Modifying a path into the shortest path

Xiaowei Duan

Delft University of Technology

Modifying a path into the shortest path

by

Xiaowei Duan

to obtain the degree of Master of Science
in Electrical Engineering
Track Wireless Communication and Sensing
at the Delft University of Technology,
to be defended publicly on Tuesday December 20, 2022 at 11:00 AM.

Student number:	5337593
Project duration:	February, 2022 – December, 2022
Thesis committee:	Prof.dr.ir. P.F.A. Van Mieghem, TU Delft,chair
	Dr. J.L.A. (Johan) Dubbeldam, TU Delft
	Ir. Rogier Noldus, Ericsson,TU Delft
Supervisors:	Prof.dr.ir. P.F.A. Van Mieghem, TU Delft
	Zhihao Qiu, TU Delft, daily supervisor

To my mom

PREFACE

This thesis, *Modifying a path into the shortest path*, represents the end of my two and a half years of study at the Delft University of Technology. I completed the Master of Science degree in Electrical Engineering with this project at the Network Architectures and Services (NAS) group. Studying at the Delft University of Technology is unforgettable for me.

Firstly, I would like to express my sincere gratitude to my supervisor, Professor Piet Van Mieghem for his support and constant guidance during the entire thesis period. Secondly, I would like to thank my daily supervisor, Ir. Zhihao Qiu, for his weekly brainstorming sessions, guidance on the structure of my thesis and helpful advice about this project. I would like to thank Erik de Vries for his guidance on how to execute simulations and codes using QCE clustering. In addition, I would like to thank Professor Johan Dubbeldam for being on my thesis committee. Notably, I am particularly grateful to Ir. Rogier Noldus for his valuable suggestions for my thesis. Words cannot express how thankful I am. I am fortunate to obtain their help and it must be the most treasured experience I had during my study at the Delft University of Technology.

During my thesis time, a lot of things in my life changed. I have also had moments of disappointment and confusion. I would like to thank my parents for their support and encouragement. I would also like to thank my friend Yuan Hua for her company, and my friend Ying Jin as well as Xuemeng Tian for their support behind the scenes. Finally, I would also like to thank myself. Thanks for my persistence, optimism and efforts.

Xiaowei Duan
Delft, December 2022

Abstract

Modifying a path into the shortest path can be dealt with by the method to process the inverse shortest path problem (ISPP). ISPP is a problem based on graph theory, that is to design link weights in a graph to satisfy that given paths are the shortest between the corresponding node pairs. It can be used in networks of complex systems to solve practical problems such as re-routing in transportation systems and reallocating resources in IP networks. This thesis proposes two new methods based on the simplex algorithm, the split path method (Sp) and the limit constraints method (Lc), to solve the inverse shortest path problem with one predefined path (ISPP-S). The different approaches used to find the constraints in these two methods affect the obtained optimal solution and running time. By analyzing the simulation results (running time, adjustment of link weight as well as the relationship between path length and running time) of these two algorithms in graphs with various structures, we can evaluate the efficiency of Sp and Lc and summarize their applicable networks. After modification, these two methods can be utilized to solve the extended inverse shortest path problem with multiple target paths (ISPP-M) and with target paths in a spanning tree (ISPP-T). In addition, the applicability of these two algorithms is also extended from directed graphs to undirected graphs. The application of Sp and Lc algorithms in a variety of empirical networks is also discussed. Through the analysis of the above problems and the experimental results, we discover that Sp is more suitable to solve ISPP with relatively few constraints (ISPP-S or ISPP in small-sized networks); Lc performs better when solving ISPP with relatively more constraints (ISPP-M or ISPP in large-sized networks).

Key words: The inverse shortest path problem, Network, Graph theory, Mathematical optimization, Simplex algorithm

Contents

1

Introduction

A complex system is a system composed of individual parts or components linked together in some way [1]. A society with billions of people, a transportation system with thousands of roads and a nervous system containing a large number of neurons are all complex systems. Actually, we are surrounded by complex systems. How to describe, understand or even control complex systems and reveal their fundamental laws and principles, network science may provide an answer.

Network science is an emerging discipline that has its origins in graph theory. Many practical problems in complex systems can be transformed into mathematical problems based on graph theory. The inverse shortest path problem (ISPP), the topic of this thesis, is a graph-theory based problem. We can solve ISPP in complex systems to improve or control these systems.

1.1. Motivations

Modifying a path into the shortest path is known as the inverse shortest path problem (ISPP) [2][3][4][5], which can be dealt with by the method to process ISPP. The inverse shortest path problem is motivated by real-world traffic models. When people travel from one place to another, distance is not the only factor that influences their choice of route. Various factors (such as tolls on the route, traffic congestion, the number of service centers, etc.) are concerned by them when selecting routes. Taking all factors into account, people would have an assessment of the specific cost for each route. In their opinion, the cost of the route they choose is the lowest. However, the routes chosen by them tend to be different from what the road network planner had preconceived based on relevant knowledge. So when planners analyze chosen routes, they need to redistribute the priori cost of routes to match the assessments of cost that people have in their minds. At the same time, the change before and after reallocation should not be too great. This is because people's perceived costs are not entirely accurate and only provide bounds on the travel cost. The previous costs set by planners are still informative and cannot be completely ignored.

Not only the traffic in the real road networks, but also the traffic flow in an IP network is a motivating example for studying the inverse shortest path problem. In the IP network, the data package should be allocated the shortest path to be transferred by the operator, which heavily affects the network's performance [6]. When the performance of the network is not ideal, adjusting weights is one method and the administrator can adjust the weight setting to ensure that the data package transmission path is the shortest. But how to configure administrative weights is unclear, and it is hard for the administrator to immediately adjust network parameters to keep the entire network working efficiently. This problem can be solved as an ISPP by mathematical programming in a short time.

These two motivating examples not only indicate reasons for studying the inverse shortest path problem but also point out two important parameters that we should focus on during the study. One is the amount of adjustment, and the other is the processing time. However, previous studies mainly focus on the algorithm to solve the inverse shortest path problem, and research on these parameters is insufficient.

1.2. OBJECTIVES

The objective of the thesis is to explore the inverse shortest path problem (ISPP), including

- Understanding the content of ISPP.
- Proposing new algorithms to solve ISPP.
- Implementing previous algorithms and new algorithms in Matlab language; choosing graphs with different structures as experimental subjects and experimenting with all algorithms; analyzing and comparing their experimental results (running time and weight adjustment).
- Modifying algorithms to apply to ISPP extension problems and applying these algorithms to solve ISPP in empirical networks.

