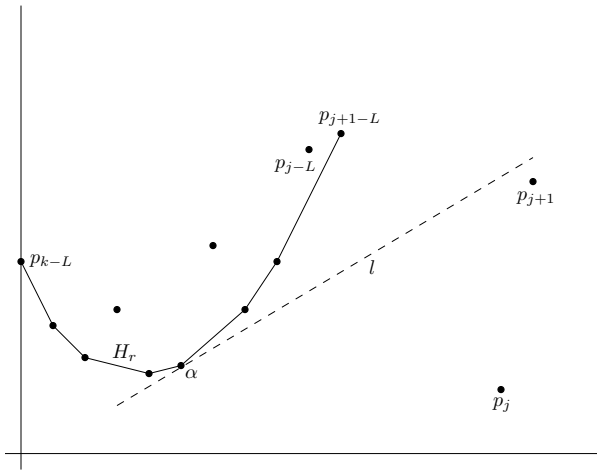


Figure 2.7: *LR pass: p_{j+1-L} is above, while p_{j+1} is on or below l*

Case 3: p_{j+1-L} is on or below l (Figure 2.8).

H_r is updated. Let l' be a line through p_{j+1-L} and parallel to l. Let p_{j+1} be above

l'; reset $l = \overline{p_{j+1-L}p_{j+1}}$ and $\alpha = p_{j+1-L}$.

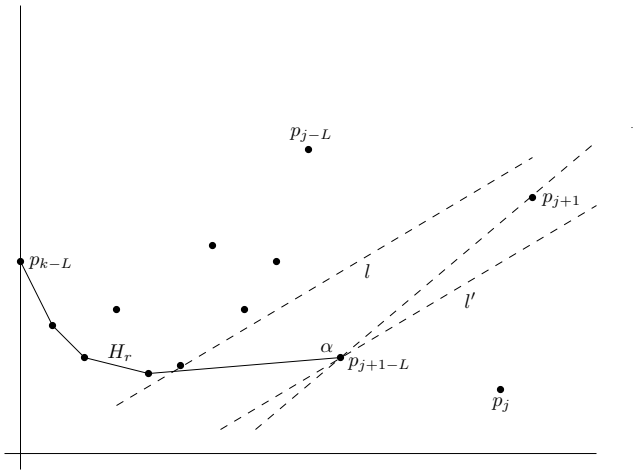


Figure 2.8: LR pass: p_{j+1-L} *is on or below* l*, and* p_{j+1} *is above* l'

Case 4: p_{j+1-L} is on or below l, and p_{j+1} is on or below l' (Figure 2.9).

H_r is updated. Set l to l' and $\alpha = p_{j+1-L}$.

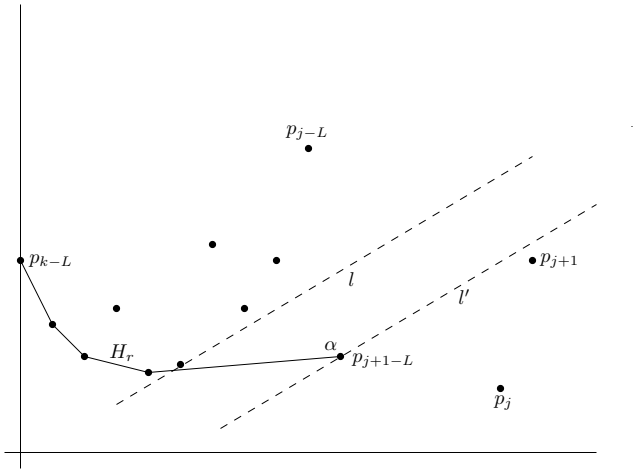


Figure 2.9: LR pass: p_{j+1-L} *is on or below* l*, and* p_{j+1} *is on or below* l'

Each point in the left window $\{p_{k-L}, p_{k+1-L}, \ldots, p_{k+U-2L}\}$ is added to an H_r once, and deleted at most once from a subsequent H_r. For a new point p_j, if α does not move right, the cost of computation is constant and is charged to the point p_{j-L} that is added to the

hull. We note that α never moves clockwise. Now we consider the case in which α move counterclockwise. Each point on H_r is accessed at most once during the recomputation of α, since it never moves clockwise. The cost of recomputing α is charged to the points passed over as we move counterclockwise on the updated H_r from the current α, and the cost of deleting the points on H_r on the left of α are charged to them. Thus, each point p_i in the left window is charged at most 3 times: 2 times for insertion into and deletion from H_r and once for being passed over by α.

Since α never moves backward in this pass, we do not need to maintain the part of H_r that lies on the left of α. The algorithm for the LR pass, called MDS-LRPASS, is given in Algorithm 1.

We note that at the end of traversal of H_r in step 2.1.2.1, if an edge of H_r coincides with the new tangent line, we select the right end point of that edge as the point of contact p_i.

2.3.2 The RL pass

This pass needs more careful handling. In this pass, we consider the right end points p_j, $j \in [k, k+U-L-1]$, in right-to-left fashion. For each new right end point p_j, $j \in [k, k+U-L-1]$, we incrementally compute the lower convex hull $H_l = LCH(\{p_{j-U}, p_{j-U+1}, \ldots, p_{k-L-1}\})$. In other words, for each p_j, H_l is updated by insertion of a new point p_{j-U} on the left end of it. As in LR pass, we maintain 2 parameters to aid the incremental computation: a tangent line l to the current hull H_l with the maximum slope found so far, and the point

Algorithm 1 Algorithm for LR Pass

1: **procedure** MDS-LRPASS(s, L, U, k)
Input: s is the array of prefix sum for the input sequence. L and U are respectively lower and upper bounds. k is the index of the first element of the current batch of right end elements.
Output: Maximum density segment l in LR pass for the current batch of elements.
2: $H_r \leftarrow LCH(\{p_{k-L}\})$ ▷ p_i is the point (i, s_i). H_r is the lch.
3: $\alpha \leftarrow p_{k-L}$ ▷ α is the left end point of the current maximum slope line segment.
4: $l \leftarrow \overline{\alpha p_k}$ ▷ l is the current maximum slope line segment. It is stored as a pair of points.
5: **for** $j \leftarrow k+1$ to $k+U-L$ **do**
6: **if** p_{j-L} is above l **then**
7: $H_r \leftarrow LCH(H_r \cup \{p_{j-L}\})$
8: **if** p_j is above l **then** ▷ l and α are not updated if p_j is on or below l.
9: Starting from α, traverse H_r counterclockwise to find the new point of contact p_i on it with the tangent line passing through p_j, and delete from H_r those points that are passed over by α.
10: $\alpha \leftarrow p_i$
11: $l \leftarrow \overline{\alpha p_j}$
12: **end if**
13: **else** ▷ p_{j-L} is on or below l
14: $H_r \leftarrow \{p_{j-L}\}$
15: $\alpha \leftarrow p_{j-L}$
16: Set l to the line parallel to l and passing through α
17: **if** p_j is above l **then** ▷ l is not updated further if p_j is on or below l
18: $l \leftarrow \overline{\alpha p_j}$
19: **end if**
20: **end if**
21: **end for**
22: **return** l
23: **end procedure**

of contact α of l with the current hull H_l. We always represent l by a pair of points. The slope of l is the current maximum density for this batch of p_j.

Initially, $H_l = \{p_{k-L-1}\}$, $l = \overline{p_{k-L-1}p_{k+U-L-1}}$ and $\alpha = p_{k-L-1}$. For the current right end point p_j, $j \in [k+1, k+U-L-1]$, let H_l, l and α be as shown in Figure 2.10. For the next right end point p_{j-1}, we update H_l, l and α. H_l is updated by inserting the point p_{j-1-U} on the left, i.e., $H_l = LCH(\{p_{j-1-U}\} \cup H_l)$. The updated H_l is traversed counterclockwise from α (or from the newly inserted hull point p_{j-1-U} - if α is deleted from H_l) to find the new tangent line l having the maximum slope found so far, and the new point of contact α on H_l with the updated l. Again, there are 4 cases as follows:

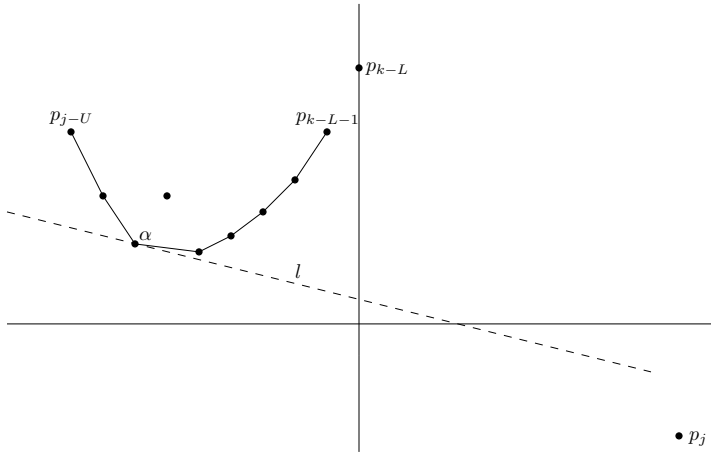


Figure 2.10: *RL pass: The lower hull, l and α for the right end point p_j, $j \in [k+1, k+U-L-1]$*

Case 1: p_{j-1-U} is on or above l, and p_{j-1} is above l (Figure 2.11).

H_l is updated. We traverse H_l counterclockwise from α to find a tangent to it from p_{j-1}. We reset l to this tangent line and α to the point of contact between updated l and H_l.

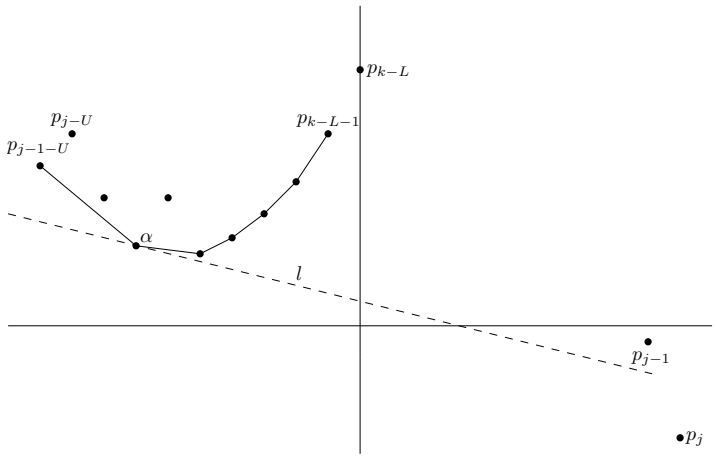


Figure 2.11: RL pass: p_{j-1-U} *on or above* l*, and* p_{j-1} *is above* l

Case 2: p_{j-1-U} is on or above l, and p_{j-1} is on or below l (Figure 2.12).

H_l is updated. However, α and l remain unchanged.

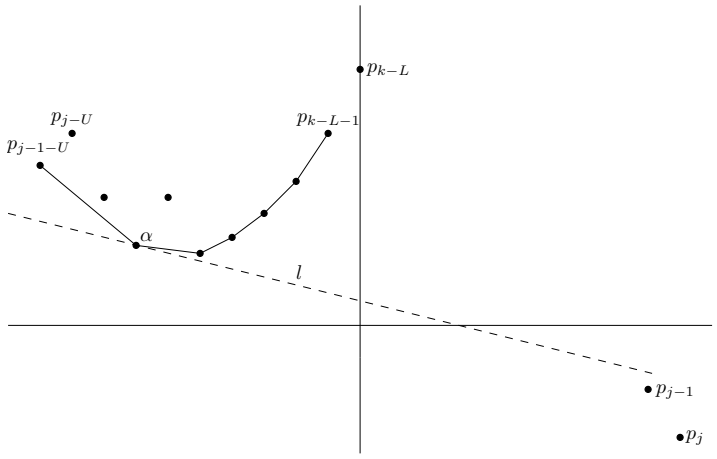


Figure 2.12: RL pass: p_{j-1-U} *is on or above* l *and* p_{j-1} *is on or below* l

Case 3: p_{j-1-U} is below l (Figure 2.13).

H_l is updated. Let l' be a line through p_{j-1-U} and parallel to l. Let p_{j-1} be above l'.

There will be only one point, viz., p_{j-1-U}, on the updated H_l that is on the left side of α. We traverse the updated H_l from p_{j-1-U} counterclockwise from α to the point of contact of the tangent from p_{j-1} to the new H_l, while α and l are updated to the new

tangent and the point of contact respectively. In this case, on the left of α at most one point, viz., the newly added point p_{j-1-U}, is checked to find α. Consequently, α can move left by at most one point.

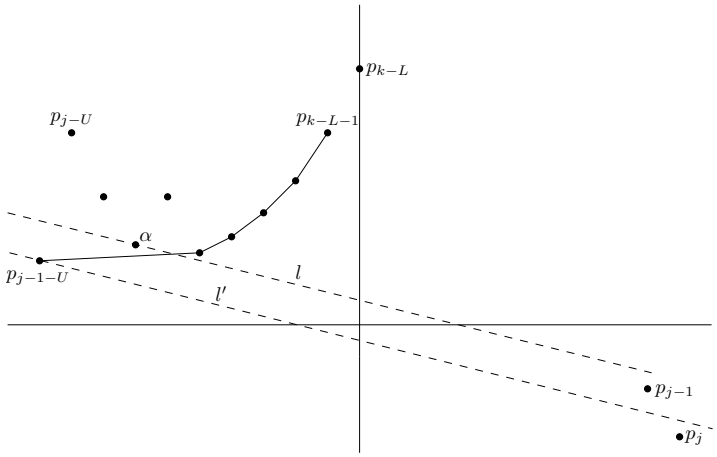


Figure 2.13: RL pass: p_{j-1-U} *is below* l *and* p_{j-1} *is above* l'

Case 4: p_{j-1-U} is below l, and p_{j-1} is on or below l' (Figure 2.14).

H_l is updated as in Case 3. We reset l to l' and α to p_{j-1-U}.

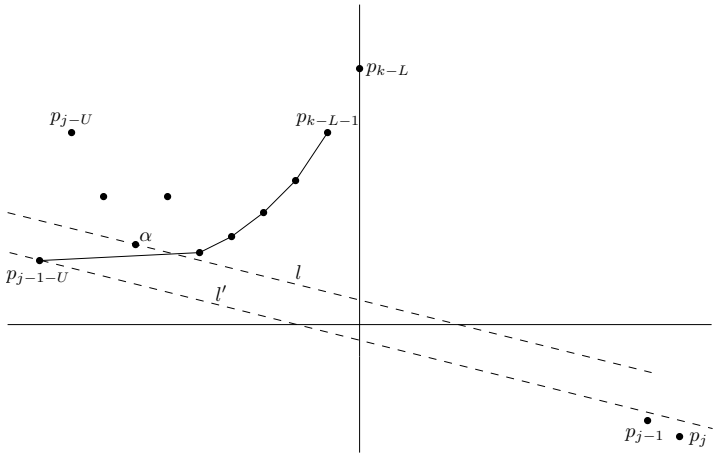


Figure 2.14: RL pass: p_{j-1-U} *is below* l *and* p_{j-1} *is on or below* l'

Time complexity analysis for this pass is exactly the same as that for the LR pass, except that for a new point p_j, α may move clockwise on H_l exactly by one position. If it

does move clockwise, then it moves to p_{j-U}. This cost is charged to the new point p_{j-U} in the left window. Thus, each point p_i in the left window is charged at most 4 times: 2 times for insertion into and deletion from H_l, once when α moves clockwise to it and once when α passes over it.

We note that once α moves clockwise and passes over a point p_i on H_l, it never moves back to that point again, or to any point lying on its left in the current H_l. Consequently, those points cannot be in contention for α anymore. We delete them from current H_l and do not consider them for future H_l. The algorithm for the RL pass, called MDS-RLPASS, is given in Algorithm 2.

We note that at the end of traversal of H_r in steps 2.1.1 and 2.2.4.1, if an edge of H_r coincides with the new tangent line, we select the right end point of that edge as the point of contact α.

2.3.3 The Algorithm

For the first batch of $U - L + 1$ points p_j, $j \in [L, U]$, we make an LR pass only. For the remaining points at the end, right end points are p_j, $j \in [k, n]$. First, we make an LR pass with p_i, $i \in [k - L + 1, n - L + 1]$ and LCH($\{p_{k-L}, p_{k-L+1}, \ldots, p_{n-L}\}$). Next, we make an RL pass as follows: Left end points are p_i, $i \in [k - U + 1, k - L]$. Construct LCH H_l for the left end points $p_{k-L-1}, p_{k-L-2}, \ldots, p_{n-U}$. Draw tangent from p_n to this hull. The tangent line is l and the point of contact is α. Delete from H_l the points that are on the right side of α. Now we make RL pass starting from right end point p_{n-1} and left end point p_{n-U-1}.

Algorithm 2 Algorithm for RL pass

procedure MDS-RLPASS(s, L, U, k)
Input: s is the array of prefix sum for the input sequence. L and U are respectively lower and upper bounds. k is the index of the first element of the current batch of right end elements.
Output: Maximum density segment l in RL pass for the current batch of elements.
    $H_l \leftarrow LCH(\{p_{k-L-1}\})$ ▷ p_i is the point (i, s_i). H_l is the lch.
    $\alpha \leftarrow p_{k-L-1}$ ▷ α is the left end point of the current maximum slope line segment.
    $l \leftarrow \overline{\alpha p_{k+U-L-1}}$ ▷ l is the current maximum slope line segment. It is stored as a pair of points.
    for $j \leftarrow k + U - L - 2$ to k **do**
        if p_{j-U} is on or above l, and p_j is above l **then** ▷ If p_{j-U} is on or above l, and p_j is on or below l, then none of H_l, l and α is updated
            Starting from α, traverse H_l counterclockwise to find the new point of contact p_i on it with the tangent line passing through p_j, and delete from H_l those points that are passed over by α
            $\alpha \leftarrow p_i$
            $l \leftarrow \overline{\alpha p_j}$
        else ▷ p_{j-U} is below l
            $H_l \leftarrow LCH(\{p_{j-U}\} \cup H_l)$
            $\alpha \leftarrow p_{j-U}$
            Set l to the line parallel to l and passing through α
            if p_j is above l **then** ▷ If p_j is on or below updated l, then none of α and l is updated again
                Starting from α, traverse H_l counterclockwise to find the new point of contact p_i on it with the tangent line passing through p_j, and delete from H_l those points that are passed over by α
                $\alpha \leftarrow p_i$
                $l \leftarrow \overline{\alpha p_j}$
            end if
        end if
    end for
    return l
end procedure

It stops when $j = k$ and $i = k - U$. We call this algorithm as MDS-RIGHTRESIDUAL.

The algorithms for maximum density segment, called MDS, is given in Algorithm 3.

Algorithm 3 Algorithm for maximum density segment problem

1: **procedure** MDS(A, L, U)
input: A is the input sequence. L and U are respectively lower and upper bounds.
Output: Maximum density segment l of A.
2: $n \leftarrow |A|$ ▷ n is the number of elements in A
3: $s_0 \leftarrow 0$ ▷ s is the array of prefix sum for the input sequence A.
4: **for** $i = 1, i \leftarrow i + 1$ till $k \leqslant n$ **do**
5: $s_0 \leftarrow s_i + A[i]$
6: **end for**
7: $l \leftarrow$ MDS-LRPASS(s, L, U, L) ▷ l is the current maximum density line segment. It is stored as a pair of points.
8: $b \leftarrow U - L + 1$
9: **for** $k = U + 1, k \leftarrow k + b$ till $k \leqslant n$ **do** ▷ One iteration for each batch of $U - L + 1$ elements and final iteration for the batch of residual elements (if any). Exit when $k > n$
10: $l' \leftarrow$ MDS-RLPASS(s, L, U, k) ▷ l' is the maximum slope line segment returned by MDS-RLPASS
11: **if** $slope(l) < slope(l')$ **then**
12: $l \leftarrow l'$
13: **end if**
14: $l' \leftarrow$ MDS-LRPASS(s, L, U, k) ▷ l' is the maximum slope line segment returned by MDS-LRPASS
15: **if** $slope(l) < slope(l')$ **then**
16: $l \leftarrow l'$
17: **end if**
18: **if** $k - s < n$ **then**
19: $l' \leftarrow$ MDS-RIGHTRESIDUAL($s, L, U, k - b, n$) ▷ l' is the maximum slope line segment returned by MDS-RIGHTRESIDUAL
20: **if** $slope(l) < slope(l')$ **then**
21: $l \leftarrow l'$
22: **end if**
23: **end if**
24: **end for**
25: **return** l
26: **end procedure**

2.3.4 Analysis

Each batch of $U - L + 1$ points in the left index window is considered at most twice by MDS algorithm: once for an LR pass of a batch of $U - L + 1$ right end points and once for an RL pass of a batch of $U - L$ right end points. As mentioned above the cost charged to each of these left end points are constant for each pass. Each of the right end points is accessed at most twice and that cost is charged to the respective point. Consequently, the time complexity is in $O(n)$. Thus, we have the following theorem:

Theorem 1 *Given a sequence A of n real numbers and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, MDS algorithm finds the maximum density segment of A from among all the segments of A of length at least L and at most U in $O(n)$ time and $O(U - L)$ space in an online manner.*

2.3.5 Improved Algorithm

The MDS algorithm works for all L and U with $0 \leqslant L \leqslant U \leqslant n$. Huang [44] proved the following result:

Observation 2 *If $R \subset A$ is a maximum density segment of length at least $2L$, then R can be obtained by merging 2 adjacent segments of length at least L with the highest densities.*

We prove a similar result below:

Lemma 2 *Let A be a sequence of length n, and L be a positive number such that $L \leqslant n$.*

One of the maximum density segments of A of length at least L must be of length at most $2L-1$.

Proof. Let $R = A[i,j]$ be a maximum density segment of length $l \geqslant 2L$. Let ρ denotes the density function. Let $\rho(R) = d$. We shall show that there is a subsegment $R' \subset R$ of length at least L such that $\rho(R') \geqslant d$.

Let us consider the subsegment $R_1 = A[i, i+L-1]$ of R of length L. Let $\rho(R_1) = d_1$. If $d_1 \geqslant d$, then R_1 is the required subsegment R' and we are done.

Otherwise, we consider $R_2 = R - R_1$. Let $\rho(R_2) = d_2$. Since $|R| \geqslant 2L$, we have $|R_2| = l - L \geqslant 2L - L = L$ and

$$d_2 = \frac{ld - Ld_1}{l-L} = d + \frac{L(d-d_1)}{l-L} > d, \text{since } d > d_1 \text{ and } l > L$$

Therefore, R_2 is the required subsegment R'. □

When $U \geqslant 2L$, for a right end point p_j we do not need to consider the left end points p_i such that $i \leqslant j - 2L$ by the above lemma. For this case, we improve our above algorithm by restricting the size of the batch of right end points p_j to L. If the batch consists of p_j, $j \in [k, k+L]$, the set of left end points are p_i, where $i \in [k-L, k]$ for the LR pass and $i \in [k-2L, L]$ for the RL pass. The improved algorithm, called MDS-IMPROVED, is given in Algorithm 4.

We have the following theorem:

Theorem 3 *Given a sequence A of n real numbers and two real numbers L and U with*

Algorithm 4 Improved algorithm for maximum density segment problem

```
1: procedure MDS-IMPROVED(A, L, U)
   Input: A is the input sequence. L and U are respectively lower and upper bounds.
   Output: Maximum density segment l of A.
2:     if U > 2L − 1 then
3:         U ← 2L − 1                ▷ If U is larger than 2L − 1, reset it to 2L − 1.
4:     end if
5:     l ← MDS(A, L, U)              ▷ Solve the problem by MDS.
6:     return l
7: end procedure
```

$1 \leqslant L \leqslant U \leqslant n$, MDS-IMPROVED algorithm finds the maximum density segment of A from among all the segments of A of length at least L and at most U in $O(n)$ time and $O(min\{U - L, L\})$ space in an online manner.

2.3.6 Implementation

In the implementation of our algorithms described above, no division is needed except once such as for reporting the final result of maximum density for the whole problem. We always represent the tangent line l by the pair of points p_i and p_j through which it passes. To represent the line l' passing through a point p'_i and parallel to l, we determine the translation that translates p_i to p'_i, and make the same translation to p_j to find the point p'_j. Then the line l' is represented by the pair of points p'_i and p'_j. We do not need to maintain the slope μ. Instead we compare the slopes of a pair of line segments. To compare the slopes of 2 line segments, all we need to do is to determine if it is a left turn or a right turn. This can be done without division.

Our algorithm and Chung and Lu's [22] algorithm's run time have been compared using random number sequence data and real DNA sequence data (see Tables 2.1 and 2.2

Table 2.1: Comparison with random numbers

N	L	U	Our (millisec)	Chung (millisec)	Chung/Our
50,000	200	500	9.1	8.1	0.89
50,000	5,200	20,500	7.6	7.1	0.94
50,000	8,200	32,500	6.5	6.7	1.03
10,000,000	2,500	25,000	185	154	0.83
10,000,000	12,500	125,000	176	154	0.88
10,000,000	62,500	625,000	145	147	1.02

Table 2.2: Comparison with real DNA sequence data

N	L	U	Our (millisec)	Chung (millisec)	Chung/Our
10,000	80	640	1.6	1.0	0.63
10,000	320	2560	1.4	1.2	0.860.94
10,000	1,280	10,240	1.5	1.0	0.67
450,000	200	2,000	68	55	0.81
450,000	2,000	200,000	60	53	0.88
450,000	40,000	400,000	59	53	0.90

respectively). It is found that our algorithm performs better when the difference between U and L is very large in comparison to the number of inputs N, except for one case of DNA sequence data. In other cases Chung and Lu's algorithm performs better.

2.3.7 Non-uniform Length

The above algorithm for uniform length can be extended to solve the general problem where the lengths $l_i, i = 1, ..., n$, are arbitrary. For this we define the cumulative lengths $L_i, i = 0, ..., n$, as $L_0 = 0$ and $L_i = l_1 + ... + l_i$, for $i = 1, ..., n$. Then the density $\mu_{i,j}$ of a segment $A[i, j]$ can be written as

$$\mu_{i,j} = \frac{s_j - s_{i-1}}{L_j - L_{i-1}}.$$

For each element $a_i, i = 1, ..., n$, in the sequence A we get the point (L_i, s_i), $i = 1, ..., n$,

in the plane. We have the initial point $(L_0, s_0) = (0, 0)$. Then the problem to find the feasible segments with the maximum density is reduced to finding the feasible pairs of points with the maximum slope. And this can be solved by a simple modification to our above algorithm for uniform lengths. The only difference is that the abscissas of the consecutive points (L_i, s_i) and (L_{i+1}, s_{i+1}) are l_{i+1} distance away instead of unit distance away. Its time and space complexity will be $O(n)$ and $O(\frac{U-L}{l_{min}})$ respectively. Thus, we have the following theorem:

Theorem 4 *Given a sequence A of n pairs of real numbers $(a_i, l_i), i = 1, ..., n$, and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, our geometric algorithm as described above finds the maximum density segment of A from among all the segments of A of length at least L and at most U in $O(n)$ time and $O(\frac{U-L}{l_{min}})$ space in an online manner.*

2.3.8 Extension to Higher Dimensions

Using the method of [10, 70] the 2-dimensional problem is reduced to $\binom{m}{2} + m$ 1-dimensional problems for an $m \times n$ input matrix. We solve each of them using the above algorithm. The time complexity will be $O(m^2 n)$.

Theorem 5 *Given a 2-dimensional $m \times n$ matrix A of pairs of real numbers $(a_{ij}, l_{ij}), i = 1, ..., m; j = 1, ..., n$, and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, there exists an algorithm to find the maximum density subarray of A from among all the subarrays of A of length at least L and at most U in $O(m^2 n)$ time and $O(mU)$ space.*

Proof. Similar to the proof of Theorem 3 of Brodal and Jorgensen [13] and is omitted. □

The above algorithm can be extended to any dimension d in a straight forward way.

Theorem 6 *Given a d-dimensional $n_1 \times n_2 \times ... \times n_d$ matrix A of pairs of real numbers and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, there exists an algorithm to find the maximum density subarray of A from among all the subarrays of A of length at least L and at most U in $O(n_1 \Pi_{i=2}^{d} n_i^2)$ time and $O(U \Pi_{i=2}^{d} n_i)$ space.*

Proof. Similar to the proof of Theorem 4 of Brodal and Jorgensen [13] and is omitted. □

To avoid being repetitive, we note that the algorithms described in the following sections can all be extended to higher dimensions using the same reduction technique.

2.4 k Maximum Density Segments

2.4.1 Algorithm for $k \in O(\lg^2(U-L))$

Let us assume that $k \in O(\lg^2(U-L))$. The above MDS algorithm is repeated k times for each batch of $U-L+1$ points to find at least k maximum density segments with right end points in the batch. In each iteration the maximum density segment for the iteration is found. Keeping the left end point of the maximum density segment found in the current iteration fixed, all the feasible segments with right end point within the current batch of $U-L+1$ points are selected and the left end point is deleted at the end of the iteration. The algorithm for a batch of $U-L+1$ points is described in Algorithm 5. We call this algorithm as KMDS-SMALLK. X and Y are the sets of respectively $2U-2L+1$ and

$U - L + 1$ number of left and right end points of MDS algorithm. D is the set of k number of candidate maximum density segments found so far.

Algorithm 5 Algorithm for k-maximum density segment problem when $k \in O(\lg^2(U-L))$

1: **procedure** KMDS-SMALLK(X, Y, L, U, D, k)
Input: X and Y are the sets of left and right end points respectively. L and U are lower and upper bounds respectively. k is the required number of maximum density segments.
Output: The set D of k number of candidate maximum density segments and corresponding densities found so far.
2: **for** $i = 1$ to k **do**
3: Find the maximum density segment from among the feasible segments with left end points in X and right end points in Y using the MDS algorithm (Algorithm 3).
4: Let x be the left end point of this segment. For all feasible segments with left end point x, i.e., $(x, y'), y' \in Y$, insert $(x, y', d(x, y'))$ in D.
5: From D select the k maximum density segments using a linear time selection algorithm [12] and update D with these k elements.
6: Delete x from X.
7: **end for**
8: **return** D
9: **end procedure**

Clearly, each iteration costs $O(U - L)$ time. k iterations in a pass cost $O(k(U - L))$ time. The total cost per right end point is in $O(k)$. Thus, we have the following theorem:

Theorem 7 *Given a sequence A of n real numbers, two integers L and U with $1 \leqslant L \leqslant U \leqslant n$, and an integer $k \in O(\lg^2(U-L))$, MDS-SMALLK algorithm finds the k maximum density segments of A from among all the segments of A of length at least L and at most U in $O(kn)$ time and $O(U - L)$ space in an online manner.*

2.4.2 Algorithm for $k \in \omega(\lg^2(U-L)) \cap o(n(U-L)/\lg^2(U-L))$

For $k \in \omega(\lg^2(U-L)) \cap o(n(U-L)/\lg^2(U-L))$ we present an improved algorithm. For simplicity, we assume that the sequence elements are of unit length, i.e., $l_i = 1, i = 1, ...n$.

As before, for each batch of $U - L + 1$ points we make LR and RL passes to consider all the feasible segments with right points in the batch. But the segments are processed in a different way. Let the batch of right end points be in the index window $[b, b + U - L]$. In the LR pass the left end points of all the feasible segments are in the index window $[b-U+1, b-L+1]$, and in RL pass they are in $[b-L+2, b+U-2L+1]$. Then the LR and RL passes will consider all the feasible segments with right end points in the index window $[b, b + U - L]$. The LR and RL passes are similar. We describe LR pass for the batch.

Grouping the feasible segments

Here we describe the grouping of feasible segments for the LR pass. A group of feasible segments is represented by the pair $I_l \times I_r$ where I_l and I_r are the index windows for respectively the consecutive left end points and the consecutive right end points according to the x-coordinate such that all the combinations of segments with left end points in I_l and right end points in I_r are feasible. The main advantage of this grouping is that for each right end point p_j with $j \in I_r$, all the segments $\overline{p_i p_j}$ with $i \in I_l$ is a feasible segment. All the segments represented by a group $I_l \times I_r$ will be processed in one batch. Processing of a group of feasible segments will be described next. We shall call I_l the left index window and I_r the right index window.

For example, in Figure 2.15 the large square identifies a group of 64 segments which will be represented by $[b - U + 8, b - U + 15] \times [b, b + 7]$. We do not construct the segments explicitly, just identify the pair of index windows.

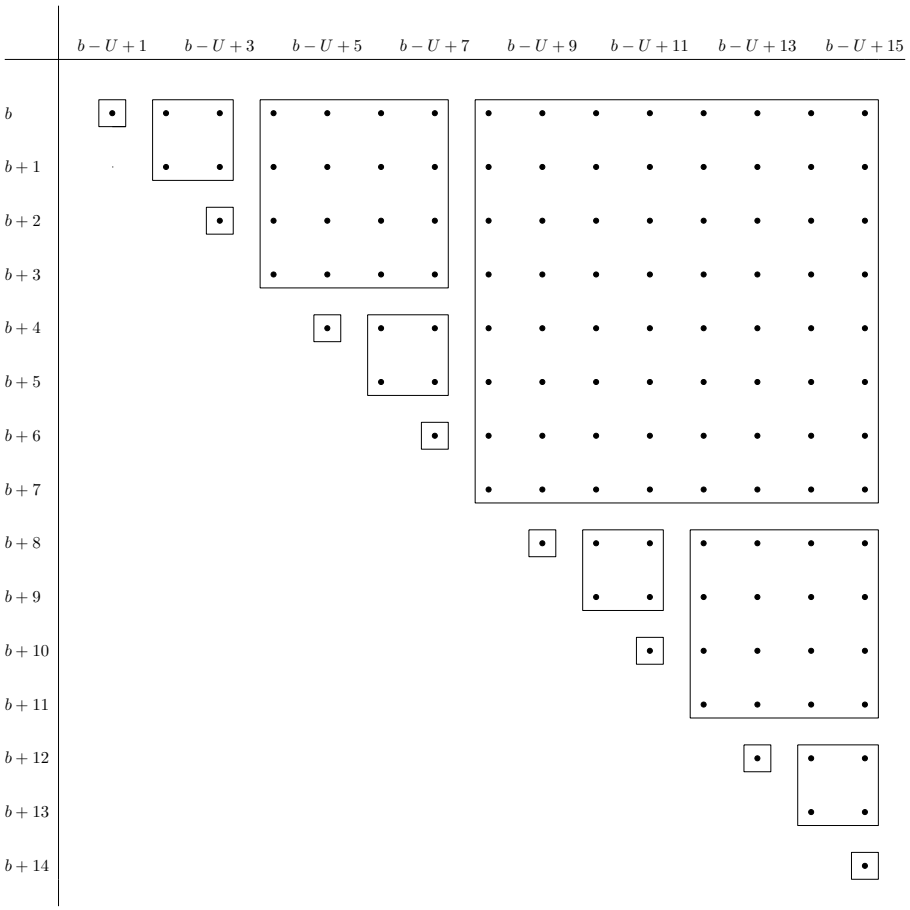


Figure 2.15: *Grouping of feasible segments*

We scan I_l in left-to-right fashion. First, for the single right end point p_b, we make a group of all the feasible segments with the single left end point p_{b-U+1}. The group of feasible segments is $[b-U+1, b-U+1] \times [b, b]$ (Figure 2.15).

Next, we make the following 2 groups of feasible segments: $[b-U+2, b-U+3] \times [b, b+1]$ and $[b-U+3, b-U+3] \times [b+2, b+2]$. This completes scanning 2 more left end points $p_i \in [p_{b-U+2}, p_{b-U+3}]$. After this scan all the feasible segments with consecutive 3 left end points starting from p_{b-U+1} and consecutive 3 right end points starting from p_b have been completely grouped.

Next, we make the following 4 groups of feasible segments: $[b-U+4, b-U+7] \times [b, b+3]$, $[b-U+5, b-U+5] \times [b+4, b+4]$, $[b-U+6, b-U+7] \times [b+4, b+5]$ and $[b-U+7, b-U+7] \times [b+6, b+6]$. This completes scanning 4 more left end points $p_i \in [p_{b-U+2}, p_{b-U+3}]$. After this scan all the feasible segments with consecutive 7 left end points starting from p_{b-U+1} and consecutive 7 right end points starting from p_b have been completely grouped.

At the end of the i-th step we have grouped all the combinations of segments generated by $2^i - 1$ consecutive right end points and the same number of consecutive left end points such that they are feasible. We note that for each of the groups of feasible segments generated by the above algorithm, the left and right index windows have the same length, and that the length of the index windows are in powers of 2. For simplicity, let us assume that $U - L + 1 = 2^s - 1$ for some positive integer s. After s steps all the feasible segments with consecutive $2^s - 1$ left end points starting from p_{b-U+1} and ending at p_{b-L+1}, and the same number of consecutive right end points starting from p_b and ending at p_{b+U-L} have been completely grouped. Thus, all the feasible segments corresponding to the LR pass have been completely grouped. We note that all the G_is are mutually disjoint in the sense that no pair of G_is have any common segment.

Lemma 8 *The above algorithm constructs groups of feasible segments G_i, $i = 1, ..., U-L+1$ such that $\cup_{i=1}^{U-L+1} G_i$ is the set of all feasible segments in the LR pass and all the G_is are mutually disjoint.*

The two characteristics of G_is mentioned in Lemma 8 ensures that the segments in each

group can be processed independently of the other groups and that we need to process the G_is only.

In the above grouping procedure we do not consider the segments, but their indices. It will take constant time to construct a group. For $2^s - 1$ right end points, $2^s - 1$ groups of feasible segments will be created.

From Figure 2.15 we see that the above grouping can be done by the recursive algorithm KMDS-GROUPING in Algorithm 6. The inputs l_s, r_s and m are respectively the starting indices of left and right windows, and the length of them.

Algorithm 6 Algorithm for grouping points

1: **procedure** KMDS-GROUPING(l_s, r_s, m)
Input: l_s and r_s and m are the starting indices of left and right windows respectively. m is the length of them.
Output: The set of pairs of left and right intervals for the indices of the input sequences. For each pair length of left and right intervals are the same and are of the form 2^i where i is an integer.
2: **if** $m > 1$ **then**
3: $m' \leftarrow \frac{m+1}{2}$
4: **return** KMDS-GROUPING$(l_s, r_s, m' - 1) \cup \{([l_s + m' - 1, l_s + m - 1], [r_s, r_s + m' - 1])\} \cup$ KMDS-GROUPING$(l_s + m', r_s + m', m' - 1)$
5: **else**
6: **return** $\{([l_s, l_s], [r_s, r_s])\}$
7: **end if**
8: **end procedure**

Grouping in the RL pass will be similar. Thus, we have the following lemma:

Lemma 9 *All the groups G_i for each of LR and RL passes can be created in $O(U - L)$ time and space.*

Organizing the points

Now we describe the processing of a group of feasible segments. Let $G = I_l \times I_r$ be a group of feasible segments where $|I_l| = |I_r| = m = 2^t$ for some positive integer t. Then $|G| = |I_l| \times |I_r| = 2^{2t}$. Let Q and R be the sets of points having index windows I_l and I_r respectively. Then $|Q| = |R| = m = 2^t$.

First, we organize the points in Q. We use Overmars and van Leeuwen's [57] algorithm, with a simple modification, to construct the *lch* (lower convex hull) of Q by composition. In our computation Q will not change. We do not need insertion or deletion operation for the convex hull.

By construction of the geometric problem all the points are already sorted by x-coordinates, and are vertically separated (i.e., no pair of points lie on the same vertical line). In fact, all the n input points are separated by unit distance in x-coordinate, and consequently all the points of Q are separated by unit distance in x-coordinates.

The convex hull is constructed iteratively. In the first iteration, we construct 2^{t-1} *lch*s of 2 consecutive points each. In the 2nd iteration, we construct 2^{t-2} *lch*s of 2^2 consecutive points each by composing pairs of adjacent constituent *lch*s of 2 consecutive points each. In the 3rd iteration, we construct 2^{t-3} *lch*s of 2^3 consecutive points each by composing pairs of adjacent constituent *lch*s of 2^2 consecutive points each. We continue this for t iterations. The information of all of these constituent *lch*s as well as the *lch* of Q is stored in a balanced binary search tree, say C. This tree will be called LCH Tree. Its leaf vertices represent

the points of Q. Direct parents of the leaves represent the next higher level of *lch*s. Direct parents of these parents represent the next higher level of *lch*s and so on. The root represent the *lch* of Q. We denote the *lch* of Q by H^1 and a *lch* at i-th level and j-th position from left by H^i_j.

First, we describe a naive algorithm for finding the k-maximum density segments. In Overmars and Leeuwen's [57] algorithm each vertex u of C is associated with a concatenable queue [2] to store the information about a portion of the *lch* of all the points stored in the leaves of the subtree rooted at u. This was necessary to update C after each insertion and deletion. But we do not insert or delete any points in Q. Once constructed we do not need to modify C. With each vertex $u \in C$, we associate an array Q_u instead of a concatenable queue. Q_u stores the same information as the concatenable queue, viz., the left or right part of the contour of the *lch*, of the points at the leaves of the subtree rooted at u, that is not a part of the contour of the *lch* associated with the father of (u). Contents of a vertex c of C are as follows:

1. $f(c)$ - a pointer to the father of c (if any).

2. lchild(c) - a pointer to the left child of c.

3. rchild(c) - a pointer to the right child of c.

4. Q_c - an array containing the *lch* of the set of points in the subtree of c.

5. b_l - left end point of the bridge.

6. b_r - right end point of the bridge.

The time for the construction of the arrays at any level of C from its immediate lower level is bounded by $O(m)$ and there are $\lg m$ levels in the tree. Thus, we have the following Lemma which is similar to Proposition 4.1 of Overmars and Leeuwen [57]:

Lemma 10 *The tree C for a set of m points can be constructed in $O(m \lg m)$ time and $O(m)$ space.*

Proof. Similar to the proof of Corollary 3.3 and Proposition 4.1 of Overmars and Leeuwen [57] and is omitted. □

The time needed for the construction of the convex hull is blown up by a factor of $\lg m$. But it will help searching the k maximum density segments efficiently.

Searching

Now we describe searching for k maximum density segments for the group of segments in G. Let us assume that the LCH Tree C of all levels of *lch*s of Q have already been constructed. For a right end point $p_j \in R$, the maximum density segment is found by drawing tangent to the top most level *lch* H^1 (Figure 2.16). For simplicity, we assume that there is only one point of contact always. But this assumption is not essential for the method being described in the following. Because, if there are multiple points of contact, say s number of points of contact $p_i, p_{i'}, p_{i''}$, ... etc., then all of them will correspond to the same density. If necessary, all of them will be selected first at no extra cost. Only then, the search needs

to find another segment of lower density by following all of $p_i, p_{i'}, p_{i''}$, ..., . If this total cost is averaged over the p_is, then it will be the same as that of following each of some s points with different tangents separately.

Let the single point of contact be p_1^j (p_{j-L-12} in Figure 2.16). We want to find the next maximum density segment with the same right end point p_j. Let the left end point of this segment be p_2^j (p_{j-L-11} in Figure 2.16). We need to find it. Clearly, it lies either on the contour of, or interior to H^1. It will be:

- either, one of the 2-adjacent points $p_{j_1'}$ and $p_{j_1''}$ of p_{j_1} (one on the left and the other on the right of p_{j_1}) on H^1;
- or, interior to H^1 and its x-coordinate lies between the x-coordinates of $p_{j_1'}$ and $p_{j_2''}$, i.e., $x_{j_1'} < x_{j_2} < x_{j_1''}$.

We search neighbourhood of p_{j_1} by successively reducing the size of the neighbourhood until we reach 2 adjacent points with x-coordinates $x_{j_1} - 1$ and $x_{j_1} + 1$.

By construction, the contour of lower hull H^1 consists of a portion of the contour of each of H_1^2 and H_2^2. They are joined by an edge, called bridge [57], between the 2 nearest end points of those portions. So, p_1^j must lie either on H_1^2 or on H_2^2. Let it lie on H_2^2.

By construction of H, any pair of *lch*s at the same level are mutually disjoint, $H_{j_1}^i \cap H_{j_2}^i = \phi$ for all j_1 and j_2 with $j_1 \neq j_2$. Since p_1^j lies on the contour of H_2^2, p_2^j can either be the point of contact of tangent from p_j to H_1^2, or on the contour or interior of H_2^2. To find

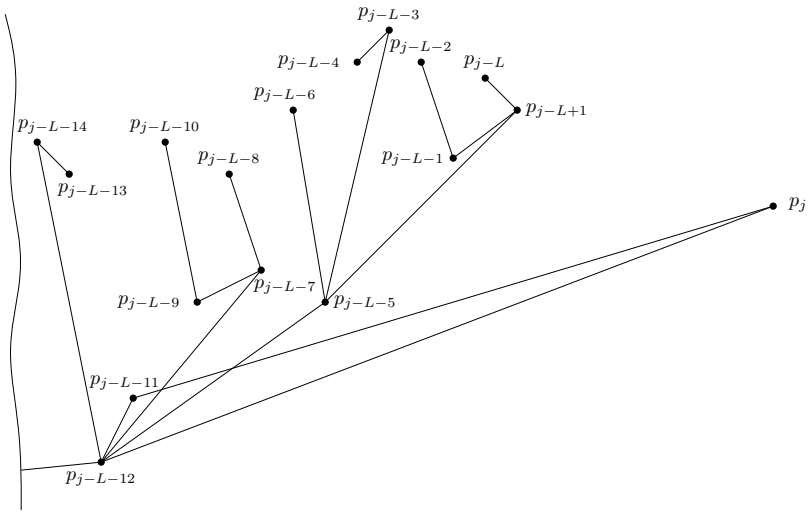


Figure 2.16: *Finding the next point w.r.t. right end point* p_j

the point of contact with H_1^2, the contour of H_1^2 can be searched in $O(\lg m)$ time using binary search on the array associated with the corresponding vertex c_1^2 in C. The second case is the same as the initial problem except for the *lch* changed from H^1 to H_2^2. Thus, the problem is solved recursively. There are $\lg m$ recursions. In each recursion the tangent point to the contour of one *lch* is found by using binary search on the array of points of the contour. Total time for searching p_2^j is $O(\lg^2 m)$.

For each point $p_j \in R$, we find the length constrained maximum density segment with p_j as the right end point. This is done in $O(\lg m)$ time by drawing tangent from p_j to the top level *lch* H^1 (stored at the root of C). The tangent point is found by using a binary search of the array associated with the root of C. For each p_j a vertex v_{j_1} is constructed for the maximum density segment w.r.t. p_j. Since the tangents to H^1 from multiple points in R may have the same point of contact, the same point in H^1 may be left end points for multiple vertices $v_{j_1}, v_{k_1}, ..., etc.$, having distinct right end points $p_j, p_k, ..., etc.$ respectively.

A maximum heap T is constructed using v_{j_1}, $j \in I_r$, as its vertices and the density of a segment as the order of the heap (Figure 2.17). The heap is initially constructed as a balanced binary search tree with the exception that each vertex has a null middle children. The middle children will point to an implicit heap that will be initialized and expanded as needed.

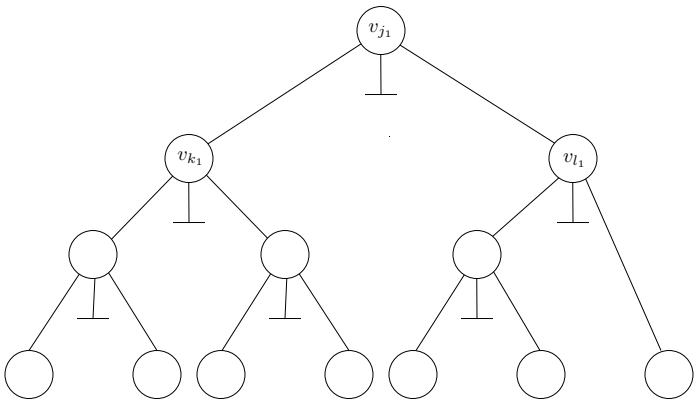


Figure 2.17: *Search tree*

From the heap the k maximum density elements are selected by using Frederickson's [33] heap selection algorithm. A middle child will be explicitly constructed only when Fredrickson's [33] algorithm reaches there. Each of these vertices will have a maximum of $\lg m$ number of children. A child vertex will be created for each vertex of C that is visited during search. After the initial construction of T, we will never create a left child or a right child of any of its initial vertices.

Let t be any vertex of T. Let p_i and p_j be the left and right end points corresponding to t. Let p_i be the tangent point on the *lch* H_r^q. Then t will contain a pointer to c_r^q, where c_r^q represents the *lch* H_r^q. For each vertex u_{j1} of T, $j \in I_r$, all the vertices of its middle

subtrees as well as itself represent the segments for which right end points are the same point p_j. Contents of a vertex t of T are as follows:

1. $f(t)$ - a pointer to the father of t (if any).

2. $lchild(t)$ - a pointer to the left child of t.

3. $rchild(t)$ - a pointer to the right child of t.

4. $c(t)$ - a pointer to the vertex c of C, where by searching the lch at c the search has selected p_i as the left end point of a segment

5. p_j - right end point of a segment.

6. p_i - left end point of a segment with right end point p_j. As mentioned before, its value is selected by searching the vertex pointed by $c(t)$.

7. ρ - slope of $p_i p_j$.

Let Fredrickson's [33] algorithm wants to access the children of a vertex v of T. Accessing the left and right children is straightforward. To access the middle children, it creates them first. Let v corresponds to the tangent point on H_r^q w.r.t. p_j. First, we search for the next maximum density segment w.r.t. p_j. We search for the tangent point on lchs at lower levels than H_r^q. The search is conducted by the recursive algorithm discussed above. A child vertex of v is created for the tangent point on each of the lower level lchs from p_j.

Then Fredrickson's [33] algorithm searches those vertices. The algorithm selects k maximum density segments in this way. The time at each vertex is blown up by a factor of $\lg^2 m$.

We call the naive algorithm for expanding T as EXPAND. It is formally given in Algorithm 7. For each vertex c in C we store the entire *lch* of the set of points in its subtree in its associated array Q_c. We do not need C^*. Each vertex t of T has a pointer t_c to an associated vertex c in C. EXPAND is called when Frederickson's [33] algorithm wants to access the children of t. It is not called from t if t_c points to a leaf vertex in C. EXPAND calls the algorithm CONTACTPOINT in Algorithm 8 to find the tangent point on a hull from a right end point.

Lemma 11 *The naive algorithms described above selects the k-maximum density segments correctly.*

Proof. We need to show that: (i) there is no duplication of vertices in expanded T, (ii) the expanded T is a max heap and (iii) no vertex of the k-maximum density segment vertices are missed by Fredrickson's [33] selection algorithm.

During construction of T, the roots of T_j have been inserted in T in maximum heap order, and by construction, there is no duplicate root vertices T_js in T.

Now we consider the vertices that are expanded as Fredrickson's [33] algorithm wants to access them. These vertices are expanded from the roots of T_j. Let us consider a tree rooted at T_j wrt a right end point p_j. We shall show that for each point p_i, only one vertex of T_j is associated with it. Let a vertex associated with p_i in T_j be t_i^j. We shall show that

Algorithm 7 Algorithm for expanding the search tree

procedure EXPAND(t, c)
Input: t is a vertex of search tree T. c is a vertex of LCH Tree C.
Output: NULL
▷ When Frederickson's [33] algorithm wants to access the children of t, this algorithm creates the children of t. t has a pointer to c. c stores the entire *lch* of the set of points in its subtree.
    create a child vertex t' of t
    $p_j(t' \leftarrow p_j(t)$
    if $lchild(c)$ is a leaf of C **then**
        Create a child vertex t'' of t
        $p_j(t'' \leftarrow p_j(t)$
        $c(t') \leftarrow$ lchild(c)
        $c(t'') \leftarrow$ rchild(c)
        $p_i(t') \leftarrow p_i(lchild(c))$
        $p_i(t'') \leftarrow p_i(rchild(c))$
        $d(t') \leftarrow$ slope($p_i(t')p_j(t)$)
        $d(t'') \leftarrow$ slope($p_i(t'')p_j(t)$)
    else
        if $x(t) > x_{b_l}(c)$ **then**
            $c(t') \leftarrow lchild(c)$
            CONTACTPOINT($t', lchild(c), b_l(c)$)
            EXPAND($t, rchild(c)$)
        else
            $c(t') \leftarrow rchild(c)$
            CONTACTPOINT($t', rchild(c), b_r(c)$)
            EXPAND($t, lchild(c)$)
        end if
    end if
end procedure

Algorithm 8 Algorithm for finding the tangent point on a *lch*.

procedure CONTACTPOINT(t', c', b)
Input: t' is a vertex of search tree T. c' is a vertex of LCH Tree C. b is the end point of bridge.
Output: NULL
    binary search $Q_c^{'}$ to find the point of contact p_i' on the left over part of $H_{c'}$ in $Q_c{}'$ from $p_j(t')$
    if slope($p_i'p_j$) < slope(b, p_j) **then**
        $p_i' \leftarrow b$
    end if
    $p_i(t') \leftarrow p_i'$
    $d(t') \leftarrow$ slope($p_i(t')p_j(t')$)
end procedure

no other vertex in the tree T_j can be associated with t_i^j.

We note that a vertex is created in T wrt only a point of contact on an *lch*. Let t_i^j is created wrt the point of contact p_h on the *lch* H^α.

By construction, t_i^j must have been created only when p_h is found as the point of contact with the highest level super hull among all the super hulls of p_h, for which p_h is the point of contact. So, H^α is the highest level super hull among all the super hulls of p_h for which p_h is the point of contact.

Again, we note that when a vertex t_β^j wrt point of contact p_i on *lch* H^β is expanded, a child vertex of t_β^j is created for each subhull [down to the single point subhull (that is associated with a leaf vertex of C)] of H^β (on which p_β^j does not lie). In other words, when a vertex t_β^j wrt a point of contact p_1^j on *lch* H^β is expanded, a child vertex of t_β^j is created for each subhull [down to the single point subhull (that is associated with a leaf vertex of C)] of H^β, whose brother subhull's contour contains the point p_β.

Thus, p_α can be created only as a unique child vertex of t_β^j. Therefore, t_α^j is unique only if t_β^j is unique. By induction on parent vertex and the fact that T_j is a tree, it follows that t_α^j is unique.

Since H^α is a subhull of H^β, and p_h and p_i are points of contact of the tangents from p_j to H^α and H^β respectively, we have $\text{slope}(p_h p_j) \leqslant \text{slope}(p_i p_j)$. So, max heap order is preserved for creating t_α^j as a child of t_β^j.

To prove (iii) we assume that Fredrickson's [33] algorithm has expanded the tree T^j

completely. Let p_h be an arbitrary left end point. We shall show that there exists a vertex in T^j.

Let H^α be the highest level superhull of p_h among all the super hulls of p_h which have p_h as the point of contact of the tangent from p_j. Following the argument similar to the above we can show that a vertex wrt p_h must have been created in T^j, if there exists a vertex t^j_β in T^j wrt to the point of contact p_i of the tangent from p_j to the highest level super hull of p_i, among all the super hulls of p_i which have p_i as the point of contact of the tangent from p_j to them. The result follows by induction and the fact that each point has a super hull and that all the super hulls are composed hierarchically into the single highest level super hull H. □

Lemma 12 *After constructing the LCH Tree C of a group G of size m, the k maximum density segments can be found from G in $O(k \lg^2 m)$ time.*

From Lemmas 10 and 12 we have:

Lemma 13 *For a group G of size m, the k maximum density segments can be found in $O(m \lg m + k \lg^2 m)$ time and $m \lg m$ space.*

Let $U - L + 1 = 2^s - 1$. There will be $\frac{U-L+2}{2^{i+1}}$ groups of size $2^i - 1$. Total cost for the construction of *lch*s for the LR pass is

$$\sum_{i=0}^{s-1} \frac{U-L+2}{2^{i+1}} O(2^i \lg 2^i) = O((U-L) \lg^2 (U-L))$$

To find the k maximum density segments for the LR pass the heap T is constructed from all the groups. Then Fredrickson's [33] algorithm is used to search the k maximum density segments from it. The CH Trees of the groups are searched as described above. For each pass of each batch, the k maximum density segments are updated using a linear time selection algorithm [12]. If $k > \theta(U - L)$ a single heap T is constructed for all the passes and all the batch. There will be $(U - L)\lg(U - L)$ vertices in the tree. Fredrickson's [33] algorithm is used to search the k maximum density segments from it as before. We have the following theorem:

Theorem 14 *For $k \in \omega(\lg^2(U - L)) \cap o(n(U - L)/\lg^2(U - L))$, there exists an algorithm for solving the k maximum density segments problem in $O((n + k)\lg^2(U - L))$ time.*

We now improve the above algorithms for expanding T. Its asymptotic time complexity remains the same though. Each vertex v_{j_i} of T is associated with a binary search tree C_j^* which contains a copy of all the search paths in C that have been traversed to date w.r.t. p_j. Let v be any vertex of T. Let p_i and p_j be the left and right end points corresponding to v. Let p_i be the tangent point on the *lch* H_r^q. Then v will contain a pointer to c_r^q and c_r^{*q}, where c_r^q represents the *lch* H_r^q and C_r^{*q} is its corresponding vertex in C^*.

For each vertex v_{j1} of T, $j \in I_r$, all the vertices of its middle subtrees as well as itself represent the segments for which right end points are the same point p_j. All these vertices are associated with the same binary search tree C_j^*.

For each vertex v_{ji} of T, the keys at each vertex of associated tree C^* stores the informa-

tion of the points that have already been selected as a left end point for a maximum density segment. This indicates that the points have been removed from contention w.r.t. p_j. But no point is deleted from the arrays associated with the vertices of C, because those points may be in contention w.r.t. right end points other than p_j. Since the points of any *lch* H_j^i that are selected as left end points for maximum density segments must be consecutive on H_j^i, we store only the indices of Q_j^i corresponding to the left end and the right points of this sequence of points as keys in the corresponding vertex of C_j^*. Let the names of these keys be *left* and *right*.

For a point $p_j \in R$, as the search for a maximum density segment goes down a path in C from the root towards a leaf and finds a maximum density segment at a vertex u in C, a similar path from the root of C^* as well as the bordering vertices are created or updated in C^*, if needed. The point selected from u is removed from contention for p_j and is recorded accordingly in the corresponding vertex in C^* by updating either *left* or *right*. When a new vertex of C^* is created and no point is removed from contention from the corresponding hull, *left* and *right* are set to zero. Clearly, time for updating C^* is $O(\lg m)$.

For searching on or interior to a *lch* H^α, we always look for 3 adjacent points on it. Let the 3 points on it be p_j^1, p_j^2 and p_j^3 in order of increasing x-coordinates. The x-coordinate of the next maximum density segment must lie on or between the x-coordinates of p_j^1 and p_j^3. If we do not need to go into its interior, then we find the tangent point to it from p_j. The tangent point is obviously p_j^2.

Now consider the case when we need to go into its interior of H^α. If p_j^2 is not one of the 2 bridge points on H_α, then those 3 points must lie on one of the 2 constituent *lch*s of H_α. We search that *lch* and the problem is the same as the original problem except for the change in *lch*.

Otherwise, let 1 point lies on the constituent *lch* H_1^α on the left and the rest 2 points lie on the constituent *lch* H_2^α on the right. For H_1^α we find the tangent point p_j^4 on the left over portion of it from p_j using binary search on the array Q_1^α (which stores the leftover portion on the right of the *lch*) associated to c_1^α that represents H_1^α. Then the tangent point on will be the one between P_j^1 and p_j^4 with the maximum slope. Corresponding to that tangent point we create a vertex in each of C^* and T.

For H_2^α, the two points are p_j^2 on the left and p_j^3 on the right. The point p_j^5 that is adjacent to p_j^1 on the left on H_2^α is the right most point on the remaining portion of H_2^α. Consequently, it must be the current last point in the associated array c_j^α. We select that point from the associated array c_2^α. Corresponding to p_j^5 on H_2^α, we create a vertex in each of C^* and T. Now the 3 adjacent points on H_2^α are p_j^5, p_j^2 and p_j^3 in left-to-right order and the problem is similar to the original problem except for the change in *lch* and the 3 points. The computation will by symmetric if one point lies on H_2^α and the rest 2 lies on H_1^α.

The above algorithms can be easily modified, in a way similar to that in Section 2.3.7 for length constrained maximum density segment problem with non-uniform length, to solve the problem with non-uniform length. The algorithm is efficient when $k \in \omega(\lg^2(U-L)) \cap$

$o(n(U-L)/\lg^2(U-L))$. When $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$ any brute force algorithm will be optimal as long as the selection is done in linear time in the number of feasible segments. We describe a brute force algorithm in the following subsection.

2.4.3 Algorithm for $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$

For $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$ a brute force algorithm as described in the following is optimal in time. From $O(n(U-L))$ number of all the possible feasible segments, k maximum density segments are selected using linear time selection algorithm [12]. Its time complexity is clearly $O(n(U-L))$. To minimize space usage the sequence is scanned from left to right. For each element $a_j \in A$ all the feasible segments $A[i,j]$ with right end element being a_j are considered. The segments are inserted into a set D of candidate maximum density segments. As soon as k new segments are inserted into D, k number of maximum density segments are selected from it using linear time selection algorithm [12], and D is updated with these k maximum density segments. Its time complexity is clearly $O(n(U-L))$. We have the following theorem:

Theorem 15 *Given a sequence A of n real numbers, two integers L and U with $1 \leqslant L \leqslant U \leqslant n$, and one integer $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$, the above algorithm finds the k maximum density segments of A from among all the segments of A of length at least L and at most U in $O(n(U-L))$ time and $O(k)$ space in an online manner.*

2.5 Maximum Sum Segment

As before we solve the problem in batch mode with $U - L + 1$ elements in each batch. For each batch of elements we consider all the feasible segments with right end element in the batch. Analogous to Observation 1 for the maximum density segment problem, we have the following observation:

Observation 3 *For an element $a_j, U \leqslant j \leqslant n$, let G^j be the set of the candidate left end elements a_i of all feasible segments. If G_1^j and G_2^j are any 2 subsets of G^j such that $G^j = G_1^j \cup G_2^j$, then*

$$\max_{a_i \in G^j} \sum_{t=i}^{j} a_t = \max\{\max_{a_i \in G_1^j} \sum_{t=i}^{j} a_t, \max_{a_i \in G_2^j} \sum_{t=i}^{j} a_t\}.$$

For each batch of $U - L + 1$ number of right end elements we make 2 passes as before. Let the batch of elements be $a_b, ..., a_{b+U-L}$, where $b = U, U + (U - L + 1), U + 2(U - L + 1), ..., U + \lfloor \frac{n-U+1}{U-L+1} \rfloor (U - L + 1), b \geq U$.

First, we consider LR pass for the batch. For each right end element $a_j \in A[b, b+U-L]$ the feasible segments are $A[b - L + 1, j], A[b - L + 2, j], ..., A[j - L + 1, j]$. The set of sums of these segments is represented by Q_{LR}^j and is defined as:

$$Q_{LR}^j = \{(i, j, s[i, j]) | i = b - L + 1, b - L + 2..., j - L + 1; s[i, j] = \sum_{t=i}^{j} a_t\}.$$

It can be incrementally defined as:

$$Q_{LR}^{j} = \{(j-L+1, j, s[j-L+1, j])\} \cup \{(i, j, s+a_j) | (i, j-1, s) \in Q_{LR}^{j-1}\}. \tag{2.1}$$

The set Q_{LR}^{j} is constructed from Q_{LR}^{j-1} by adding a_j to each element of Q_{LR}^{j-1} and inserting an additional element. Since adding a constant number does not change the relative order of a set of numbers, the same element will be the maximum element for the set before and after the addition. We have

$$\max Q_{LR}^{j} = \max\{\max Q_{LR}^{j-1} + a_j, s[j-L+1, j]\}$$

If we denote the maximum of Q_{LR}^{j} by M_{LR}^{j}, $j = b, ..., b+U-L$, then the above relation becomes

$$M_{LR}^{j} = \max\{M_{LR}^{j-1} + a_j, s[j-L+1, j]\}$$

To find the maximum value M_{LR}^{j} for Q_{LR}^{j} we need the information about the maximum element M_{LR}^{j-1} of Q_{LR}^{j-1}. The algorithm, called MSS-LR, is given in Algorithm 9.

Lemma 16 *Given a sequence A of n real numbers and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, MSS-LR finds the maximum sum segment of A from among all the segments of A of length at least L and at most U in LR pass for a batch of right end elements of size $U-L+1$ in $O(U-L)$ time and $O(U-L)$ space.*

Proof. The array s of input prefix sum is computed in the preprocessing step in $O(U-L)$

Algorithm 9 Algorithm for LR pass for maximum sum segment problem

```
1: procedure MSS-LR(A, s, L, U, b)
   Input: A is the input sequence, s is the array of prefix sum of A, and L and U are
   respectively lower and upper bounds. b is the index of the first right end element of the
   current batch of right end elements.
   Output: Maximum sum M, and indices l and r of left and right elements of the
   maximum sum segement in LR pass for the current batch of right end elements.
2:     M ← s[b] − s[b − L]                          ▷ M is the current maximum sum.
3:     l ← b − L
4:     r ← b
5:     for t ← b + 1 to b + U − L do
6:         if A[t] > 0 then
7:             M ← M + A[t]
8:             r ← t
9:         end if
10:        if M < s[t] − s[t − L] then
11:            l ← t − L
12:            r ← t
13:            M ← s[t] − s[t − L]
14:        end if
15:    end for
16:    return (M, l, r)
17: end procedure
```

time. Line 2 of MSS-LR initializes the maximum sum M in constant time. For each right end element a_t, M is updated in lines 7-8 and/or 11-13 correctly in constant time. Thus, total time for the batch of $U - L + 1$ right end elements is $O(U - L)$.

Updating in lines 7-8 corresponds to all the feasible segments with right end element a_t except the segment $A[t - L + 1, t]$. Updating in lines 11-13 corresponds to the segment $A[t - L + 1, t]$. Thus, the algorithm correctly finds the maximum sum segment among all the feasible segments in LR pass with a batch of $U - L + 1$ right end elements. □

Now we consider RL pass for the batch. For each right end element $a_j \in A[b, b + U - L]$ the feasible segments are $A[i, j], i = b - L + 1, b - L, ..., j - U + 1$. The set of sums of these

segments is represented by Q^j_{LR} and is defined as:

$$Q^j_{RL} = \{(i, j, s[i, j] | i = j - U + 1, j - U + 2, ..., b - U + 1; s[i, j] = \sum_{t=i}^{j} a_t\}.$$

It can be incrementally defined as:

$$Q^j_{RL} = \{(j - U + 1, j, s[j - U + 1, j])\} \cup \{(i, j, s - a_{j+1}) | (i, j + 1, s) \in Q^{j+1}_{RL}\}. \quad (2.2)$$

The set Q^j_{RL} is constructed from Q^{j+1}_{RL} by subtracting a_{j+1} from each element of Q^{j+1}_{RL} and inserting an additional element. Since subtracting a constant number does not change the relative order of a set of numbers, the same element will be the maximum element for the set before and after the subtraction. We have

$$\max Q^j_{RL} = \max\{\max Q^{j+1}_{RL} - a_{j+1}, s[j - U + 1, j]\}$$

If we denote the maximum of Q^j_{RL} by M^j_{RL}, $j = b + U - L, ..., b$, then the above relation becomes

$$M^j_{RL} = \max\{M^{j+1}_{RL} - a_{j+1}, s[j - U + 1, j]\}$$

To find the maximum value M^j_{RL} for Q^j_{RL} we need the information about the maximum element M^{j+1}_{RL} of Q^{j+1}_{RL}. The algorithm, called MSS-RL, is given in Algorithm 10.

Lemma 17 *Given a sequence A of n real numbers and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, MSS-RL finds the maximum sum segment of A from among all the*

Algorithm 10 Algorithm for RL pass for maximum sum segment problem

```
1: procedure MSS-RL(A, s, L, U, b)
   Input: A is the input sequence, s is the array of prefix sum of A, and L and U are
   respectively lower and upper bounds. b is the index of the first right end element of the
   current batch of right end elements.
   Output: Maximum sum M, and indices l and r of left and right elements of the
   maximum sum segement in RL pass for the current batch of right end elements.
2:     M ← s[b + U − L] − s[b − L + 1]                    ▷ M is the current maximum sum.
3:     l ← b − L + 1
4:     r ← b + U − L
5:     for t ← b + U − L − 1 to b do
6:         if A[t] < 0 then
7:             M ← M − A[t]
8:             r ← t
9:         end if
10:        if M < s[t] − s[t − U + 1] then
11:            l ← t − U + 1
12:            r ← t
13:            M ← s[t] − s[t − U + 1]
14:        end if
15:    end for
16:    return (M, l, r)
17: end procedure
```

segments of A of length at least L and at most U in RL pass for a batch of right end elements of size $U - L + 1$ in $O(U - L)$ time and $O(U - L)$ space.

Proof. Similar to the proof of Lemma 16. □

By Lemmas 16 - 17 and Observation 3 we have

Theorem 18 *Given a sequence A of n real numbers and two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, our algorithm as described above finds the maximum sum segment of A from among all the segments of A of length at least L and at most U in $O(n)$ time and $O(U - L)$ space in online manner.*

2.6 k Maximum Sum Segments

We use a simple modification of Brodal and Jorgensen's [13] method to solve the k Maximum Sum Segments problem. As before we solve the problem in batch mode with $U - L + 1$ elements in each batch. For each batch of elements we consider all the feasible segments with right end elements in the batch. So, for each batch of $U - L + 1$ right end points we make 2 passes as before. For each pass we shall use Brodal and Jorgensen [13] algorithm to construct a partially persistent [29] max-heap ordered binary tree using vertex copying technique. The heap implicitly contains all the feasible segments with their respective sums. The heap is a modified version of the self adjusting heap (skew heap) of Sleator and Tarjan [64] such that it supports insertions in amortized constant time. Brodal and Jorgensen [13] called it *Iheap*. It will take $O(U - L)$ time and space to build it. From the heap the k maximum sum elements are selected by using Frederickson's [33] binary heap selection algorithm. It will take $O(k)$ time. For each pass of each batch of $U - L - 1$ right end elements we update the k maximum sum segments by using a linear time selection algorithm [12].

First, we consider LR pass for the batch. For each right end element $a_j \in A[b, b+U-L]$ the set Q^j_{LR} of sums of all the feasible segments are incrementally defined in equation (2.1). To avoid adding a_j to each element of Q^{j-1}_{LR} explicitly we represent the set of sums Q^j_{LR} implicitly by a pair $\langle H^j_{LR}, j\rangle$, where H^j_{LR} contains left end indices and j is the right end index of the segments whose sums constitute Q^j_{LR}. Here H^j_{LR} is a version of a partially

persistent *Iheap* representing all the segments whose sums constitute the set Q^j_{LR}. The right end index for the heap can be inserted to all the vertices of the heap by setting corresponding value of j. Then the set of sums Q^j_{LR} can be computed using the prefix sums as follows: $Q^j_{LR} = \{(s_j - s_{t-1}) | t \in H^j_{LR}\}$. The pair $\langle H^j_{LR}, j\rangle$ is incrementally defined as follows:

$$\langle H^b_{LR}, b\rangle = \langle \{b - L + 1\}, b\rangle,$$

$$\langle H^{j+1}_{LR}, j+1\rangle = \langle H^j_{LR} \cup \{j - L + 2\}, j+1\rangle.$$

where $j = b, b+1, ..., b+U-L-1$.

To construct $\langle H^{j+1}_{LR}, j+1\rangle$ from $\langle H^j_{LR}, j\rangle$ a vertex with key value $j - L + 1$ is inserted into H^j_{LR}. Since the new version of the heap is constructed using partial persistence, H^j_{LR} remains intact after this insertion. To evaluate the sum for a vertex in the version H^{j+1}_{LR} of the *Iheap* the right end index $j+1$ of all the segments in H^{j+1}_{LR} are found from the access pointer to this version. Then the sum is evaluated by subtracting the corresponding prefix sums. Since the relative order of the sums in Q^j_{LR} does not change in Q^{j+1}_{LR} and since only one new element is inserted into Q^{j+1}_{LR}, the time to construct H^{j+1}_{LR} from H^j_{LR} is to insert the new element in H^j_{LR}. Figure 2.18 shows a partially persistent *Iheap* and its access pointers corresponding to the LR-pass of the sequence (2, 5, -6, 3, -5, 2, 7, 3, 2, 4, -16, 6) with $L = 7$ and $U = 12$. The copy pointers, the inverse pointers and the version stamps of the vertices are not shown. The heap is constructed using vertex-copying technique.

From the heap the k largest sums are selected by using Frederickson's [33] binary heap

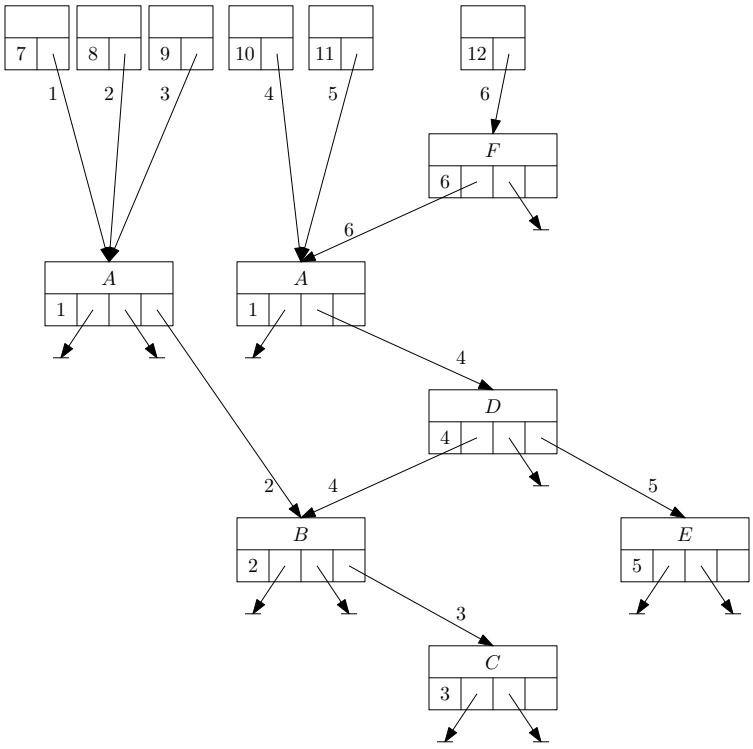


Figure 2.18: *Partially persistent Iheap and its access pointers corresponding to the LR pass of the sequence* $(2, 5, -6, 3, -5, 2, 7, 3, 2, 4, -16, 6)$ *with* $L = 7$ *and* $U = 12$. *The copy pointers, the inverse pointers and the version stamps of the vertices are not shown.*

selection algorithm. The algorithm visits vertices in top-down fashion. Before any vertex in a heap H^j_{LR} is visited by the algorithm it is explicitly constructed and the newly constructed vertex is visited. The right end element index j is moved downward and sum for the vertex is evaluated as the selection algorithm moves downward. For example, when Fredrickson's [33] algorithm follows the access pointer corresponding to the 8th element a_8 ($j = 8$) (Figure 2.18), the root A^2 of the 2nd version *Iheap* H^2 is explicitly created. In this vertex the corresponding values of i and j, i.e., 1 and 8 respectively, are stored as keys. The sum $s[1, 8]$, i.e., 11, is stored as another key. If the search follows the right child of this vertex, i.e., A^2 of version 2, then another vertex B^2 is created as a right child of A^2. In this

vertex 2 and 8 are stored as keys for the values of i and j respectively, and 9 is stored as a key for the value of the sum $s[2,8]$. B^2 will be a leaf of H since B is a leaf in version 2.

A complete heap H is constructed on top of all the heaps H^j, $j = b, ..., b+U-L$ [13], where the key values of all the top $U-L$ vertices have been set to ∞. Frederickson's [33] algorithm starts from the root of H and selects $U-L+k$ largest sum vertices in H. From them k largest sum elements are selected using linear time selection algorithm [12].

Now we consider RL pass for the batch. This pass is similar to the LR pass except that the set Q^j_{RL} of sums of all the feasible segments are defined in equation (2) and that Q^j_{RL} is implicitly defined as follows:

$$\langle H_{RL}^{b+U-L-1}, b+U-L-1\rangle = \langle\{b-L\}, b+U-L-1\rangle,$$

$$\langle H_{RL}^{j-1}, j-1\rangle = \langle H_{RL}^{j} \cup \{j-U+1\}, j-1\rangle.$$

where $j = b+U-L-1, b+U-L-2, ..., b+1$. Here H^j_{RL} contains all the left end indices and j is the right end index of all the segments whose sums constitute Q^j_{RL}.

For each pass of every batch of points the k maximum sum segments are updated using linear time selection algorithm [12] to provide the solution at the end. Thus, we have the following theorem:

Theorem 19 *Given a sequence A of n real numbers, two integers L and U with $1 \leqslant L \leqslant U \leqslant n$, and one integer $k \leqslant U-L$, there exists an algorithm to find the k maximum sum segments of A from among all the segments of A of length at least L and at most U in $O(n)$*

time and $O(U)$ space.

When $k \geqslant U - L$ we use the above algorithm for each group of $\lceil \frac{k}{U-L+1} \rceil$ number of batches of $U - L + 1$ consecutive elements of the sequence. For simplicity, let us assume that $n = U + m\lceil \frac{k}{U-L+1} \rceil (U - L + 1)$, where m is an integer. We include all the feasible segments corresponding to the right end elements $a_j \in A[L, U]$ in the first group. For a group, we insert into H all the feasible segments with right end points in that group. Select the k maximum sum segments from the heap using Frederickson's heap selection algorithm [33]. The set of k maximum sum segments is updated using linear time selection algorithm [12]. Thus, we have the following theorem:

Theorem 20 *Given a sequence A of n real numbers, two real numbers L and U with $1 \leqslant L \leqslant U \leqslant n$, and one integer $k > U - L$, there exists an algorithm to find the k maximum sum segments of A from among all the segments of A of length at least L and at most U in $O(n + k)$ time and $O(k)$ space.*

2.7 Finding All the Segments with Some Content Requirement

In genomic sequence analysis at times it is necessary to find all the segments in a sequence with some user specified minimum sum or density requirements [44].

2.7.1 Finding All the Segments Satisfying a Sum Lower Bound

Let σ be some user specified lower bound for sum. We use the algorithm in Section 4.1 to construct partially persistent *Iheap* [13] and select largest value vertices from it using Frederickson's heap selection algorithm [33]. The only change is that vertices are selected from the heap in iteration. In t-th iteration 2^t largest value vertices are selected and their minimum sum is found. The iteration stops when the minimum sum in an iteration is less than σ. Then all the segments with sum at least σ are reported.

Constructing the heap takes $O(U - L)$ time. Let h_b be the size of the output from the b-th batch and $2^{s-1} \leq h_b < 2^s$, for some $s \in I^+$. There will be s number of iterations of the selection algorithm. The b-th iteration takes $O(2^b)$ time. Total time over all the iterations is $O(\sum_{t=1}^{s} 2^t) = O(2^{t+1}) = O(h_b)$, where s is the number of iterations. Total time for b-th batch with $U - L + 1$ number of elements in the batch is $O(U - L + h_b)$. For n inputs the time will be $O(n + h)$, where h is the total number of outputs.

Theorem 21 *Given a sequence A of n real numbers, two integers L and U with $1 \leqslant L \leqslant U \leqslant n$, and one real number σ, All the segments of A of length at least L and at most U and sum at least σ can be found in $O(n + h)$ time and $O(U - L + h)$ space in an online manner, where h is the number of output.*

2.7.2 Finding All the Segments Satisfying a Density Lower Bound

Following Liu and Chao [51] we transfer the problem to the problem of finding segments satisfying a lower bound of 0 for sum. Let δ be some user specified lower bound for the density. A segment $A[i,j]$ has density of at least δ iff $\sum_{t=i}^{j}(a_t - \delta l_t) \geq 0$, i.e., iff the sequence segment $(a_i - \delta l_i, a_{i+1} - \delta l_{i+1}, ..., a_j - \delta l_j)$ has sum of at least 0. Then the modified problem is to find the length constrained segments of the sequence $A' = ((a'_t, l_t)|a'_t = a_t - \delta l_t, (a_t, l_t) \in A)$ such that the sum is non-negative. Let us consider a batch of points $(a'_k, l_k), (a'_{k+1}, l_{k+1}), ..., (a'_l, l_l)$ such that $L \leq \sum_{t=k}^{l} \leq U - L$. The new problem is solved by the algorithm of Section 2.7.1.

Theorem 22 *Given a sequence A of n real numbers, two integers L and U with $1 \leqslant L \leqslant U \leqslant n$, and one real number δ, all the segments of A of length at least L and at most U and density at least δ can be found in $O(n+h)$ time and $O(U-L+h)$ space in an online manner, where h is the number of output.*

2.8 Summary

In this chapter, some problems concerning the search for the interesting regions in a sequence are considered. We have presented linear time algorithms for both the problems of length-constrained maximum sum segments and length-constrained maximum density segments. The algorithms have been extended to find the k length-constrained maximum sum segments and k length-constrained maximum density segments problems. They have

also been extended to find all the segments satisfying a user specified sum or density lower bound in linear time. We indicate the extensions of our algorithms to higher dimensions. Our algorithms facilitate efficient solutions for all these problems in higher dimensions. All the algorithms can be extended in a straightforward way to solve the problems with non-uniform length.

The algorithms have applications in several areas of biomolecular sequence analysis including finding CG-rich regions, TA and CG-deficient regions, regions rich in periodical three-base pattern, post processing sequence alignment, annotating multiple sequence alignments and computing length constrained ungapped local alignment.

It would be interesting to study if there is any linear time algorithm for the k length-constrained maximum density segments problem. It can also be investigated to find more efficient algorithms for the problems in higher dimensions. It remains open to improve the trivial lower bounds for these cases.

Chapter 3

Point Placement Problem: Improved Algorithms

3.1 Introduction

3.1.1 The Problem

Let $P = \{p_1, p_2, ..., p_n\}$ be a set of n distinct points on a line L. In this chapter, we address the problem of determining a unique placement (up to translation and reflection) of the p_i's on L, by querying distances between some pairs of points p_i and p_j, $1 \leq i, j \leq n$.

The resulting queries can be represented by a *point placement graph* (*ppg*, for short), $G = (V, E)$, where V and E are the sets of vertices and edges respectively such that each point $p_i \in P$ is represented as a vertex $v_i \in V$ and each edge $e \in E$ joins a pair of vertices v_i and v_j in V if the distance between the corresponding two points p_i and p_j on L is known. Each edge $e \in E$ is assigned the length that is equal to the distance between its adjacent vertices. We shall use p_i to denote a point on L as well as a vertex of G.