1.3. THESIS OUTLINE

The structure of this thesis is as follows:

- Chapter 2: We mainly introduce ISPP and previous studies about it. This chapter starts with the basics of graph theory. Then, we explain ISPP specifically based on knowledge of graph theory. Finally, we provide a brief overview of relevant works on ISPP and introduce two previously known algorithms, the quadratic programming method (Qp) and the column generation method (Cg), in detail.
- Chapter 3: We propose two new algorithms—the split path method (Sp) and the limited constraints method (Lc), and describe specifics about these algorithms. We also provide their flowcharts, pseudo codes, and complexity analysis.
- Chapter 4: We test four methods (Qp, Cg, Sp, Lc) for solving the ISPP with a single prescribed path (ISPP-S) in four different types of directed graphs. We analyze the efficiency of each algorithm by its running time and the total weight adjustment.

- Chapter 5: We extend the ISPP-S to the ISPP with several target paths (ISPP-M) as well as the ISPP with target paths in a spanning tree (ISPP-T). We also discuss the feasibility of Sp and Lc in undirected graphs. Simulations and experimental results of these extension problems are included.
- Chapter 6: We simulate some empirical networks and use the four algorithms described above to solve specific ISPP in these networks.
- Chapter 7: The numerous results obtained in the former chapters are summarized and some conclusions are drawn from the experiments. We also give some suggestions for further investigation.

2 THE INVERSE SHORTEST PATH PROBLEM

This chapter introduces the inverse shortest path problem (ISPP), its related work and existing algorithms to solve the ISPP. In order to clarify ISPP, we present relevant knowledge of graph theory beforehand because the definition of ISPP involves some of these terms.

2.1. GRAPH THEORY

A diagram composed of a collection of points with lines connecting them can depict the structure of real-world networks [1]. The concept of graphs is derived mathematically from such structures using points, representing "nodes", and lines, representing "links". The feature of each object in networks can be described as properties of graphs and quantitated by graph metrics.

2.1.1. BASIC STRUCTURAL PROPERTIES

The topology of a network can be represented by a graph G. Nodes in the graph can specify objects in the network, and the node set is denoted by $\mathscr{N}$. Links connecting nodes can represent connections between two objects. The link set can be represented as $\mathscr{L}$. An adjacency matrix A, a square $N \times N$ matrix with elements of 0 or 1, can be used to represent the graph [7]. If there is no link between node i and node j, $a_{ij} = 0$; else, $a_{ij} = 1$. In some cases, each link in the graph has an associated value: the link weight w. This type of graph is called a "weighted graph". The link weight is often given a specific physical meaning (such as traffic flow, email latency, construction costs, etc.) to describe the properties of the connection between two objects. A weighted matrix W which has the same dimension as matrix A is defined, and its elements are w. If $a_{ij} = 0$, $w_{ij} = 0$, there is no link from node i to node j.

A graph can be defined as a directed or undirected graph, which depends on whether its links have a fixed direction. In the directed graph, the link from node i to node j (l_{ij})

is different from the link from node j to node i (l_{ji}). So the weight of these two links a_{ij} may not be equal to a_{ji}. But in the undirected graph, $a_{ij} = a_{ji}$. Then we introduce some parts of the graph:

- **Path**: Paths are basic entities in connecting two nodes in a graph G [8], denoted by P. There are no repeated nodes or links in a path and the path length pl is the number of links contained in the path [9].

- **Shortest path tree**: The concept of the shortest path tree is based on the spanning tree. A spanning tree of an undirected graph G is a tree that contains all of the nodes of G. If a spanning tree T is a shortest path tree, the weight of the path from the root node r to every node t in this tree should be the shortest between node pairs (r, t) in graph G.

2.1.2. TOPOLOGICAL METRICS

The primary metrics we used in this thesis are listed below.

- **Degree**: The degree of nodes d determines how strongly a group of nodes are tied. The node degree d indicates the number of a node's neighbors, which is the number of nodes connecting to it. So $d \in [0, N-1]$. This definition is aimed at the undirected graph. For a directed graph, the degree of each node can be divided into 'in-degree' (the number of links terminating at this node) and 'out-degree'(the number of links starting from this node).

  The degree distribution is a description of the degree of all nodes in the graph. For a random graph, the node degree is a variable. When randomly selecting a node, the probability of its degree d equals to k ($k = 1, 2, ...$ for a connected graph) is the degree distribution $\Pr(d = k)$. Mathematically,

$$\Pr(d = k) = \frac{N_k}{N} \tag{2.1}$$

  where N_k is the number of nodes whose degree is equal to k, N is the number of nodes in the whole graph.

- **Betweenness**: Betweenness is a crucial parameter when studying the shortest path problem. The betweeness of a link l is defined as

$$B_l = \frac{\sigma_{ij}(l)}{\sigma_{ij}} \tag{2.2}$$

  where σ_{ij} is the number of shortest paths from node i to j, and $\sigma_{ij}(l)$ is the number of shortest paths between i and j traversing link l [8].

2.1.3. TYPE OF GRAPHS

After explaining the common metrics of graph theory, we introduce four graphs with different structures as test subjects for the subsequent experiments. We mainly state their features as well as how to build them up.

ER GRAPH

Erdős–Rényi (ER) random graph is first proposed by Paul Erdős and Alfréd Rényi [10], which is constructed by taking N nodes and generating a link between each pair of nodes with probability p.

The specific generation process of ER graph is as follows.

- Set the value of connection probability p.
- Select an arbitrary pair of nodes i and j.
- Choose a value uniformly from 0 to 1. If this value is less than or equal to the probability p, a link exists between nodes i and j.
- Repeat the above steps until all node pairs are selected.

This model is defined as $G(N, p)$, which is the related variants of ER random graph model.