A *ppg* G is *line rigid* or just *rigid* if its vertices have a unique placement on a line. Thus, the original problem reduces to the construction of a rigid *ppg*. The density ρ of a *ppg* G is

defined as $\rho(G) = \frac{|E|}{|V|}$.

Let us take some simple examples to illustrate the ideas involved. Suppose we have just 3 points $\{p_1, p_2, p_3\}$ on a line whose positions we want to know. Three different *ppg*s, up to relabelling, are possible (omitting the trivial case when $E = \emptyset$) as shown in Figure 3.1 below. Figure 3.1(a) corresponds to the situation when the distance between a pair of points, say p_1 and p_2, is known. For Figure 3.1(b), the distances between 2 pairs of points, say $\{p_1, p_2\}$ and $\{p_2, p_3\}$, are known. Figure 3.1(c) is the *ppg* when all the pairwise distances are known.

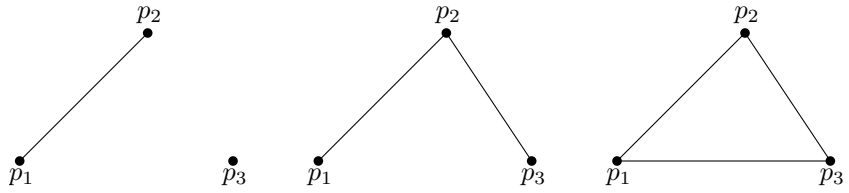


Figure 3.1: Some point placement graphs for 3 points

Clearly, for the *ppg* of Figure 3.1(a) a unique placement is not possible since the point p_3 can be anywhere relative to p_1 and p_2. The same is true of Figure 3.1(b): say, we place p_1 and p_2 first, but then the position of p_3 relative to p_2 is ambiguous. However, a unique placement is possible for the triangular *ppg* of Figure 3.1(c) as long as the length of one edge is the sum of or absolute difference between the lengths of the other two. Thus, if we first place p_1 and then place p_2 to p_1's right, p_3 will be placed between p_1 and p_2 if the sum of its distances from p_1 and p_2 is $|p_1p_2|$, and to the left of p_1 or to the right of p_2 if the absolute difference between the distances is equal to $|p_1p_2|$. In other words, the *ppg* of Figure 3.1(c) is rigid.

The last case suggests a simple algorithm using triangle as the basic component of a *ppg* for the unique placement of n points. Query the distance between two points, say p_1 and p_2. The position of each of the remaining points p_i, $i \geq 3$ is determined by querying the distances from p_i to p_1 and p_2; p_i lies between p_1 and p_2 if the sum of the distances is equal to $|p_1p_2|$, and to the left of p_1 or to the right of p_2 if the difference between the distances is equal to $|p_1p_2|$. The corresponding *ppg* shown in Figure 3.2 is then rigid. The number of queries made is $2n-3$, which is of the form $\alpha n+\beta$. Here $\alpha(=2)$ represents the asymptotic density of the *ppg* which is the limit of the number of edges per vertex as the number of vertices n goes to ∞. However, the density of the triangle *ppg* is 1.

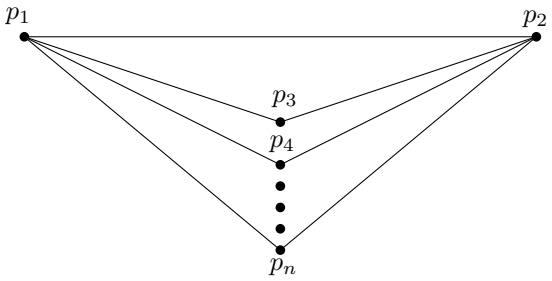


Figure 3.2: Query graph using triangles

The principal goal is to make α as small as possible. With this in mind, let us look at the more complicated and illuminating case when we have 4 points. Many different *ppg*'s are possible. We can dispense with those that have fewer than 4 edges since in these cases a unique placement is clearly not possible. Figure 3.3 below shows the possible *ppg*'s, up to relabelling, with 4 and 5 edges.

The *ppg* of Figure 3.3(a) is not rigid, for while the triangle formed by p_1, p_2 and p_3 is

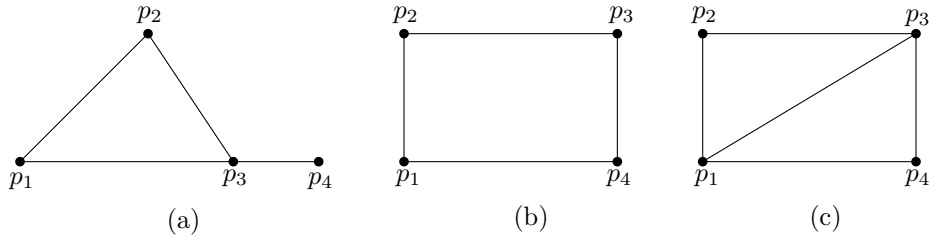


Figure 3.3: Some point placement graphs for 4 points

rigid, the point p_4 can be placed to the left or right of p_3, making the placement non-unique. The *ppg* of Figure 3.3(b) is interesting in that if the two pairs of opposite edges are equal then there is no unique placement. This is easily seen by drawing the *ppg* as a rectangle as shown in Figure 3.4(a) below and then giving a horizontal right shear to the top edge p_2p_3 so that p_2 and p_3 lie on the same line as p_1 and p_4, giving us the linear configuration shown in Figure 3.4(b). A horizontal left shear produces the linear configuration shown in Figure 3.4(c), which cannot be obtained from the linear configuration of Figure 3.4(b) by translation and/or reflection.

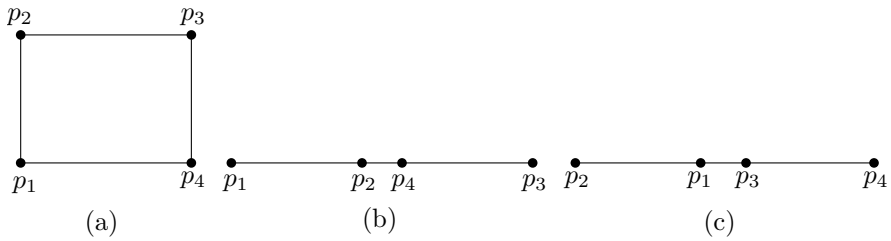


Figure 3.4: Point placement graph in the shape of a quadrilateral (a) with opposite edges being equal have 2 placements as shown in (b) and (c)

The *ppg* of Figure 3.3(c) is rigid since we have 2 triangles attached to the edge p_1p_3, each of which is rigid. Thus, it is the *ppg* of Figure 3.3(b) for which we have a structural rigidity condition, namely, $|p_1p_2| \neq |p_3p_4|$ or $|p_2p_3| \neq |p_1p_4|$ [20]. This means that if we want to

extend our previous algorithm for the unique placement of n points, by first placing two vertices, say, p_1 and p_2 on L and then building rigid quadrilaterals by querying distances from p_1 and p_2 with respect to two new vertices at a time, we must make sure that we meet the structural condition on the rigidity of each new quadrilateral.

If we try to construct a quadrilateral *ppg* (Figure 3.3(b)) in one round, its edges may not satisfy its rigidity condition, because all its 4 edges are connected and we cannot choose a suitable length for an edge to satisfy the rigidity condition on it. Suppose we want to satisfy the rigidity condition $|p_2p_3| \neq |p_1p_4|$. This is possible if the edges p_2p_3 and p_1p_4 are not paired. Then we will have options for choosing some suitable length for either p_2p_3 or p_1p_4. Suppose we want to provide this option for p_2p_3. Then for each p_1p_4 we must have candidate edges for p_2p_3. If we do not query p_3p_4 in the first round, then p_2p_3 and p_1p_4 are not paired, and it is possible to provide candidate edges for p_2p_3 after the first round of query. It is to be noted that the quadrilateral will be rigid irrespective of the lengths of the edges p_1p_2 or p_3p_4. So, there is no problem in querying for the length of the edge p_1p_4 in the second round.

So, we need to query the lengths of the edges in 2 rounds to build a rigid *ppg* using quadrilateral as the basic component. After the first round of query, we can select p_2p_3 with a suitable length and can check that the rigidity condition on it is satisfied.

Here is a 2-round algorithm due to Damaschke [26]. Let the number of points be $n = 2b + 4$, where b is a positive integer. In the first round, we make $2b + 3$ distance

queries represented by the edges in the graph in Figure 3.5. There are b leaf children p_i $(i = 3, ..., b + 2)$ rooted at p_1 and $b + 2$ leaf children p_j $(j = b + 3, ..., 2b + 4)$ rooted at p_2.

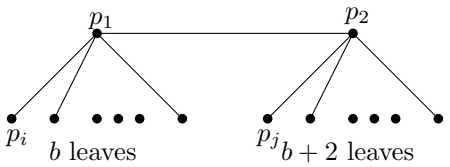


Figure 3.5: Query graph for first round in a 2-round algorithm using quadrilaterals

In the second round, for each edge p_1p_i $(i = 3, ..., b + 2)$ we find an edge p_2p_j rooted at p_2 satisfying the rigidity condition $|p_1p_i| \neq |p_2p_j|$. We can ensure this condition by having 2 extra edges at p_2, in view of the following basic observation [27]:

Observation 4 *At most two equal length edges can be incident to any vertex in a ppg.*

By Observation 4, there are at most 2 edges p_2p_j such that $|p_1p_i| = |p_2p_j|$. So, for each edge p_1p_i, an edge p_2p_j will always be found such that $|p_1p_i| \neq |p_2p_j|$. Then in the second round of query, for each i $(i = 3, ..., b+2)$, we query the distance p_ip_j to form a quadrilateral $p_1p_ip_jp_2$. It will be rigid since $|p_1p_i| \neq |p_2p_j|$. It will fix the positions of p_i and p_j relative to p_1 and p_2. For each of the 2 unused leaves p_j, the distance p_1p_j is queried in the second round to form the triangle $p_1p_jp_2$. It will fix the position of p_j relative to p_1 and p_2.

The number of queries made over the two rounds to construct this rigid *ppg* is $3b + 5$, i.e., $3n/2 - 1$. There are two noteworthy points: (a) the value of asymptotic density α is reduced from 2 for the first algorithm to 3/2 for the second, and (b) there is a price for this - we have to query the edges in two rounds. It is interesting to note that the density of the

quadrilateral *ppg* is 1, but it is not intrinsically rigid.

What if the number of points is greater than 6 but odd? Let $n = 2b + 5$, where b is a positive integer. We make an unique placement of the first $2b + 4$ vertices using the above algorithm, and query the distances of the last odd vertex from any two vertices. Distance queries for this vertex can be made in either of the 2 rounds.

3.1.2 Motivation

The motivation for studying this problem stems from the fact that it arises in diverse areas of research such as computational biology, learning theory, computational geometry, etc.

In learning theory [26] this problem is one of learning a set of points on a line non-adaptively, when learning has to proceed based on a fixed set of given distances, or adaptively when learning proceeds in rounds, with the edges queried in one round depending on those queried in the previous rounds.

The version of this problem studied in computational geometry is known as the turnpike problem. The description is as follows. On an expressway stretching from town A to town B there are several gas exits; the distances between all pairs of exits are known. The problem is to determine the geometric locations of these exits. This problem was first studied by Skiena *et al.* [63] who proposed a practical heuristic for the reconstruction. A polynomial time algorithm was given by Daurat *et al.* [28].

In computational biology, it appears in the guise of the restriction site mapping problem. Biologists discovered that certain restriction enzymes cleave a DNA sequence at specific

sites known as restriction sites. For example, it was discovered by Smith and Wilcox [65] that the restriction enzyme Hind II cleaves DNA sequences at the restriction sites GTGCAC or GTTAAC. In lab experiments, by means of fluorescent in situ hybridization (FISH experiments), biologists are able to measure the lengths of such cleaved DNA strings. Given the distances (measured by the number of intervening nucleotides) between all pairs of restriction sites, the task is to determine the exact locations of the restriction sites.

The turnpike problem and the restriction mapping problem are identical, except for the unit of distance involved; in both of these we seek to fit a set of points to a given set of interpoint distances. As is well-known, the solution may not be unique and the running time is polynomial in the number of points. While the point placement problem, prima facie, bears a resemblance to these two problems it is different in its formulation - we are allowed to make pairwise distance queries among a distinct set of labeled points. It turns out that it is possible to determine a unique placement of the points up to translation and reflection in time that is linear in the number of points.

The 3-dimensional version of this problem has application in the area of molecular conformation. Often, the experimental data about the conformational state of molecules are available in terms of interatomic distances. Majority of energy functions can also be expressed in terms of interatomic distances. The problem is to determine the conformational space of a molecule from these distance data and chirality constraints.

3.1.3 Prior Work

Early research on this problem was reported in [58, 53]. In this chapter, our first principal reference is [26], where it was shown that both the jewel and $K_{2,3}$ are rigid, and also how to build large rigid *ppg* of density 8/5 out of the jewel. A jewel is a graph with the set of vertices $\{X, Y, Z, A, B, P, Q\}$ and the set of edges $\{YZ, XA, AY, YB, BX, XP, PZ, ZQ, QX\}$ (see Figure 3.6). A $K_{2,3}$ is a graph with the set of vertices $\{X, Y, Z, A, B\}$ and the set of edges $\{XA, YA, ZA, XB, YB, ZB\}$. In a subsequent paper, Damaschke [27] proposed a randomized 2-round strategy that needs $(1 + o(1))n$ distance queries with high probability and also showed that this is not possible with 2-round deterministic strategies.

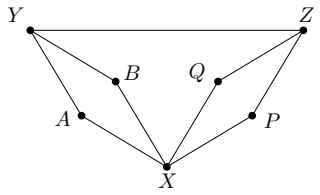


Figure 3.6: A jewel

Our second principal reference is the work of Chin *et al.* [20] who improved many of the results of [26]. Their principal contributions are the 2-round and 3-round construction of rigid graphs of density 4/3 and 5/4 using respectively 5-cycle and 6-cycle as the basic component, and a lower bound on the number of queries necessary in any 2-round algorithm. They also introduced the idea of a layer graph which is useful in finding the conditions for rigidity of a *ppg*. A layer graph is defined as follows:

Definition 3.1.1 *We first choose two orthogonal directions* **x** *and* **y** *(actually, any 2 non-*

parallel directions will do). A graph G admits a layer graph drawing if the following 4 properties are satisfied:

P1 Each edge e of G is parallel to one of the two orthogonal directions $\mathbf{x}$ and $\mathbf{y}$.

P2 The length of an edge e is the distance between the corresponding points on L.

P3 Not all edges are along the same direction (thus a layer graph has a two-dimensional extent).

P4 When the layer graph is folded onto a line, by a rotation either to the left or to the right about an edge of the layer graph lying on this line, no two vertices coincide.

Chin *et al.* [20] proved the following result about a layer graph:

Theorem 23 *A ppg is rigid iff it cannot be drawn as a layer graph.*

3.1.4 Contribution

In this chapter, we show how to construct in 2 rounds a rigid *ppg* on n points, using an instance of a 5:5 jewel as the basic component. The number of edges queried during this construction is $10n/7 + O(1)$. We extend this result to 6:6 jewels, constructing in 2 rounds a rigid *ppg* with $4n/3 + O(1)$ queries. This improves the result in [20] for constructing a *ppg* with $4n/3 + O(\sqrt{n})$ queries in 2 rounds using 5-cycles. We also improve substantially the lower bound on any 2-round algorithm from $17n/16$ in [20] to $12n/11$. In Chapter 4 we improve the lower bound and upper bound to $9n/8$ and $9n/7 + O(1)$ respectively. The results are summarized in Table 3.1.

Table 3.1: Comparison of results for lower bound and upper bound

	No of rounds	Upper bound	Lower bound
Damaschke [26, 27]	1	$8n/5 + O(1)$	$4n/3$
	2	$3n/2 + O(1)$	$30n/29$
Chin *et al.* [20]	2	$4n/3 + O(\sqrt{n})$	$17n/16$
	3	$5n/4 + O(\sqrt{n})$	
This dissertation	2	$10n/7 + O(1)$ (5:5 jewel) [5]	$9n/8$
	2	$4n/3 + O(1)$ (6:6 jewel) [3]	
	2	$9n/7 + O(1)$ (3-path)	

3.2 Generalized Jewels

The examples described in Section 3.1.1 demonstrates well how small *ppg*'s that are inherently rigid or rigid under some structural conditions can be glued together into a large rigid *ppg*. In this section we introduce a type of *ppg*, called an $m : n$ jewel, several copies of which we plan to glue together to form a large rigid *ppg*.

A generic $m : n$ jewel consists of an m-vertex cycle C_1 and another n-vertex cycle C_2 that are joined by a strut going between two vertices Y (of C_1) and Z (of C_2), and hinged at a third common vertex, X (Figure 3.7). An instance of an $m : n$ jewel is obtained by the placement of the vertices that describe the cycles C_1 and C_2.

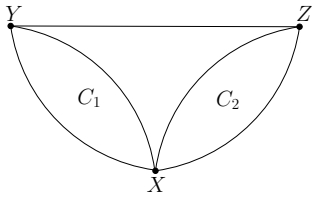


Figure 3.7: A generic $m : n$ jewel

To attain our goal we need to determine the structural conditions (if any) that make a chosen instance of the $m : n$ jewel rigid. In the next section, we obtain structural conditions

under which chosen instances of the $m : n$ jewels remain rigid for small values of m and n by drawing them as layer graphs and applying Theorem 23. Before we do that, we establish a few useful facts about the generic $m : n$ jewel. The first is as follows.

Theorem 24 *If cycles C_1 and C_2, consisting of m and n vertices respectively, are rigid then so is any $m : n$ jewel made up of these two cycles.*

Proof. Since C_1 and C_2 are rigid their respective vertices have unique linear layouts. Then in order for an $m : n$ jewel to have a layer graph drawing these placements would have to be in the orthogonal directions $\mathbf{x}$ and $\mathbf{y}$. Suppose the vertex Y is placed on the $\mathbf{x}$-axis and the vertex Z on the $\mathbf{y}$-axis, then the edge $\overline{YZ}$ of the $m : n$-jewel is not parallel to either the $\mathbf{x}$ or the $\mathbf{y}$ direction. Hence the $m : n$ jewel cannot be drawn as a layer graph and must, therefore, be rigid by Theorem 23. □

As a direct consequence of the theorem we have the following corollary:

Corollary 25 *If an $m : n$ jewel has a layer graph representation then in this representation at least one of C_1 or C_2 is a layer graph.*

In order to obtain the structural conditions that make a cycle rigid, we draw all possible layer graph representations of it and find the structural conditions for the rigidity of each of these. The logical AND of all these conditions is our answer. The second corollary is this:

Corollary 26 *The union of the set of all the structural conditions that make C_1 rigid with those that make C_2 rigid, constitute a sufficient set of structural conditions that make an $m : n$ jewel rigid.*

We shall take this route in the next two sections to obtain the structural conditions for the rigidity of chosen instances of the $m : n$ jewels for some small values of m and n.

It should be noted that a cycle with a fixed set of n_x **x**-parallel edges and thus a fixed set of n_y **y**-parallel edges can be drawn as a layer graph in different ways. They are all considered to be equivalent. For example, the three layer graph drawings of a 5-cycle in Figure 3.8 are considered to be equivalent. From now on, for an equivalent class of layer graphs we shall draw just one of them - not all. We shall not use the term class either. By a particular layer graph, we shall mean the class of layer graphs that are equivalent to it. Thus, two layer graph drawings of an n-vertex cycle are *distinct* from each other if at least one edge has different orientations in the two graphs.

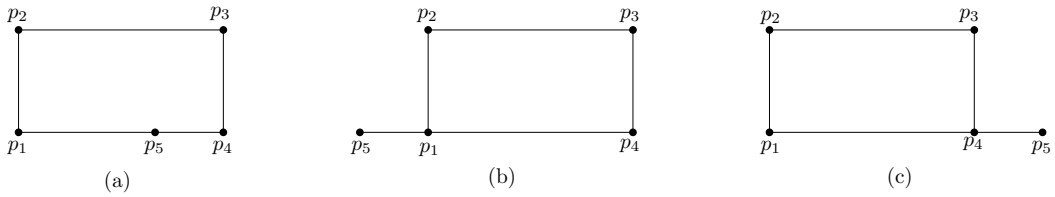


Figure 3.8: Equivalent layer graphs for a class of layer graphs of a 5-cycle

As we shall resort to exhaustive enumerations of all the layer graph representations of a cycle, the following theorem [5] is useful for checking that we have the correct number.

Theorem 27 *There are $2^{n-1} - \frac{n^2-n+2}{2}$ different layer graph representations of an n-vertex*

cycle.

3.2.1 4:4 and 5:4 Jewels

The following observation is fundamental. A formal proof can be found in [26].

Observation 5 *A 4-cycle $XAYB$ is rigid if $|XA| \neq |YB|$ or $|XB| \neq |YA|$.*

The jewel in Figure 3.6 has two 4-cycles joined together. It is an instance of a generic $4:4$ jewel.

To begin with, we prove the following theorem.

Theorem 28 *The 4:4 jewel of Figure 3.6 is rigid.*

Proof. We claim that cycles $XAYB$ and $XQZP$ are both rigid. Let the edge YZ is **x**-parallel. Three cases arise:

Case 1 The 4-cycle $XAYB$ is rigid, while the 4-cycle $XQZP$ has a layer graph representation.

Since $XQZP$ is a 4-cycle evidently its layer graph can be a rectangle only. Let the vertices of the rigid 4-cycle $XAYB$ lie on the **x**-parallel line through Y. Then for the rectangular layer graph $XQZP$ the diagonally opposite vertices X and Z lie on an **x**-parallel line collinear with YZ. Consequently, $XQZP$ cannot have a 2-dimensional extent. This violates property P1 of a layer graph. Thus the 4-cycle $XQZP$ cannot be drawn as a layer graph.

To complete the argument assume that the vertices of the rigid 4-cycle $XAYB$ lie on the **y**-parallel line through Y. Then the only way we can draw the 4-cycle $XQZP$ as a layer graph such that X and Z are non-adjacent is to place one of the two vertices P or Q at Y. As this violates property P4 that a layer graph should have, the 4-cycle $XQZP$ can not be drawn as a layer graph.

Thus the 4-cycle $XQZP$ does not have a layer graph representation when the 4-cycle $XAYB$ is rigid.

Case 2 An identical argument as in Case 1 proves that a layer graph representation of the 4-cycle $XAYB$ is impossible when the 4-cycle $XQZP$ is rigid.

Case 3 Finally, assume both the 4-cycles have layer graph representations.

Evidently, each of these is a rectangle only. As X and Y are non-adjacent vertices, they are diagonally opposite vertices of the rectangle $XAYB$. Likewise, X and Z are diagonally opposite vertices of the rectangle $XQZP$.

The arguments adduced for Case 1 can once again be used to show that it is not possible to draw the 4-cycle $XAYB$ as a layer graph if X lies on the **x**- or **y**-parallel lines passing through Y or on any of the **x**- or **y**-parallel lines passing through Z.

Assume otherwise. Now, X and Y are diagonally opposite vertices of the rectangle $XAYB$ while X and Z are diagonally opposite vertices of the rectangle $XQZP$. Therefore a vertex of the 4-cycle $XAYB$ must coincide with a vertex of the 4-cycle

$XQZP$ on an **x**-parallel line collinear with YZ. As this violates property P4 that a layer graph should have, the cycles $XQZP$ and $XQZP$ cannot have simultaneous layer graph representations.

Thus, none of the two 4-cycles of the jewel has a layer graph representation. By Theorem 23 both the cycles are rigid, and by Theorem 24 the 4:4 jewel is rigid. □

Unlike the 4:4 jewel of Figure 3.6, the 5:4 jewel of Figure 3.9 is not intrinsically rigid. As a prelude to our discussion in the following sections, it is interesting to find the structural conditions (or simply conditions) that make it rigid.

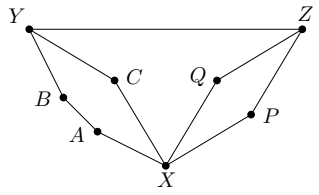


Figure 3.9: An instance of a 5 : 4 jewel

We first determine the conditions that make the cycle $XABYC$ rigid. By Theorem 27, there are five distinct layer graph representations of the 5-cycle $XABYC$, shown in Figure 3.10. As remarked earlier, each is a canonical representative of an entire class of layer graph representations; referring to Figure 3.10(a) for example, other representations can be obtained by varying the position of A on the supporting line of $\overline{XB}$.

It is impossible to extend the layer graph representations of the 5-cycle $XABYC$ shown in Figures 3.10(a) and 3.10(b) into a layer graph representation of the entire 5:4 jewel of Figure 3.9. without one of the vertices P or Q coinciding with one of the vertices B or C.

However, it is possible to extend each of the layer graph representations of Figures 3.10(c) - 3.10(e) into a layer graph representation of our 5:4 jewel. The layer graph representations of Figures 3.10(c) - 3.10(e) can be prevented by insisting on the condition $|XC| \neq |AB|, |XA| \neq |YB|, |YC| \neq |AB|$ respectively. By Theorem 23, these collectively constitute a set of sufficient conditions for the line rigidity of the 5-cycle $XABYC$.

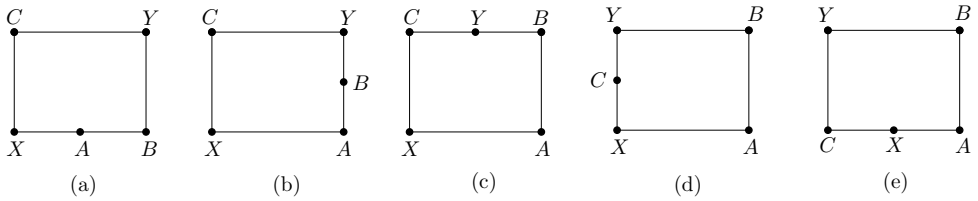


Figure 3.10: Different layer graph representations of a 5-cycle

For the 4-cycle $XPZQ$ the rigidity condition is $|XP| \neq |ZQ|$ (Observation 5). Thus, by Corollary 26, the set of sufficient conditions for the rigidity of the 5:4 jewel of Figure 3.9 is $\{|XC| \neq |AB|, |XA| \neq |YB|, |YC| \neq |AB|, |XP| \neq |ZQ|\}$.

We note in passing that for each of the configurations in Figures 3.10(c) - 3.10(e), we have an alternate condition that prevents its drawing as shown. Thus for example $|XA| \neq ||CY| \pm |YB||$ also prevents the layer graph drawing of Figure 3.10(c). With the help of the label mapping (X, C, Y, B, A) to $(p_3, p_4, p_5, p_1, p_2)$ we can see that this condition encapsulates the 3 different conditions corresponding to the 3 equivalent layer graph representations shown in Figure 3.8. In such situations, whenever possible, we choose the simpler condition, unless the other one is more useful for the construction of a *ppg*.

Theorem 29 *The 5:4 jewel of Figure 3.9 is rigid if its edges satisfy the set of conditions*

$\{|XC| \neq |AB|, |XA| \neq |YB|, |YC| \neq |AB|, |XP| \neq |ZQ|\}$.

3.3 Algorithm Based on a 5:5 Jewel

We next consider the more complex case of the 5:5 jewel of Figure 3.11. From now on, we will refer to it simply as the 5:5 jewel. By Theorem 27 there are exactly 5 distinct layer graph representations of a 5-cycle (see Figure 3.10). Thus, the set of 5 distinct conditions in Lemma 30 are sufficient to ensure the rigidity of the 5-cycle $XABYC$.

Lemma 30 *A 5-cycle $XABYC$ is rigid if its edges satisfy the following conditions:*

$$|XC| \neq |YB|, |XA| \neq |YC|, |XC| \neq |AB|, |XA| \neq |YB|, |YC| \neq |AB| \tag{3.1}$$

Proof. A formal proof appears in [20]. □

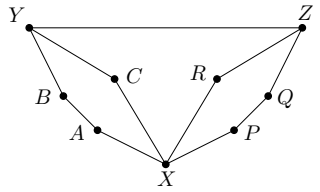


Figure 3.11: An instance of the 5:5 jewel

For the 5-cycle $XPQZR$ these conditions are:

$$|XR| \neq |ZQ|, |XP| \neq |ZR|, |XR| \neq |PQ|, |ZR| \neq |PQ|, |XP| \neq |ZQ|.$$

By Corollary 4, these 10 conditions collectively constitute a sufficient set of conditions for the line-rigidity of the 5:5 jewel.

Our goal is to glue several copies of the 5:5 jewel of Figure 3.11 into a large *ppg*, as we did for the case of quadrilaterals in Section 3.1.1. All of these will have a common strut YZ. As each jewel will account for 7 new vertices in lieu of 10 new edge queries, we expect α to be 10/7. This indeed turns out to be the case. The challenge here is to design the *ppg* in such a way that the rigidity conditions are satisfied for every jewel.

The rigidity conditions for a cycle, in their current form, involve all its edges. This requires to query the lengths of all of its edges in the first round to check if the rigidity conditions are satisfied. This does not provide us with the flexibility of choice that we need to meet the rigidity conditions in a 2-round algorithm. The edge lengths may not satisfy the conditions. If any condition is not satisfied then the cycle and thus the whole jewel may not be rigid because our set of conditions is sufficient (Theorem 24). Now, the 2-dimensional stretch of a layer graph gives a pointer - we can avoid involving one edge of a cycle from all the rigidity conditions for it. We shall avoid AB and PQ from the rigidity conditions for the two 5-cycles. Then the cycles will be rigid irrespective of the lengths of those edges. And the rigidity conditions for the cycles will involve all of their other edges. Again, in each rigidity condition we need to have at least one edge in it for which we can choose edge length, from among the options for edge lengths for that particular edge, that satisfies the condition. We shall provide options for choosing each of the edges YB and ZQ.

There will be rigidity conditions of each cycle that will not involve these edges, i.e., YB or ZQ. We cannot meet those rigidity conditions in a 2-round algorithm. We need to avoid

other edge(s) from the rigidity conditions of a cycle and/or provide options for choosing edge(s) for a cycle. We shall avoid XC and XR from the rigidity conditions for the two cycles. Then we shall have options for choosing edges YC and ZR to satisfy the rigidity conditions.

Thus, we shall avoid AB, PQ, XC and XR from the rigidity conditions. For each 5-cycle we shall replace each of its rigidity conditions that involve any of these edges. We shall replace that condition by a set of condition(s) that prevent the cycle from being drawn as the layer graph representation that corresponds to that condition.

Looking ahead slightly, Figure 3.18 describes the structure of our proposed *ppg*. It has a pool of edges hanging from each end of the strut YZ and a set of 2-pronged subgraphs. The lengths of the edges of this *ppg* are queried in the first round. In the second round, we join each 2-pronged subgraph to a pair of edges incident to Y and another pair of edges incident to Z to form a 5:5 jewel, making sure that all the rigidity conditions satisfied.

Over the rest of this section we show how to replace the rigidity conditions of the 5-cycle $XABYC$ that involve XC and/or AB with rigidity conditions that exclude these edges. To replace a condition we shall find another set of conditions that prevents the drawing of the 5-cycle $XABYC$ as a layer graph in the configuration corresponding to that condition. For example, to replace the condition $|XC| \neq |YB|$, corresponding to the layer graph of Figure 3.10)(a), we shall find a set of conditions that prevent the drawing of the layer graph of the 5-cycle in the configuration of Figure 3.10)(a).

Our first attempt will be to use other edges in the layer graph drawing corresponding to a given rigidity condition involving XC and/or AB. If this does not suit our purpose, the basic strategy will be to embed the layer graph drawing corresponding to such a rigidity condition into all possible layer graph drawings of the 5:5 jewel and derive a rigidity condition from each such embedding.

The rigidity conditions that we will consider for replacement are:

$$|XC| \neq |YB|,\ |XC| \neq |AB|,\ |YC| \neq |AB|$$

3.3.1 Replacing $|XC| \neq |AB|$

This condition has been derived from the layer graph drawing shown in Figure 3.10(c). This figure shows that an alternate rigidity condition is

$$|XA| \neq ||YB| \pm |YC||, \tag{3.2}$$

which we use to replace $|XC| \neq |AB|$.

3.3.2 Replacing $|XC| \neq |YB|$

This rigidity condition corresponds to the layer graph drawing of Figure 3.10(a). $||XA| \pm |AB|| \neq |YC|$ is an alternate rigidity condition corresponding to the layer graph drawing in Figure 3.10a) of the 5-cycle $XABYC$. However, it involves the edge AB that we wish to avoid. We shall find an alternate set of rigidity conditions. For this, we find all possible layer graph drawings of the 5:5 jewel in which the layer graph of Figure 3.10(a) is embedded. Then we find conditions which prohibit those layer graph drawings. Consequently, those

conditions will replace $|XC| \neq |YB|$, because there will be no layer graph for the 5:5 jewel in which the layer graph of Figure 3.10(a) is embedded. We shall follow this method whenever we cannot use any rigidity condition for a 5-cycle $XABYC$ or $XPQZR$ that involves some edges of the corresponding cycle only. We have the following lemma for the replacement of the current condition:

Lemma 31 *The 5-cycle $XABYC$ of the 5:5 jewel of Figure 3.11 cannot be drawn as the layer graph of Figure 3.10(a) if the edges of the jewel satisfy the following conditions:*

$$\{|ZR| \neq |YB|, |ZR| \neq |YC|\} \tag{3.3}$$

Proof. We argue below that there are exactly 4 possible layer graph drawings of the 5:5 jewel in which the layer graph of Figure 3.10(a) lies embedded. Two cases arise depending on the orientations of YZ:

- YZ is horizontal (Figure 3.12)

  Z is necessarily distinct from C, while YZ and YB are mutually perpendicular. Consider the edges on the path XRZ of the 5:5 jewel. If XR were vertical, then ZR would have to be horizontal, forcing R to coincide with C. Thus, XR must be horizontal and consequently, RZ must be vertical.

  Next, we consider the edges on the path $XPQZ$. XP can be horizontal or vertical. If XP is horizontal then PQ must be vertical, else Q and R will coincide. This forces

QZ to be horizontal giving us the layer graph of Figure 3.12(a).

If XP is vertical, then PQ must be horizontal; otherwise, Q will coincide with C. This forces QZ to be vertical, giving us the layer graph of Figure 3.12(b).

In these layer graphs, the edges YC and YZ are on a horizontal line CYZ, and are parallel to XR. The vertical edges XC and ZR connect the parallel edges. So, we must have $|XC| = |ZR|$. Thus, these layer graphs are not possible if $|ZR| \neq |YB|$.

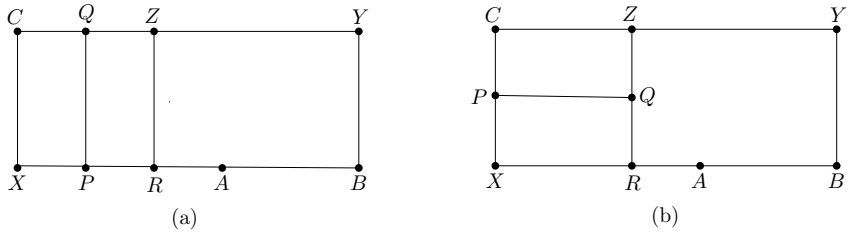


Figure 3.12: Replacing the condition $|XC| \neq |YB|$ when YB and YZ are mutually perpendicular

- YZ is vertical (Figure 3.13)

  Identical arguments as adduced for the case when YZ was assumed horizontal, gives us the layer graph drawings of Figure 3.13(a) and Figure 3.13(b).

  For both the configurations of Figure 3.13 the edges XC and XR are on a vertical line XRC, while the edges YB and YZ are on a vertical line BZY. The edge YC is horizontal and connects those two parallel lines. The edge ZR is horizontal and connects the two vertical lines XRC and BZY. So, we must have $|ZR| = |YC|$. Thus, these layer graphs are not possible if $|ZR| \neq |YC|$.

  It follows that there is no layer graph for the 5:5 jewel in which the layer graph in

Figure 3.10(a) of the 5-cycle $XABYC$ is embedded if the edges of the jewel satisfy Eq. 3.3. Hence, the 5-cycle $XABYC$ of the 5:5 jewel of Figure 3.11 cannot be drawn as the layer graph of Figure 3.10(a) if the edges of the jewel satisfy the conditions in Eq. 3.3.

□

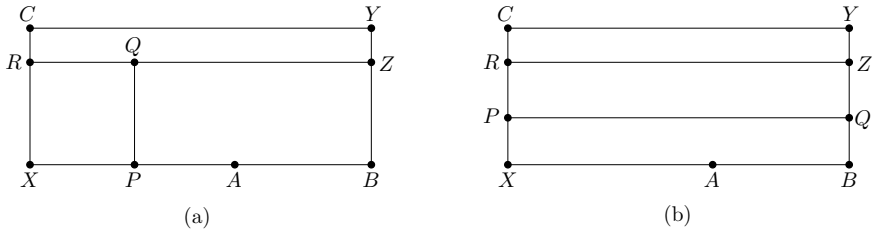


Figure 3.13: Replacing the condition $|XC| \neq |YB|$ when YB and YZ are collinear

3.3.3 Replacing $|YC| \neq |AB|$

This rigidity condition corresponds to the layer graph drawing of Figure 3.10(e). We argue below that there are exactly 12 possible layer graph drawings of the 5:5 jewel in which the layer graph of Figure 3.10(e) lies embedded. There are 2 main cases to consider.

- YZ is vertical and YB is orthogonal to it:

  The path XRZ is made up of a vertical segment XR, followed by a horizontal segment ZR, else R will coincide with C. If we consider the path XPQ, by a similar argument when XP is horizontal PQ must be vertical. If QZ were vertical, then P would have to coincide with C. Thus, QZ is horizontal. This gives us the layer graph drawing of Figure 3.14(a).

If XP is vertical, we can argue similarly as in the last paragraph that PQ must be horizontal and QZ vertical. This gives us the layer graph drawing of Figure 3.14(b).

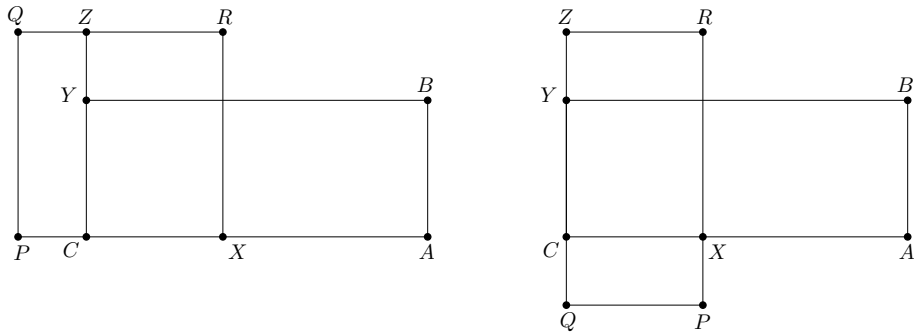


Figure 3.14: Replacing the condition $|YC| \neq |AB|$ when YZ and YB are perpendicular to each other. There is only one position for R.

$\{|YB| \neq ||XA| \pm |XC||\}$ is an alternate rigidity condition for the 5-cycle $XABYC$ with the layer graph drawing as in (Figure 3.10(e)). This condition however involves the edge XC that we wish to avoid. For both the layer graph drawings of Figure 3.14, YB and YZ being mutually perpendicular, the edges YC and YZ are on a line CYZ, and they are parallel to XR. So, we must have $|XC| = |ZR|$. Using this, we get the replacement rigidity condition $\{|YB| \neq ||XA| \pm |ZR||\}$.

- YB and YZ are collinear:

  3 subcases arise depending upon the orientations of ZR and XR.

  - ZR is perpendicular to YB and YZ, and XR is perpendicular to ZR (Figure 3.15):

    In this case there are 4 distinct placements of the edges XP, PQ and QZ giving

rise to 4 distinct layer graph drawings of the 5:5 jewel (Figure 3.15(a)-(d)).

In all the 4 layer graph drawings the edges YZ and XR are horizontal and collinear, while the edge ZR is vertical and connects those two parallel edges. The edges YB and XA are horizontal and collinear, while the edge AB is vertical and connects those two parallel edges. YZ and YB are collinear, and so are XR and XA. Therefore, we must have $|AB| = |ZR|$ and the replacement rigidity condition for this subcase is $|YC| \neq |ZR|$.