Figure 2.1 plots the degree distribution of a ER graph $G(1000, 2ln(N)/N)$ (a 1000-node ER graph with connection possibility $p = 2ln(N)/N$). The degree distribution of the ER graph is Binomial distribution.

$$\Pr(d=k) = \binom{N-1}{k} p^k (1-p)^{N-1-k} \tag{2.3}$$

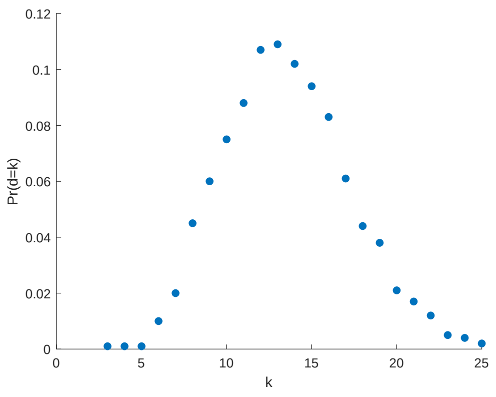


Figure 2.1: The degree distribution of ER graph

BA GRAPH

Barabási–Albert (BA) graph is a scale-free network, in which most nodes are only connected to a few nodes (hubs). The degree of hubs is larger than the degree of other nodes.

The process of constructing BA graph is as follows,

- First, make m_0 nodes in the graph as hubs and connect them to each other to generate a complete graph.

- Then, gradually add one new node at a time. Generate m_1 ($m_1 < m_0$) links between the new node and old nodes.
- The new node prefers to connect nodes with a high degree. The probability of old node i with degree d_i being connected is $\Pr_i = \frac{d_i}{\sum_{j=1}^{n} d_j}$ (n is the number of nodes in current graph).

The constructed BA graph can be denoted by $G(N, m_0, m_1)$.

The degree distribution of BA graph should obey Power-law.

$$\Pr(d = k) \sim k^{-\gamma} \tag{2.4}$$

Where γ is called the power law exponent, whose value is typically in the range $[2,3]$ in the real world.

For a BA graph $G(1000, 5, 1)$, after computing $\Pr(d = k)$ and taking the logarithm of both sides of the formula 2.4, we plot them in Figure 2.2. Figure 2.2 depicts a line with a slope equal to the power law exponent γ, where $\gamma = 2.373$.

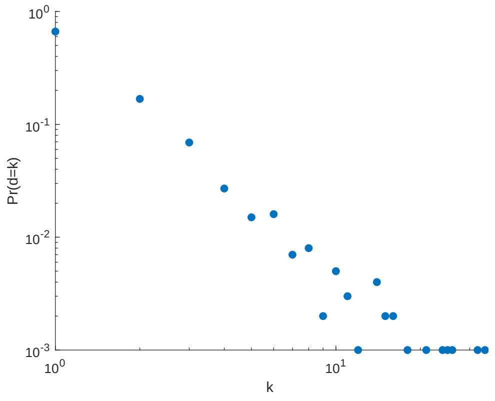


Figure 2.2: The degree distribution of BA graph

2D LATTICE GRAPH

In graph theory, a lattice graph is a graph whose drawing is embedded in Euclidean space $\mathbb{R}^n$, forming a regular tiling. Compared to other types of graphs, the structure of 2D lattice graph is fixed and regular.

Due to the special structure of 2D lattice graph, we can construct it from a geometric perspective. Every node in the graph corresponds to a point in the plane with integer coordinates. The range of x-coordinates is $[1, n_0]$, the range of y-coordinates is $[1, n_1]$, and $n_0 \times n_1 = N$ (N is the number of nodes in graph). If the distance between two nodes is equal to 1, there is a link formed between them in the direction of increasing value of the coordinate axis. 2D lattice graph can be denoted by $G(N, n_0 \times n_1)$.

In Figure 2.3, the 100-node 2D lattice graph can be represented as $G(100, 10 \times 10)$. The degree of four vertices (red points) is 2, the degree of nodes on the sides (yellow points) is equal to 3, and the degree of rest nodes is 4. This law holds for all 2D lattice graphs of any size.

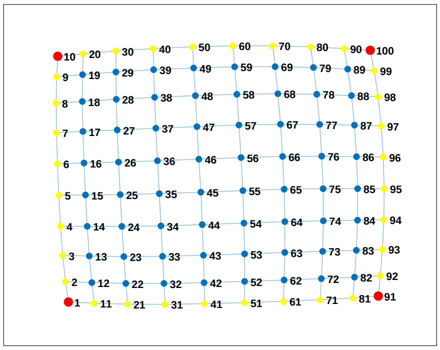


Figure 2.3: The structure of 2D lattice graph

CONFIGURATION GRAPH

The configuration model is a model of a random graph with a specified degree sequence, rather than degree distribution [11].

The steps for generating a configuration graph with N nodes are as follows:

- Consider a degree sequence $\mathscr{K}$, whose elements k_i uniformly distributed in range $[1, K]$ (K and k_i are positive integers, $i \in [1, N]$).
- Set each node has k_i half links. To ensure that the graph can be successfully constructed, the sum of k_i should be even.
- Uniformly select a pair of nodes at random and merge their half links into one. Then from the remaining half link set, continue to choose two half links. Repeat this step until there are no more half links.

The generated configuration graph may have self-loops or multi-links. But with the number of nodes increasing, the number of self-loops and multi-links will decrease.

We build a configuration graph $G(1000, [1, 10])$ with 1000 nodes and a degree range of $[1, 10]$. Figure 2.4 indicates that its degree distribution is uniform.

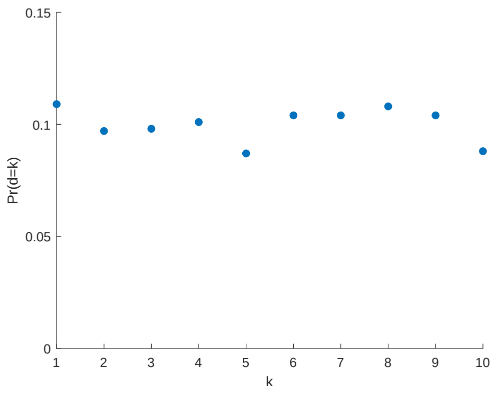


Figure 2.4: The degree distribution of configuration graph

2.2. THE INVERSE SHORTEST PATH PROBLEM

With priori knowledge of graph theory, we now concentrate on the inverse shortest path problem (ISPP).

ISPP is to reassign link weights such that predetermined paths become the shortest between corresponding origins and destinations, and the adjustment of the graph's weight is minimized [5].

Consider a graph $G(\mathscr{N},\mathscr{L},W)$ with N nodes, L links, and link weighted matrix W with elements w_l $(l \in \mathscr{L})$, and a set of n paths P_n are given. The source of the ith path P_i is denoted by s_i and the destination of it is t_i $(i \in [1,n])$.