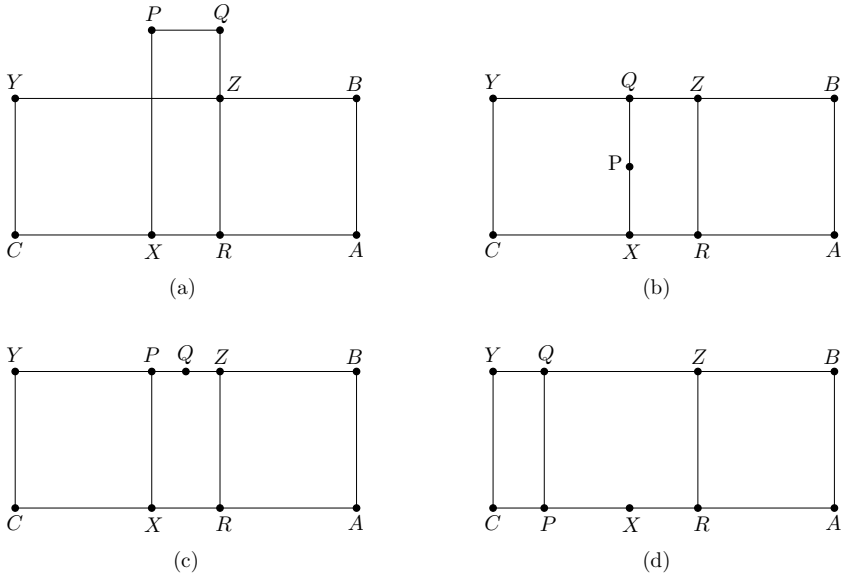


Figure 3.15: Replacing the condition $|YC| \neq |AB|$ when YB and YZ are collinear, ZR is perpendicular to BYZ and XR is perpendicular to ZR

– ZR is perpendicular to YB and YZ, and XR and ZR are collinear:

In this case XP, PQ and QZ can be placed in 2 distinct configurations (Figure 3.16). In these configurations of the jewel the 5-cycle $XABYC$ cannot be drawn as a layer graph in the present configuration if $||XA| \pm |XC|| \neq |YB|$. In both the configurations of the layer graph of the jewel YC and XRZ are parallel,

and both of XC and YZ connect them. We must have $|XC| = |YZ|$. We can rewrite the condition as $||XA| \pm |YZ|| \neq |YB|$ for this subcase.

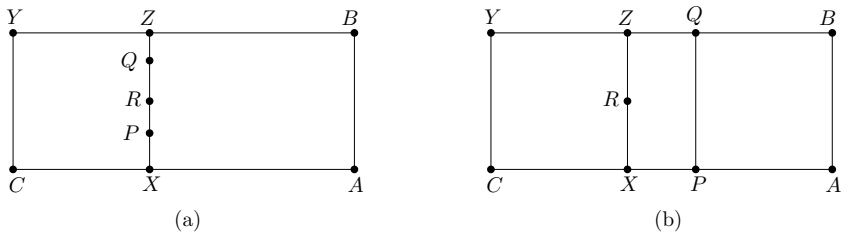


Figure 3.16: Replacing the condition $|YC| \neq |AB|$ when YB and YZ are collinear, and ZR and XR are perpendicular to BYZ

– ZR is collinear with YB and YZ (Figure 3.17):

  In this case, XR is necessarily perpendicular to ZR, while XP, PQ and QZ can be in 4 distinct configurations. In all of these, the 5-cycle $XABYC$ cannot be drawn as a layer graph in the present configuration if $||XA| \pm |XC|| \neq |YB|$. Since $|XC| = ||YZ| \pm |ZR||$ in all 4 layer graphs, the condition can be replaced by $||XA| \pm |YZ| \pm |ZR|| \neq |YB|$ for this subcase.

Thus, we have the following lemma.

Lemma 32 *The 5-cycle $XABYC$ of the 5:5 jewel of Figure 3.11 cannot be drawn as a layer graph in the configuration of Figure 3.10(e) if the edges of the jewel satisfy the following conditions:*

$$|YB| \neq ||XA| \pm |ZR||, |YC| \neq |ZR|, ||XA| \pm |YZ|| \neq |YB|, ||XA| \pm |YZ| \pm |ZR|| \neq |YB|.$$

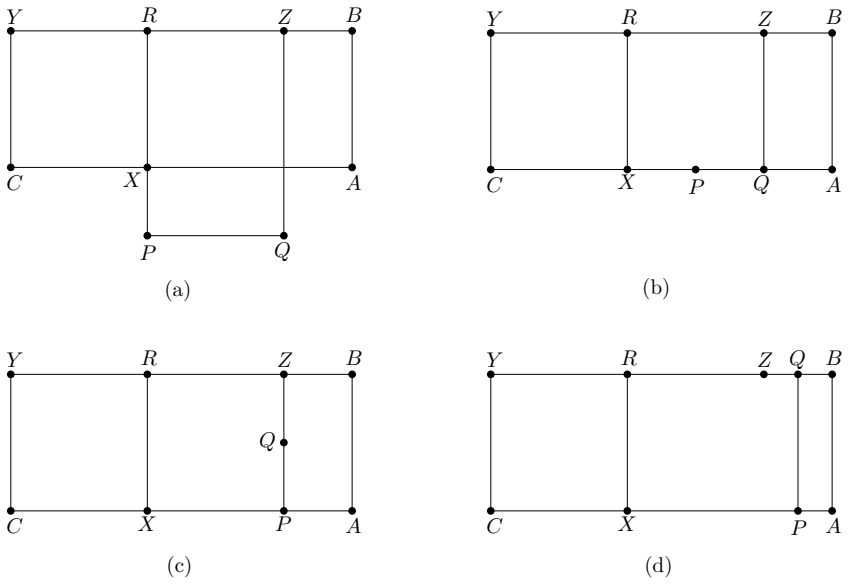


Figure 3.17: Replacing the condition $|YC| \neq |AB|$ when YB and YZ are collinear and ZR is collinear with them. XR can only be perpendicular to ZR.

3.3.4 Rigidity Conditions

From (1), (2), Lemma 31 and Lemma 32, we have the following result for the line-rigidity of the 5-cycle $XABYC$ of the 5:5 jewel:

Lemma 33 *The 5-cycle $XABYC$ of the 5:5 jewel $XABYCPQZR$ of Figure 3.11 is rigid if the edges of the jewel satisfy the following set of conditions:*

$$\{|ZR| \neq |YB|, |ZR| \neq |YC|, |XA| \neq |YC|, |XA| \neq ||YB| \pm |YC||, |XA| \neq |YB|,$$

$$|ZR| \neq ||YB| \pm |XA||, |ZR| \neq ||YB| \pm |XA| \pm |YZ||, |YB| \neq ||XA| \pm |YZ||\}.$$

We thus have an amplified set of sufficient conditions to satisfy.

Similarly, we have the following result for the line-rigidity of the other 5-cycle $XPQZR$ of the 5:5 jewel:

Lemma 34 *The 5-cycle $XPQZR$ of the 5:5 jewel $XABYCPQZR$ of Figure 3.11 is rigid if the edges of the jewel satisfy the following set of conditions:*

$$\{|YC| \neq |ZQ|, |YC| \neq |ZR|, |XP| \neq |ZR|, |XP| \neq ||ZQ| \pm |ZR||, |XP| \neq |ZQ|,$$

$$|YC| \neq ||ZQ| \pm |XP||, |YC| \neq ||ZQ| \pm |XP| \pm |YZ||, |ZQ| \neq ||XP| \pm |YZ||\}.$$

By Corollary 4, the union of the two sets of conditions in Lemmas 33 and 34 constitutes a set of sufficient conditions for the line rigidity of the 5:5 jewel of Figure 3.11. Taking care of one overlapping condition between the two sets of 8 conditions, we have 15 distinct conditions for the line-rigidity of the 5:5 jewel and hence the following lemma.

Lemma 35 *The 5:5 jewel $XABYCPQZR$ of Figure 3.11 is rigid if its edges satisfy the following set of conditions:*

1. $|YB| \notin \{|XA|, ||XA| \pm |YZ||\}$,
2. $|YC| \notin \{|XA|, ||YB| \pm |XA||\}$,
3. $|ZQ| \notin \{|XP|, |YC|, ||XP| \pm |YZ||, ||YC| \pm |XP||, ||YC| \pm |XP| \pm |YZ||\}$,
4. $|ZR| \notin \{|XP|, |YB|, |YC|, ||YB| \pm |XA||, ||YB| \pm |XA| \pm |YZ||, ||ZQ| \pm |XP||\}$.

In the next section we show how to construct a composite *ppg* made up of 5:5 jewels such that all the 15 rigidity conditions listed above are satisfied for each one of these.

3.3.5 Algorithm

We use a pair of vertices $\{Y, Z\}$ as reference vertices. We query the edge length $|YZ|$ and the pairwise distances of some other suitable vertices in the first round. All the vertices will be placed relative to Y and Z. Now we consider the second round. We select vertices in groups of 7 vertices each in such a way that the pairwise distances of the union of each group of vertices $\{X, A, B, C, P, Q, R\}$ and $\{Y, Z\}$ satisfy the conditions in Lemma 35. Then we query the remaining necessary pairwise distances of the union to form a 5:5 jewel. The jewel will be rigid by Lemma 35 irrespective of the lengths of the edges AB, CX, PQ and RX, since no condition of the lemma involves any of these edges. The unused vertices are made rigid by using triangle as the *ppg*.

Algorithm 3.1. First a bit of nomenclature. To indicate the affiliations of the vertices X, A, B, C, P, Q, R to different copies of a 5:5 jewel, we use the following indexing scheme: $X \to X_i$, $A \to A_i$, $B \to B_j$, $C \to B_k$, $P \to P_i$, $Q \to Q_m$ and $R \to Q_l$.

Let the number of points be $n = 7b + 30$, where b is a positive integer. In the first round, we make $6b+29$ distance queries represented by the edges in the graph in Figure 4.7. There are $2b + 6$ leaf children $B_j (j = 1, ..., 2b + 6)$ rooted at Y and $2b + 22$ leaf children $Q_l (l = 1, ..., 2b + 22)$ rooted at Z. The remaining $3b$ vertices are organized into groups of 3 as (A_i, X_i, P_i) $(i = 1, ..., b)$ and the distances $|A_iX_i|$ and $|X_iP_i|$, $(i = 1, ..., b)$ are queried.

In the second round, for each 2-link (A_iX_i, X_iP_i) we find a pair of edges YB_j and YB_k, rooted at Y satisfying Conditions 1 and 2 of Lemma 35; next, we find a pair of edges ZQ_m

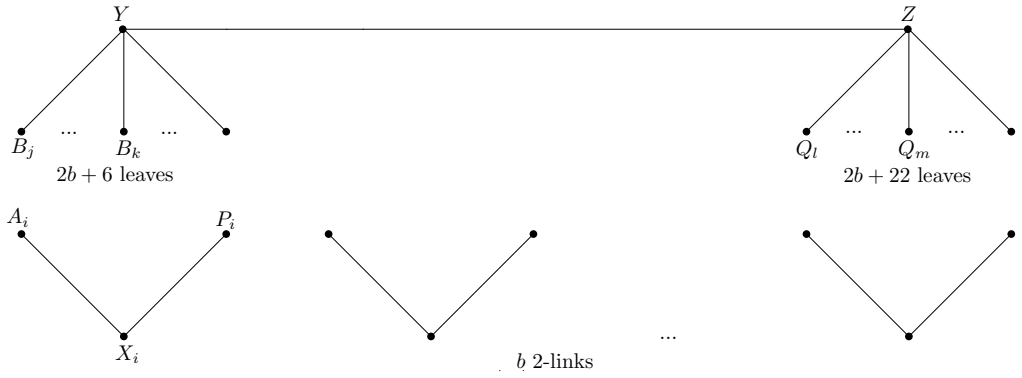


Figure 3.18: Queries in the first round for 2-round algorithm using 5:5 jewel as the basic component

and ZQ_l, rooted at Z satisfying Conditions 3 and 4 of Lemma 35.

Then for each $i, (i = 1, ..., b)$, we query the distances $|A_iB_j|$, $|X_iB_k|$, $|X_iQ_l|$ and $|P_iQ_m|$ to form a 5:5 jewel $X_iA_iB_jYB_kP_iQ_mZQ_l$. Its edges will satisfy all the rigidity conditions of Lemma 35.

For each of the 6 unused leaves B_j of the tree rooted at Y, we query the distance $|B_jZ|$ to form the triangle YB_jZ. Likewise, for each of the 22 unused leaves Q_l of the tree rooted at Z we query the distance $|Q_lY|$ to form the triangle YQ_lZ. □

The following theorem establishes the correctness of our algorithm.

Theorem 36 *The ppg constructed by Algorithm 3.1 is rigid.*

Proof. Let us consider an arbitrary 2-link (P_iX_i, X_iA_i). We show that the 5:5 jewel constructed by Algorithm 1 using the edges of this 2-link is rigid.

Let us consider the selection of the edge YB_j for the jewel in the second round. From

Condition 1 of Lemma 35, $|YB_j|$ cannot be equal to $|X_iA_i|$, $||X_iA_i|+|YZ||$ or $||X_iA_i|-|YZ||$. By Observation 4 there can be at most 2 edges rooted at Y that are equal to a given length. Hence there are at most 6 edges rooted at Y that do not qualify to be chosen as YB_j. By adding 6 extra leaves at Y we provide the room needed to choose YB_j for each of the 2-links (P_iX_i, X_iA_i) with $i = 1, ..., b$, so that the rigidity conditions on this edge are satisfied.

An identical argument shows that the 6 additional leaves at Y enables us to choose YB_k in the second round so that the rigidity conditions on this edge are satisfied for each of the 2-links (P_iX_i, X_iA_i) with $i = 1, ..., b$.

Consider next the selection of the edge ZQ_m for the jewel in the second round. From Condition 3 of Lemma 35, $|ZQ_m|$ cannot be equal to $|X_iP_i|$, $|YB_k|$, $|X_iP_i|+|YZ|$, $||X_iP_i|-|YZ||$, $|YB_k|+|YZ|$, $||YB_k|-|YZ||$, $|X_iP_i|+|YB_k|+|YZ|$, $||X_iP_i|-|YB_k|+|YZ||$, $||X_iP_i|+|YB_k|-|YZ||$, $||X_iP_i|-|YB_k|-|YZ||$. Again from Observation 1 it follows that there are at most 20 edges rooted at Y that do not qualify to be chosen as ZQ_m. Adding 22 extra leaves at Z provides us with the room needed to choose ZQ_m for each of the 2-links (P_iX_i, X_iA_i) with $i = 1, ..., b$, so that the rigidity conditions on this edge are satisfied.

There will be at most 20 edges ZQ_m rooted at Z that do not satisfy the conditions on it as stated in Lemma 35 (by Observation 4). In addition to the $2b$ edges necessary to construct the b jewels there are 22 extra edges rooted at Z. So, for each set of 2-link (P_iX_i, X_iA_i) and 3-link (B_jY, B_kY, YZ) with $i = 1, ..., b$ (B_jY depends on P_iX_i and X_iA_i, and B_kY depends on P_iX_i, X_iA_i and B_jY), we can always find an edge YQ_m that satisfies

the condition on it as stated in Lemma 35.

Finally, consider the second-round selection of the edge ZQ_l for the jewel. From Condition 4 of Lemma 35 there are 11 rigidity conditions on $|ZQ_l|$, and hence by Observation 4 will be at most 22 edges ZQ_l rooted at Z are not eligible to be chosen. In addition to the $2b$ edges necessary to construct the b jewels there are 22 extra edges rooted at Z. So, for each set of 2-link (P_iX_i, X_iA_i) and 4-link (B_jY, B_kY, YZ, ZQ_m) with $i = 1, ..., b$ the 22 extra edges rooted at Z provide us with the latitude to find an edge ZQ_l always that satisfies the rigidity conditions on it.

So, for each 2-link (A_iX_i, X_iP_i) we can always find edges YB_j, YB_k, ZQ_l and ZQ_m for the 5:5 jewel of Figure 3.11 such that the conditions for rigidity (Lemma 35) are satisfied. Each of the b 5:5 jewels of Figure 3.11 with YZ as an edge is constructed in the second round by satisfying the rigidity conditions of Lemma 35. So, they are rigid and, for each $i, (i = 1, ..., b)$, the positions of $X_i, A_i, B_j, B_k, P_i, Q_m$ and Q_l are fixed relative to Y and Z. Each of the remaining 6 leaves of Y forms a triangle (YB_j, B_jZ, ZY) with YZ as an edge. So, their positions are fixed relative to Y and Z. Each of the remaining 22 leaves of Z forms a triangle (ZQ_l, Q_lY, YZ) with YZ as an edge. So, their positions are fixed relative to Y and Z.

Hence, the whole *ppg* is rigid. □

Theorem 37 $10n/7 + 99/7$ *queries are sufficient to place n distinct points on a line in two rounds.*

Proof. We need $6b + 29$ queries in the first round and $4b + 28$ queries in the second round. In total $10b + 57$ pairwise distances are to be queried for the placement of $7b + 30$ points. We have $10b + 57 = 10/7 * (7b + 30) - 300/7 + 57 = 10n/7 + 99/7$. □

It is worth noting that our algorithm needs at least 37 points to work. When we have fewer points we can switch to the quadrilateral algorithm, described in the Introduction. The 2-round 5-cycle algorithm of Chin *et al.* [20] a total of $4/3n + 34/3\sqrt{n}$ queries for the placement of n points. Thus our 5:5 jewel algorithm does better when $n \leq 4076$. This provides the motivation for considering 6:6 jewels, which we do next.

3.4 Algorithm Based on a $6:6$ Jewel

The principal ideas underlying this algorithm are similar to the algorithm based on 5:5 jewel of the last section. So we will skip the repetitive details when there is no scope for confusion.

Figure 3.19 shows the *ppg* for an instance of the 6:6 jewel that we shall use in the construction of our composite *ppg*. For brevity we will refer to left cycle as C_1 and the right cycle as C_2, and by 6:6 jewel we will mean the instance shown.

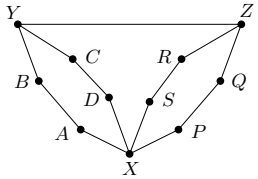


Figure 3.19: A $6:6$ jewel

By Theorem 27, the 6-cycle $XABYCD$ has 16 different layer graph representations

(Figure 3.20), giving us the following 16 conditions for its line-rigidity. The layer graphs can be grouped into 4 groups depending on the number of edges on each side:

1. $|YC| \neq |XD|, |YB| \neq |CD|, |YC| \neq |AB|, |YB| \neq |XA|, |XD| \neq |AB|, |XA| \neq |CD|$,
2. $|YB| \neq |XD|, |AB| \neq |CD|, |YC| \neq |XA|$,
3. $|YB| \neq ||YC| \pm |XD||, |YC| \neq ||XA| \pm |XD||, |YB| \neq ||XD| \pm |CD||, |YB| \neq ||XA| \pm |XD||, |YB| \neq ||YC| \pm |XA||, |XA| \neq ||YC| \pm |CD||$,
4. $|XA| \neq ||YB| \pm |CD||$.

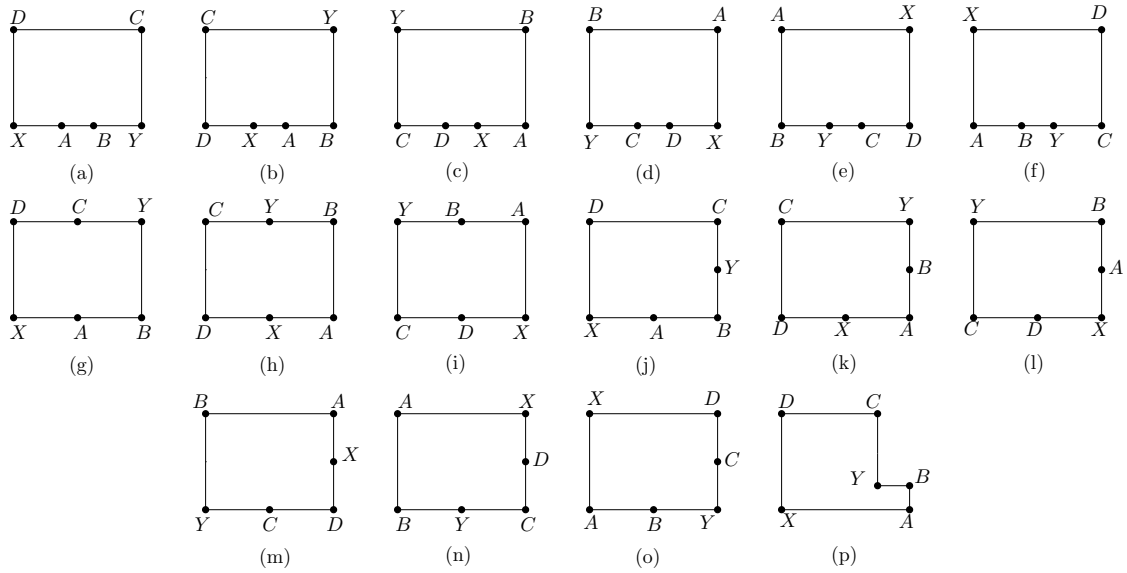


Figure 3.20: Different layer graph representations of a 6-cycle

Similarly, we have another set of 16 conditions for the line-rigidity of the cycle C_2, viz.,

1. $|ZR| \neq |XS|, |ZQ| \neq |RS|, |ZR| \neq |PQ|, |ZQ| \neq |XP|, |XS| \neq |PQ|, |XP| \neq |SR|$,
2. $|ZQ| \neq |XS|, |ZR| \neq |XP|, |PQ| \neq |RS|$,

3. $|ZQ| \neq ||ZR| \pm |XS||, |ZR| \neq ||XP| \pm |XS||, |ZQ| \neq ||XS| \pm |RS||, |ZQ| \neq ||XP| \pm |XS||, |ZQ| \neq ||ZR| \pm |XP||, |XP| \neq ||ZR| \pm |RS||,$

4. $|XP| \neq ||ZQ| \pm |RS||$.

By Corollary 26, the conjunction of these two sets of conditions constitutes a set of sufficient conditions for the line-rigidity of the 6:6 jewel above.

3.4.1 Replacing Conditions

We would like to make the 6:6 jewel rigid irrespective of the lengths of the edges AB, CD, PQ and RS, as this allows us to query the remaining edges in such a way that the rigidity conditions are satisfied. Towards this goal, we reformulate 16 conditions (8 from each cycle) involving these edges with alternate sets of conditions, satisfying which we also satisfy the replaced ones.

We use the left cycle, $C_1 = XABYCD$, as a running example to demonstrate these replacements.

Replacing $|AB| \neq |CD|$:

The layer graph for the 6-cycle C_1 corresponding to this condition is shown in Figure 3.20(h). From the figure it is evident that we can replace this with the condition

$$||YB| \pm |YC|| \neq ||XA| \pm |XD|| \tag{3.4}$$

since this will also prevent the layer graph drawing of the cycle as in Figure 3.20(h).

Replacing $|XA| \neq |CD|$:

The layer graph of C_1 corresponding to this condition is shown in Figure 3.20(f). To replace this condition we follow a similar strategy as for the 5:5 jewel, except for a small twist: we draw all possible layer graphs of the 6:6 jewel, excluding the chain $XSRZ$, in which the layer graph of Figure 3.20(f) is embedded. The condition $|XA| \neq |CD|$ is then amplified into the set of conditions that prevent the drawing of the layer graph representation of the 6-cycle corresponding to this condition (Figure 3.20(f)). Two cases arise, depending on whether YZ is horizontal or vertical.

- YZ is horizontal:

  Here Z and X have different x and y coordinates. XP, PQ and QZ can have 4 different orientations as shown in Figures 3.21(a) - 3.21(d). The following conditions will prevent the layer graph drawings of the 6-cycle $XABYCD$ in Figure 3.20(f), when YZ is horizontal: $|ZQ| \neq ||XA| \pm |XP||$ (Figure 3.21(a)), $||YC| \pm |YZ|| \neq ||XD| \pm |XP||$ (Figure 3.21(b)), $|ZQ| \neq |XA|$ (Figure 3.21(c)) and $||ZQ| \pm |YC| \pm |YZ|| \neq ||XD| \pm |XP||$ (Figure 3.21(d)).

- YZ is vertical and $|YZ| = |XA|$:

  In this case only one layer graph is possible as shown in Figure 3.22. We can replace $|XA| \neq |CD|$ with $|YZ| \neq |XA|$. This will prevent the layer graph drawing of the 6-cycle $XABYCD$ in Figure 3.20(f) when YZ is vertical and $|YZ| = |XA|$.

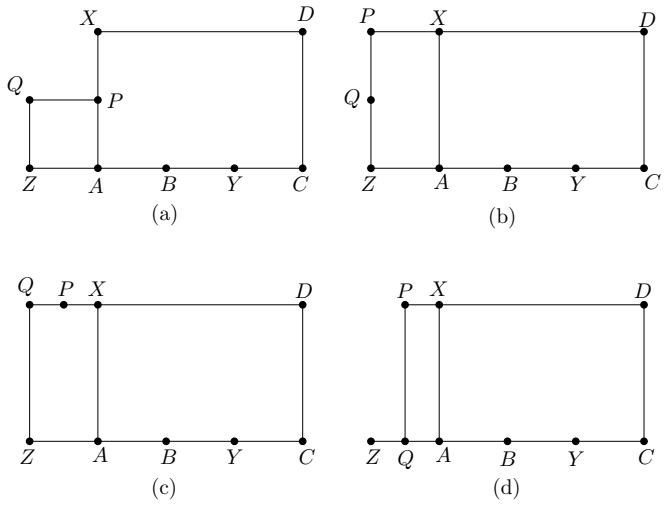


Figure 3.21: Replacing condition $|XA| \neq |CD|$ when YZ is horizontal

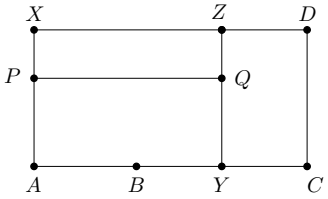


Figure 3.22: Replacing condition $|XA| \neq |CD|$ when YZ is vertical and $|YZ| = |XA|$

- YZ is vertical and $|YZ| \neq |XA|$:

  Here Z and X have different x and y coordinates. XP, PQ and QZ can have 6 different orientations as shown in Figure 3.23(a) - 3.23(f). These layer graphs give rise to the following set of conditions that prevents the layer graph drawing of the 6-cycle $XABYCD$ as in Figure 3.20(f), when YZ is vertical and $|YZ| \neq |XA|$: $||ZQ| \pm |YZ|| \neq |XA|$ (Figure 3.23(a)), $|YC| \neq ||XD| \pm |XP||$ (Figure 3.23(b)), $||ZQ| \pm |YZ|| \neq ||XA| \pm |XP||$ (Figure 3.23(c)), $||ZQ| \pm |YC|| \neq ||XD| \pm |XP||$ (Figure 3.23(d)), $||ZQ| \pm |YC|| \neq |XD|$ (Figure 3.23(e)) and $|YZ| \neq ||XA| \pm |XP||$ (Figure 3.23(f)).

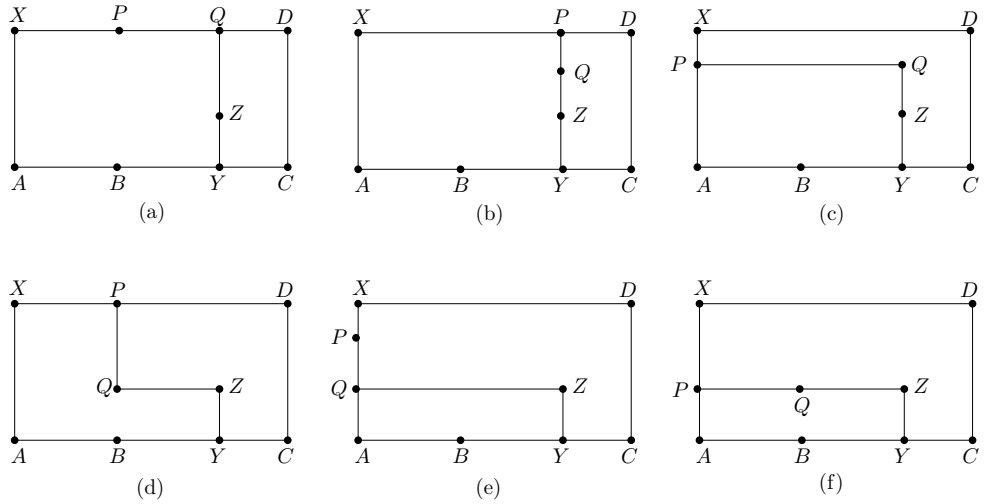


Figure 3.23: Replacing condition $|XA| \neq |CD|$ when YZ is vertical and $|YZ| \neq |XA|$

Thus, we have the following lemma, the proof of which is similar to the proof of Lemma 31 and is omitted:

Lemma 38 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(f) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||XA| \pm |XP||, ||YC| \pm |YZ|| \neq ||XD| \pm |XP||, |ZQ| \neq |XA|, ||ZQ| \pm |YC| \pm |YZ|| \neq ||XD| \pm |XP||, |YZ| \neq |XA|, ||ZQ| \pm |YZ|| \neq |XA|, |YC| \neq ||XD| \pm |XP||, ||ZQ| \pm |YZ|| \neq ||XA| \pm |XP||, ||ZQ| \pm |YC|| \neq ||XD| \pm |XP||, ||ZQ| \pm |YC|| \neq |XD|, |YZ| \neq ||XA| \pm |XP||\}$.

Replacing $|XD| \neq |AB|$:

The layer graph of the 6-cycle corresponding to this condition is as shown in Figure 3.20(e). This layer graph is the same as that in Figure 3.20(f) if we interchange A with D and B

with C. By this interchange of the labels in Lemma 38 we have the following lemma for the replacement of condition:

Lemma 39 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(e) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||XD| \pm |XP||, |YB| \neq ||YZ| \pm |XA| \pm |XP||, |ZQ| \neq ||YB| \pm |YZ| \pm |XA| \pm |XP||, |ZQ| \neq |XD|, |YZ| \neq |XD|, |ZQ| \neq ||YZ| \pm |XD| \pm |XP||, |ZQ| \neq ||YB| \pm |XA||, |YZ| \neq ||XD| \pm |XP||, |ZQ| \neq ||YB| \pm |XA| \pm |XP||, |ZQ| \neq ||YZ| \pm |XD||, |YB| \neq ||XA| \pm |XP||\}$.

Replacing $|YC| \neq |AB|$:

The layer graph of the 6-cycle corresponding to this condition is as shown in Figure 3.20(c). Figure 3.24 shows all the possible layer graphs of the 6:6 jewel, excluding the chain $XSRZ$, in which the layer graph of Figure 3.20(c) is embedded (different configurations for P and Q are combined in the same figure). From Figure 3.24 we see that the condition $|YC| \neq |AB|$ can be replaced by the following conditions:

- $|ZQ| \neq ||YC| \pm |XP||$, $|ZQ| \neq ||YB| \pm |YZ| \pm |XA||$, $|YC| \neq |XP|$, $|ZQ| \neq ||YB| \pm |YZ| \pm |XA| \pm |XP||$, $|ZQ| \neq |YC|$, $|YB| \neq ||YZ| \pm |XA| \pm |XP||$ (Figure 3.24(a)),
- $|YB| \neq ||YZ| \pm |XA||$ (Figure 3.24(b)) and

- $|ZQ| \neq ||YB| \pm |XA||$, $|YC| \neq ||YZ| \pm |XP||$, $|ZQ| \neq ||YC| \pm |YZ| \pm |XP||$, $|ZQ| \neq ||YB| \pm |XA| \pm |XP||$ (Figure 3.24(c)).

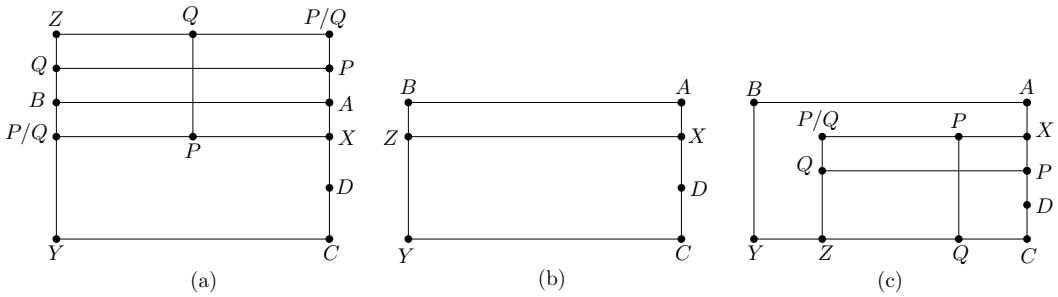


Figure 3.24: Replacing condition $|YC| \neq |AB|$

Thus, we have the following lemma, the proof of which is similar to the proof of Lemma 31 and is omitted:

Lemma 40 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(c) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||YC| \pm |XP||, |ZQ| \neq ||YB| \pm |YZ| \pm |XA||, |YC| \neq |XP|, |ZQ| \neq ||YB| \pm |YZ| \pm |XA| \pm |XP||, |ZQ| \neq |YC|, |YB| \neq ||YZ| \pm |XA| \pm |XP||, |YB| \neq ||YZ| \pm |XA||, |ZQ| \neq ||YB| \pm |XA||, |YC| \neq ||YZ| \pm |XP||, |ZQ| \neq ||YC| \pm |YZ| \pm |XP||, |ZQ| \neq ||YB| \pm |XA| \pm |XP||\}$.

Replacing $|YB| \neq |CD|$:

The layer graph of the 6-cycle corresponding to this condition is as shown in Figure 3.20(b). This layer graph is the same as that in Figure 3.20(c) if we interchange A with D and B

with C. By this interchange of the labels in Lemma 40 we have the following lemma for the replacement of this condition:

Lemma 41 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(b) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||YB| \pm |XP||, |ZQ| \neq ||YC| \pm |YZ| \pm |XD||, |YB| \neq |XP|, |ZQ| \neq ||YC| \pm |YZ| \pm |XD| \pm |XP||, |ZQ| \neq |YB|, |YC| \neq ||YZ| \pm |XD| \pm |XP||, |YC| \neq ||YZ| \pm |XD||, |ZQ| \neq ||YC| \pm |XD||, |YB| \neq ||YZ| \pm |XP||, |ZQ| \neq ||YB| \pm |YZ| \pm |XP||, |ZQ| \neq ||YC| \pm |XD| \pm |XP||\}$.

Replacing $|YC| \neq ||XA| \pm |CD||$:

The layer graph of the 6-cycle corresponding to this condition is as shown in Figure 3.20(o). Figure 3.25 shows all the possible layer graphs of the 6:6 jewel, excluding the chain $XSRZ$, in which the layer graph of Figure 3.20(o) is embedded. From Figure 3.25 we see that the condition $|YC| \neq ||XA| \pm |CD||$ can be replaced by the following conditions:

- $|ZQ| \neq ||YZ| \pm |XD| \pm |XP||$, $|ZQ| \neq |XA|$, $|YZ| \neq ||XD| \pm |XP||$, $|ZQ| \neq ||XA| \pm |XP||$ (Figure 3.25(a)),
- $|ZQ| \neq ||YZ| \pm |XA| \pm |XP||$, $|ZQ| \neq |XD|$, $|YZ| \neq ||XA| \pm |XP||$, $|ZQ| \neq ||XD| \pm |XP||$ (Figure 3.25(b)).

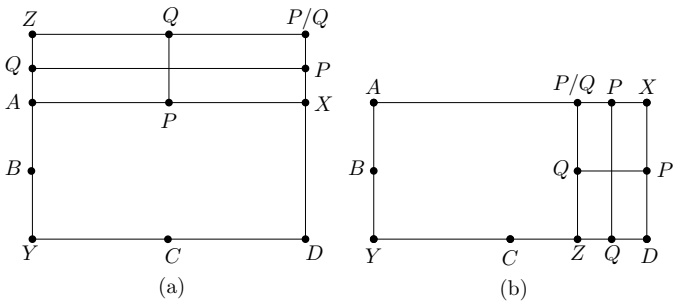


Figure 3.25: Replacing condition $|YC| \neq ||XA| \pm |CD||$

Thus, we have the following lemma, the proof of which is similar to the proof of Lemma 31 and is omitted.

Lemma 42 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(o) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||YZ| \pm |XD| \pm |XP||, |ZQ| \neq |XA|, |YZ| \neq ||XD| \pm |XP||, |ZQ| \neq ||XA| \pm |XP||, |ZQ| \neq ||YZ| \pm |XA| \pm |XP||, |ZQ| \neq |XD|, |YZ| \neq ||XA| \pm |XP||, |ZQ| \neq ||XD| \pm |XP||\}$.

Replacing $|YB| \neq ||XD| \pm |CD||$:

The layer graph of the 6-cycle corresponding to this condition is as shown in Figure 3.20(l). Figure 3.26 shows all the possible layer graphs of the 6:6 jewel, excluding the chain $XSRZ$, in which the layer graph of Figure 3.20(l) is embedded. From Figure 3.26 we see that the condition $|YB| \neq ||XD| \pm |CD||$ can be replaced by the following conditions:

- $|ZQ| \neq ||YB| \pm |YZ| \pm |XP||$, $|ZQ| \neq |YC|$, $|YB| \neq ||YZ| \pm |XP||$, $|ZQ| \neq ||YC| \pm$

$|XP||$ (Figure 3.26(a));

- $|ZQ| \neq ||YC| \pm |YZ| \pm |XP||$, $|ZQ| \neq |YB|$, $|YC| \neq ||YZ| \pm |XP||$, $|ZQ| \neq ||YB| \pm |XP||$ (Figure 3.26(b)).

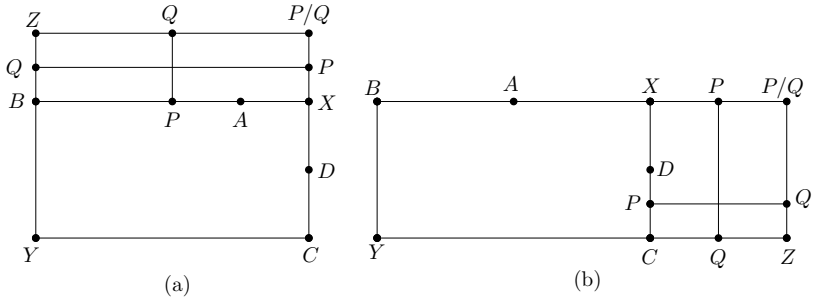


Figure 3.26: Replacing condition $|YB| \neq ||XD| \pm |CD||$

Thus, we have the following lemma, the proof of which is similar to the proof of Lemma 31 and is omitted.

Lemma 43 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(l) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||YB| \pm |YZ| \pm |XP||, |ZQ| \neq |YC|, |YB| \neq ||YZ| \pm |XP||, |ZQ| \neq ||YC| \pm |XP||, |ZQ| \neq ||YC| \pm |YZ| \pm |XP||, |ZQ| \neq |YB|, |YC| \neq ||YZ| \pm |XP||, |ZQ| \neq ||YB| \pm |XP||\}$.

Replacing $|YB| \neq ||XA| \pm |CD||$

The layer graph of the 6-cycle corresponding to this condition is as shown in Figure 3.20(p). Figure 3.27 shows all the possible layer graphs of the 6:6 jewel, excluding the chain $XSRZ$,

in which the layer graph of Figure 3.20(p) is embedded. From Figure 3.27 we see that the condition $|YB| \neq ||XA| \pm |CD||$ can be replaced by the following conditions:

- $|ZQ| \neq ||YC| \pm |XD| \pm |XP||$, $|ZQ| \neq ||YB| \pm |YZ| \pm |XA||$, $|YC| \neq ||XD| \pm |XP||$, $|ZQ| \neq ||YB| \pm |YZ| \pm |XA| \pm |XP||$, $|ZQ| \neq ||YC| \pm |XD||$, $|YB| \neq ||YZ| \pm |XA| \pm |XP||$ (Figure 3.27(a));

- $|YB| \neq ||YZ| \pm |XA||$ (Figure 3.27(b));

- $|YC| \neq ||YZ| \pm |XD||$ (Figure 3.27(c));

- $|YB| \neq ||XA| \pm |XP||$, $|ZQ| \neq ||YC| \pm |XD||$, $|ZQ| \neq ||YB| \pm |XA| \pm |XP||$, $|ZQ| \neq ||YB| \pm |XA||$, $|YC| \neq ||YZ| \pm |XD| \pm |XP||$, $|ZQ| \neq ||YC| \pm |YZ| \pm |XD| \pm |XP||$ (Figure 3.27(d)).

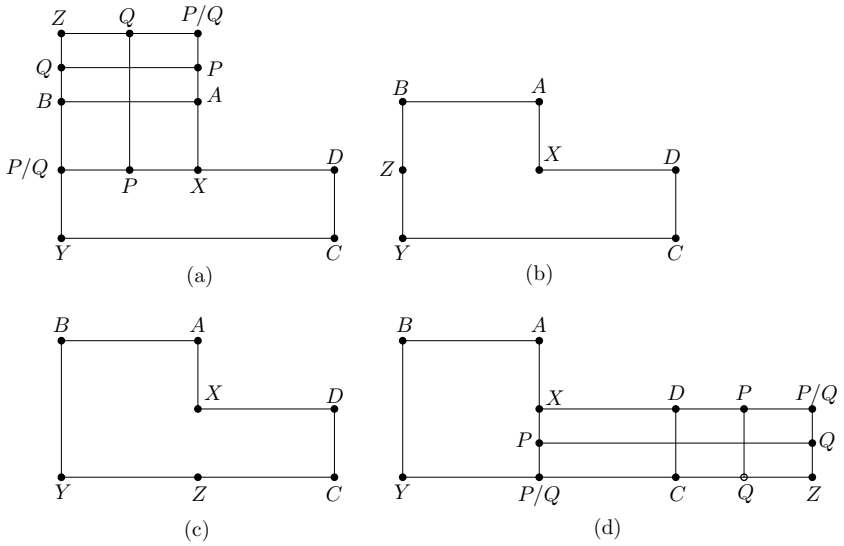


Figure 3.27: Replacing condition $|YB| \neq ||XA| \pm |CD||$

Thus, we have the following lemma, the proof of which is similar to the proof of Lemma 31 and is omitted.