The goal of the ISPP is to make target paths P_i become the shortest path between node s_i to node t_i with the new link weighted matrix W', w'_l represents its elements. Meanwhile, the sum of link weight changes should be minimal. The weighted adjustment of whole graph can be expressed as $\sum_{l\in\mathscr{L}} |w'_l - w_l|$.

Then the inverse shortest path problem can be formulated as

$$\min \sum_{l\in\mathscr{L}} |w'_l - w_l| \tag{2.5}$$

$$\text{whereby}\begin{cases} P_i \text{ is the shortest path from } s_i \text{ to } t_i \text{ with } W', i = 1,2,\dots,n \\ w_l, w'_l \geq 0, l \in \mathscr{L} \end{cases}$$

When $n = 1$, we defined it as the ISPP with a **single target path** (ISPP-S); when $n > 1$, it can be considered the ISPP with **multiple target paths** (ISPP-M).

Our study starts from the ISPP-S and then extends to the ISPP-M as well as its special case, making a spanning tree become to the **shortest path tree** (ISPP-T).

2.3. RELATED WORK

After introducing the definition of ISPP, now we review some related work that has been carried out.

In the past few decades, there have been some developments in the study of the inverse shortest path problem. Burton and Toint [12] initially used the quadratic programming under the l_2 norm to solve the inverse shortest path problem in 1992. Since then, more and more scientists have started to study ISPP. In 1995, Xu [4] found that the feasible weight vectors of ISPP can form the extreme directions of the polyhedral cone and explored the relationship between ISPP and the minimum cut problem. In the same year, Zhang et al. [3] applied the column generation algorithm to solve the ISPP formulated as the linear programming problem under l_1 norm. In 2003, András Faragó [13] focused on the application of ISPP for route finding in high-speed telecommunication networks. Mobarakeh Karimi [14] studied ISPP with multiple objectives under the bottle type weighted hamming distance in 2018.

This section mainly discusses two typical algorithms to solve ISPP. One is proposed by Burton and Toint[12] and based on a quadratic programming model, which we call the quadratic programming method (Qp). The other uses linear programming, designed by Zhang [3], and is called the column generation method (Cg).

2.3.1. THE QUADRATIC PROGRAMMING METHOD

Burton and Toint[12] suggest this method based on the Goldfarb-Idnani (GI) method for convex quadratic programming. Goldfarb and Idnani [15] proposed it as a variant of the dual quadratic programming method. The GI method works by computing a series of optimal solutions to a dual quadratic programming problem using an active set of constraints from the original problem. Before we introduce Qp, we'll go over some of the details of the GI algorithm.

PRIOR KNOWLEDGE

- **Dual quadratic programming problem**

  The form of a quadratic programming problem can be represented as follows:

$$min \quad L(x) = \tfrac{1}{2}x^{T}Qx + \lambda^{T}(Ax - b) + c^{T}x \tag{2.6}$$

  The form of its dual problem is like:

$$max \quad g(\lambda) = -\tfrac{1}{2}\lambda^{T}AQ^{-1}A^{T}\lambda - (c^{T}Q^{-1}A^{T} + b^{T})\lambda \tag{2.7}$$

  According to the weak duality of the dual QP problem, the objective function value of the primal problem is always no less than that of its dual problem; that is to say, the dual optimality means the primal feasibility (the optimal solution of the dual problem is the feasible solution of the primal problem).

- **The active set method**

  The active set algorithm is applied in convex optimization to find the active constraints that function in the calculation. This method reduces the number of constraints during each iteration, which simplifies the computation. The value of the Lagrange multiplier corresponding to the constraint is used to determine whether this constraint is still binding. Constraints with negative Lagrange multipliers should be dropped from the active set. In this algorithm, inequality constraints should be converted into equality constraints [2], which is easier to calculate.

  The process of the active set algorithm is depicted in Figure 2.5.

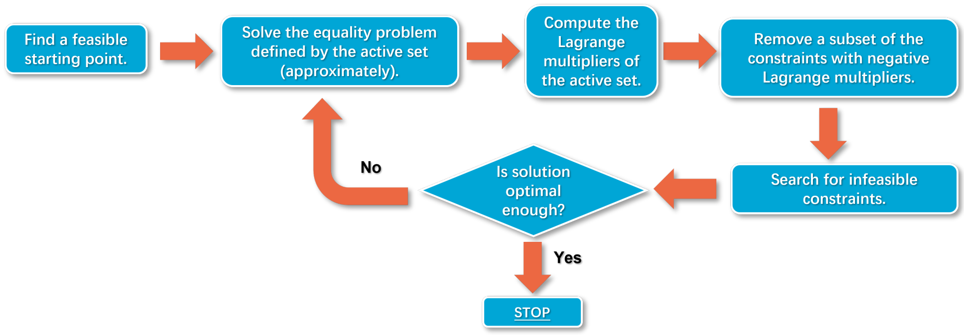

Find a feasible starting point.
Solve the equality problem defined by the active set (approximately).
Compute the Lagrange multipliers of the active set.
Remove a subset of the constraints with negative Lagrange multipliers.
Search for infeasible constraints.
Is solution optimal enough?
No
Yes
STOP

Figure 2.5: The procedure of the active set method [16]

ALGORITHM DESCRIPTION

Define a directed graph $G(\mathcal{N}, \mathcal{L}, W)$ with N nodes, L links, and link weighted matrix W. A predefined path P_{ij} is given, and node i, j are the source and destination of P_{ij} respectively. Path P^*_{ij} is defined as the shortest path between node i and node j.

Qp is based on the GI algorithm which is a dual quadratic programming method, so it is necessary to keep the dual feasibility when searching for the optimal solution. The steps of Qp are as follows:

- Step 0: Initialization. Transform the primal problem to the dual problem. Empty the active set and let Lagrange multipliers u be equal to 0.
- Step 1: Find constraints and check whether path P_{ij} is the shortest path between node i to node j; if yes, the algorithm stops; if not, select one violated constraint q.
- Step 2: Compute the primal and dual step directions d and r, which determine the direction of approximation to the optimal solution. When $d = 1$, its corresponding link weight increases; when $d = -1$, its corresponding link weight decreases; and when $d = 0$, the link weight is maintained.
- Step 3: Determine the maximum steplength t_f by the dual direction r. The actual step length should not exceed this value to maintain dual feasibility. When t_f is infinite, the algorithm stops and there is no solution to the original problem.
- Step 4: Calculate the steplength t_c which meets qth constraints. The actual steplength t is the smaller of t_f and t_c ($t = min[t_f, t_c]$). Take the step and obtain new link weights,

  $$w'_l = w_l + td, l \in \mathcal{L} \tag{2.8}$$

  Then update Lagrange multipliers u.
- Step 5: If $t \neq t_f$, add this constraint q to the active set. Update associated parameters, return to step 1, and check whether all constraints have been satisfied.
- Step 6: If the step length t is equal to the maximum step length t_f, the constraint corresponding to the Lagrange multiplier $u < 0$ no longer has a binding effect and should be removed from the active set. Then go to step 2.

The complete flowchart of Qp can be seen in Appendix B.1.

EVALUATION

Qp can solve ISPP relatively effectively. But in some cases, Qp cannot find the optimal solution, which is called the infeasible solution case. It is related to the steplength and step direction chosen for each iteration.

There are two main reasons for the infeasible solution case:

- In the first iteration, the steplength t is calculated without considering the restriction that it cannot exceed the link weights. Actually, the value of each link weight changed is equal to the steplength t. If $t > w_l$ and $d_l = -1$, combining with equation 2.8, w_l may be negative (the weight of link l is negative). Therefore, it is possible to form a ring with negative weight, making it hard to find a solution. When link weights become larger, this situation may improve.

- There are two ways to make the target path become the shortest path: decreasing the weights of links on P_{ij} and increasing the weights of links on other paths between node i and node j. Qp uses both of them, and the increasing or decreasing values are all equal to the steplength t. Additionally, the total weight of the target path P_{ij} and that of the shortest path P^*_{ij} are forced to be balanced, which is more challenging to obtain the optimal solutions.

2.3.2. THE COLUMN GENERATION METHOD

Different from Burton's algorithm based on the convex quadratic programming, Zhang [3] formulates ISPP as a special linear programming (LP) problem. As the size of the graph grows, the number of constraints also increases, which makes the basic LP model take more time to solve many inequalities. Zhang uses column generation algorithm to solve the dual problem of LP. Column generation algorithm makes the time to solve the LP problem significantly reduced. So Cg is efficient in solving ISPP with a large number of constraints. To make it easier to understand Cg, we explain the knowledge in advance.

PRIOR KNOWLEDGE

- **Column generation algorithm**

  Column generation algorithm is the core of the column generation method (Cg), it refers to the LP algorithm designed to solve problems in which there are a huge number of variables compared to the number of constraints [17]. It limits the original problem (master problem) to a restricted master problem. After obtaining the optimal solutions to the restricted master problem, check whether all constraints are satisfied. If yes, the algorithm stops, and the original optimal solution is found. If not, a variable that may improve the objective function is merged into the LP model. The correlative coefficient column of this variable is also added to the coefficient matrix of LP model. This process is repeated until the optimal solution is obtained.

- **Dual problem of linear programming**

  Cg also utilizes the dual problem of LP. If the original problem is too difficult to solve, we can simplify the calculation by solving its corresponding dual problem. The original problem and its corresponding dual problem are as follows:

  The original problem:

$$\max \sum_{j=1}^{n} c_j x_j \tag{2.9}$$

$$\text{whereby}\begin{cases} \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i & (i = 1, \cdots, m) \\ x_j \geqslant 0 & (j = 1, \cdots, n) \end{cases}$$

The dual problem:

$$\min \sum_{i=1}^{m} b_i y_i \tag{2.10}$$

$$\text{whereby} \begin{cases} \sum_{i=1}^{m} a_{ij} y_i \geqslant c_j & (j = 1, \cdots, n) \\ y_i \geqslant 0 & (i = 1, \cdots, m) \end{cases}$$

The dual problem of LP has some properties that are used in Cg:

- Property 1: The number of constraints in the original problem is the same as the variables' number in the dual problem.
  When the number of the primal problem's constraints is too large, the dual problem can induce some algorithmic simplifications.
- Property 2: The dual of the dual is the primal itself. We explain it with an example. The dual problem of formula 2.9 is formula 2.10. Then we find the dual problem of formula 2.10,

$$\max \sum_{j=1}^{n} c_j z_j \tag{2.11}$$

$$\text{whereby} \begin{cases} \sum_{j=1}^{n} a_{ij} z_j \leqslant b_i & (i = 1, \cdots, m) \\ z_j \geqslant 0 & (j = 1, \cdots, n) \end{cases}$$

  The solution z of formula 2.11 is the same as the solution x of formula 2.9 (primal problem). As a result, we can acquire the primal optimal solution.

ALGORITHM DESCRIPTION

Consider an undirected graph $G(\mathcal{N}, \mathcal{L}, W)$ with N nodes, L links and a link weight matrix W. A restricted path P_{ij} linking node i and node j is given.

The process of the whole algorithm is as follows:

- Initially process the original problem and transform it into the standard form of its dual problem. A set of slack variables is introduced to make all constraints become equality constraints, so that the simplex method can be used.

$$\max \quad \tilde{c}^T v \tag{2.12}$$

$$\text{whereby} \begin{cases} \tilde{A} v = 1_{2L} \\ v \geqslant 0 \end{cases} \tag{2.13}$$

  where v is the variable vector containing the dual variables and the slack variables. $\tilde{c}$ is the coefficient matrix consisting of the link weight, and $\tilde{A}$ is the coefficient matrix obtained by constraints.

- Using the simplex algorithm finds the optimal basis B as well as its coefficient c_B. The dual solution π of equation 2.12 can be calculated by B and c_B.

$$\pi = [\pi^{1^T}, \pi^{2^T}] = c_B^T B^{-1} \tag{2.14}$$

According to Property 2 of the dual problem, π is the primal optimal solution of the restricted master problem. π^1 is the link weight increase vector, π^2 is the vector of decrease. And the new link weights can be expressed as

$$w'_l = w_l + \pi^1_l - \pi^2_l, l \in \mathscr{L} \tag{2.15}$$

- If any constraints are violated, choose one randomly and find its coefficient column associated with the variables (a_{k_0,j_0} and c_{k_0,j_0}). Merge them with the corresponding coefficient matrix, $\tilde{A} = [\tilde{A}, a_{k_0,j_0}]$, $\tilde{c} = [\tilde{c}, c_{k_0,j_0}]$.

- Repeat the above steps until all constraints are satisfied.