Lemma 44 *The 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19 cannot be drawn as a layer graph in the configuration of Figure 3.20(p) if the edges of the jewel satisfy the following conditions:*

$\{|ZQ| \neq ||YC| \pm |XD| \pm |XP||, |ZQ| \neq ||YB| \pm |YZ| \pm |XA||, |YC| \neq ||XD| \pm |XP||, |ZQ| \neq ||YB| \pm |YZ| \pm |XA| \pm |XP||, |ZQ| \neq ||YC| \pm |XD||, |YB| \neq ||YZ| \pm |XA| \pm |XP||, |YB| \neq ||YZ| \pm |XA||, |YC| \neq ||YZ| \pm |XD||, |YB| \neq ||XA| \pm |XP||, |ZQ| \neq ||YC| \pm |XD||, |ZQ| \neq ||YB| \pm |XA| \pm |XP||, |ZQ| \neq ||YB| \pm |XA||, |YC| \neq ||YZ| \pm |XD| \pm |XP||, |ZQ| \neq ||YC| \pm |YZ| \pm |XD| \pm |XP||\}$.

3.4.2 Rigidity Conditions

From Eqs. (4)-(5) and Lemmas 38 - 44 we have the following lemma for the line-rigidity of the 6-cycle $XABYCD$ of the 6:6 jewel of Figure 3.19:

Lemma 45 *The 6-cycle $XABYCD$ of the 6:6 jewel $XABYCDPQZRS$ of Figure 3.19 is rigid if the edges of the jewel satisfy the following conditions:*

1. $|YZ| \notin \{|XA|, |XD|, ||XA| \pm |XP||, ||XD| \pm |XP||\}$;
2. $|YB| \notin \{|XA|, |XD|, |XP|, ||XA| \pm |XD||, ||XA| \pm |XP||, ||XA| \pm |YZ||, ||XP| \pm |YZ||, ||XA| \pm |XP| \pm |YZ||\}$;

3. $|ZQ| \notin \{|XA|, |XD|, |YB|, ||XA| \pm |XP||, ||XD| \pm |XP||, ||XA| \pm |YZ||, ||YB| \pm |XA||, ||XD| \pm |YZ||, ||YB| \pm |XP||, ||XA| \pm |XP| \pm |YZ||, ||XD| \pm |XP| \pm |YZ||, ||YB| \pm |XA| \pm |XP||, ||YB| \pm |XA| \pm |YZ||, ||YB| \pm |XP| \pm |YZ||, ||YB| \pm |XA| \pm |XP| \pm |YZ||\}$;

4. $|YC| \notin \{|XD|, |XA|, |XP|, |ZQ|, ||ZQ| \pm |XD||, ||ZQ| \pm |XP||, ||YB| \pm |XA||, ||YB| \pm |XD||, ||XA| \pm |XD||, ||XD| \pm |XP||, ||XD| \pm |YZ||, ||XP| \pm |YZ||, ||XD| \pm |XP| \pm |YZ||, ||ZQ| \pm |XD| \pm |XP||, ||ZQ| \pm |XD| \pm |YZ||, ||ZQ| \pm |XP| \pm |YZ||, ||ZQ| \pm |XD| \pm |XP| \pm |YZ||\}$.

Similarly, we have the following lemma for the line-rigidity of the other 6-cycle $XPQZRS$ of the 6:6 jewel:

Lemma 46 *The 6-cycle $XPQZRS$ of the 6:6 jewel $XABYCDPQZRS$ of Figure 3.19 is rigid if the edges of the jewel satisfy the following conditions:*

1. $|YZ| \notin \{|XP|, |XS|, ||XA| \pm |XP||, ||XA| \pm |XS||\}$;

2. $|YB| \notin \{|XP|, |XS|, ||XA| \pm |XP||, ||XP| \pm |YZ||, ||XS| \pm |YZ||, ||XA| \pm |XS||, ||XA| \pm |XP| \pm |YZ||, ||XA| \pm |XS| \pm |YZ||\}$;

3. $|ZQ| \notin \{|XA|, |XS|, |YB|, |XP|, ||XA| \pm |XP||, ||XA| \pm |YZ||, ||YB| \pm |XA||, ||YB| \pm |XP||, ||XP| \pm |XS||, ||XP| \pm |YZ||, ||XA| \pm |XP| \pm |YZ||, ||YB| \pm |XA| \pm |XP||, ||YB| \pm |XA| \pm |YZ||, ||YB| \pm |XP| \pm |YZ||, ||YB| \pm |XA| \pm |XP| \pm |YZ||\}$;

4. $|ZR| \notin \{|XS|, |XP|, |XA|, |YB|, ||ZQ| \pm |XP||, ||ZQ| \pm |XS||, ||XP| \pm |XS||, ||YB| \pm$

$|XS||, ||XA| \pm |XS||, ||XA| \pm |YZ||, ||YB| \pm |XA||, ||XS| \pm |YZ||, ||XA| \pm |XS| \pm |YZ||, ||YB| \pm |XA| \pm |XS||, ||YB| \pm |XS| \pm |YZ||, ||YB| \pm |XA| \pm |YZ||, ||YB| \pm |XA| \pm |XS| \pm |YZ||\}$.

By Corollary 26, the union of the two sets of conditions in Lemmas 45 and 46 constitutes a set of sufficient conditions for the line-rigidity of the 6:6 jewel of Figure 3.19. Taking care of overlapping conditions between the two sets of conditions, we have 74 distinct conditions for the line-rigidity of the 6:6 jewel and hence the following lemma:

Lemma 47 *The 6:6 jewel $XABYCDPQZRS$ of Figure 3.19 is rigid if its edges satisfy the following conditions:*

1. $|YZ| \notin \{|XA|, |XD|, |XP|, |XS|, ||XA| \pm |XP||, ||XD| \pm |XP||, ||XA| \pm |XS||\}$;
2. $|YB| \notin \{|XA|, |XD|, |XP|, |XS|, ||XA| \pm |XD||, ||XA| \pm |XP||, ||XA| \pm |YZ||, ||XP| \pm |YZ||, ||XS| \pm |YZ||, ||XA| \pm |XS||, ||XA| \pm |XP| \pm |YZ||, ||XA| \pm |XS| \pm |YZ||\}$;
3. $|ZQ| \notin \{|XA|, |XS|, |XD|, |YB|, |XP|, ||XA| \pm |XP||, ||XD| \pm |XP||, ||XA| \pm |YZ||, ||YB| \pm |XA||, ||XD| \pm |YZ||, ||YB| \pm |XP||, ||XP| \pm |XS||, ||XP| \pm |YZ||, ||XA| \pm |XP| \pm |YZ||, ||XD| \pm |XP| \pm |YZ||, ||YB| \pm |XA| \pm |XP||, ||YB| \pm |XA| \pm |YZ||, ||YB| \pm |XP| \pm |YZ||, ||YB| \pm |XA| \pm |XP| \pm |YZ||\}$;
4. $|YC| \notin \{|XD|, |XA|, |XP|, |ZQ|, ||YB| \pm |XA||, ||YB| \pm |XD||, ||XA| \pm |XD||, ||XD| \pm |XP||, ||XD| \pm |YZ||, ||XP| \pm |YZ||, ||ZQ| \pm |XD||, ||ZQ| \pm |XP||, ||YB| \pm |XA| \pm$

$|XD||, ||XD| \pm |XP| \pm |YZ||, ||ZQ| \pm |XD| \pm |XP||, ||ZQ| \pm |XD| \pm |YZ||, ||ZQ| \pm |XP| \pm |YZ||, ||ZQ| \pm |XD| \pm |XP| \pm |YZ||\}$;

5. $|ZR| \notin \{|XS|, |XP|, |XA|, |YB|, ||ZQ| \pm |XP||, ||ZQ| \pm |XS||, ||XP| \pm |XS||, ||YB| \pm |XS||, ||XA| \pm |XS||, ||XA| \pm |YZ||, ||YB| \pm |XA||, ||XS| \pm |YZ||, ||ZQ| \pm |XP| \pm |XS||, ||XA| \pm |XS| \pm |YZ||, ||YB| \pm |XA| \pm |XS||, ||YB| \pm |XS| \pm |YZ||, ||YB| \pm |XA| \pm |YZ||, ||YB| \pm |XA| \pm |XS| \pm |YZ||\}$.

In the next section we show how a composite *ppg* can be constructed by satisfying all the 74 conditions for each such jewel.

3.4.3 Algorithm

It is interesting to note that the substitution mechanism has generated rigidity conditions on the strut YZ (Condition 1 of Lemma 47). This implies that, unlike the case for a 5:5 jewel, we will need a pool of vertices S for which the pairwise distances of all the pairs of points corresponding to the vertices in S are known after the first round of query, and from which we choose the end vertices Y and Z of a strut YZ in order to meet the rigidity conditions on YZ. We make the vertices in S rigid in the first round. Then the pairwise distances of all the pairs of points corresponding to the vertices in S are known after the first round of query. We make the remaining 9 vertices of each 6:6 jewel rigid in the second round.

We have to choose the size of S carefully. Since there are 10 conditions on the length of

an YZ, from Observation 4 it follows that there must be at most 21 edges incident to the end vertex Y, when we are looking for the other end vertex Z of a strut.

However, if we use $|S| = 22$ for the selection of Z for a particular Y, it may happen that all the 6:6 jewels get attached to the same vertex $Z \in S$. This hinders our goal of obtaining a better value for α than previously known.

We need to attach 6:6 jewels evenly to all the vertices in S so that the same number of edges can be attached to each of them in the first round, and all of those edges, except for a constant number, are used to attach 6:6 jewels. In other words, we need to attach the 6:6 jewels to the vertices in S in such a way that the numbers of 6:6 jewels attached to any two vertices differ by at most a constant number.

To specify the number of basic components attached to a vertex in a rigid set S in the first round we use the term *valence*. We denote the set of rigid vertices in round 1 with valence d as S_d.

Now we describe our algorithm to select a pair of vertices Y and Z in S to attach 6:6 jewels. To attach a 6:6 jewel we always select a vertex in S with the lowest valence as the first vertex (say, Y). Of the remaining vertices in S, at most 20 vertices may not be acceptable for Z, because of the conditions on YZ (Condition 1 of Lemma 47). From among the rest $|S| - 1$ vertices in S that satisfy the conditions on YZ, we select the one that has the lowest valence, as Z. This method is followed to attach each 6:6 jewel to the vertices in S, while the 6:6 jewels are attached sequentially.

The following lemma tells us how big S must be.

Lemma 48 *A set S of 42 vertices is sufficient to ensure that the valences of two vertices in S differ by at most 1.*

Proof. Initially, all the vertices in S have valence 0, i.e., $|S_0| = 42$ and $|S_i| = 0$ for each i. Our algorithm selects pairs of vertices (Y, Z) in S_0 to attach 6:6 jewels. To select a pair of vertices (Y, Z) in S_0 by satisfying 10 different length restrictions (Condition 1 of Lemma 47) in all, we need a buffer of vertices in S_0 of size at least 20. Thus, our algorithm selects 11 pairs of vertices (Y, Z) from S_0. This way our algorithm attaches the first 11 6:6 jewels to 22 vertices in S_0. Then, we have $|S_0| = 20$ and $|S_1| = 22$.

Next, for each new 6:6 jewel our algorithm selects at least 1 vertex (as Y, say) from S_0 until it is empty. The valence of this vertex will be raised to 1. If the second vertex is not found in S_0, it will be selected from S_1, raising its valence to 2. Now, we have $|S_0| = 0$, $|S_1| \geq 22$ and the rest vertices are in S_2.

We note that when both the vertices of a pair are chosen from S_0 then $|S_1|$ is increased by 2. Consequently, $|S_1|$ remains even. Now we consider the case when only one vertex is chosen from S_0. Since $|S_1| \geq 22$, we can choose the other vertex from S_1. This increases $|S_1|$ by 1 and decreases it by 1 at the same time. Thus, in both the cases $|S_1|$ will always be an even number.

If $|S_1| > 20$ we choose pairs of vertices for Y and Z from S_1 until $|S_1| = 20$. Eventually, we have $|S_0| = 0$, $|S_1| = 20$ and $|S_2| = 22$. Thus, at some point of time all the vertices in S

have at most 2 consecutive valences, viz., 1 and 2. At any point of time before that, they may have at most 3 consecutive valences, viz., 0 - 2.

We shall show that our algorithm attach the 6:6 jewels in such a way that at any point of time the vertices in S will have at most 3 consecutive valences, and that at some point of time they will have at most 2 consecutive valences only. For this we use induction to show that if we start with vertices in S in the state of a valence distribution (S_d, S_{d+1}), and attach the basic components according to our algorithm, then at some point of time the state of the valences will be (S_{d+1}, S_{d+2}). We assume that $|S_d| = 20$ and $|S_{d+1}| = 22$.

We attach 6:6 jewels until $|S_d| = 0$. For each new 6:6 jewel we choose at least 1 vertex from S_d and at most 1 vertex from S_{d+1} with a total of 2 vertices to form a 6:6 jewel. Then we have $|S_{d+1}| \geq 22$. The rest vertices are of valence $d+2$. We argue as above to show that $|S_{d+1}|$ will always be even.

If $|S_{d+1}| > 20$, then for each new 6:6 jewel our algorithm uses a pair of vertices in S_{d+1} as in the initial round above until $|S_{d+1} = 20|$. The rest 22 will be of valence $d+2$. Now the situation is the same as it was at the start of induction except that the levels of valences have been increased by 1. □

The set S of 42 vertices can be set as the vertices of 8 4:4 jewels hanging from a common strut. Since each 4:4 jewel is rigid so is this configuration. edges of this *ppg*.

From Condition 2 of Lemma 47 we see that we need 48 extra edges for the selection of an YB that satisfies all the conditions on it as stated in the lemma. Similarly, by Conditions

3, 4 and 5 of Lemma 47 we need 98 extra edges for ZQ, 96 extra edges for YC and 96 extra edges for ZR respectively. Thus, 98 extra edges at Y and Z will suffice to satisfy all the conditions on these edges. In addition to these extra 98 edges we need 2 more edges to accommodate the difference of 1 6:6 jewel that can be attached to them. Thus, we need a total of 100 extra edges at each of the 42 vertices of S.

The main idea underlying the algorithm below is to construct multiple copies of a 6:6 jewel over two rounds to ensure their rigidity. We use the set of vertices S as reference vertices. Any set of 42 vertices is chosen as S. The pair of vertices $\{Y, Z\}$ that make up the strut YZ (see Figure 3.19) of a 6:6 jewel, is chosen from the set S. As part of the first round, a rigid layout of S is fixed by attaching eight 4:4 jewels of Figure 3.6 from a common strut. The common strut of the 4:4 jewels joins two vertices of S. Pairwise distances of some other suitable vertices are also queried in the first round.

Now we consider the second round. Let $S' = V \setminus S$ be the complement of S. In the second round, the positions of all the vertices of S' are fixed relative to the vertices in S by first selecting groups of 9 vertices each from S' and placing them relative to a pair of vertices $\{Y, Z\}$ of S. For this, we select a vertex $Y \in S$ which has the lowest valence of 6:6 jewel of Figure 3.19 and a 5-link (X, A, D, P, S). Then we select a vertex $Z \in S$ such that it has the lowest valence of 6:6 jewel of Figure 3.19 and that $|YZ|$ satisfies all the conditions of rigidity on it as stated in Condition 1 of Lemma 47. Thereafter, the vertices B, C, Q and R of S' are selected such that the conditions of rigidity on $|YB|$, $|ZQ|$, $|YC|$ and $|ZR|$ as

stated in respectively Conditions 2, 3, 4 and 5 of Lemma 47 are satisfied. Then we query the remaining necessary pairwise edge distances $|AB|, |CD|, |PQ|$ and $|RS|$ of the group to form a 6:6 jewel. The jewel will be rigid by Lemma 47 irrespective of the lengths of the edges AB, CD, PQ and RS, since no condition of the lemma involves any of these edges. The unused vertices of S' are made rigid by using 4-cycle as the *ppg*.

Algorithm 3.2. As in Algorithm 3.1, we use the following indexing scheme: $X \rightarrow X_i$, $A \rightarrow A_i$, $B \rightarrow B_j$, $C \rightarrow B_k$, $D \rightarrow D_i$, $P \rightarrow P_i$, $Q \rightarrow Q_m$, $R \rightarrow Q_l$, $S \rightarrow S_i$, $Y \rightarrow Y_u$ and $Z \rightarrow Y_v$.

Let the total number of points be n. We attach b 6:6 jewels (Figure 3.19) to each of 20 fixed vertices in S and $b+1$ to the remaining 22. This gives us a total of $21b+11$ jewels.

In the first round, we make distance queries represented by the edges of the graph in Figure 3.28. All the vertices Y_u $(u = 1, ..., 42)$ (or, Y_v, $v = 1, ..., 42$) in the subgraph enclosed by the rectangle are made line rigid in the first round by using the 4:4 jewel of Figure 3.6 as the *ppg*. There are 8 4:4 jewels (Figure 3.6) attached to a common strut, 42 vertices and 65 edges in the subgraph. There are $2b+100$ leaf children rooted at each of the vertices Y_u $(u = 1, ..., 42)$ (or, Y_v, $v = 1, ..., 42$) to attach b or $b+1$ 6:6 jewels (Figure 3.19). Since there will be $21b+11$ 6:6 jewels we have $21b+11$ groups of 5 vertices $(A_i, D_i, S_i, P_i, X_i)$ $(i = 1, ..., 21b+11)$. We query the distances $|A_iX_i|$, $|D_iX_i|$, $|S_iX_i|$ and $|P_iX_i|$, $(i = 1, ..., 21b+11)$ in the first round. We will make a total of $168b+4309$ pairwise distance queries in the first round for the placement of $n = 189b+4297$ points.

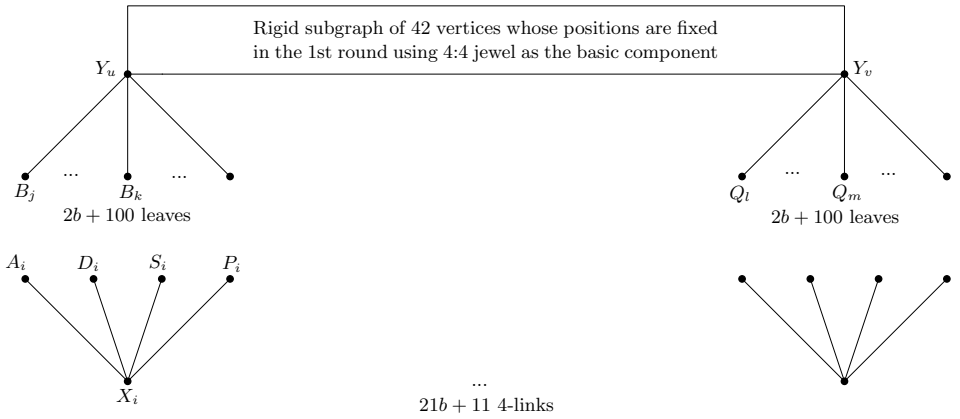


Figure 3.28: Queries in the first round for 2-round algorithm using 6:6 jewel as the basic component

In the second round, for each 4-link $(A_i, D_i, S_i, P_i, X_i), i = 1, ..., 21b + 11$, we construct a 6:6 jewel (Figure 3.19), satisfying all its rigidity conditions as in Lemma 47. For each such 4-link we select a vertex Y_u, from the subgraph of 42 fixed vertices $Y_u/Y_v(u, v = 1, ..., 42; u \neq v)$, that has the lowest valency of 6:6 jewel of Figure 3.19. Since all the 42 vertices $Y_u, u = 1, ..., 42$, are fixed in the first round, for any pair of such fixed vertices $(Y_u, Y_v)(u, v = 1, ...42; u \neq v)$ we can find the distance $|Y_uY_v|$. So, for each pair of vertices $(Y_u, Y_v)(u, v = 1, ..., 42; u \neq v)$, we shall use (Y_u, Y_v) as an edge in the construction of the 6:6 jewel of Figure 3.19. Now from the subgraph of 42 fixed vertices we select another vertex $Y_v(v \neq u)$ such that the length $|Y_uY_v|$ satisfies all the conditions of rigidity on it as stated in Condition 1 of Lemma 47 and that it has the lowest valency of 6:6 jewel of Figure 3.19 among all such qualifying vertices. We note that we can always find such vertex Y_v, because there will be at most 20 edges Y_uY_v whose length do not satisfy the rigidity conditions on it (Condition 1 of Lemma 47) whereas we have 41 vertices for choosing the vertex Y_v.

Then we find an edge Y_uB_j rooted at Y_u satisfying the conditions of rigidity on it as stated in Condition 2 of Lemma 47, then we find another edge Y_vQ_m rooted at Y_v satisfying the conditions of rigidity on it as stated in Condition 3 of Lemma 47, then we find another edge Y_uB_k rooted at Y_u satisfying the rigidity conditions on it as stated in Condition 4 of Lemma 47 and, finally, we find another edge Y_vQ_l rooted at Y_v satisfying the rigidity conditions on it as stated in Condition 5 of Lemma 47. Then for each $i, (i = 1, ..., 21b+11)$, we query the distances $|A_iB_j|$, $|D_iB_k|$, $|S_iQ_l|$ and $|P_iQ_m|$ to form a 6:6 jewel $X_iA_iB_jY_uB_kD_iP_iQ_mY_vQ_lS_i$. Its edges will satisfy all the rigidity conditions of Lemma 47. Thus, all the $21b+11$ 4-links will be consumed to construct $21b+11$ jewels. For this $84b+44$ edges will be queried.

There will be unused leaves B_j (or Q_l) numbering 100 for each of 20 fixed vertices Y_u $(u = 1, ..., 42)$ (or, Y_v, $v = 1, ..., 42$) and 98 for each of 22 fixed vertices Y_u $(u = 1, ..., 42)$ (or, Y_v, $v = 1, ..., 42$). The total number of such unused vertices is 4156. We use a 4-cycle *ppg* to fix them in the second round. As before, for each pair of vertices $(Y_u, Y_v)(u, v = 1, ..., 42; u \neq v)$, we shall use Y_uY_v as an edge in the construction of the 4-cycle. For each unused vertex B_j rooted at Y_u we find another vertex Q_l rooted at Y_v such that $|Y_uB_j| \neq |Y_vQ_l|$. Then the 4-cycle $B_jY_uY_vQ_l$ will be rigid (Observation 2). Then we query the distance $|B_jQ_l|$ to complete the 4-cycle.

Note that we can always find a vertex like Q_l. For after repeated selection of such matching pairs of edges there may remain at most 2 edges Y_uB_j rooted at Y_u of length

equal to that of the same number of edges rooted at Y_v (Observation 4). In such a situation we switch the matching to match such edges rooted at Y_u with edges other than those same length edge/s rooted at Y_v - this is always possible because there are at most 2 edges rooted at Y_v that have the same length (Observation 4).

For 4156 unused vertices (after the construction of the 6:6 jewel) there will be 2078 4-cycles, and 2078 edges will be queried to complete the 4-cycles. The total number of queries in the second round will be $(84b + 44) + 2078$, i.e., $84b + 2122$.

Theorem 49 *The ppg constructed by Algorithm 3.2 is rigid.*

Proof. The proof is similar to that of Theorem 36 for the line rigidity of the *ppg* constructed by Algorithm 3.1. □

The number of queries in the first and second rounds are $168b + 4309$ and $84b + 2122$ respectively. Thus, in 2 rounds a total of $252b + 6431$ pairwise distances are to be queried for the placement of $189b + 4297$ points. It is interesting to note that our algorithm would need at least 4486 points to work, which makes it reasonably practical. When we have fewer points we can use Algorithm 3.1 instead.

Now, $252b + 6431 = (252/189) * (189b + 4297) - (4/3) * 4297 + 6431 = 4n/3 + (19293 - 17188)/3 = 4n/3 + 2105/3$. Thus, we have the following theorem:

Theorem 50 $4n/3 + 2105/3$ *queries are sufficient to place n distinct points on a line in two rounds.*

A consequence of the last theorem is that our 6:6 jewel algorithm is better than the 5-cycle algorithm of Chin *et al.* [20] for $n \geq 11851$.

3.5 Lower Bound for Two Rounds

In this section we revisit the adversarial argument given by [20] to establish a lower bound on 2-round algorithms. We show that a deeper analysis improves the lower bound substantially.

Let $\mathcal{A}$ denote any 2-round algorithm and $\mathcal{B}$ an adversary. The latter sets edge lengths for *ppg* in each of the 2 rounds and returns the distance between any two points queried by $\mathcal{A}$. $\mathcal{B}$ can also assign value to the distance between a pair of points *not queried* by A. While $\mathcal{A}$'s goal is to make as few distance queries as possible, $\mathcal{B}$ tries to maximize the density of the *ppg*.

In the *first round*, $\mathcal{A}$ queries the distances between pairs of vertices corresponding to the edges E_1 of the *ppg*, $G_1 = (V, E_1)$. In response, $\mathcal{B}$ returns queried edge-lengths consistent with the following 3-part strategy. We call a vertex of degree at least 3 in a *ppg* G as a *heavy* vertex in G.

*S*1**.** *$\mathcal{B}$ fixes the layout of all heavy vertices in G_1 and sets the lengths of the edges in G_1 incident to these vertices.*

*S*2**.** *For each vertex of degree 2 in G_1 that is connected to a vertex of degree 1 in G_1, the length of one of the two edges incident to the degree 2 vertex is set to a fixed value $c > 0$.*

S3. Let $\mathcal{P}_k = < p_1, p_2, ..., p_k > (k \geq 2)$ *be a maximal path of degree 2 vertices* $p_i, i = 1, \ldots, k$ *in* G_1. *Let* p_0 *and* p_{k+1} *be non-degree 2 vertices in* G_1 *adjacent to* p_1 *and* p_k *respectively. First* $\mathcal{B}$ *sets* $|p_{i-1}p_i| = |p_{i+1}p_{i+2}|$ *for* $i = 1$ *(mod 3). If both* p_0 *and* p_{k+1} *are heavy vertices in* G_1, *then it sets* $|p_ip_{i+1}| = |p_{i-1}p_{i+2}|$ *for* $i = 1$ *(mod 3) and also fixes the layout of the vertices* $p_i, i = 0$ *(mod 3). Otherwise, if at least one of them, say* p_{k+1}, *is of degree one in* G_1 $\mathcal{B}$ *sets* $|p_kp_{k+1}| = |p_{k-2}p_{k-1}|$. *Also, except for the edges whose length is* c, $\mathcal{B}$ *sets the lengths of the rest of the edges to lie between* $2c$ *and* $3c$.

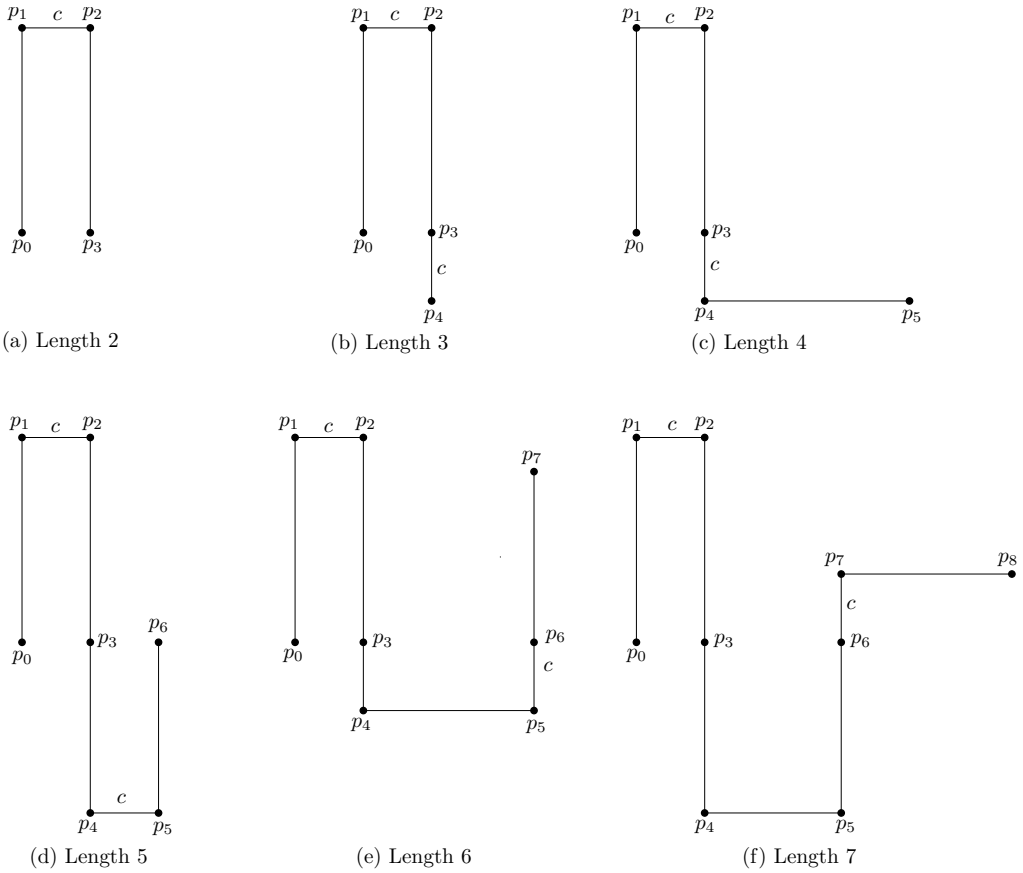
Lemma 51 *Strategies S2 and S3 of* $\mathcal{B}$ *are mutually consistent.*

Proof. Consider a path $\mathcal{P}_k$ of degree 2 vertices in G_1 such that both p_0 and p_{k+1} have degree 1. If $k = 1$, only $S2$ comes into play and in this case $\mathcal{B}$ sets $|p_1p_2| = c$. For all $k \geq 4$, $\mathcal{B}$ sets $|p_1p_2| = c$, $|p_{k-1}p_k| = c$ in accordance with $S2$ and the lengths of all other edges in accordance with $S3$. Figures 3.29(c) - 3.29(f) serve as examples of this length assignment since for any k, the total number of edges is a multiple of 3 as in Figure 3.29(d), or a multiple of 3 plus 1 as in Figure 3.29(e) or a multiple of 3 plus 2 as in Figure 3.29(f).

For $k = 2$ and $k = 3$ $\mathcal{B}$ makes the length assignments as shown Figures 3.29(a) - 3.29(b), which are again consistent with $S2$ and $S3$.

If p_0 is heavy, then $\mathcal{B}$ does not have to set $|p_1p_2|$ to c. □

In the *second round*, $\mathcal{A}$ queries the distances between new pairs of vertices corresponding



(a) Length 2 (b) Length 3 (c) Length 4

(d) Length 5 (e) Length 6 (f) Length 7

Figure 3.29: The residual parts of maximal paths of degree 2 vertices that will satisfy $S2$

to the edges in E_2 of the *ppg* $G_2 = (V, E_1 \cup E_2)$. In response, $\mathcal{B}$ returns queried edge lengths consistent with the following strategy:

S4. *Let* $\mathcal{P}_k =< p_1, p_2, ..., p_k >$ $(k \geq 2)$ *be a maximal path of degree 2 vertices of length at least 2 in* G_1. *Let* p_0 *and* p_{k+1} *be non-degree 2 vertices adjacent to* p_1 *and* p_k *respectively.*

If at least one of them, say p_{k+1}, *is of degree 1 in the first round and if, for some* i *with* $i = 1$ *(mod 3) and* $i < k$, *no edge is connected to either* p_i *or* p_{i+1} *in the second round by the algorithm then* $\mathcal{B}$ *sets* $|p_i p_{i+1}| = |p_{i-1} p_{i+2}|$ *for one of those values of* i *in*

the second round.

Or, if no edge is connected to either p_{k-1} or p_k in the second round by $\mathcal{A}$, then $\mathcal{B}$ sets $|p_{k-1}p_k| = |p_{k-2}p_{k+1}|$.

An important observation is in order: the above strategies of $\mathcal{B}$ do not prevent $\mathcal{A}$ from making a linear placement of the vertices of a maximal path of degree 2 vertices that joins a heavy vertex to a vertex of degree 1 in distinct positions.

Let p_0 be a heavy vertex. Consider all the maximal paths $\mathcal{P}_k$ of degree 2 vertices incident to p_0, whose other end is of degree 1. For each path, $\mathcal{B}$ computes the sum of the lengths of all the edges in the path. Let l_{max} be the maximum of all the sums. $\mathcal{B}$ maintains an interval of this length on either side of p_0 free from the placement of the vertices that lie on a path $\mathcal{P}_k$ incident to p_0 whose other end is a heavy vertex. This is ensured as follows:

1. The distance between p_0 and an adjacent heavy vertex is at least l_{max}.

2. Let $\mathcal{P}_k =< p_0, p_1, p_2 >$. In this case, $\mathcal{B}$ sets $|p_0p_1| > l_{max}$. If $\mathcal{P}_k =< p_0, p_1, ..., p_{k+1} >$, where $k > 1$, $\mathcal{B}$ sets $|p_0p_1| = |p_2p_3| > l_{max}$ and $|p_1p_2| > 2|p_0p_1|$. This ensures that all the vertices of the prefix segment $< p_0, p_1, p_2, p_3 >$ of the path is at a distance farther than l_{max} away from p_0. Clearly the remaining vertices on $\mathcal{P}_k$, however placed, will also be at a distance farther than l_{max}.

The strategies adopted by $\mathcal{B}$ bound the lengths of maximal paths formed by degree 2 vertices in G_2. The precise results are given in the next 3 lemmas.

Lemma 52 *In G_2, the length of a longest chain of consecutive edges from E_1 that terminate on a heavy vertex at each end of the chain is 4.*

Proof. Let p_0 and p_{k+1} be non degree 2 vertices adjacent to a maximal path $\mathcal{P}_k =< p_1, p_2, ..., p_k > (k \geq 2)$ of degree 2 vertices of length k in G_1.

We first consider the case when both of p_0 and p_{k+1} are heavy vertices of G_1.

Given strategy $S3$ of $\mathcal{B}$, if for an $i < k$ with $i = 1 \pmod 3$ $\mathcal{A}$ attaches no edge to either p_i or p_{i+1} in the second round then their positions will be ambiguous. Thus, the lemma is settled for this case.

Consider the case when p_{k+1} is of degree 1. In view of strategies $S3$ and $S4$ of $\mathcal{B}$, $\mathcal{A}$ must attach an edge at p_i or p_{i+1} in the second round, for $i < k$ and $i = 1 \pmod 3$, to make the placements of these vertices unambiguous. Thus, the lemma is settled for this case also. □

Lemma 53 *A maximal path $\mathcal{P}_k$ of degree 2 vertices in G_2 that contains at least one edge of E_2 can have at most 2 consecutive edges of E_1.*

Proof. Let $\mathcal{P}_k (k \geq 2)$ be a maximal path of degree 2 vertices in G_1, and p_0 and p_{k+1} be non degree 2 vertices adjacent to p_1 and p_k respectively, where one of p_0 and p_{k+1} be of degree 1 in G_1.

Suppose p_0 is of degree 1 in G_1. In view of strategy $S3$ of $\mathcal{B}$, if no edge is connected to either p_i or p_{i+1} for some $i = 1 \pmod 3$ then following strategy $S4$, $\mathcal{B}$ will set $|p_i p_{i+1}| =$

$|p_{i-1}p_{i+2}|$ for one of those values of i in the second round. Thus, there must be an edge connected to either p_i or p_{i+1} for all $i = 1 \pmod 3$. In particular, $\mathcal{A}$ must add an edge to be incident to p_1 or p_2 (when $i = 1$).

If p_{k+1} is of degree 1 then following strategy $S3$ the adversary sets $|p_kp_{k+1}| = |p_{k-2}p_{k-1}|$ in the first round. If $\mathcal{A}$ attaches no edge to either p_{k-1} or p_k in the second round, then following $S4$, $\mathcal{B}$ sets $|p_{k-1}p_k| = |p_{k-2}p_{k+1}|$. This makes the placements of the vertices p_{k-1} and p_k will ambiguous (Observation 2). Thus $\mathcal{A}$ must attach an edge to p_{k-1} or p_k to preempt $\mathcal{B}$.

Thus, for both the cases, there will be at most 2 vertices of degree at most 2 at an end of a path of degree 2 vertices of G_1, if the end vertex is of degree 1. The algorithm will place them in the second round by introducing edge/s to one or both of them. Thus, in a maximal path of degree 2 vertices in G_2 that contains at least one edge from E_2 there can be at most 2 consecutive edges from E_1. □

Lemma 54 *The number of vertices in any maximal path of degree 2 vertices in G_2 is at most 3.*

Proof. If a maximal path of degree 2 vertices of G_2 consists of edges from E_1 only then by Lemma 52 its length is at most 3.

Now we consider maximal path of degree 2 vertices of G_2 that contains at least one edge from E_2. In such a path there cannot be three consecutive edges from E_1 (Lemma 53). Suppose the number of degree 2 vertices in such a maximal path is 4. Let the vertices

be p_1, p_2, p_3 and p_4. Let p_0 and p_5 be heavy vertices adjacent to p_1 and p_4 respectively. Since any maximal path of degree 2 vertices in G_2 can have at most 2 consecutive edges from E_1 the edges p_0p_1, p_1p_2, p_2p_3, p_3p_4 and p_4p_5 can be from E_1 or E_2 in the following 5 combinations:

1. E_2, E_1, E_2, E_1, E_1
2. E_2, E_1, E_1, E_2, E_1
3. E_1, E_2, E_1, E_2, E_1
4. E_1, E_1, E_2, E_2, E_1
5. E_1, E_1, E_2, E_1, E_1

For combination 1, $\mathcal{B}$ can set the length of the 2 edges in E_2 so that $|p_0p_5| = |p_1p_2|+|p_2p_3|$ and $|p_0p_1| = |p_4p_5| - |p_3p_4|$ (Figure 3.30). Then by Theorem 23 the 6-cycle $p_0p_1p_2p_3p_4p_5$ would not be rigid. Similarly, for the combinations 2-4 $\mathcal{B}$ can make the graph ambiguous. As for combination 5, following $S2$ $\mathcal{B}$ can set $|p_1p_2| = |p_3p_4| = c$, and can set the length of p_2p_3 in the second round in such a way that $|p_2p_3| = |p_4p_5| + |p_5p_0| + |p_0p_1|$ (Figure 3.31). The 6-cycle $p_0p_1p_2p_3p_4p_5$ would not be rigid then (Theorem 23). □

The *density* of a *ppg*, $G = (V, E)$ is defined as the ratio $|E|/n$, where $n = |V|$. We establish the following lower bound on the density of a *ppg* constructed by any 2-round algorithm.

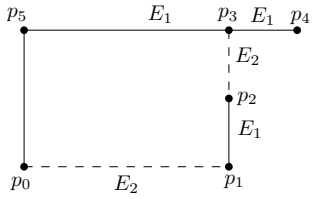


Figure 3.30: Maximal path of degree 2 vertices in G_2 for the combination of edges E_2, E_1, E_2, E_1, E_1

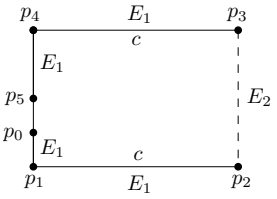


Figure 3.31: Maximal path of degree 2 vertices in G_2 for the combination of edges E_1, E_1, E_2, E_1, E_1

Theorem 55 *Any deterministic 2-round algorithm for solving the 1-dimensional point placement problem requires at least* $12n/11$ *queries in the worst case.*

Proof. Let each edge of G have weight 1, which we split evenly between the vertices in V that define it. If w_i is the accumulated weight of the i-th vertex, clearly $\sum_{i=1}^{n} w_i = |E|$ so that $n * min_i\{w_i\} \leq |E|$. Thus $min_i\{w_i\}$ is a lower bound on the density.

We can get a more precise estimate. Observe that a *ppg* has 2 types of vertices, heavy ones (already defined before) and vertices lying on maximal paths of degree 2 vertices that we call *light* vertices. If an edge joins two light vertices or two heavy vertices then the edge weight is divided equally between the vertices. Otherwise, the light vertex gets $1/2 + g$ of the weight and the heavy vertex $1/2 - g$ of the weight , where $0 \leq g \leq 1/2$.

The density of a heavy vertex is at least $3(1/2 - g)$. As for light vertices, we note that by Lemma 54 each maximal path of degree 2 vertices has length k, where $k \leq 3$. The total edge weight of such a path is $2(1/2 + g) + (k - 1)$. Thus, the average density of each vertex in such a path is $1 + 2g/k$. It is minimum when $k = 3$. Thus, the density of a light vertex is at least $1 + 2g/3$.

The minimum average density for all vertices in G_2 is thus

$$\max \min\{3/2 - 3g, 1 + 2g/3\} = 12/11$$

when $g = 3/22$. □

3.6 Summary

In this chapter, 2-round algorithms based on 5:5 and 6:6 jewels have been presented and the lower bound have been improved from $17n/16$ to $12n/11$. The algorithms have been implemented and they correctly found the uniqe position of points in each case. In the next chapter, we further improve both the lower bound and the upper bound of the problem.