The flowchart for Cg can be found in Appendix B.2.

EVALUATION

- Due to using the dual method, the number of variables in the original problem equals the number of constraints in its dual problem. Solving ISPP with a few constraints in graphs of considerable size tends to take a long time. Actually, the number of links (variables) in real-world networks tends to be much larger than the number of constraints, which may lead to the inefficiency of Cg in practical applications. More details are provided in Chapter 6.

- Cg needs to introduce slack variables, and the number of introduced slack variables is relevant to the number of links. When the graph contains a large number of links, Cg requires more memory to store variables and run.

2.3.3. ALGORITHMIC FRAMEWORK FOR SOLVING ISPP

From above analysis of two algorithms, we can summarize the framework of algorithms for solving ISPP. The input is a graph, and the output is a graph with the same structure but different link weights.

- First find the constraints.

- Then model ISPP as an optimization problem. Express the constraints mathematically and find the optimal solution.

- Finally, check whether all constraints are satisfied. If yes, stop; otherwise, repeat the above steps.

The flowchart of this framework is shown in Appendix B.3.

2.3.4. OBSERVATION

By analyzing Qp and Cg, we know that there are two ways to make the target path P_{ij} the shortest, decreasing weight of one or more links on path P_{ij} and increasing weight of links on the initial shortest path between node i and node j. Compared to the former, the latter cannot guarantee P_{ij} to be the shortest.

For example, in the graph of G given in Figure 2.6, we assign weights to each link. Path $P_{ij} = [i, s, j]$ is the identified path represented by solid black lines. The shortest path between node i and node j is solid lines in blue $(i-k-j)$. The constraint can be expressed as an inequality:

$$w_{is} + w_{sj} \leq w_{ik} + w_{kj} \tag{2.16}$$

As Figure 2.6 shows, in the left graph, the weight of path $i-s-j$: $|P_{ij}| = w_{is} + w_{sj} = 4$, whereas the shortest path from i to j is path $i-k-j$: $|P'_{ij}| = w_{ik} + w_{kj} = 2$. After optimization, the weight of link l_{ik} increased to 3 and the inequality 2.16 is satisfied. However, in the right graph, the shortest path is still $i-j$ rather than $i-s-j$.

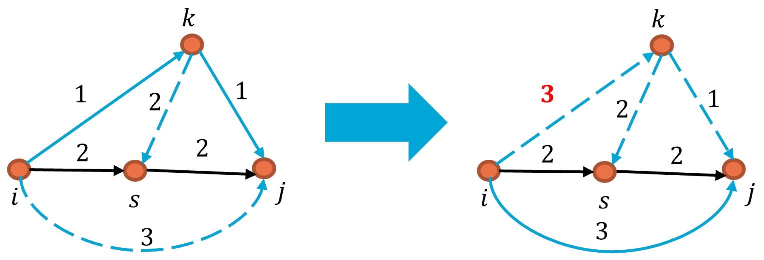


Figure 2.6: An example to explain observation

As a result, reducing the weight of one or more links on the prescribed path P_{ij} is likely a better approach because this method makes the path P_{ij} be the shortest path between nodes i and node j in most cases. The **betweenness** for some or more links on this identified path increases (the number of shortest paths in graph G that traverse one or more links on this identified path has increased).

3
ALGORITHM DESIGN

After analyzing and evaluating two existing algorithms, the quadratic programming method (Qp) and the column generation method (Cg), we try to propose new algorithms to solve the inverse shortest path problem (ISPP) more efficiently. The efficiency of an algorithm to solve ISPP can be quantified as two parameters: the running time t_r and the total weight adjustment Δ. A shorter running time means a feasible solution can be obtained faster in practical applications and we aim to design a new algorithm to solve ISPP rapidly.

Based on the algorithmic framework for solving ISPP in section 2.3.3, the running time is determined by the time spent on constraint discovery, solving the optimization problem, and checking whether all constraints are satisfied. We concentrate on reducing the number of active constraints (constraints used in optimization) to shorten the time of optimization. However, simply minimizing the number of constraints used may take more time because the obtained solution in each iteration may not be the optimal solution, which results in more iterations. In order to reduce the total running time, a trade-off should be made between the number of active constraints and the number of iterations.

In order to simplify solving the optimal solution, two existing methods (Qp and Cg) select one constraint in each iteration and combine it with the old constraints. In addition, they apply the dual problem. Our methods, the split path method (Sp) and the limited constraints method (Lc), differ in that they select constraints in distinct ways, replace the old constraints with new ones, and directly solve the original problem using the simplex algorithm.

3.1. THE SIMPLEX METHOD

Our algorithms are both based on the simplex method, which is widely used to solve linear programming problems. Dantzig [18] proposed it based on the idea of step-by-step descent along the edges of the convex polyhedron from one vertex to an adjacent one.

The complete steps of the simplex method are:

- Express the problem to standard mathematical form of the simplex method
- Introduce slack variables, transform inequality constraints into equality constraints.
- Pivot operation, change the basis.
- Check for optimality and identify optimal values

3.2. SPLIT PATH METHOD

The basic technique of this method is to find the constraints by splitting the path, so we call it the split path method (Sp).

3.2.1. ALGORITHM DESCRIPTION

We first assume a directed graph $G(\mathcal{N}, \mathcal{L}, W)$, together with a path P_{ij} between the origin i and the destination j. The links on path P_{ij} are stored in the set $\mathcal{P}_0$, which are represented by solid black lines in Figure 3.1.

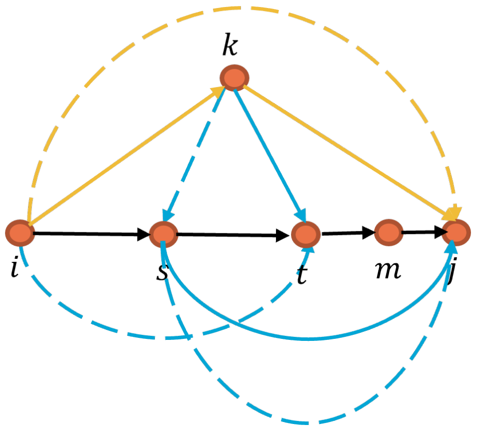


Figure 3.1: An example of graph

SELECT ACTIVE CONSTRAINTS

We select specific constraints by splitting path P_{ij} , which is the core of the split path method. The selected constraints are defined as active constraints and are used to calculate the optimal solution.