Chapter 4

Improved Algorithm and Lower Bound for Point Placement Problem

4.1 Introduction

In Chapter 3 we proposed a 2-round algorithm that query $4n/3 + O(1)$ edges to construct rigid *ppg* on n points using 6:6 jewels as the basic components. In this chapter, we present a 2-round algorithm that queries $9n/7 + O(1)$ edges to construct a rigid *ppg* on n points, using 3 paths of degree two vertices of length 2 each with a common vertex as the basic component, bettering a result of [20] that uses 5-cycles. More significantly, we improve the lower bound on any 2-round algorithm to $9n/8$.

4.2 A 2-round Algorithm Based on a 3-path Graph

In this section, we describe a 2-round algorithm that queries $9n/7+O(1)$ edges to construct a rigid *ppg* on n points. We use the graph in Figure 4.1 with density 6/5 as the basic building block. It can be construed as 3 paths $p_1q_1r_1s$, $p_2q_2r_2s$ and $p_3q_3r_3s$ with a common

terminal vertex s. Hence we call it a 3-path graph and formally define it as $G_{3p} = (V_{3p}, E_{3p})$ where

$$V_{3p} = \{p_1, q_1, r_1, p_2, q_2, r_2, p_3, q_3, r_3, s\} \text{ and}$$

$$E_{3p} = \{p_1q_1, q_1r_1, r_1s, p_2q_2, q_2r_2, r_2s, p_3q_3, q_3r_3, r_3s, p_1p_2, p_2p_3, p_3p_1\}.$$

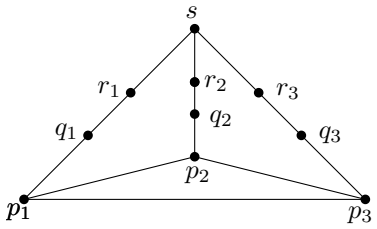


Figure 4.1: The 3-path graph

Since the G_{3p} can be drawn as a layer graph (see Figure 4.4), by Theorem 23 it is not intrinsically rigid. Indeed, there does not exist an intrinsically rigid graph of density $6/5$ in view of the lower bound of $4/3$ on any rigid graph with the exception of the graphs $K_3, K_4^-, K_{2,3}$ and the jewel [26].

To find a set of conditions that make G_{3p} rigid, first we fix the placements of p_1, p_2 and p_3. Next, we find conditions that make the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ rigid. Relative to the fixed placement of p_3 and s we have a 4-cycle (p_3, q_3, r_3, s) with a virtual edge between s and p_3. Adding a condition that makes this 4-cycle rigid to the set of rigidity conditions of the 7-cycle, gives us a set of conditions that makes G_{3p} rigid.

Let us describe how we construct a rigid G_{3p}. First, we make p_1, p_2 and p_3 rigid in the first round query. In the second round, let us first we make the 7-cycle rigid by choosing the edges of G_{3p} in such a way that the rigidity conditions for it are satisfied. Then we make

the virtual 4-cycle rigid by choosing the edges of G_{3p} in such a way that rigidity conditions for this cycle are satisfied. Clearly, the G_{3p} constructed thus will be rigid. It is evident that this way of choosing the edges of G_{3p} in the second round is equivalent to choosing its edges, in the second round, by satisfying the union of the conditions for line rigidity of the 2 cycles. We shall follow the latter method.

To find the rigidity conditions for the 7-cycle, we resort to exhaustive enumerations of all the layer graph representations of the cycle. By Theorem 27, a 7-cycle has 42 different layer graph representations. We find conditions that prohibit the drawing of any them. By Theorem 23, they constitute the set of conditions for the line rigidity of the 7-cycle. All of the 42 layer graphs for a 7-cycle can be grouped into 6 groups based on the number of edges on each side. For the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$, a representative layer graph for each of those groups is shown in Figure 4.2.

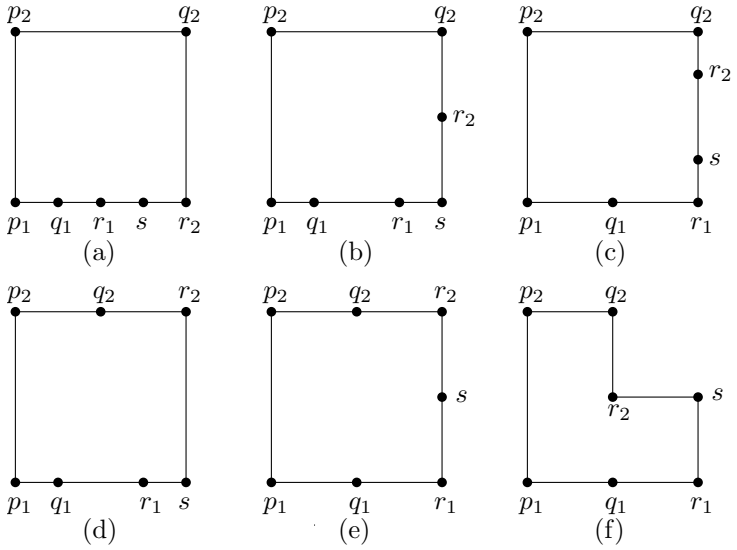


Figure 4.2: An example of each group of layer graph for the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$

From these layer graphs we deduce the conditions for line rigidity of this cycle. The result is summarized in the following lemma (Lemma 56). As an example, we consider the group of 7 layer graphs represented by Figure 4.2(a). Each layer graph in this group has 4 edges on one side of the layer graph. The layer graphs are shown in Figure 4.3. From these layer graphs we deduce the following rigidity conditions: $|p_1q_1| \neq |r_1s|, |q_1r_1| \neq |sr_2|, |r_1s| \neq |r_2q_2|, |sr_2| \neq |q_2p_2|, |r_2q_2| \neq |p_2p_1|, |q_2p_2| \neq |p_1q_1|$ and $|p_2p_1| \neq |q_1r_1|$.

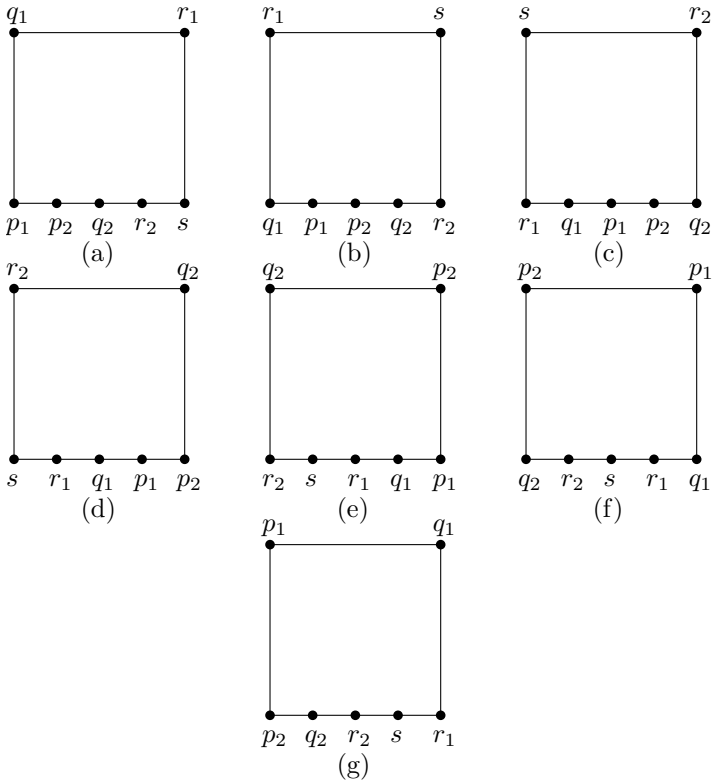


Figure 4.3: A group of layer graphs for the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ where 4 edges lie on one edge of the layer graph

Lemma 56 *A 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *is rigid if*

1. $|p_1p_2| \neq |q_2r_2|$, $|p_1p_2| \neq |q_1r_1|$, $|p_2q_2| \neq |r_2s|$, $|p_1q_1| \neq |r_1s|$, $|q_2r_2| \neq |r_1s|$, $|q_1r_1| \neq |r_2s|$, $|p_1q_1| \neq |p_2q_2|$.

2. $||p_1p_2|\pm|p_2q_2|| \neq |r_2s|$, $||p_2q_2|\pm|q_2r_2|| \neq |r_1s|$, $||p_1p_2|\pm|p_1q_1|\pm|p_2q_2|| \neq |r_1s|$, $|p_1q_1| \neq ||r_1s|\pm|r_2s||$, $|p_1p_2| \neq ||q_1r_1|\pm|r_1s||$, $||p_1q_1|\pm|q_1r_1|| \neq |p_2q_2|$, $||p_1q_1|\pm|p_1p_2|| \neq |q_2r_2|$.

3. $||p_1p_2|\pm|p_1q_1|| \neq |r_1s|$, $||p_1q_1|\pm|q_1r_1|| \neq |r_2s|$, $||p_1p_2|\pm|p_1q_1|\pm|p_2q_2|| \neq |r_2s|$, $|p_2q_2| \neq ||r_1s|\pm|r_2s||$, $|p_1p_2| \neq ||q_2r_2|\pm|r_2s||$, $||p_2q_2|\pm|q_2r_2|| \neq |p_2q_2|$, $||p_2q_2|\pm|p_1p_2|| \neq |q_1r_1|$.

4. $|p_1p_2| \neq |r_2s|$, $|p_1p_2| \neq |r_1s|$, $|p_2q_2| \neq |r_1s|$, $|p_1q_1| \neq |r_2s|$, $||p_1q_1| \pm |p_2q_2| \pm |p_1p_2|| \neq ||r_1s| \pm |r_2s||$, $|p_2q_2| \neq |q_1r_1|$, $|p_1q_1| \neq |q_2r_2|$.

5. $|p_2q_2| \neq ||p_1p_2| \pm |r_1s||$, $|p_1q_1| \neq ||p_1p_2| \pm |r_2s||$, $|p_1q_1| \neq ||p_1p_2| \pm |r_1s| \pm |r_2s||$, $|p_2q_2| \neq ||p_1p_2|\pm|r_1s|\pm|r_2s||$, $|p_1q_1| \neq ||q_2r_2|\pm|r_2s||$, $|p_2q_2| \neq ||q_1r_1|\pm|r_1s||$, $|p_1p_2| \neq ||r_1s| \pm |r_2s||$.

6. $||p_1q_1|\pm|q_1r_1|| \neq ||p_2q_2|\pm|r_2s||$, $||p_2q_2|\pm|q_2r_2|| \neq ||p_1q_1|\pm|r_1s||$, $|q_1r_1| \neq ||p_2q_2|\pm|r_2s||$, $|q_2r_2| \neq ||p_1q_1| \pm |r_1s||$, $|p_2q_2| \neq ||p_1q_1| \pm |r_1s||$, $|p_1q_1| \neq ||p_2q_2| \pm |r_2s||$, $|p_2q_2| \neq ||p_1q_1| \pm |r_1s| \pm |r_2s||$.

Proof. The proof is similar to the proof of the corresponding lemma for 5-cycle given by Chin *et al.* [20], and is omitted. □

The above conditions involve all the edges of the 7-cycle. If we query the lengths of all the edges of the cycle in the first round of a 2-round algorithm, the edge lengths may not satisfy all the rigidity conditions. It is evident from the 2-dimensional stretch of layer graph that we can avoid the length of an edge from all the conditions of rigidity for a cycle.

We avoid q_1r_1 and q_3r_3 from the conditions of rigidity for the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ and the virtual 4-cycle (p_3, q_3, r_3, s) respectively. Then the conditions for rigidity of each of these cycles will involve all the other edges in the corresponding cycle.

Again, for each rigidity condition we need to have at least one edge in the condition such that we can choose an edge with suitable length, that satisfies the rigidity condition, as that edge, from among the options for edges with different lengths for that particular edge. Thus, we need to have choices for edge lengths of some edges so that we can avoid some edge lengths for some edges according to the conditions for rigidity. We provide these choices for p_1q_1 and p_3q_3.

Since the rigidity conditions will involve neither q_1r_1 nor q_3r_3, G_{3p} will be rigid irrespective of the lengths of those edges. If we query q_1r_1 and q_3r_3 in the second round then we can create enough choices for p_1q_1 and p_3q_3 in the first round to satisfy any rigidity condition involving any of them.

There will be rigidity conditions for the 7-cycle that will not involve these edges, i.e., p_1q_1 and p_3q_3. We cannot meet those rigidity conditions in a 2-round algorithm. So, we need to avoid some other edge(s) from the rigidity conditions of the cycle and/or provide options for choosing some other edge(s) for the cycle. For the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ there will be rigidity conditions involving p_2q_2 and r_2s, p_2q_2 and r_1s, and all the 3 edges p_2q_2, r_1s and r_2s. We can meet all those conditions, if we provide sufficient options for choosing the length of p_2q_2. We can provide choices for edge lengths of the edge p_2q_2 in the

first round if we do not query the edge q_2r_2 of the 7-cycle in the first round.

Thus, we do not query the lengths of the edges q_1r_1, q_2r_2 and q_3r_3 in the first round. We query them in the second round. We find a set of sufficient conditions for rigidity for G_{3p} that does not involve these edges. The 7-cycle have rigidity conditions involving either q_1r_1 or q_2r_2. We replace each of its rigidity conditions that involve any of these edges. We replace each such condition by a set of condition(s) that prevents the cycle from being drawn as the layer graph representation that corresponds to that condition. Then we can satisfy all the rigidity conditions irrespective of the lengths of these edges which will be reported in the second round.

Among the 42 conditions in Lemma 56 for line rigidity of the 7-cycle, 20 conditions involve either q_1r_1 or q_2r_2 of the 7-cycle that we want to avoid in the conditions. We replace each of these conditions by a set of conditions that prevents the 7-cycle from being drawn as the layer graph representation that corresponds to that condition. By Theorem 23, the set of all these new conditions and the ones that are not replaced will constitute the rigidity conditions for the 7-cycle. As stated before, if the 7-cycle is rigid then the (p_3, q_3, r_3, s) will be a 4-cycle in the second round and it can be made rigid by imposing the condition [20]:

$$|p_3q_3| \neq |r_3s|. \tag{4.1}$$

This condition together with the rigidity conditions for the 7-cycle will constitute the rigidity conditions for the whole G_{3p}.

In the next subsection, we show how to replace the above mentioned 20 conditions that involve the edges q_1r_1 and q_2r_2 of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ with the ones that do not involve them. To this end, for each of these conditions, first we try to find use other edges of the cycle in the layer graph representation corresponding to that condition. If this fails then we embed the layer graph representation corresponding to that condition into all possible layer graph representations of the whole G_{3p}, and derive a rigidity condition from each such embedding.

4.2.1 Replacing Conditions

As an example of replacing conditions we shall replace the first condition, viz., $|p_1p_2| \neq |q_2r_2|$.

Replacing $|p_1p_2| \neq |q_2r_2|$

The rigidity condition $|p_1p_2| \neq |q_2r_2|$ corresponds to the layer graph of Figure 4.2(a). To replace this condition we find a set of conditions that prevent the drawing of layer graph of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ in the configuration of Figure 4.2(a).

Lemma 57 *The 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ of G_{3p} cannot be drawn as the layer graph of Figure 4.2(a) if the edges of G_{3p} satisfy the following set of conditions:*

$\{|p_1p_3| \neq ||p_3q_3| \pm |r_3s||, |p_1p_3| \neq |r_3s||, ||p_3q_3| \pm |sr_2|| \neq |p_2q_2|, ||p_3q_3| \pm |sr_2| \pm |sr_3|| \neq |p_2q_2|\}$.

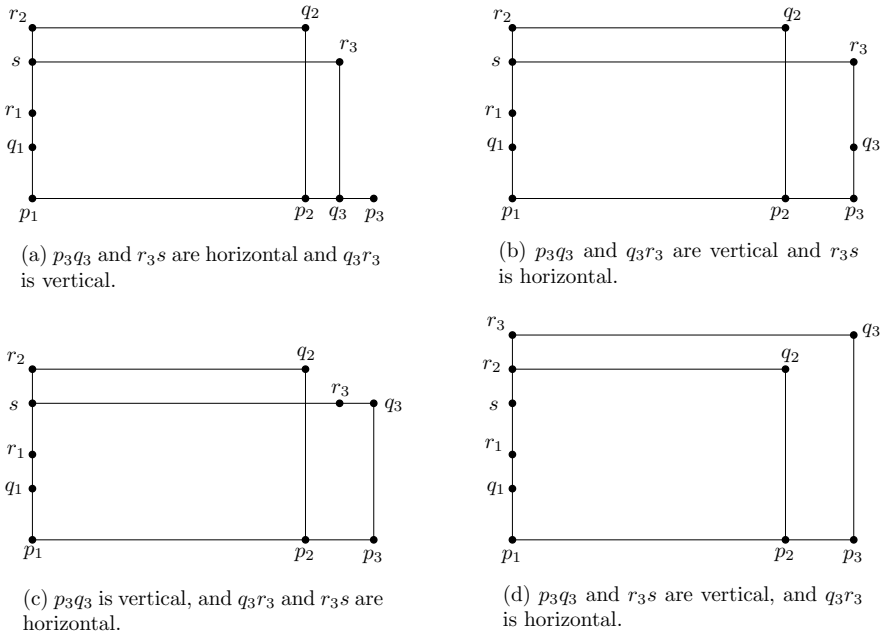
Proof. We consider all possible layer graphs of G_{3p} in which the 7-cycle appears in the

above fixed configuration. For each such layer graph of G_{3p}, we find the condition or set of conditions that prevents G_{3p} from being drawn as a layer graph of that configuration and, a fortiori, the embedded 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ in the configuration of Figure 4.2(a). This new set of conditions acts as a replacement for the condition $|p_1p_2| \neq |q_2r_2|$ since that set will prevent the drawing of the layer graph of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ in the corresponding configuration in Figure 4.2(a).

Since p_1, p_2 and p_3 are made rigid in the first round, they must lie on a line and their positions must be unique (up to translation and reflection) after the first round. In the present configuration of the 7-cycle (Figure 4.2(a)), p_1, q_1, r_1, s and r_2 are on the same side of the layer graph. Since p_1 and s are collinear and they lie on a line perpendicular to the line through p_1, p_2 and p_3, the edges p_3q_3, q_3r_3 and r_3s can have 4 distinct configurations giving rise to 4 distinct layer graph representations (Figure 4.4) of the whole G_{3p} with the layer graph of the 7-cycle being in the configuration of Figure 4.2(a). Thus, in order to be able to draw the layer graph of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ in the configuration of Figure 4.2(a) the layer graph of the whole G_{3p} must have one of the four distinct configurations as shown in Figure 4.4.

First, we consider the configuration where p_3q_3 and r_3s are horizontal, and q_3r_3 is vertical (Figure 4.4a). The condition $|p_1p_2| \neq |q_2r_2|$ prevents the 7-cycle from being drawn as a layer graph of present configuration. However, it involves the edge q_2r_2 which we need to avoid. In the present configuration of the layer graph of the G_{3p} p_1, q_1, r_1, s and r_2 are on a line

which is parallel to p_2q_2 and q_3r_3. So, we must have $|q_2r_2| = ||p_2p_3| \pm |p_3q_3| \pm |r_3s||$. Using this the condition becomes $|p_1p_2| \neq ||p_2p_3| \pm |p_3q_3| \pm |r_3s||$. Since $||p_1p_2| \pm |p_2p_3|| = |p_1p_3|$ the condition reduces to $|p_1p_3| \neq ||p_3q_3| \pm |r_3s||$. If we ensure this condition then we must have $|p_1p_2| \neq |q_2r_2|$ in the present configuration of G_{3p}. Thus, G_{3p} in general and the 7-cycle in particular cannot be drawn as a layer graph in the present configurations of the 7-cycle and G_{3p}.



(a) p_3q_3 and r_3s are horizontal and q_3r_3 is vertical.

(b) p_3q_3 and q_3r_3 are vertical and r_3s is horizontal.

(c) p_3q_3 is vertical, and q_3r_3 and r_3s are horizontal.

(d) p_3q_3 and r_3s are vertical, and q_3r_3 is horizontal.

Figure 4.4: Layer graphs of G_{3p} when the layer graph of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ has 4 edges p_1q_1, q_1r_1, r_1s and sr_2 on one side

Now we consider the case when p_3q_3 and q_3r_3 are vertical, and r_3s is horizontal (Figure 4.4b). In the present configuration of the layer graph of G_{3p}, p_1, q_1, r_1, s and r_2 are on a line, and p_3q_3 and q_3r_3 are on a line. Those lines are parallel and they are parallel to p_2q_2. So, we must have $|q_2r_2| = ||p_2p_3| \pm |r_3s||$. Using this the condition becomes $|p_1p_2| \neq ||p_2p_3| \pm |r_3s||$. We have $||p_1p_2| \pm |p_2p_3|| = |p_1p_3|$. Using this the rigidity condition

$|p_1p_2| \neq |q_2r_2|$ becomes $|p_1p_3| \neq |r_3s|$.

Next, we consider the case when p_3q_3 is vertical, and q_3r_3 and r_3s are horizontal (Figure 4.4c). The condition $||p_1q_1| \pm |q_1r_1| \pm |r_1s| \pm |sr_2|| \neq |p_2q_2|$ prevents the 7-cycle from being drawn as a layer graph of present configuration. However, it involves the edge q_1r_1 which we need to avoid. In the present configuration of the layer graph of G_{3p} p_1, p_2 and p_3 are on a line, and q_3, r_3 and s are on a line. The lines are parallel. So, we must have $||p_1q_1| \pm |q_1r_1| \pm |r_1s|| = |p_3q_3|$. Using this the condition becomes $||p_3q_3| \pm |sr_2|| \neq |p_2q_2|$.

Finally, we consider the case when p_3q_3 is vertical, q_3r_3 is horizontal and r_3s is vertical (Figure 4.4d). In the present configuration of the layer graph of G_{3p}, p_1, p_2 and p_3 are on a line. The line is parallel to q_3r_3. So, we must have $||p_1q_1| \pm |q_1r_1| \pm |r_1s| \pm |sr_3|| = |p_3q_3|$. Using this the rigidity condition $||p_1q_1| \pm |q_1r_1| \pm |r_1s| \pm |sr_2|| \neq |p_2q_2|$ becomes $||p_3q_3| \pm |sr_3| \pm |sr_2|| \neq |p_2q_2|$.

It follows that there is no layer graph for G_{3p} in which the layer graph in Figure 4.2(a) of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ is embedded if the edges of G_{3p} satisfy the conditions in the statement of this lemma. So, the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ of G_{3p} cannot be drawn as the layer graph of Figure 4.2(a) if the edges of G_{3p} satisfy those conditions. □

Similarly, we can replace the remaining 3 conditions corresponding to the layer graphs in group 1 and involving the edges q_1r_1 and q_2r_2. The result is summarized in the following lemma:

Lemma 58 *The 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ of G_{3p} cannot be drawn as the layer graphs*

corresponding to the conditions $|p_1p_2| \neq |q_1r_1|$, $|q_2r_2| \neq |r_1s|$ *and* $|q_1r_1| \neq |r_2s|$ *if the edges of* G_{3p} *satisfy the following conditions:*

1. $||p_1p_3| \pm |r_3s|| \neq |p_3q_3|$, $|r_3s| \neq ||p_2p_3| \pm |r_2s| \pm |p_2q_2||$, $|p_3q_3| \neq |r_1s|$ *and* $|p_3q_3| \neq ||r_1s| \pm |r_3s||$.

2. $||p_2p_3| \pm |r_3s|| \neq |p_3q_3|$, $|r_3s| \neq ||p_1p_3| \pm |r_1s| \pm |p_1q_1||$, $|p_3q_3| \neq |r_2s|$ *and* $|p_3q_3| \neq ||r_2s| \pm |r_3s||$.

3. $||p_3q_3| \pm |r_3s|| \neq |p_2p_3|$, $|p_2p_3| \neq |r_2s|$, $|p_3q_3| \neq ||p_1q_1| \pm |r_1s||$ *and* $||p_3q_3| \pm |r_3s|| \neq ||p_1q_1| \pm |r_1s||$.

Proof. Similar to the proof of Lemma 57. □

Similarly, we can replace the 4 conditions corresponding to the layer graphs in group 2 and involving the edges q_1r_1 and q_2r_2. The result is summarized in the following lemma:

Lemma 59 *The 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *of* G_{3p} *cannot be drawn as the layer graphs corresponding to the conditions* $||p_2q_2| \pm |q_2r_2|| \neq |r_1s|$, $|p_1p_2| \neq ||q_1r_1| \pm |r_1s||$, $||p_1q_1| \pm |q_1r_1|| \neq |p_2q_2|$ *and* $||p_1q_1| \pm |p_1p_2|| \neq |q_2r_2|$ *if the edges of* G_{3p} *satisfy the following conditions:*

1. $||p_3q_3| \pm |r_3s|| \neq ||p_2p_3| \pm |r_2s||$, $||p_2p_3| \pm |r_2s|| \neq |r_3s|$, $|p_3q_3| \neq |r_1s|$ *and* $||p_3q_3| \neq |r_1s| \pm |r_3s||$.

2. $||p_3q_3| \pm |r_3s|| \neq |p_2p_3|$, $|p_2p_3| \neq |r_3s|$, $|p_1q_1| \neq |r_1s|$, $|p_2q_2| \neq |r_2s|$, $|p_3q_3| \neq |p_1q_1|$ *and* $||p_3q_3| \pm |p_1q_1| \neq |r_3s||$.

3. $||p_3q_3| \pm |r_3s| \pm |r_1s|| \neq |p_1p_3|$, $|p_2q_2| \neq |r_2s|$, $|p_1p_3| \neq |r_1s| \pm |r_3s|$, $|p_2q_2| \neq |p_3q_3|$, $|p_1q_1| \neq |r_1s|$ *and* $|p_2q_2| \neq ||p_3q_3| \pm |r_3s||$.

4. $|p_3q_3| \pm |r_3s| \pm |p_1q_1| \neq |p_1p_3|$, $|p_2q_2| \neq |r_2s|$, $|p_1q_1| \pm |p_1p_3| \neq |r_3s|$, $|p_2q_2| \pm |r_2s| \neq |p_3q_3|$, $|p_1q_1| \neq |r_1s|$ *and* $||p_2q_2| \pm |r_2s|| \neq ||p_3q_3| \pm |r_3s||$.

Proof. Similar to the proof of Lemma 57. □

Replacing $||p_2q_2| \pm |p_1p_2|| \neq |q_1r_1|$

Now we replace the condition $||p_2q_2| \pm |p_1p_2|| \neq |q_1r_1|$ of group 3. Corresponding layer graph of the 7-cycle as well as all the possible configurations of G_{3p} for this case are shown in Figure 4.5. From the figure we obtain the replacement conditions as before:

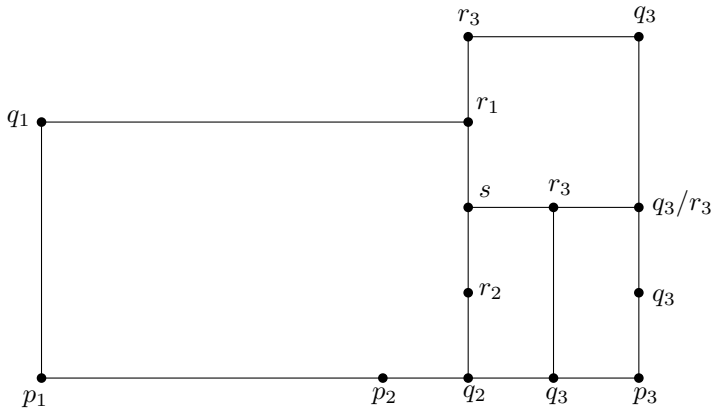


Figure 4.5: 2 edges are on one side of layer graph and 3 edges are on its adjacent side

Lemma 60 *The 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *of* G_{3p} *cannot be drawn as the layer graph of Figure 4.5 corresponding to the condition* $||p_2q_2| \pm |p_1p_2|| \neq |q_1r_1|$ *if the edges of* G_{3p}

satisfy the following conditions:

$|p_3q_3| \pm |r_3s| \pm |p_2q_2| \neq |p_2p_3|$, $|p_1q_1| \neq |r_1s|$, $|p_2q_2| \pm |p_2p_3| \neq |r_3s|$, $|p_1q_1| \pm |r_1s| \neq |p_3q_3|$, $|p_2q_2| \neq |r_2s|$ *and* $|p_1q_1| \pm |r_1s| \neq |p_3q_3| \pm |r_3s|$.

Proof. Similar to the proof of Lemma 57. □

Similarly, we can replace the remaining 3 conditions corresponding to the layer graphs in group 3 and involving the edges q_1r_1 and q_2r_2. The result is summarized in the following lemma:

Lemma 61 *The 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *of* G_{3p} *cannot be drawn as the layer graphs corresponding to the conditions* $||p_1q_1| \pm |q_1r_1|| \neq |r_2s|$, $|p_1p_2| \neq ||q_2r_2| \pm |r_2s||$ *and* $||p_2q_2| \pm |q_2r_2|| \neq |p_2q_2|$ *if the edges of* G_{3p} *satisfy the following conditions:*

1. $||p_3q_3| \pm |r_3s| \pm |r_2s|| \neq |p_2p_3|$, $|p_1q_1| \neq |r_1s|$, $|p_2p_3| \neq |r_2s| \pm |r_3s|$, $|p_1q_1| \neq |p_3q_3|$, $|p_2q_2| \neq |r_2s|$ *and* $||p_1q_1| \neq |p_3q_3|| \pm |r_3s|$.

2. $||p_3q_3| \pm |r_3s|| \neq |p_1p_3|$, $|p_1p_3| \neq |r_3s|$, $|p_2q_2| \neq |r_2s|$, $|p_1q_1| \neq |r_1s|$, $|p_3q_3| \neq |p_2q_2|$ *and* $||p_3q_3| \pm |p_2q_2| \neq |r_3s||$.

3. $||p_3q_3| \pm |r_3s|| \neq ||p_1p_3| \pm |r_1s||$, $|p_1p_3| \pm |r_1s| \neq |r_3s|$, $|p_3q_3| \neq |r_2s|$ *and* $||p_3q_3| \neq |r_2s| \pm |r_3s||$.

Proof. Similar to the proof of Lemma 57. □

Similarly, we can replace the 2 conditions corresponding to the layer graphs in group 4 and involving the edges q_1r_1 and q_2r_2. The result is summarized in the following lemma:

Lemma 62 *The 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ of G_{3p} cannot be drawn as the layer graphs corresponding to the conditions $|p_2q_2| \neq |q_1r_1|$ and $|p_1q_1| \neq |q_2r_2|$. if the edges of G_{3p} satisfy the following conditions:*

1. $||p_3q_3| \pm |r_3s| \pm |r_1s|| \neq ||p_1p_3| \pm |p_1q_1||$, $||r_3s| \pm |r_1s|| \neq ||p_1p_3| \pm |p_1q_1||$, $|p_3q_3| \neq |p_2q_2|$, $||p_3q_3| \pm |r_3s|| \neq |p_2q_2|$ *and* $||p_1q_1| \pm |p_1p_3|| \neq |r_1s|$.

2. $||p_3q_3| \pm |r_3s| \pm |r_2s|| \neq ||p_2p_3| \pm |p_2q_2||$, $||r_3s| \pm |r_2s|| \neq ||p_2p_3| \pm |p_2q_2||$, $|p_3q_3| \neq |p_1q_1|$, $||p_3q_3| \pm |r_3s|| \neq |p_1q_1|$ *and* $||p_2q_2| \pm |p_2p_3|| \neq |r_2s|$.

Proof. Similar to the proof of Lemma 57. □

Similarly, we can replace the 2 conditions corresponding to the layer graphs in group 5 and involving the edges q_1r_1 and q_2r_2. The result is summarized in the following lemma:

Lemma 63 *The 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ of G_{3p} cannot be drawn as the layer graphs corresponding to the conditions $|p_1q_1| \neq ||q_2r_2| \pm |r_2s||$ and $|p_2q_2| \neq ||q_1r_1| \pm |r_1s||$ if the edges of G_{3p} satisfy the following conditions:*

1. $||p_2q_2| \pm |p_3q_3| \pm |p_2p_3|| \neq |r_3s|$, $||p_2q_2| \pm |p_2p_3|| \neq |r_3s|$, $|p_1q_1| \neq |p_3q_3|$ *and* $|p_1q_1| \neq ||p_3q_3| \pm |r_3s||$.

2. $||p_1q_1| \pm |p_3q_3| \pm |p_1p_3|| \neq |r_3s|$, $||p_1q_1| \pm |p_1p_3|| \neq |r_3s|$, $|p_2q_2| \neq |p_3q_3|$ *and* $|p_2q_2| \neq ||p_3q_3| \pm |r_3s||$.

Proof. Similar to the proof of Lemma 57. □

Replacing $||p_1q_1| \pm |q_1r_1|| \neq ||p_2q_2| \pm |r_2s||$

Now we replace the condition $||p_1q_1| \pm |q_1r_1|| \neq ||p_2q_2| \pm |r_2s||$ of group 6. Corresponding layer graph of the 7-cycle as well as all the possible configurations of G_{3p} for this case are shown in Figure 4.6. From the figure we obtain the replacement conditions as before:

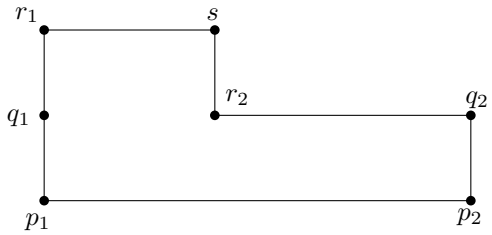


Figure 4.6: The layer graph is staircase shaped with p_1q_1 and q_1r_1 on one side

Lemma 64 *The 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *of* G_{3p} *cannot be drawn as the layer graph of Figure 4.6 corresponding to the condition* $||p_1q_1| \pm |q_1r_1|| \neq ||p_2q_2| \pm |r_2s||$ *if the edges of* G_{3p} *satisfy the following conditions:*

$||p_3q_3| \pm |r_3s| \pm |r_1s|| \neq |p_1p_3|$, $||r_3s| \pm |r_1s|| \neq |p_1p_3|$, $|p_3q_3| \neq ||p_2q_2| \pm |r_2s||$, $|p_3q_3| \neq ||p_2q_2| \pm |r_2s| \pm |r_3s||$ *and* $|p_1p_3| \neq |r_1s|$.

Proof. Similar to the proof of Lemma 57. □

Similarly, we can replace the remaining 3 conditions corresponding to the layer graphs in group 6 and involving the edges q_1r_1 and q_2r_2. The result is summarized in the following

lemma:

Lemma 65 *The 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *of* G_{3p} *cannot be drawn as the layer graphs corresponding to the conditions* $||p_2q_2| \pm |q_2r_2|| \neq ||p_1q_1| \pm |r_1s||$, $|q_1r_1| \neq ||p_2q_2| \pm |r_2s||$ *and* $|q_2r_2| \neq ||p_1q_1| \pm |r_1s||$ *if the edges of* G_{3p} *satisfy the following conditions:*

1. $||p_1q_1| \pm |p_1p_3|| \neq ||p_3q_3| \pm |r_1s| \pm |r_3s||$, $||p_1q_1| \pm |p_1p_3|| \neq ||r_1s| \pm |r_3s||$, $|p_3q_3| \neq ||p_2q_2| \pm |r_2s||$, $|p_3q_3| \neq ||p_2q_2| \pm |r_2s| \pm |r_3s||$ *and* $|p_1q_1| \neq ||p_1p_3| \pm |r_1s||$.

2. $||p_2q_2| \pm |p_2p_3|| \neq ||p_3q_3| \pm |r_2s|| \pm |r_3s|$, $||p_2q_2| \pm |p_2p_3|| \neq |r_2s| \pm |r_3s|$, $|p_3q_3| \neq ||p_1q_1| \pm |r_1s||$, $|p_3q_3| \neq ||p_1q_1| \pm |r_1s| \pm |r_3s||$ *and* $|p_2q_2| \neq ||p_2p_3| \pm |r_2s||$.

3. $||p_3q_3| \pm |r_3s| \pm |r_2s|| \neq |p_2p_3|$, $||r_3s| \pm |r_2s|| \neq |p_2p_3|$, $|p_3q_3| \neq ||p_1q_1| \pm |r_1s||$, $|p_3q_3| \neq ||p_1q_1| \pm |r_1s| \pm |r_3s||$ *and* $|p_2p_3| \neq |r_2s|$.

Proof. Similar to the proof of Lemma 57. □

4.2.2 Rigidity Conditions

From Eq. 4.1 and Lemmas 56 - 65 we have the following lemma for the rigidity of the 7-cycle $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ of G_{3p}.

Lemma 66 *The 7-cycle* $(p_1, q_1, r_1, s, r_2, q_2, p_2)$ *of* G_{3p} *is rigid if the edges of* G_{3p} *satisfy the following conditions:*

1. $|p_1p_2| \notin \{|r_1s|, |r_2s|, ||r_1s| \pm |r_2s||\}$,

2. $|p_2p_3| \notin \{|r_2s|, |r_3s|, ||r_2s| \pm |r_3s||\}$,

3. $|p_3p_1| \notin \{|r_3s|,\ |r_1s|,\ ||r_3s| \pm |r_1s||\}$,

4. $|p_1q_1| \notin \{|r_1s|,\ |r_2s|,\ ||r_1s| \pm |r_2s||,\ ||p_1p_2| \pm |r_1s||,\ ||p_1p_2| \pm |r_2s||,\ ||p_1p_3| \pm |r_1s||,$ $||p_1p_3| \pm |r_3s||,\ ||p_1p_2| \pm |r_1s| \pm |r_2s||,\ ||p_1p_3| \pm |r_1s| \pm |r_3s||\}$,

5. $|p_2q_2| \notin \{|r_1s|,\ |r_2s|,\ |p_1q_1|,\ ||r_1s| \pm |r_2s||,\ ||p_1p_2| \pm |r_1s||,\ ||p_1p_2| \pm |r_2s||,\ ||p_2p_3| \pm |r_2s||,$ $||p_2p_3| \pm |r_3s||,\ ||p_1q_1| \pm |r_1s||,\ ||p_1q_1| \pm |r_2s||,\ ||p_1p_2| \pm |r_1s| \pm |r_2s||,\ ||p_2p_3| \pm |r_2s| \pm |r_3s||,$ $||p_1q_1| \pm |r_1s| \pm |r_2s||,\ ||p_1q_1| \pm |p_1p_2| \pm |r_1s||,\ ||p_1q_1| \pm |p_1p_2| \pm |r_2s||,\ ||p_1q_1| \pm |p_1p_2| \pm$ $|r_1s| \pm |r_2s||\}$,

6. $|p_3q_3| \notin \{|r_1s|,\ |r_2s|,\ |r_3s|,\ |p_1q_1|,\ |p_2q_2|,\ ||r_2s| \pm |r_3s||,\ ||r_3s| \pm |r_1s||,\ ||p_1p_3| \pm |r_3s||,$ $||p_2p_3| \pm |r_3s||,\ ||p_1q_1| \pm |r_1s||,\ ||p_1q_1| \pm |r_3s||,\ ||p_2q_2| \pm |r_2s||,\ ||p_2q_2| \pm |r_3s||,\ ||p_1p_3| \pm$ $|r_1s| \pm |r_3s||,\ ||p_2p_3| \pm |r_2s| \pm |r_3s||,\ ||p_1q_1| \pm |r_1s| \pm |r_3s||,\ ||p_2q_2| \pm |r_2s| \pm |r_3s||,$ $||p_1q_1| \pm |p_1p_3| \pm |r_3s||,\ ||p_2q_2| \pm |p_2p_3| \pm |r_3s||,\ ||p_1q_1| \pm |p_1p_3| \pm |r_1s| \pm |r_2s||,\ ||p_2q_2| \pm$ $|p_2p_3| \pm |r_2s| \pm |r_3s||\}$.

The union of the two sets of conditions in Eq. 4.1 and Lemma 66 constitutes a set of sufficient conditions for the rigidity of G_{3p}. Taking care of overlapping conditions between the two sets of conditions, we have 55 distinct conditions for the rigidity of G_{3p} and hence the following lemma:

Lemma 67 *The G_{3p} having the vertices p_1, p_2 and p_3 rigid in the first round, is rigid if its edges satisfy the conditions mentioned in Lemma 66.*

4.2.3 Algorithm

As mentioned before, we make triplet of vertices (p_1, p_2, p_3) of each G_{3p} rigid in the first round. But we have rigidity conditions on the edges p_1p_2, p_2p_3 and p_3p_1 (Conditions 1-3 of Lemma 67). This implies that we need a pool of vertices, S, for which the pairwise distances of all the pairs of points corresponding to the vertices in S are known after the first round of query, and from which we choose the triplet of vertices (p_1, p_2, p_3) in order to meet the rigidity conditions on p_1p_2, p_2p_3 and p_3p_1. We make the vertices in S rigid in the first round. Then the pairwise distances of all the pairs of points corresponding to the vertices in S are known after the first round of query. We make the remaining 7 vertices of each G_{3p} rigid in the second round.

To select triplet of vertices in S as (p_1, p_2, p_3) of a G_{3p}, let us select any vertex of S as p_1. Then let us find another vertex of S, we denote it as p_2, satisfying the conditions on the length $|p_1p_2|$ mentioned in Condition 1 of Lemma 67. The length of p_1p_2 cannot be equal to at most 4 different lengths. By Observation 4, each length can be attained by at most 2 edges incident on p_1. Thus, at most 8 edges will not satisfy the conditions on $|p_1p_2|$. We need at least 8 extra vertices, i.e., we need to have a total of at least 9 more vertices, other than p_1, in S as candidate for p_2.