First we classify other paths (except P_{ij}) between node i and node j into to two groups. Group 1 contains orange paths which do not have overlapping links with path P_{ij}, which is the path $i-k-j$ and $i-j$ in the graph G. While the rest of the paths from node i to node j belong to Group 2.

Then we restrict link l_{st} and divide the predetermined path P_{ij} into two parts P^1_{ij} and P^2_{ij}. In Figure 3.1, P^1_{ij} is the subpath $i-s-t-m$, and P^2_{ij} is $s-t-m-j$. We should ensure that the weight of every subpath in P_{ij}, either from the start link l_{is} to the restrict link l_{st}

or from the restrict link l_{st} to the last link l_{mj} of P_{ij}, is less than the weight of all paths with the same source and destination as each subpath. As a result, these constraints can produce a set of inequalities.

Group 1 contains these inequalities,

$$\begin{cases} w_{is} + w_{st} + w_{tm} + w_{mj} \le w_{ik} + w_{kj} \\ w_{is} + w_{st} + w_{tm} + w_{mj} \le w_{ij} \end{cases} \tag{3.1}$$

And Group 2 consists of these inequalities,

$$\begin{cases} w_{is} + w_{st} \le w_{it} \\ w_{is} + w_{st} \le w_{ik} + w_{kt} \\ w_{is} + w_{st} \le w_{ik} + w_{ks} + w_{st} \\ w_{st} + w_{tm} + w_{mj} \le w_{sj1} \\ w_{st} + w_{tm} + w_{mj} \le w_{sj2} \end{cases} \tag{3.2}$$

With the longer prescribed path and more complicated graph structure, these inequalities are not enough. So we need to traverse all links (except the start link l_{is} and end link l_{mj}) on the target path and restrict them one by one.

As the size of the graph continues to grow, the number of inequalities generated by constraints also increases. To reduce the running time and simplify calculations, we can ignore some constraints and only maintain the requirement that these (sub)paths should be shorter than their corresponding shortest paths (solid lines in Figure 3.1) instead of comparing all paths with the same source and destination (both solid and dotted lines in Figure 3.1). The shortest path in Group 1 is stored in the set $\mathscr{Q}$, and in Group 2 is stored in sets $\mathscr{P}_1$ and $\mathscr{P}_2$, respectively. These three sets can be combined into the set Ω. So after some inequalities dropped, the merged inequality set is as follows,

$$\begin{cases} w_{is} + w_{st} \le w_{ik} + w_{kt} \\ w_{st} + w_{tm} + w_{mj} \le w_{sj1} \\ w_{is} + w_{st} + w_{tm} + w_{mj} \le w_{ik} + w_{kj} \end{cases} \tag{3.3}$$

Then we restrict other links, such as l_{tm}, and treat them the same way. The number of constraints is related to the path length (pl). When desired path P_{ij} contains one link or two links ($pl_{ij} = 1$ or 2), there is one constraint: the weight of P_{ij} should be less than that of the shortest path from node i to node j. As a result, the number of constraints n_c in each iteration is as follows:

$$n_c = \begin{cases} 1 + 2 \times (pl_{ij} - 2), & \text{if } pl_{ij} > 1 \\ 1, & \text{if } pl_{ij} = 1 \end{cases} \tag{3.4}$$

SOLVE THE LINEAR PROGRAMMING PROBLEM

With n_c inequalities and the objective function $\min \sum_{l \in \mathscr{L}} |w'_l - w_l|$, an LP model can be formed. We apply the simplex algorithm to solve it.

Due to the simplex algorithm requests the standard form of the objective function, we introduce two intermediate variables, x_l^+ and x_l^-, $l \in \mathscr{L}$.

$$x_l^+ = \begin{cases} w_l' - w_l & \text{if } w_l' > w_l \\ 0 & \text{if } w_l' < w_l \end{cases} \tag{3.5}$$

$$x_l^- = \begin{cases} w_l - w_l' & \text{if } w_l > w_l' \\ 0 & \text{if } w_l < w_l' \end{cases} \tag{3.6}$$

So, the objective function can be transformed as :

$$|w_l' - w_l| = x_l^+ + x_l^-, \qquad l \in \mathscr{L} \tag{3.7}$$

and

$$\min \sum_{l\in\mathscr{L}} |w_l' - w_l| = \min \sum_{l\in\mathscr{L}} x_l^+ + \sum_{l\in\mathscr{L}} x_l^- \tag{3.8}$$

Then we can consider x_l^+ as the element of the increasing column vector X^+ and x_l^- as the element of the decreasing column vector X^-. the coefficient matrix c is all-one matrix with $2L$ elements (L is the number of links in graph).

Constraints can be represented as

$$\sum_{l\in\mathscr{P}_0} (x_l^+ - x_l^- + w_l) \leqslant \sum_{l\in\Omega} (x_l^+ - x_l^- + w_l) \tag{3.9}$$

$$x_l^+ \geqslant 0, \quad l \in \mathscr{L}$$

$$0 \leqslant x_l^- \leqslant w_l, \quad l \in \mathscr{L}$$

The complete matrix form of this linear programming problem is,

$$\min c^T [X^+; X^-] \tag{3.10}$$

$$\text{whereby} \begin{cases} [A, -A][X^+; X^-] \leqslant b \\ X^+ \geqslant 0, 0 \leqslant X^- \leqslant W \end{cases}$$

where c is the $2L \times 1$ all-ones matrix. Matrices A and $-A$ are the coefficient matrices of variables X^+ and X^-, respectively.

$$A_l = \begin{cases} 1, & \text{if } l \in \mathscr{P}_0 \text{ and } l \notin \Omega \\ -1, & \text{if } l \in \Omega \text{ and } l \notin \mathscr{P}_0 \\ 0, & \text{if } l \in \mathscr{P}_0 \cap \Omega \end{cases} \tag{3.11}$$

The size of matrix A is $n_c \times L$. The elements of column matrix b can be obtained by $\sum_{l\in\Omega} w_l - \sum_{l\in\mathscr{P}_l} w_l$, and the order of b is $n_c \times 1$.

To solve this linear programming problem, we need to add the slack variables z to generate equation constraints and use the simplex algorithm (the number of variables z equals the number of constraints n_c).

$$\min c'^T x \tag{3.12}$$

$$\text{whereby} \begin{cases} A'x = b' \\ x \geqslant 0 \end{cases}$$

where $x = [X^+; X^-; z]$; $A' = [A, -A, I]$, I is the identity matrix; $b' = [b; 0]$, 0 is the all-zero column vector.