After p_2 is selected, let us find another vertex of S, we denote it as p_3, from the remaining vertices of S such that the conditions on $|p_2p_3|$ in Condition 2 of Lemma 67 are satisfied. By Observation 4, at most 8 edges will not satisfy the conditions on $|p_2p_3|$. This warrants the

set S to have at least 8 extra vertices other than p_1, p_2 and p_3. The vertex p_3 selected this way by satisfying the conditions on p_2p_3 must also have to satisfy the conditions on p_3p_1 mentioned in Condition 3 of Lemma 67. By Observation 4, at most 8 edges will not satisfy the conditions on $|p_3p_1|$. This warrants the set S to have at least 8 more extra vertices, i.e., a total of 16 extra vertices, other than p_1, p_2 and p_3. Then it is ensured that a triplet of vertices in S can be found as (p_1, p_2, p_3) of a G_{3p}.

But if S has only 19 vertices for the selection of p_is it may happen that all the G_{3p}s are attached to the same triplets. This hinders our goal of obtaining a better value for α than previously known. We need to attach G_{3p}s evenly to all the vertices of S so that the same number of edges can be attached to each of them in the first round and all of those edges, except for a constant number, are used to attach the basic components. In other words, we need to attach the 3-paths to the vertices in S in such a way that the numbers of G_{3p}s attached to any two vertices differ by at most a constant number. To specify the number of G_{3p}s attached to a vertex in S we shall use the term valence. We denote the set of vertices with valence d as S_d.

Now we describe our algorithm to select triplets of vertices in S to attach G_{3p}s. To attach a G_{3p} we always select a vertex in S with the lowest valence as the first vertex (say p_1). Of the remaining vertices of S, at most 8 vertices may not be acceptable for the second vertex (say p_2), because of the conditions on p_1p_2. From among the rest $|S| - 1$ vertices that satisfy the conditions on p_1p_2 we select the one that has the lowest valence, as p_2.

Of the rest $|S| - 2$ vertices of S, at most 16 may not be acceptable for the last vertex, say p_3, because of the conditions on p_2p_3 and p_3p_1. From among the rest vertices that satisfy the conditions on p_2p_3 and p_3p_1 we choose the one that has the lowest valence, as p_3. This method is followed to attach each G_{3p} to the vertices in S, while G_{3p}s are attached sequentially. The following lemma tells us how big S must be:

Lemma 68 *A set S of 35 vertices is sufficient to ensure that the valences of any two vertices in S differ by at most 2.*

Proof. Initially, all the vertices in the pool have valence 0. To pick three vertices of minimum valence (0 in this case) to beat 16 different length restrictions in all, we need a buffer of size at least 16. Thus we can pick 6 triplets (p_1, p_2, p_3) of valence 0, until the buffer limit is reached. At the end of this cycle, 18 vertices have valence 1 and 17 vertices have valence 0. The next cycle begins by picking pairs (p_1, p_2) from the pool of 17 vertices of valence 0, as long as we have a buffer of size 8. Since we do not have enough vertices in the buffer to ensure that the third vertex p_3 is from the valence 0 pool, we might have to pick these from the pool of valence 1 vertices. So up to 4 vertices can have valence 2. Thus, we have $|S_0| \leq 9$ and $|S_2| \leq 4$. The rest of the vertices are of valence 1, i.e., $|S_1| \geq 22$.

Next we attach G_{3p}s until $|S_0| \leq 2$. This will attach at most 7 G_{3p}s. Then we have $|S_0| \leq 2$ and $|S_2| \leq 18$, and consequently, $|S_1| \geq 15$. Again, we attach G_{3p}s until $|S_0| = 0$. This will attach at most 2 G_{3p}s. Then we have $|S_0| = 0$ and $|S_3| \leq 2$, and consequently, $|S_1 \cup S_2| \geq 33$. Thus, at some point of time all the vertices in S have at most 3 consecutive

valences, viz., 1, 2 and 3. At any point of time before that, they may have at most 4 consecutive valences, viz., 0 - 3.

We shall show that at any point of time the vertices in S will have at most 4 consecutive valences, and that at some point of time they will have at most 3 consecutive valences only. For this we use induction to show that if we start with vertices in S in the state of a valence distribution(S_d, S_{d+1}, S_{d+2}), and attach G_{3p}s according to our algorithm, then at some point of time the state of the valences will be $(S_{d+1}, S_{d+2}, S_{d+3})$. We assume that $|S_d \cup S_{d+1}| \leq 18$. Otherwise, we attach G_{3p}s until $|S_d \cup S_{d+1}| \leq 18$.

First, we consider the cases for which $|S_d| \leq 9$. Then $|S_{d+1} \cup S_{d+2}| \geq 26$ with $|S_d \cup S_{d+1} \cup S_{d+2}| = 35$. We attach G_{3p}s until $|S_d| = 0$. For each new G_{3p}, at least 1 vertex of S_d will be moved to S_{d+1}, and at most 2 vertices of S_{d+2} will be moved to S_{d+3}. It is clear that at most 9 G_{3p}s will be attached, and that there will always be at least 19 vertices in $S_d \cup S_{d+1} \cup S_{d+2}$ until there is no vertex in S_d. We have $|S_d| = 0$, $|S_{d+1} \cup S_{d+2}| \geq 17$ and $|S_{d+3}| \leq 18$. Thus, the valences of all the vertices will become $d+1$, $d+2$ and $d+3$.

Now we consider the worst case for which $|S_d| = 18$ and $|S_{d+1}| = 0$. They imply that $|S_{d+2}| = 17$. We attach G_{3p}s until $|S_d| \leq 10$. At most 4 G_{3p}s will be attached. We group all the possible situations into 2 subcases. First, we consider the subcase when 2 vertices are used from S_d for each new G_{3p}. Exactly 4 G_{3p}s will be attached using 8 vertices from S_d. We have $|S_d| = 10$ and $|S_{d+1}| \geq 5$ with $|S_d \cup S_{d+1}| \geq 15$, and $|S_{d+3}| \leq 4$. After attachment of 1 more G_{3p} we have $|S_d| \leq 8$ and $|S_{d+1}| \geq 6$ with $|S_d \cup S_{d+1}| \geq 14$, and $|S_{d+3}| \leq 5$.

Now we attach G_{3p}s until $|S_d| \leq 5$. Clearly, at most 3 G_{3p}s will be attached, and we have $|S_d| \leq 5$ and $|S_{d+1}| \geq 3$ with $|S_d \cup S_{d+1}| \geq 8$ (because at most 6 valence $d+1$ vertices will be raised to valence $d+2$ vertices), and $|S_{d+3}| \leq 8$ (because at most 3 valence $d+2$ vertices will be raised to valence $d+3$ vertices). As long as there are at least 19 vertices in $S_d \cup S_{d+1} \cup S_{d+2}$, all the 3 vertices of a new G_{3p} will be chosen from that union. No vertices will be used from S_{d+3}, and hence no vertex's valence will be raised to $d+4$. We attach G_{3p}s until $|S_d| = 0$. It is evident that at most 5 G_{3p} will be attached, and we have $|S_d| = 0$, $|S_{d+1} \cup S_{d+2}| \geq 17$ and $|S_{d+3}| \leq 18$.

Now we consider the other subcase which consists of the remaining possible situations. For this case, 3 or 4 G_{3p}s will be attached. It can be easily seen that $|S_d| \leq 9$ and $|S_{d+1}| \geq 6$ with $|S_d \cup S_{d+1}| \geq 15$, and $|S_{d+3}| \leq 3$. We attach G_{3p}s until $|S_d| \leq 6$. It can be easily checked that at most 3 G_{3p}s will be attached, and we have $|S_d| \leq 6$ and $|S_{d+1}| \geq 3$ with $|S_d \cup S_{d+1}| \geq 9$, and $|S_{d+3}| \leq 6$. We attach G_{3p}s until $|S_d| = 0$.It is evident that at most 6 G_{3p}s will be attached, and we have $|S_d| = 0$, $|S_{d+1} \cup S_{d+2}| \geq 17$ and $|S_{d+3}| \leq 18$.

It can be easily shown that for all the other combinations of number of vertices in valences d and $d+1$ subject to a maximum of 18, all the vertices will be elevated to at most 3 consecutive valences $d+1$, $d+2$ and $d+3$. The calculations will be similar to the above. □

We make the above set S of 35 vertices rigid in the first round by using jewels of Damaschke [26] as the *ppg*. We create 6 jewels hanging from a common strut that is

incident on 2 vertices of S. This will make 32 vertices rigid. For this we need to query the lengths of 49 edge. We make the remaining 3 vertices rigid by using triangle as the *ppg*. For each of these 3 vertices we query its distance from each of the pair of vertices that are incident on the strut. There will be 6 more queries for edge lengths. Thus, we shall query a total of 55 edges in the first round to make the 35 vertices of S rigid in that round.

The conditions on p_1q_1, p_2q_2 and p_3q_3 in serial numbers respectively 3, 4 and 5 of Lemma 67 will not be satisfied by at most 40, 90 and 122 edges respectively (by Observation 4). In addition to the 122 extra edges needed at each of p_i's to satisfy the conditions on $|p_1q_1|$, $|p_2q_2|$ and $|p_3q_3|$ we need 2 more extra edges incident on each of p_i to accommodate the difference of 2 between the number of basic components that can be attached to the p_i's. Thus, we need a total of 124 extra edges incident on each of the vertices $p_i, i = 1, ..., 35$ of S. We shall attach $3b$, $3b + 1$ or $3b + 2$ (where b is a positive integer) number of G_{3p}s to each vertex in S. This requires us to have $3b + 124$ edges incident on each of p_i's in S. In the worst case there will be at most 18 vertices in S with valence $3b$, no vertices in S with valence $3b + 1$ and the remaining vertices with valence $3b + 2$. Thus, we shall be able to construct a total of at least $35b + 11$ number of G_{3p}s from the edges provided for p_iq_i at all the p_i's in S. Now we describe the algorithm to construct a composite *ppg* made up of G_{3p}s such that all the rigidity conditions listed in Lemma 67 are satisfied for each of them.

Algorithm 4.1. Let the total number of vertices be $n = 245b + 4,419$, where b is a positive integer. We attach at least $3b$ and at most $3b + 2$ numbers of G_{3p}s (Figure 4.1)

to each of 35 rigid vertices in S subject to the condition that the total number of such components being $35b + 11$.

In the first round, we make distance queries represented by the edges of the graph in Figure 4.7. All the vertices p_i $(i = 1, ..., 35)$ in the subgraph enclosed by the rectangle are elements of S and are made rigid in the first round by using the jewel of [26] as the *ppg*. There are 6 jewels attached to a common strut in the subgraph. Residual 3 vertices are made rigid by using triangle as the *ppg*. They are attached to the common strut. There are a total of 55 edges in the subgraph. There are $b + 124$ leaf children rooted at each of the vertices p_i, p_j, or p_k $(i, j, k = 1, ..., 35)$ of S to attach $3b$, $3b + 1$ or $3b + 2$ G_{3p}s (Figure 4.1). Since there will be $35b + 11$ G_{3p}s we make $35b + 11$ groups of 4 vertices $(r_{il}, r_{jl}, r_{kl}, s_l)$, $(l = 1, ..., 35b + 11)$. We query the distances $|r_{il}s_l|$, $|r_{jl}s_l|$ and $|r_{kj}s_l|$, $(l = 1, ..., 35b + 11)$ in the first round. We will make a total of $210b + 4,428$ pairwise distance queries in the first round for the placement of $n = 245b + 4,419$ vertices.

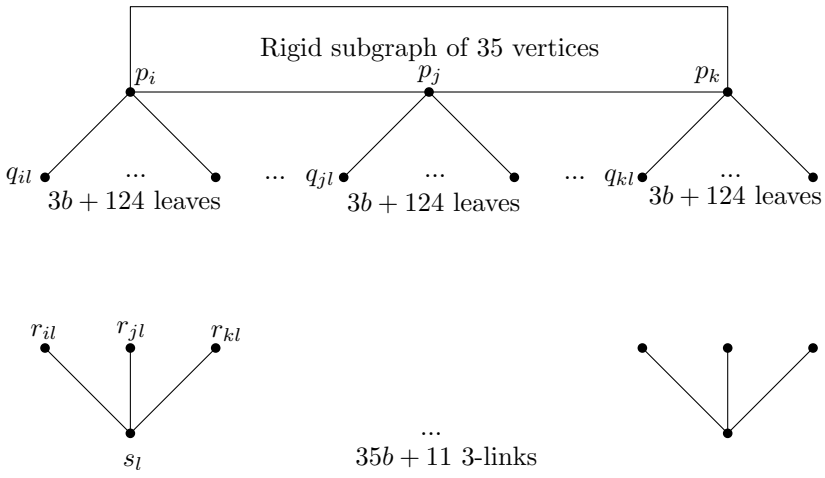


Figure 4.7: Queries in the first round for 2-round algorithm using G_{3p} as the basic component

In the second round, for each 3-link $(r_{il}, r_{jl}, r_{kl}, s_l), l = 1, ..., 35b + 11$, we construct a G_{3p} (Figure 4.1), satisfying all its rigidity conditions as in Lemma 67. For each such 3-link we select a vertex p_i, from the subgraph of 35 vertices of S that has the lowest valence of G_{3p} of Figure 4.1. Since all the 35 vertices $p_i, i = 1, ..., 35$, are rigid in the first round, for any pair of such fixed vertices $(p_i, p_j)(i, j = 1, ...35; i \neq j)$ we can find the distance $|p_i p_j|$. So, for each pair of vertices $(p_i, p_j)(i, j = 1, ..., 35; i \neq j)$, we shall use (p_i, p_j) as an edge in the construction of the G_{3p} of Figure 4.1.

Now from the subgraph of 35 vertices of S we select another vertex $p_j(j \neq i)$ such that the length $|p_i p_j|$ satisfies all the 4 conditions of rigidity on it as stated in serial number 1 of Lemma 67 and that it has the lowest valence of G_{3p} of Figure 4.1 among all such qualifying vertices. We note that we can always find such vertex p_j, because there will be at most 8 edges $(p_i p_j)$ whose lengths do not satisfy the rigidity conditions on it (Lemma 67) whereas we have 34 more vertices for choosing the vertex p_j. Similarly, from the subgraph of 35 vertices of S we select another vertex $p_k(k \neq i, k \neq j)$ such that the length $|p_j p_k|$ satisfies all the 4 conditions of rigidity on it as stated in serial number 2 of Lemma 67 and the length $|p_k p_i|$ satisfies all the 4 conditions of rigidity on it as stated in serial number 3 of Lemma 67, and that it has the lowest valency of G_{3p} of Figure 4.1 among all such qualifying vertices. We note that we can always find such vertex p_k, because there will be at most 16 vertices p_k such that the lengths of the edges $p_j p_k$ and $p_k p_i$ do not satisfy the rigidity conditions on them (Lemma 67) whereas we have 33 more vertices for choosing the vertex p_k.

Then we find an edge p_iq_{il} rooted at p_i satisfying the 20 conditions of rigidity on it as stated in serial no. 4 of Lemma 67, then we find another edge p_jq_{jl} rooted at p_j satisfying the 45 conditions on it as stated in serial no. 5 of Lemma 67 and finally, we find another edge p_kq_{kl} rooted at p_k satisfying the 61 conditions on it as stated in serial no. 6 of Lemma 67.

Then for each $l, (l = 1, ..., 35b + 11)$, we query the distances $|q_{il}r_{il}|$, $|q_{jl}r_{jl}|$ and $|q_{kl}r_{kl}|$ to form a G_{3p} with vertices $p_i, p_j, p_k, q_{il}, q_{jl}, q_{kl}, r_{il}, r_{jl}, r_{kl}$ and s_l. Its edges will satisfy all the rigidity conditions of Lemma 67. Thus, all the $35b + 11$ 3-links will be consumed to construct $35b + 11$ G_{3p}s. For this $105b + 33$ edges will be queried in the second round.

There will be unused leaves $q_{il}/q_{jl}/q_{kl}$ numbering 4,307 in total for the 35 vertices of S. We use a 4-cycle *ppg* [26] to fix 4,306 of them and a triangle *ppg* to fix the rest 1 vertex in the second round. As before, for each pair of vertices $(p_i, p_j)(i, j = 1, ..., 35; i \neq j)$, we shall use (p_i, p_j) as an edge in the construction of the 4-cycle. For each unused vertex q_{il} rooted at p_i we find another vertex q_{jl} rooted at p_j such that $|p_ip_{il}| \neq |p_jp_{jl}|$. Then the 4-cycle $p_iq_{il}q_{jl}p_j$ will be rigid (Observation 5). Then we query the distance $|q_{il}q_{jl}|$ in the second round to complete the 4-cycle. Note that we can always find a vertex like q_{jl}. For, after repeated selection of such matching pairs of edges there may remain at most 2 edges p_iq_{il} rooted at p_i of length equal to that of the same number of edges rooted at p_j (Observation 4). In such a situation we switch the matching to match such edges rooted at p_i with edges other than those same length edge(s) rooted at p_j - this is always possible because there are at most 2 edges rooted at p_j that have the same length (Observation 4). To make the remaining

1 leave vertex rigid we query in the second round its distance from any vertex of S other than its parent vertex.

For 4,307 unused vertices (after the construction of the G_{3p}s) 2,153 4-cycles and 1 triangle will be constructed. 2,153 edges will be queried to complete the 4-cycles and 1 edge will be queried to construct the triangle. The total number of queries in the second round will be $(105b + 33) + 2,153 + 1$, i.e., $105b + 2,187$.

Theorem 69 *The ppg constructed by Algorithm 4.1 is rigid.*

Proof. The proof is similar to that of Theorem 36 for the line rigidity of the *ppg* constructed by Algorithm 3.1. □

The number of queries in the first and second rounds are $210b + 4,428$ and $105b + 2,187$ respectively. Thus, in 2 rounds a total of $315b + 6,615$ pairwise distances are to be queried for the placement of $245b + 4,419$ points. Now, $315b + 6,615 = (315/245) * (245b + 4419) - (9/7) * 4419 + 6615 = 9n/7 + (46305 - 39771)/7 = 9n/7 + 6534/7$. Thus, we have the following theorem:

Theorem 70 $9n/7 + 6534/7$ *queries are sufficient to place n distinct points on a line in two rounds.*

4.3 An Improved Lower Bound for Two Rounds

In this section we improve the lower bound for a 2-round algorithm to $\frac{9}{8}$. Our argument is adversarial as in [27, 20]. Let $\mathcal{A}$ denote any 2-round algorithm. We imagine that an

adversary $\mathcal{B}$ sets edge lengths for *ppg* in each of the 2 rounds with the intention of maximizing its density and returns the distance between any two points queried by $\mathcal{A}$. Let the set of edges queried in the first and second round be E_1 and E_2 respectively. Then $G_1 = (V, E_1)$ is the query graph for the first round, while $G_2 = (V, E_1 \cup E_2)$ is the *ppg* G.

Let $< p_1, p_2, ..., p_k >$ denote a path of distinct degree 2 vertices in G_i. We call it simply as *degree 2 path* in G_i. We note that each vertex of a degree 2 path in G_i is of degree 2 in G_i. Let p_0 and p_{k+1} be the vertices adjacent to p_1 and p_k respectively. If both p_0 and p_{k+1} are not of degree 2, then the path is a maximal path of degree 2 vertices in G_i, and p_0 and p_{k+1} are called start and end vertices respectively of the path. We call a maximal path of degree 2 vertices in G_i simply as *degree 2 maximal path* in G_i. We call a vertex of degree at least 3 in G_i as a *heavy* vertex in G_i. If both the start and end vertices of a degree 2 maximal path in G_1 are heavy in G_1 then we call the maximal path as *class A path*. We define the *length* of a degree 2 path in a graph G_i as the number of vertices in the path in G_i. We note that an edge has a path length of 0. We use $\mathcal{P}_k$ to denote a degree 2 path of length k.

A connected subgraph H of G_i $(i = 1, 2)$ is called a *handle* [27] in G_i if the layout of H is ambiguous in the i-th round, though the layout of all the remaining vertices of G_i are fixed in round i. For example, the subgraphs $(\{p_1\}, \phi)$, $(\{p_1, p_2\}, \{p_1p_2\})$ and $(\{p_1, p_2, p_3\}, \{p_1p_2, p_2p_3\})$ are handles in the the graphs whose layer graphs are shown in

Figures 4.8(a), 4.8(b) and 4.8(c) respectively[1]. In the rest of the discussion of this section, handles play a critical role.

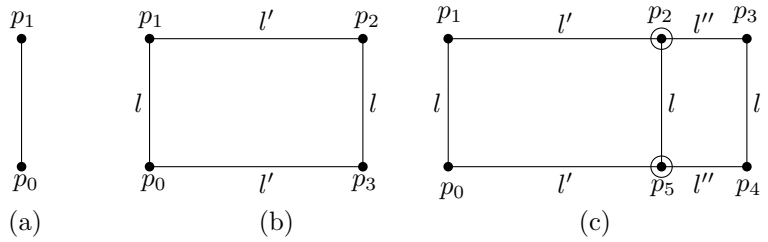


Figure 4.8: Three different handles

Lemma 71 *(a) For each handle H in G_1, the algorithm must insert an edge incident to at least one vertex of H in round 2. (b) For each potential handle H in G_2, the algorithm must insert an edge incident to at least one vertex of H in round 2.*

Proof. For each of the cases (a) and (b), suppose that the algorithm does not insert any edge at some vertices of H in round 2, then the layout of H will remain ambiguous in round 2. This contradicts the fact that G_2 is rigid. □

Let us consider round 1. An algorithm constructs a G_1 and submits it to $\mathcal{B}$. $\mathcal{B}$ assigns lengths to the edges of G_1 by creating handles in G_1 and scopes for potential handles in G_2, with the intention of forcing the algorithm to insert as many edges as possible in round 2; and returns it to the algorithm. $\mathcal{B}$ assigns lengths to the edges according to the following strategies. The algorithm is oblivious of the strategies.

*S*1**.** *The adversary fixes the layout of all heavy vertices except the following 3 types of degree 3 vertices. Let p_0 be a vertex of degree 3. The exceptions are:*

[1]Heavy vertices are circled in the figures of this section

(1) The length of each of the 3 degree 2 maximal paths in G_1 connected to p_0 is at most 1 and the other terminals of the path are not heavy. (Figure 4.9)

(2) The vertex p_0 is connected to exactly one heavy vertex by a degree 2 maximal path of length 1 in G_1 and the length of each of the remaining 2 degree 2 maximal paths in G_1 connected to p_0 is at most 1 and remaining two terminals not heavy. (Figure 4.10)

(3) The vertex p_0 is adjacent either to exactly one heavy vertex in G_1, or to the start or end vertex of a class A path; and the length of each of the remaining 2 degree 2 maximal paths in G_1 connected to p_0 is exactly 1. (Figure 4.11)

We call the vertex p_0 of exception (3) as *specialOne* vertex and its adjacent vertex in G_1 as *specialTwo* vertex if it is heavy in G_1.

The motivation for these exceptions is to provide scope to create handles in round 2 where the vertex set of the handle includes the degree 3 vertex p_0 or some vertices of some degree 2 paths in G_2 attached to p_0. In Figures 4.9, 4.10 and 4.11 the vertex p_0 is an example of exceptions (1), (2) and (3) respectively for a degree 3 vertex whose layout is not fixed by $\mathcal{B}$ in round 1. Exception (1) is used in Figures 4.19 and 4.20 to create the handle $(\{p_1, p_2\}, \{p_1p_2\})$ and $(\{p_2, p_3\}, \{p_2p_3\})$ respectively. Exception (2) is used in Figure 4.18 to create the handles with vertex set $\{p_2'', p_1'', p_0, p_1', p_2'\}$ and edge set $\{p_2''p_1'', p_1''p_0, p_0p_1', p_1'p_2'\}$.

*S*2**.** *For all degree 2 vertices, if one of the incident edges is also incident on a degree 1 vertex, the adversary sets the length of one of the two incident edges to be the same,*

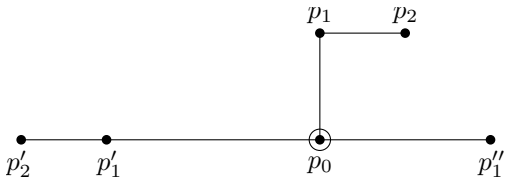


Figure 4.9: The layout of heavy vertex p_0 is not fixed in round 1

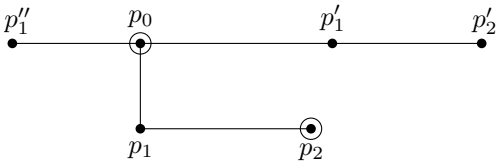


Figure 4.10: The layout of heavy vertex p_0 is not fixed in round 1

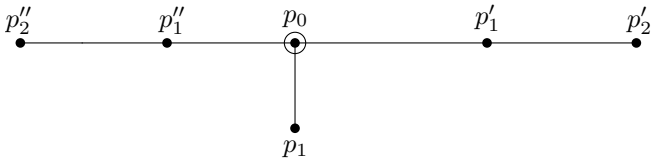


Figure 4.11: The layout of heavy vertex p_0 is not fixed in round 1

say c, over all these degree 2 vertices.

The aim of this strategy is to provide scope to create a handle in each degree 2 path of length exactly 4 in G_2 having an edge in E_2 (see Figure 4.12). Then by Lemma 71, the algorithm must insert an edge at a vertex of the handle. Consequently, the path will be divided into 2 smaller paths of degree 2 vertices.

In Figure 4.12, p_1 and p'_1 are degree 2 vertices in G_1 and their incident edges p_1p_2 and $p'_1p'_2$ are incident to the degree 1 vertices p_2 and p'_2 respectively in G_1. For the degree 2 path (p_1) of length 1, $\mathcal{B}$ sets either $|p_0p_1| = c$ or $|p_1p_2| = c$ in round 1 as per $S2$. In Figure 4.12 $\mathcal{B}$ sets $|p_1p_2| = c$ as per $S2$. Similarly, for the degree 2 path (p'_1) of length 1, $\mathcal{B}$ sets $|p'_1p'_2| = c$ in round 1 as per $S2$. In round 2, if the algorithm inserts an edge $p_2p'_2$, then the adversary sets $|p_2p'_2| = |p_1p'_1|$. This creates a handle $(\{p_2, p'_2\}, \{p_2p'_2\})$ in G_2. By

Lemma 71, the algorithm must insert an additional edge at either p_2 or p'_2. This will split the potential degree 2 path $< p_1, p_2, p'_2, p'_1 >$ of length 4 in G_2 into degree 2 paths, each of length at most 2.

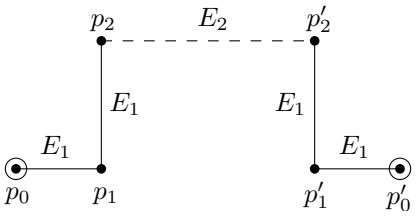


Figure 4.12: The subgraph $(\{p_2, p'_2\}, \{p_2p'_2\})$ is a potential handle in G_2

S3. For each degree 2 maximal path $\mathcal{P}_k =< p_1, p_2, ..., p_k >$, $k \geq 2$, of length at least 2 in G_1, let p_0 and p_{k+1} be non-degree 2 vertices in G_1 adjacent to p_1 and p_k respectively in G_1. (a) The adversary sets $|p_{i-1}p_i| = |p_{i+1}p_{i+2}|$ for $i = 1$ (mod 3). In addition, (b) if $\mathcal{P}_k$ is a class A path then $\mathcal{B}$ fixes the layout of $p_i, i = 0$ (mod 3) and sets $|p_ip_{i+1}| = |p_{i-1}p_{i+2}|$ for $i = 1$ (mod 3), with the exception that if p_0 is a specialOne vertex then $\mathcal{B}$ keeps option for potentially fixing p_0 such that $|p_1p_2| = |p_0p_3|$, and that if $pk + 1$ is a specialOne vertex and $k + 1 = 0$ (mod 3) then $\mathcal{B}$ keeps option for potentially fixing p_{k+1} such that $|p_{k-1}p_k| = |p_{k-2}p_{k+1}|$; finally, (c) if at least one of them, say p_{k+1}, is of degree one the adversary sets the lengths of alternate edges equal.

Strategies $S3(a)$ and $S3(b)$ aim to create handles $(\{p_i, p_{i+1}\}, \{p_ip_{i+1}\})$ for $i = 1$ (mod 3) in G_1 in each class A path (Figure 4.13). Then by Lemma 71, the algorithm must insert an edge at a vertex of each of the handles in round 2. The path will be divided into smaller degree 2 paths of length at most 3. We have the following lemma:

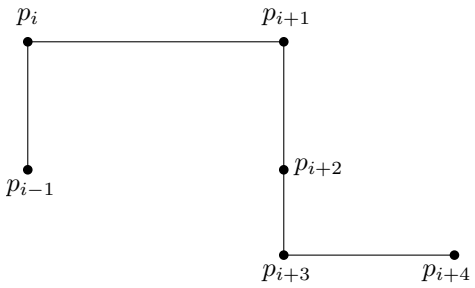


Figure 4.13: The subgraph $(\{p_i, p_{i+1}\}, \{p_i p_{i+1}\})$ is a handle in G_1

Lemma 72 *[20] For each class A path, say $< p_1, p_2, ..., p_k >$, $k \geq 2$, there exists at least one edge in E_2 incident to either p_i or p_{i+1} for $i = 1$ (mod 3) in G_2.*

*S*4**.** *(1) If a vertex, say p_0, of degree 3 has 2 degree 2 maximal paths the other ends of which are not attached to any heavy vertex, and if p_0 is incident on only one degree 2 maximal path of length 1 of which the other end is incident on a heavy vertex, then set the length of one of the edges of this third path as c.*

(2) If 2 specialOne vertices p_0 and p'_0 are adjacent in G_1 then set $|p_0 p'_0| = c$. If a specialTwo vertex p'_0 of degree 3 in G_1 has exactly 2 adjacent vertices of type specialOne then $\mathcal{B}$ sets the length of the edge incident to p'_0 and one of the specialOne vertices adjacent to p'_0 as c. Let p_0 be any specialOne vertex and the 2 degree 2 paths of length 1 attached to it be $< p_0, p_1, p_2 >$ and $< p_0, p''_1, p''_2 >$. Then $\mathcal{B}$ sets $|p_1 p_2| = |p''_1 p''_2| = c$.

Below we show that the application of $S2$ and $S3$ keeps edge lengths consistent:

Lemma 73 *Strategies $S2$ and $S3$ of $\mathcal{B}$ are mutually consistent.*

Proof. Consider a path $\mathcal{P}_k$ of degree 2 vertices in G_1 such that both p_0 and p_{k+1} have degree 1. If $k = 1$, only S2 comes into play and in this case $\mathcal{B}$ sets $|p_1 p_2| = c$. For all $k \geq 4$,

$\mathcal{B}$ sets $|p_1p_2| = c$, $|p_{k-1}p_k| = c$ in accordance with S2 and the lengths of all other edges in accordance with S3. Figures 4.14(c)-(f) serve as examples of this length assignment since for any k, the total number of edges is a multiple of 3 as in Figure 4.14(d), or a multiple of 3 plus 1 as in Figure 4.14(e). or a multiple of 3 plus 2 as in Figure 4.14(f). For $k = 2$ and $k = 3$, $\mathcal{B}$ makes the length assignments as shown in Figures 4.14(a)-(b), which are again consistent with S2 and S3.

If one of p_0 and p_{k+1}, say p_0, is heavy, then $\mathcal{B}$ does not have to set $|p_1p_2|$ to c. On the other hand, if only p_{k+1} is heavy then the length assignment is symmetrically reversed, i.e., starts from p_{k+1}. □

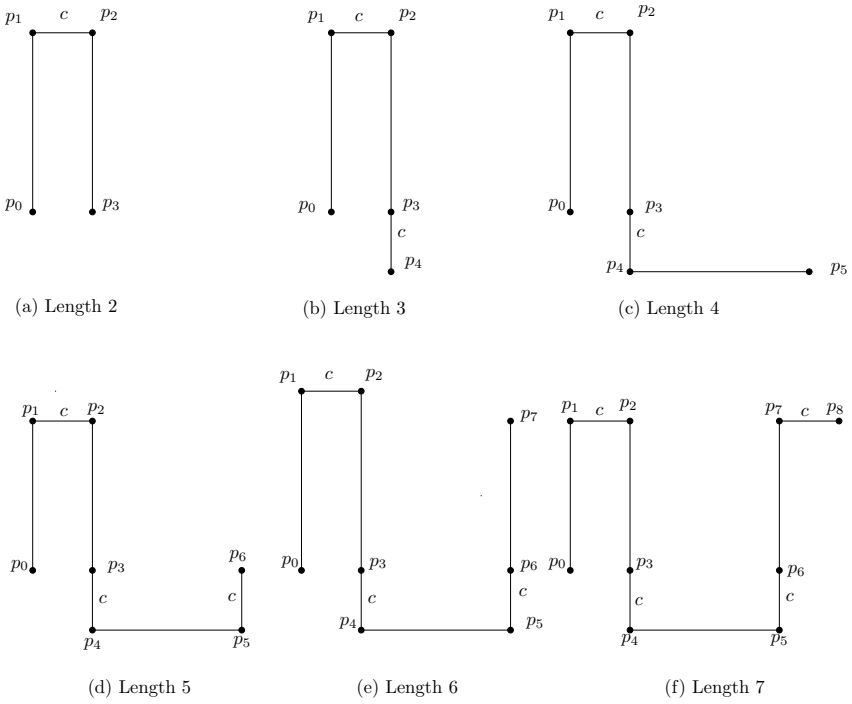


Figure 4.14: The residual parts of maximal paths of degree 2 vertices that will satisfy $S2$

Now we consider round 2. The algorithm completes the construction of G_2 by disam-

biguating existing handles in G_1 and potential handles in G_2 with the insertion of edges into G_1 so that G_2 becomes rigid and submits it to $\mathcal{B}$; $\mathcal{B}$ assigns lengths to all the edges of E_2 with an intention to make G_2 ambiguous and returns it to the algorithm.

For a maximal path of degree 2 vertices in G_2, as a consequence of S_3 there are limits on: (1) the maximum number of edges from E_1 if the path consists of edges from E_1 only (Figure 4.15 shows a degree 2 maximal path $< p_1, p_2, p_3, p_4, p_5, p_6 >$ in G_1 with both the vertices p_0 and p_7 adjacent to start and end vertices p_1 and p_{k+1} respectively being heavy), and (2) the maximum number of consecutive edges from E_1 if it contains at least one edge from E_2 (Figure 4.14 shows some degree 2 maximal paths in G_1 with none of the end vertices being heavy).

If both of p_0 and p_{k+1} are heavy in G_1, then $\mathcal{B}$ sets the above layout in such a way that if, for any i with $i = 1 \pmod 3$ and $i < k$, no edge is attached to either p_i or p_{i+1} in the second round their positions will be ambiguous. Thus, for this case the length of a maximal path of degree 2 vertices in G_2 containing only the edges in E_1 can be at most 3.

If at least one of p_0 and p_{k+1}, say p_{k+1}, is of degree one in G_1, then $\mathcal{B}$ sets the above layout in such a way that if, for any i with $i = 1 \pmod 2$ and $i < k$, no edge is attached to either p_i or p_{i+1} in the second round, they can be made ambiguous by setting $|p_i p_{i+1}| = |p_{i-1} p_{i+2}|$ in the second round. Thus, for this case the length of a maximal path of degree 2 vertices in G_2 containing only the edges in E_1 can be at most 2.

If p_{k+1} is of degree 1 in G_1 and no edge is attached to either p_{k-1} or p_k in the second

round, then the positions of p_{k-1} and p_k can be made ambiguous by setting $|p_{k-1}p_k| = |p_{k-2}p_{k+1}|$ in that round. The algorithm must attach an edge in G_2 to p_{k-1} or p_k. Still then there will be a handle with at most 2 vertices at an end of a path of degree 2 vertices if the end vertex is of degree 1. The algorithm must make them rigid in the second round by attaching an edge in E_2 to at least p_{k+1}. Thus, we have the following lemma:

Lemma 74 *In a degree 2 maximal path in G_2 that contains at least one edge from E_2, there can be at most 2 consecutive edges from E_1.*

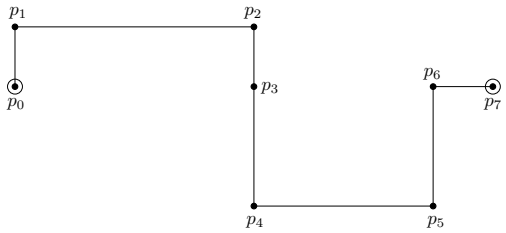


Figure 4.15: $< p_1, p_2, p_3, p_4, p_5, p_6 >$ is a degree 2 maximal path in G_1 with both the end vertices being heavy. In the second round, the algorithm has to introduce edges at p_1 or p_2 to make them unambiguous, and at p_4 or p_5 to make them unambiguous. This will reduce the length of the degree 2 maximal path in G_2.

The above results together with $S2$ and $S3$ imply that Theorem 54 holds for the *ppg* [3]. The following theorem establishes the lower bound on the density of a *ppg* for any 2-round algorithm.

Theorem 75 *Any deterministic 2-round algorithm for solving the 1-dimensional point placement problem requires at least $9n/8$ queries in the worst case.*

Proof. We determine the average density in G_2 for each type of vertices in V. For this, we categorize the vertices in V into two types: A and B as described below. For density

calculation of the vertices in V, each edge in E_1 and E_2 is split into two fractional edges. The two incident vertices of the edge owns the two fractions. Each of the following edges of E_1 is split into $\frac{5}{8}$ and $\frac{3}{8}$ fractional edges:

(1) For a degree 3 specialTwo vertex p_0 in G_1 that is adjacent to 2 specialOne vertices in G_1, one of the incident edges between p_0 and its adjacent specialOne vertices.

(2) For a degree 3 specialTwo vertex p_0 in G_1 that is adjacent to 3 specialOne vertices in G_1, all the incident edges between p_0 and its adjacent specialOne vertices.

(3) For a specialTwo vertex p_0 of degree at least 4 in G_1, each of the incident edges between p_0 and its adjacent specialOne vertices.

For each of the above 3 cases the incident specialTwo vertex p_0 owns $\frac{3}{8}$ and the incident specialOne vertex owns $\frac{5}{8}$. Each of the remaining edges in E_1 and E_2 is divided into 2 equal halves. Each of its 2 incident vertices owns 1/2 of the edge. If a vertex p_0 is divided into 2 equal halves, each half owns one half of the total number of edges owned by p_0.

A. **Vertices in class A paths**

For each class A path we compute the average density of its vertices (see Figure 4.16). The density of a class A path is the ratio of the edges owned by all the vertices in the path and the number of vertices in the path. The minimum of the densities of all the class A paths will be the minimum density of all the vertices of type A.

For a class A path of length k, there must be at least one edge in E_2 incident to

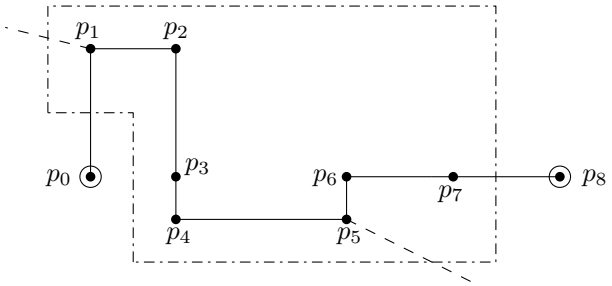


Figure 4.16: Type A vertices are on the path $< p_1, p_2, p_3, p_5, p_6, p_7 >$ of degree 2 vertices in G_1. They are enclosed by a dash dotted polygon.

a vertex in each pair of veritces (p_i, p_{i+1}) $(i = 1 \pmod 3)$, (by Lemma 72), and $\rho = \frac{1}{k}[k+\lfloor\frac{k+1}{3}\rfloor\times\frac{1}{2}] \geq \frac{1}{k}[k+[\frac{k+1}{3}-\frac{2}{3}]\times\frac{1}{2}] \geq \frac{9}{8}$ if $k \geq 4$. For $k = 2$, $\rho = \frac{1}{2}(2+\frac{1}{2}) = \frac{5}{4} > \frac{9}{8}$, for $k = 3$ $\rho = \frac{1}{3}(3+\frac{1}{2}) = \frac{7}{6} > \frac{9}{8}$. We note that no specialOne or specialTwo vertex is of type A, because each of them is a heavy vertex in G_1 but a type A vertex is a degree 2 vertex in G_1. Thus, the minimum average density of type A vertices is $\frac{9}{8}$.

B. **All the remaining vertices**

To compute the minimum density of this type of vertices we group these vertices and their adjacent edges into neighbourhoods of heavy vertices in G_2 of this type and evaluate the densities of these groups. Their minimum will be the minimum density for this type of vertices.

We call each of the following two as a *class B path*: (1) the maximal path of degree 2 vertices in G_2 that is not a part of any class A path, and (2) an edge in G_2 that is incident to at least one heavy vertex of type B. We note that all the vertices in a class

B path are of type B. There are 2 types of groups around the heavy vertices of type B based on whether the heavy vertex is connected to a vertex of type A or a heavy vertex of type B by a class B path.

Accounting for this type of vertices is as follows. (1) If class B path is attached to two heavy vertices v_1 and v_2 of type B then the path (i.e., the vertices and the edges owned by them) is divided equally, and each of the two groups around v_1 and v_2 own one half of the path. (2) Now we consider the case where the two ends of a class B path are attached to two types of vertices, say v_1 of type A and v_2 of type B. Clearly, one half of the edge incident to v_1 is owned by v_1. All the vertices and the remaining edges of the path are owned by the group of vertices around v_2.

We consider the two types of groups of type B vertices separately.

(a) **Group of type B vertices centered at a heavy vertex of type B in G_2 that is connected to heavy vertices of type B in G_2 only, by class B paths**

First, we consider group B(a) vertices that are not centered at specialOne or specialTwo vertices. The average density of a group B(a) vertices decreases as the length of any of the class B paths attached to the heavy vertex of the group increases (Figure 4.17 shows a group B(a) vertices). Consequently, for a group of vertices around a heavy vertex the contribution of average density for the group

from an attached path of degree 2 vertices decreases as the length of the path increases. Thus, the density contribution from a class B path will be minimum if the path length is the maximum. By Lemma 54 the maximum length of a path of degree two vertices is 3. The minimum density contribution from a class B path will be from a path of length 3.

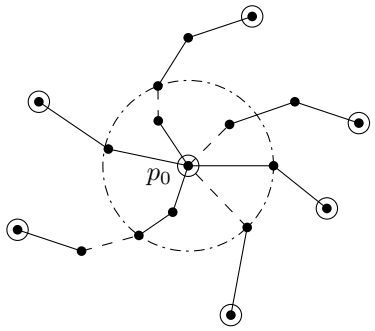


Figure 4.17: A group of vertices of type B(a) around a heavy vertex p_0 in G_2. They are enclosed by a dash dotted circle. If a vertex is on the circle then its one half belongs to this group.

There are 4 edges and 3 vertices in such a path. The density of a half of this path is $\frac{4}{2}/\frac{3}{2} = \frac{4}{3} > \frac{9}{8}$. So, the minimum density of one half of a class B path is greater than $\frac{9}{8}$. We note that one half of a class B path is owned by a B(a) group. Consequently, class B paths will not contribute to reduce the average density of a B(a) group around a heavy vertex of type B to lower than $\frac{9}{8}$. So, we only consider the groups around the heavy vertices of this group each of which has the least number of class B paths attached to the heavy vertex, i.e., which has exactly 3 class B paths attached since degree of a heavy vertex is at least 3. Let the total number of vertices in the 3 class B paths be m. Then average

density for the group around the heavy vertex is $d = \frac{\frac{m+3}{2}}{\frac{1}{2}m+1} = 1 + \frac{1}{m+2} \geq \frac{9}{8}$ for $m \leq 6$. Thus, for the groups with paths having total number of degree 2 vertices at most 6 the minumum average density will be $\frac{9}{8}$. It remains to consider the groups with total number of degree 2 vertices 7, 8 and 9, since there can be at most 3 degree 2 vertices in a path by Lemma 54, and at most 9 degree 2 vertices in the 3 paths.

For the group with 2 class B paths of length 3 and 1 class B path of length 1, placement will not be unique due to S_2 and S_4 (Figure 4.18). For path $< p_0, p_1, p_2 >$ either $|p_0p_1| = c$ or $|p_1p_2| = c$ by S_4. For path $< p_0, p_1', p_2', p_3', p_4' >$, among the edges $p_1'p_2'$ and $p_2'p_3'$ the one in E_1 will have length c by S_2. Similarly, for the path $< p_0, p_1'', p_2'', p_3'', p_4'' >$ either $|p_1''p_2''| = c$ or $|p_2''p_3''| = c$. One more edge must be attached to fix the placements of the points. Then total number of vertices for the paths at the heavy vertex p_0 will be at most 5, which is less than 6. Consequently, the density for the group around p_0 will be at least $\frac{9}{8}$.

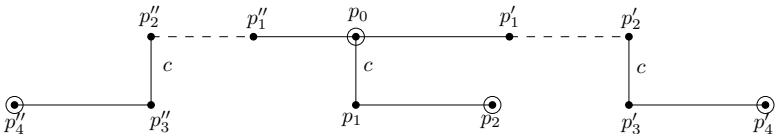
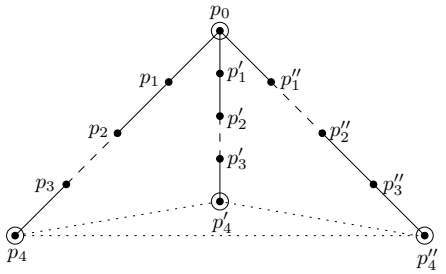


Figure 4.18: Heavy vertex of group B(a) with 2 paths of degree 2 vertices of length 3 and 1 path of degree 2 vertex of length 1

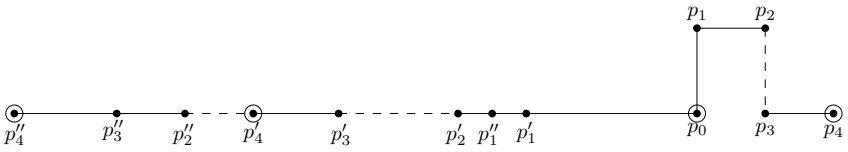
Next we consider the group with all the 3 paths of length 3. For this case the placement will not be unique (Figures 4.19 and 4.20). For Figure 4.19 there must be an edge at p_1 or p_2 of the path $< p_0, p_1, p_2, p_3, p_4 >$ to make p_1 and p_2

unambiguous (by Lemma 71). For Figure 4.20 there must be an edge at p_2 or p_3 to make p_2 and p_3 unambiguous (by Lemma 71). Similarly, there must be an extra edge for each of the other 2 paths $< p_0, p'_1, p'_2, p'_3, p'_4 >$ and $< p_0, p''_1, p''_2, p''_3, p''_4 >$. Thus, the reduced group consists of 3 degree 2 paths each of maximum length 2. There will be at most 6 degree 2 vertices in the degree 2 maximal paths at the heavy vertex p_0 and the average density for the group around p_0 will be at least $\frac{9}{8}$.

Similarly, for the group with 2 paths of length 3 and 1 path of length 2, and the group with 1 path of length 3 and 2 paths of length 2 we can show that there must be edges at the vertices of the paths that will make the total number of vertices in the degree 2 paths around p_0 at most 6. So, the minimum average density for the group around p_0 will be $\frac{9}{8}$.

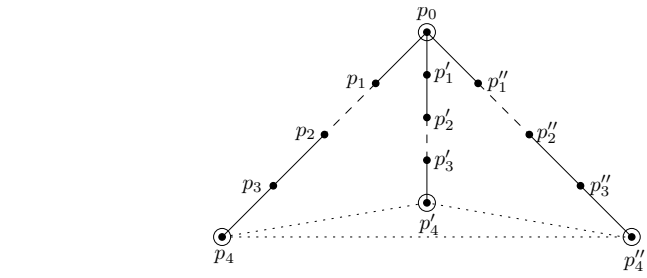


(a) Point placement graph. There is no edge between the pairs of vertices (p_0, p_4), (p_0, p'_4) or (p_0, p''_4); but there may be some edges between the pairs of vertices (p_4, p'_4), (p'_4, p''_4) and (p''_4, p_4). Dotted lines are potential edges.

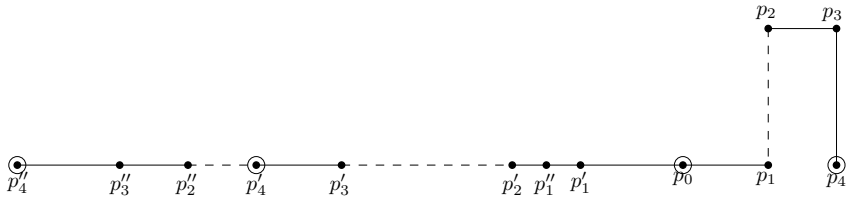


(b) Layer graph.

Figure 4.19: Point placement graph and layer graph of heavy vertex p_0 of group B(a) with 3 degree 2 maximal paths of length 3. For a degree 2 maximal path $< p_1, p_2, p_3 >$ attached to p_0, there are two consecutive edges p_0p_1 and p_1p_2 in E_1 at p_0.



(a) Point placement graph. There is no edge between the pairs of vertices (p_0, p_4), (p_0, p'_4) or (p_0, p''_4); but there may be some edges between the pairs of vertices (p_4, p'_4), (p'_4, p''_4) and (p''_4, p_4). Dotted lines are potential edges.



(b) Layer graph.

Figure 4.20: Point placement graph and layer graph of heavy vertex p_0 of group B(a) with 3 degree 2 maximal paths of length 3. For a degree 2 maximal path $< p_1, p_2, p_3 >$ attached to p_0, there is only one consecutive edge p_0p_1 in E_1 at p_0.

Thus, the minimum of the averages for this type of group is $\frac{9}{8}$.

It is found above that all the type B paths attached to a degree 3 vertex v of type $B(a)$ cannot have the maximum possible length of 3, and that the maximum of the total number of degree 2 vertices in the 3 type B paths attached to v is at most 6. And for a total of at most 6 such vertices the density of the group around v is at most 9/8.

So, we need to check the type $B(a)$ groups around degree 4 vertices in G_2. If all the 4 type B paths attached to a degree 4 vertex v of type $B(a)$ in G_2 are of maximum possible length 3, then the density of the group around v is $\rho = \frac{4\times 2}{4\times 1\frac{1}{2}+1} = \frac{8}{7} > \frac{9}{8}$. So, all the groups around degree 4 vertices of type $B(a)$

must have density greater than $\frac{9}{8}$, since the density of a shorter group B path is shorter than the density of a longer group B path. Again, since a group B path does not help reduce the density of a $B(a)$ group, density of a $B(a)$ group around a $B(a)$ vertex of degree greater than 4 must be greater than $\frac{9}{8}$.

It can be readily checked that the minimum densities of the groups of type B(a) around specialOne or specialTwo vertices is $\frac{9}{8}$. We shall check these results for the more restrictive case when they are of type B(b).

(b) **Group of type B vertices centered at a heavy vertex of type B in G_2 that is connected by at least one class B path to at least one type A heavy vertex**

First, we consider group B(a) vertices that are not centered at specialOne or specialTwo vertices. It is shown above that class B path attached to 2 type B heavy vertices does not contribute to reduce the average density of a group to lower than $\frac{9}{8}$.

Now we consider a class B path between a type B heavy vertex and a type A vertex. A heavy vertex of type B in G_2 can be connected to a vertex of type A by a maximal path of degree 2 vertices in G_2 in two ways based on whether the edge of the path incident on the type A vertex is in E_1 or E_2. If the edge is in E_1 the length of the degree 2 path is 0, because type A vertices are found only in the maximal paths of degree 2 vertices in G_1 where each end of a path is connected

to a heavy vertex of type B in G_1 by an edge from E_1. For this case one half of each end edge is counted towards the density of its adjacent vertex of type B. Clearly, this path will not contribute to reduce the density of the corresponding neighbourhood of type B vertices.

For the second case, i.e., if the edge is in E_2, the maximum length of the maximal path of degree 2 vertices in G_2 is 2 since one end of the maximal path is connected to a heavy vertex in G_2 by an edge from E_2 and since there can be at most 1 edge from E_2 and at most 2 consecutive edges from E_1 in a maximal path of degree 2 vertices in G_2 containing edges from E_1 as well as E_2 (by Lemma 74). The minimum average density of the vertices of this path is $\frac{1}{2}(2+\frac{1}{2}) = \frac{5}{4} > \frac{9}{8}$. Also this path will not contribute to reduce the density of its corresponding neighbourhood of type B vertices to lower than $\frac{9}{8}$.

So, we consider the heavy vertices of this group each of which has exactly 3 paths of degree 2 vertices in G_2. If the group of vertices around a heavy vertex of type B(b) has 2 degree 2 paths of length 3 attached to heavy vertices of type B and 1 path of degree 2 vertices attached to a heavy vertex of type A by an edge from E_2 then each of the 3 paths will have an edge from E_2. In a way similar to the case of group B(a) vertices consisting of 3 degree 2 paths (Figures 4.19 and 4.20) it can be shown that the reduced group will have density of at least $\frac{9}{8}$. For the group with 2 paths of length 3 being attached to heavy vertex of type B

and the third path of length 0 being attached to a vertex of type A by an edge from E_1, then the total number of vertices in all the 3 class B paths around the vertex is 6. It can be shown in a way similar to the discussion of group B(a) that such a group's density is at least $\frac{9}{8}$. It can be easily checked that for all other combinations of 3 maximal paths the minimum average density for the groups of vertices will be at least $\frac{9}{8}$.

Now for the reason similar to group B(a), we consider the group of vertices around a heavy vertex v of type B(b) where v is attached 3 degree 2 paths of length 3 the other ends of which are adjacent to heavy vertices of type B, and 1 path of degree 2 vertices of length 2 the other end of which is incident to a heavy vertex of type A. Density of the group around v is $\rho = \frac{3\times 2+2\frac{1}{2}}{3\times 1\frac{1}{2}+1\times 2+1} > \frac{9}{8}$.

Let us consider groups of vertices of type B(b) around specialOne vertices. We note that for a group of type B(b) around a specialOne vertex p_0, either there is a handle by strategies S1(3) and S4(2) or p_0 owns $\frac{5}{8}$ edge count of the edge incident to p_0 and its adjacent specialTwo vertex. For the former case the group around p_0 becomes smaller and the density of the reduced group is at least $\frac{9}{8}$. For the latter case $\rho = \frac{2\times 2\frac{1}{2}+\frac{5}{8}}{2\times 2+0+1} = \frac{9}{8}$.

Now we consider the specialTwo vertex p_0' of degree 3 in G_1 adjacent to p_0. If p_0 is the only specialOne vertex adjacent to p_0' then either the density of the group around p_0' is at least $\frac{9}{8}$ or p_0' too is a specialOne vertex and a handle must

have been created by $\mathcal{B}$ according to strategies S1(3) and S4(2). The reduced group around p'_0 will have density at least $\frac{9}{8}$. If there are 2 specialOne vertices adjacent to p'_0 then $\rho = \frac{1\times 2\frac{1}{2}+\frac{1}{2}+\frac{3}{8}}{1\times 2+2\times 0+1} = \frac{9}{8}$. For 3 specialOne vertices adjacent to p'_0, $\rho = \frac{3\times\frac{3}{8}}{3\times 0+1} = \frac{9}{8}$. It can be easily checked that if p'_0 is of degree at least 4 then the minimum density of the group around it will be at least $\frac{9}{8}$.

Thus, the minimum average density for all vertices in G_2 will be $\frac{9}{8}$. □

4.4 Summary

We have presented a 2-round algorithm for the point placement problem and improved the upper bound for a 2-round algorithm from $4n/3 + O(\sqrt{n})$ to $9n/7 + O(1)$. Its worst-case lower bound for 2-round algorithm is improved from the existing best $17n/16$ to $9n/8$.

Chapter 5

Detection of Potential Ligand Binding Sites

5.1 Introduction

The biological functions of proteins are the result of their interactions with other molecules such as other proteins, nucleic acids, substrates [1], coenzymes, etc [73]. These interactions generally occur in the concave regions on the surfaces of proteins. The concave regions on the outer and inner surfaces of proteins are called pockets and cavities respectively. It is important to identify the pockets and cavities in proteins. The region of a protein that binds to another molecule is called binding site (Figure 5.1). A ligand is a small molecule that binds with a protein to modulate its function. The binding occurs by intermolecular forces. Ligand binding sites are often found in the largest pockets on protein surfaces [52, 54]. Information about a ligand binding site provides valuable information about protein-ligand docking and the structure of the ligand. This helps design small molecules which can control protein functions [14, 67]. Comparative analysis of ligand binding sites helps to understand

[1] Substrates are the molecules that are bound and acted upon by enzymes.

the protein-ligand binding specificity [15][2].

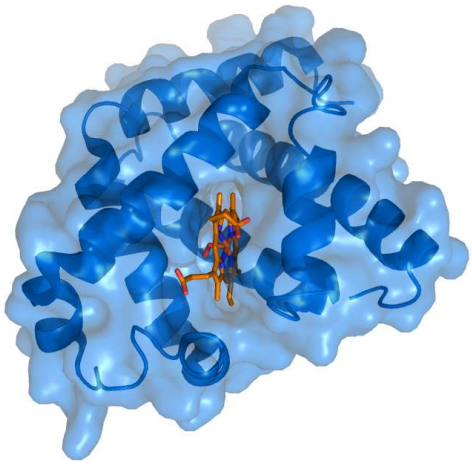


Figure 5.1: Ligand binding site on a protein [figure of Wikipedia: `http://en.wikipedia.org/wiki/Ligand_binding`]

Binding ability depends on the tertiary structure of the protein and the chemical properties of the surrounding amino acids side chains. Tertiary structure of the protein defines the pocket. Ligand binding sites are often located at the largest pockets on the protein surface. In this chapter, we present a modification of a geometric method called MSPocket [75] for finding pockets on protein surface.

5.2 Prior Work

Many computational search methods have been proposed for the identification of ligand binding sites. They can be classified into geometric approach and comprehensive approach based on the type of information used to characterize the pocket [75]. Geometric approaches

[2]Specificity is the ability to identify negative results.

use only geometric properties of proteins. A comprehensive approach, on the other hand, uses other properties of proteins as well, viz., evolutionary information, interaction energy, chemical properties of proteins, etc.

According to Zhu and Pisabarro [75], among the methods in current use, the three best ones are Fpocket [47], VICE [72] and Roll [73]. VICE [72] uses a 3D rectangular grid to represent protein (Figure 5.2). The resolution of the grid is adjustable ($1\mathring{A}$ is appropriate in most cases). Grid points occupied by protein atoms are assigned 0, while the rest are assigned 1. For each of the latter grid points, the algorithm scans in 30 directions represented by vectors of specified length and passing through the grid points. The vectors are grouped into 3 shells. In Figure 5.2 shell 1, 2 and 3 vectors are shown in red, green and blue respectively. For a grid point, if at least half of the scan directions are blocked then it is inside a pocket. Figure 5.3 shows search for 3 grid points numbered 1, 2 and 3 along shell 1 vectors $(1, 0)$, $(1, 1)$, $(0, 1)$, $(-1, 1)$, $(-1, 0)$, $(-1, -1)$, $(0, -1)$ and $(1, -1)$. Black vectors are blocked by protein and are classified as blocked. Green vectors are not blocked and have clear paths to the edges of the grid. They are classified as clear. Pink vectors are not blocked, but they do not reach the grid edge because of finite length. They are classified as stalled. Point 1 has more clear than blocked vectors, and is outside the pocket. Point 2 has more blocked than clear vectors, and is inside the pocket. Point 3 is ambiguous. It needs further examination along shell 2 vectors.

Roll [73] uses the atomic coordinates of protein. A probe of radius $2\mathring{A} - 8\mathring{A}$ ($2\mathring{A}$ is

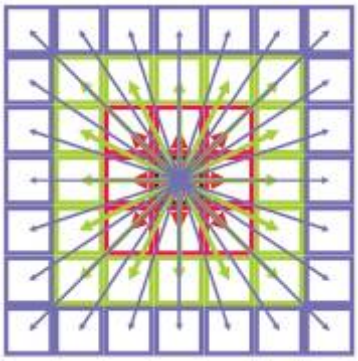


Figure 5.2: Grid points and shell vectors in VICE [Figure 1(a) of [72]]

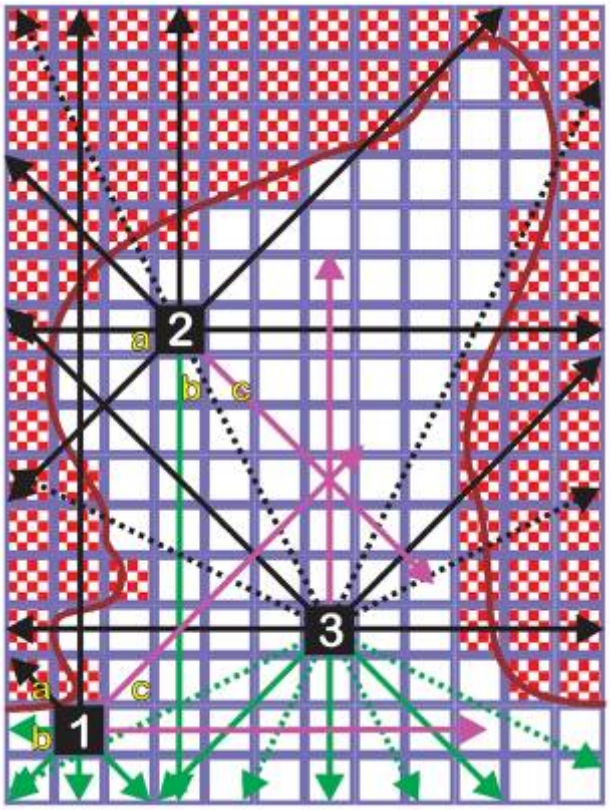


Figure 5.3: Scanning by VICE along shell 1 vectors [Figure 1(c) of [72]]

appropriate in most cases) is rolled on each slice of a 3D grid representation of a protein. The regions between probe and the protein surface are pockets. An implementation of Roll is called POCASA [73]. Fpocket [47] is a comprehensive approach. It uses geometric criteria, and uses electronegativity of protein atoms for pruning. It uses geometric objects

alpha sphere [49] to fill the space in protein. Each alpha sphere is defined by 4 atoms. Pockets correspond to ensembles of alpha spheres of intermediate radii. They must be apolar also.

Zhu and Pisabarro [75] proposed MSPocket. They claim that its performance is at par with the above 3 methods. MSPocket is a purely geometric approach to find ligand binding site on protein solvent excluded surface (SES). It does not use cubic grid representation of protein. So, it is not protein orientation dependent. In Section 5.3 we propose a modification of it. Here we describe MSPocket in detail.

Input to the MSPocket algorithm is the SES of a protein which in turn is generated by MSMS, a widely used tool for computing molecular surfaces [60]. The SES of a molecule consists of zero or more internal components and one external component of SES of a molecule. The components are reported as triangulated meshes. MSPocket processes only the external component of protein SES.

Let $G = (V, E)$ be the graph corresponding to the mesh, where V is the set of vertices in the mesh and E is the set of edges in it. Vertex normals to the mesh are used as features to identify pockets. For each vertex v, MSPocket calculates the average angle of deviation θ of the normal vectors of the adjacent vertices of v with respect to the normal vector at v, and assigns it to v. Then it selects representative vertices v' in ascending order of θ. When a vertex is selected its adjacent vertices are removed from contention. This reduces the number of vertices to about 25%.

Let V' be the set of representative vertices. Then a new graph $G' = (V', E')$ is constructed using V' as its set of vertices. Two vertices are adjacent in G' if they are adjacent in G, or one of them is adjacent to a neighbour of the other in G, or their neighbours are adjacent in G. For each pair of vertices in G', if their mutual distance is within some distance cutoff d_p (the authors of MSPocket used $d_p = 8\mathring{A}$), then they are called close vertex pairs (ClsVP). Each ClsVP is a candidate for pocket vertex pair (PktVP) or protrusion vertex pair (PtrVP). A ClsVP is selected as a PktVP if the distance between them decreases by a value greater than $0.2r$, where r is a user specified distance change ratio parameter (the authors of MSPocket suggested $r = 1.3$), as the pair is moved along their normal by a short distance ($0.2\mathring{A}$). The vertices in a PktVP are potential pocket vertices. A PktVP is considered as a PtrVP if their normals are inclined at an angle larger than some user specified parameter a_p (the authors of MSPocket used $a_p = 200^\circ$).

Finally, pocket outlier vertices are pruned from the pocket and missed vertices at the pocket bottom are included into the pocket using some neighbourhood conditions.

All the parameters are adjustable by the user. This makes it very flexible and can be adjusted according to the demand of the input. The authors claim that its performance is comparable to the existing best performing methods.

5.3 Contribution

In MSPocket each vertex of a ClsVP (A, B) is moved along its respective normal by a small distance $0.2\mathring{A}$. Let the new positions of the pair be A' and B'. The pair is considered as a PktVP if $d_{AB} - d_{A'B'} > 0.2r$, where r is a distance change ratio parameter. The vertices in a PktVP are potential pocket vertices. The larger the value of r, the more the vertex normals are pointing towards each other. We replace this constraint by the angles of normals at A and B with $\overrightarrow{AB}$ and $\overrightarrow{BA}$ respectively. They are more closely related to the angle of inclination of the two planes passing through the vertices and perpendicular to the two vertex normals. Let the angles be θ_1 and θ_2 respectively. We require that $\theta_1 < 70°$, $\theta_2 < 70°$ and $\theta_1 + \theta_2 < \theta$, where θ is a parameter that depends on θ_1 and θ_2. The value of θ is adjustable by users depending on the values of θ_1 and θ_2. In this work, we have set its value as follows: (i) for $\theta_1 < 70°$ and $\theta_2 < 70°$, $\theta_1 + \theta_2 < 85°$; (ii) for $\theta_1 < 65°$ and $\theta_2 < 65°$, $\theta_1 + \theta_2 < 90°$; (iii) for $\theta_1 < 60°$ and $\theta_2 < 60°$, $\theta_1 + \theta_2 < 95°$; and (iv) for $\theta_1 < 55°$ and $\theta_2 < 55°$, $\theta_1 + \theta_2 < 100°$. The remaining parameters of MSPocket are set to the same values that are used in MSPocket. Parameter r is not used in our method. We call this method as Modified MSP.

5.3.1 Experimental Results

We have used the same 48 bound benchmark dataset [43] that is used by MSPocket. Table 5.1 shows the success rates of top 1 and top 3 pockets that are identified by Modified

Table 5.1: Comparison of success rates in detection of ligand binding sites

Method	Top 1 (%)	Top 3 (%)
Modified MSP	79	90
MSPocket	77	90

MSP and MSPocket. For top 1 pocket Modified MSP has a slightly better success rate than that of MSPocket, while for top 3 pockets the success rates are the same.

5.4 Summary

We have modified MSPocket algorithm to make the angle constraint more closely related to the concavity of the ligand binding sites. The algorithm is tested on a small dataset. The results are encouraging. The method can be further extended and/or refined to make it more effective.

Chapter 6

Conclusions

In this dissertation, we have investigated 3 problem areas of computational biology where geometry is involved. First, we have considered the length-constrained sums/densities of DNA sequence. An optimal algorithm has been presented for the length-constrained maximum density segment problem. Experiments show that it runs significantly faster than an existing optimal algorithm [21]. It has been extended to solve the k length-constrained maximum density segments problem in $O(nk)$, $O((n+k)\lg^2(U-L))$ and $O(n(U-L))$ time when $k \in O(\lg^2(U-L))$, $k \in \omega(\lg^2(U-L)) \cap o(n(U-L)/\lg^2(U-L))$ and $k \in \Omega(n(U-L)/lg^2(U-L)) \cap O(n(U-L))$ respectively. Previously there was no non-trivial solution for this problem. The method has been extended to solve also the length-constrained maximum sum segment problem and the k length-constrained maximum sum segments problem in optimal time. We have also presented optimal algorithms to find all the length-constrained segments satisfying a sum or density lower bound. We have indicated the extensions of our algorithms to higher dimensions. Our algorithms facilitate efficient solutions for all these problems in higher dimensions. All the algorithms can

be extended in a straightforward way to solve the problems with non-uniform length.

It would be interesting to study if there is any linear time algorithm for the k length-constrained maximum density segments problem. It can also be investigated to find more efficient algorithms for the problems in higher dimensions. It remains open to improve the trivial lower bounds for these cases.

Second, we have explored the point placement problem on a line. The 3-dimensional version of it has application in the area of molecular conformation. 2-round algorithms based on 5:5 and 6:6 jewels have been presented. We have presented a 2-round algorithm based on 3-path and improved the upper bound for a 2-round algorithm from $4n/3+O(\sqrt{n})$ to $9n/7+O(1)$. Worst-case lower bound for 2-round algorithm for the problem is improved from the existing best $17n/16$ to $9n/8$. It is challenging to decrease the gap between the upper and lower bounds for two rounds even further. One can consider 7:7 and 8:8 jewels as basic component for algorithm. Improving the upper bound of $5n/4$ for three rounds [20] can also be investigated. One can study the relation between this problem and the restriction site mapping problem. Another interesting line of work is to generalize this problem to higher dimensions.

Third, we have studied the problem of detection of ligand binding sites on protein surface. We have proposed a modification of a geometric method called MSPocket [75] for detection of ligand binding sites on protein surface. We have replaced a constraint for the angular inclination of a pair of vertices in a pocket. Experimental comparison of our method

with MSPocket using benchmark dataset of 48 bound protein structures shows encouraging result. It can be investigated to extend/improve this method by incorporating interaction energy, evolutionary information, chemical properties of proteins, etc.

Bibliography

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22:207–216, June 1993.

[2] A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Pub. Co., 1974.

[3] M. S. Alam and A. Mukhopadhyay. A new algorithm and improved lower bound for point placement on a line in two rounds. In *CCCG '10: Proceedings of the 22nd Canadian Conference on Computational Geometry*, pages 229–232, 2010.

[4] M. S. Alam and A. Mukhopadhyay. A new geometric algorithm for the maximum density segment problem. In *Proceedings of the 20th Annual Fall Workshop on Computational Geometry*, Stony Brook University, Stony Brook, NY, USA, Oct. 29-30, 2010.

[5] M. S. Alam, A. Mukhopadhyay, and A. Sarker. Generalized jewels and the point placement problem. In *CCCG '09: Proceedings of the 21st Canadian Conference on*

Computational Geometry, pages 45–48, 2009.

[6] N. N. Alexandrov and V. V. Solovyev. Statistical significance of ungapped alignments. In *Pacific Symposium on Biocomputing (PSB-98)*, pages 463–472. 1998.

[7] L. Allison. Longest biased interval and longest non-negative sum interval. *Bioinformatics*, 19(10):1294–1295, 2003.

[8] S. E. Bae and T. Takaoka. Algorithms for the problem of k maximum sums and a vlsi algorithm for the k maximum subarrays problem. In *7th International Symposium on Parallel Architectures, Algorithms and Networks*, pages 247–253, Los Alamitos, CA, USA, 2004.

[9] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27:865–873, September 1984.

[10] J. Bentley. Programming pearls: perspective on performance. *Commun. ACM*, 27:1087–1092, November 1984.

[11] G. Bernardi. Isochores and the evolutionary genomics of vertebrates. *Gene*, 241(1):3 – 17, 2000.

[12] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[13] G. S. Brodal and A. G. Jorgensen. A linear time algorithm for the k maximal sums problem. In *Proc. 32nd International Symposium on Mathematical Foundations of Computer Science*, pages 442–453. Springer Verlag, Berlin, 2007.

[14] S. J. Campbell, N. D. Gold, R. M. Jackson, and D. R. Westhead. Ligand binding: functional site location, similarity and docking. *Current Opinion in Structural Biology*, 13(3):389 – 395, 2003.

[15] B. Y. Chen and B. Honig. Vasp: A volumetric analysis of surface properties yields insights into protein-ligand binding specificity. *PLoS Comput Biol*, 6(8):e1000881, 08 2010.

[16] K.-Y. Chen and K.-M. Chao. On the range maximum-sum segment query problem. In R. Fleischer and G. Trippen, editors, *Proceedings of the 15th International Symposium on Algorithms and Computation*, pages 294–305. Springer Berlin / Heidelberg, 2005.

[17] K.-Y. Chen and K.-M. Chao. On the range maximum-sum segment query problem. *Discrete Applied Mathematics*, 155(16):2043 – 2052, 2007.

[18] C.-H. Cheng, K.-Y. Chen, W.-C. Tien, and K.-M. Chao. Improved algorithms for the k maximum-sums problems. In X. Deng and D.-Z. Du, editors, *Proceedings of the 16th International Symposium on Algorithms and Computation*, pages 799–808. Springer Berlin / Heidelberg, 2005.

[19] C.-H. Cheng, K.-Y. Chen, W.-C. Tien, and K.-M. Chao. Improved algorithms for the k maximum-sums problems. *Theoretical Computer Science*, 362(1-3):162 – 170, 2006.

[20] F. Y. L. Chin, H. C. M. Leung, W.-K. Sung, and S.-M. Yiu. The point placement problem on a line - improved bounds for pairwise distance queries. In *Proceedings of the Workshop on Algorithms in Bioinformatics*, pages 372–382, 2007.

[21] K.-M. Chung and H.-I. Lu. An optimal algorithm for the maximum-density segment problem. In G. Di Battista and U. Zwick, editors, *Algorithms - ESA 2003*, pages 136–147. Springer Berlin / Heidelberg, 2003.

[22] K.-M. Chung and H.-I. Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34(2):373–387, 2005.

[23] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemeredi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.

[24] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16(5):548–558, 1983.

[25] G. Crippen and T. Havel. *Distance Geometry and Molecular Conformation.* Research Studies Press, Taunton, Somerset, England, 1988.

[26] P. Damaschke. Point placement on the line by distance data. *Discrete Applied Mathematics*, 127(1):53–62, 2003.

[27] P. Damaschke. Randomized vs. deterministic distance query strategies for point location on the line. *Discrete Applied Mathematics*, 154(3):478–484, 2006.

[28] A. Daurat, Y. Gérard, and M. Nivat. The chords' problem. *Theor. Comput. Sci.*, 282(2):319–336, 2002.

[29] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86 – 124, 1989.

[30] L. Duret, D. Mouchiroud, and C. Gautier. Statistical analysis of vertebrate sequences reveals that long genes are scarce in gc-rich isochores. *Journal of Molecular Evolution*, 40:308–317, 1995.

[31] T.-H. Fan, S. Lee, H.-I. Lu, T.-S. Tsou, T.-C. Wang, and A. Yao. An optimal algorithm for maximum-sum segment and its application in bioinformatics. In O. Ibarra and Z. Dang, editors, *Implementation and Application of Automata*, pages 46–66. Springer Berlin / Heidelberg, 2003.

[32] C. Fields and C. Soderlund. gm: a practical tool for automating dna sequence analysis. *Computer Applications in the Biosciences: CABIOS*, 6(3):263–270, 1990.

[33] G. N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation.*, 104(2):197–214, 1993.

[34] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization. *SIGMOD Rec.*, 25:13–23, 1996.

[35] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Trans. Database Syst.*, 26:179–213, 2001.

[36] S. M. Fullerton, A. Bernardo Carvalho, and A. G. Clark. Local rates of recombination are positively correlated with gc content in the human genome. *Molecular Biology and Evolution*, 18(6):1139–1142, 2001.

[37] M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu. Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics. In R. Guig and D. Gusfield, editors, *Algorithms in Bioinformatics*, pages 157–171. 2002.

[38] M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu. Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. *Journal of Computer and System Sciences*, 70(2):128 – 144, 2005.

[39] U. Grenander. *Pattern Analysis*. Springer-Verlag, New York, NY, USA, 1978.

[40] D. Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2(3):207 – 214, 1982.

[41] R. Hardison, D. Krane, D. Vandenbergh, J.-F. Cheng, J. Mansberger, J. Taddie, S. Schwartz, X. Huang, and W. Miller. Sequence and comparative analysis of the rabbit [alpha]-like globin gene cluster reveals a rapid mode of evolution in a g + c-rich region of mammalian genomes. *Journal of Molecular Biology*, 222(2):233 – 249, 1991.

[42] T. Havel, I. Kuntz, and G. Crippen. The theory and practice of distance geometry. *Bulletin of Mathematical Biology*, 45(5):665–720, 1983.

[43] B. Huang and M. Schroeder. Ligsitecsc: predicting ligand binding sites using the connolly surface and degree of conservation. *BMC Structural Biology*, 6(1):19, 2006.

[44] X. Huang. An algorithm for identifying regions of a dna sequence that satisfy a content requirement. *Computer Applications in the Biosciences: CABIOS*, 10(3):219–225, 1994.

[45] M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Inf. Process. Lett.*, 47(3):115–122, 1993.

[46] S. K. Kim. Linear-time algorithm for finding a maximum-density segment of a sequence. *Inf. Process. Lett.*, 86(6):339–342, 2003.

[47] V. Le Guilloux, P. Schmidtke, and P. Tuffery. Fpocket: an open source platform for ligand pocket detection. *BMC Bioinformatics*, 10(1):168, 2009.

[48] D. Lee, T.-C. Lin, and H.-I. Lu. Fast algorithms for the density finding problem. *Algorithmica*, 53:298–313, 2009.

[49] J. Liang, C. Woodward, and H. Edelsbrunner. Anatomy of protein pockets and cavities: measurement of binding site geometry and implications for ligand design. *Protein Science*, 7(9):1884–1897, 2008.

[50] Y.-L. Lin, T. Jiang, and K.-M. Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *Journal of Computer and System Sciences*, 65(3):570 – 586, 2002.

[51] H.-F. Liu and K.-M. Chao. Algorithms for finding the weight-constrained k longest paths in a tree and the length-constrained k maximum-sum segments of a sequence. *Theoretical Computer Science*, 407(1-3):349 – 358, 2008.

[52] N. London, D. Movshovitz-Attias, and O. Schueler-Furman. The structural basis of peptide-protein binding strategies. *Structure*, 18(2):188–199, 2010.

[53] B. Mumey. Probe location in the presence of errors: a problem from DNA mapping. *Discrete Applied Mathematics*, 104(1-3):187–201, 2000.

[54] M. Nayal and B. Honig. On the nature of cavities on protein surfaces: Application to the identification of drug-binding sites. *Proteins: Structure, Function, and Bioinformatics*, 63(4):892–906, 2006.

[55] A. Nekrutenko and W. H. Li. Assessment of compositional heterogeneity within and between eukaryotic genomes. *Genome Res.*, 10(12):1986–1995, 2000.

[56] S. Ohno. Universal rule for coding sequence construction: Ta/cg deficiency-tg/ct excess. *Proceedings of the National Academy of Sciences*, 85(24):9630–9634, 1988.

[57] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166 – 204, 1981.

[58] J. Redstone and W. L. Ruzzo. Algorithms for a simple point placement problem. In *CIAC'00: Proceedings of the 4th Italian Conference on Algorithms and Complexity*, pages 32–43, London, UK, 2000.

[59] W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, pages 234 – 241, 1999.

[60] M. F. Sanner, A. J. Olson, and J.-C. Spehner. Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.

[61] P. M. Sharp, M. Averof, A. T. Lloyd, G. Matassi, and J. F. Peden. DNA sequence evolution: the sounds of silence. *Royal Society of London Philosophical Transactions Series B*, 349:241–247, 1995.

[62] J. C. Shepherd. Method to determine the reading frame of a protein from the purine/pyrimidine genome sequence and its possible evolutionary justification. *Proceedings of the National Academy of Sciences*, 78(3):1596–1600, 1981.

[63] S. S. Skiena, W. D. Smith, and P. Lemke. Reconstructing sets from interpoint distances (extended abstract). In *SCG '90: Proceedings of the 6th Annual Symposium on Computational Geometry*, pages 332–339, New York, NY, USA, 1990.

[64] D. D. Sleator and R. E. Tarjan. Self adjusting heaps. *SIAM J. Comput.*, 15(1):52–69, 1986.

[65] H. O. Smith and K. W. Wilcox. A restriction enzyme from hemophilus influenzae. i. purification and general properties. *Journal of Molecular Biology*, 51:379–391, 1970.

[66] P. Soriano, M. Meunier-Rotival, and G. Bernardi. The distribution of interspersed repeats is nonuniform and conserved in the mouse and human genomes. *Proceedings of the National Academy of Sciences*, 80(7):1816–1820, 1983.

[67] C. Sotriffer and G. Klebe. Identification and mapping of small-molecule binding sites in proteins: computational tools for structure-based drug design. *Il Farmaco*, 57(3):243 – 251, 2002.

[68] N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, and R. Hardison. Comparison of five methods for finding conserved sequences in mul-

tiple alignments of gene regulatory regions. *Nucleic Acids Research*, 27(19):3899–3910, 1999.

[69] T. Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science*, 61:191–200, 2002.

[70] H. Tamaki and T. Tokuyama. Algorithms for the maximum subarray problem based on matrix multiplication. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 446–452, Philadelphia, PA, USA, 1998.

[71] E. N. Trifonov. Translation framing code and frame-monitoring mechanism as suggested by the analysis of mrna and 16 s rrna nucleotide sequences. *Journal of Molecular Biology*, 194(4):643 – 652, 1987.

[72] A. Tripathi and G. Kellogg. A novel and efficient tool for locating and characterizing protein cavities and binding sites. *Proteins: Structure, Function, and Bioinformatics*, 78(4):825–842, 2010.

[73] J. Yu, Y. Zhou, I. Tanaka, and M. Yao. Roll: a new algorithm for the detection of protein pockets and cavities with a rolling probe sphere. *Bioinformatics*, 26(1):46–52, 2010.

[74] Z. Zhang, P. Berman, T. Wiehe, and W. Miller. Post-processing long pairwise alignments. *Bioinformatics*, 15(12):1012–1019, 1999.

[75] H. Zhu and M. T. Pisabarro. Mspocket: an orientation-independent algorithm for the detection of ligand binding pockets. *Bioinformatics*, 27(3):351–358, 2011.

[76] S. Zoubak, O. Clay, and G. Bernardi. The gene distribution of the human genome. *Gene*, 174(1):95 – 102, 1996.

Vita Auctoris

Name: Md. Shafiul Alam

Present Address: 3459 Wells Street
Windsor, Ontario, N9C 1T6
Canada

Permanent Address: Village and Post Office - Bonkola
Upazilla - Sujanagar, District - Pabna
Bangladesh