Then, the column of matrix A' should do a linear transformation to identify its redundant parts. Assuming an orthogonal matrix O,

$$A'O^T = [B \quad N] \tag{3.13}$$

and $B \in \mathbb{R}^{m\times m}$ (B is a full rank basis), $N \in \mathbb{R}^{m\times(n-m)}$ ($m = n_c, n = 3L$).

According to the properties of the orthogonal matrix, $A'O^T Ox = b'$, so we set $Ox = \hat{x}$,

$$[B \quad N]\hat{x} = b' \tag{3.14}$$

And we let $\hat{x} = [\hat{x}_B^T \quad \hat{x}_N^T]^T$, so

$$B\hat{x}_B + N\hat{x}_N = b' \tag{3.15}$$

Here, we set $\hat{x}_N = 0$, because it can be considered as the slack variables. The basic vector $\hat{x}_B$ can be calculated, and we can get an extreme point in this situation.

Then, the basis B needs to be changed by pivot operation. In general, the pivot operation changes the positions of one non-basic vector and one basic vector. The non-basic vector is the input vector, and the basic vector is the output vector. Therefore, different extreme points can be obtained. The optimal solution can be found until the objective value cannot be lowered. We can define a parameter r_c to determine if the objective value is at its minimum.

$${r_c}^T = {c'_N}^T - {c'_B}^T B^{-1} N \tag{3.16}$$

and c'_N, c'_B are the coefficient matrices corresponding to N and B, respectively.

If $r_c^T \geqslant 0$ is always valid no matter how to change the basis, it means that the minimum is found, and the simplex algorithm should be finished.

ACCURACY CHECK

Solutions obtained under these constraints are approximations and may not satisfy all constraints in the original problem. Because ensuring the weight of the predefined (sub)path is smaller than the respective shortest path can be obtained by increasing this shortest path's weight. In this case, one of the other paths may become the shortest. The specific analysis has been mentioned in section 2.3.4.

To further explore the effectiveness of the algorithm with only one iteration, we choose the BA graph $G(N, 5, 1)$ (with 5 hubs and each time generating one link) to test and statistic their accuracy through the number of total iteration $iter$. If $iter = 1$, the algorithm can calculate the optimal solution in one go; if $iter > 1$, the solution after iteration is approximated.

As seen from Table 3.1, the accuracy of the split path method is very close to 1 (100%) in BA graph. As the number of nodes in the BA graph increases and the number of hubs maintains, the accuracy increases. So, when the BA graph size is large and the number of hubs is far more less than the the number of total nodes, we can sacrifice a little accuracy for a shorter running time. In other types of graphs, the accuracy decreases as the graph structure becomes more complex (a large number of nodes and links). But when the size of graphs is small ($N < 10$), the accuracy remains high. Related data is shown in Appendix C.

Table 3.1: The split path method's accuracy after one iteration

Nodes / Graph	10	20	50	100	200	500	1000
BA graph	0.9750	0.9806	0.9878	0.9851	0.9921	0.9924	0.9942

We aim to obtain the optimal solution such that all constraints are satisfied, there are two ways to achieve this goal.

- Use a loop and repeat the entire algorithm until all constraints are satisfied. Correspondingly, the running time may increase.
- Set the upbound of the increasing vector X^+ to 0, which is not allowed to increase link weights. This approach makes it more likely that the optimal solution will be obtained after only one iteration. But it still needs loops in some cases, and the adjustment of link weight may increase.

Sp chooses the first approach. Repeat the whole algorithm until all constraints of original problem are satisfied.

3.2.2. FLOW CHART

Figure 3.2 is the flow chart of Sp, which briefly shows the entire algorithm.

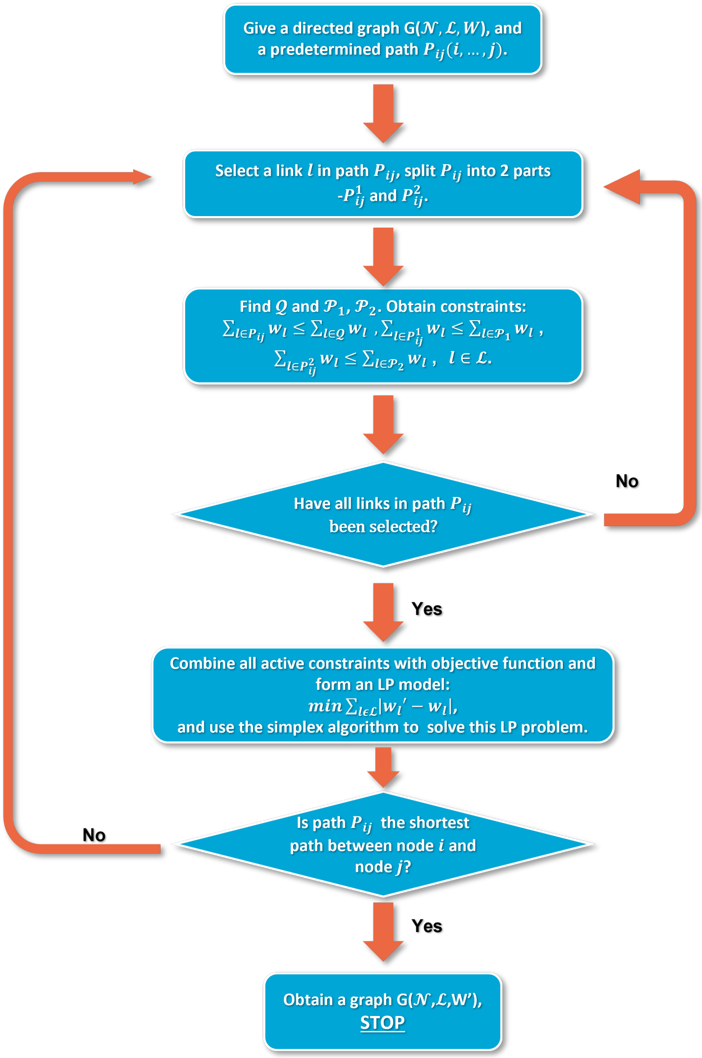

Give a directed graph G($\mathcal{N}, \mathcal{L}, W$), and a predetermined path $P_{ij}(i, \dots, j)$.
Select a link l in path P_{ij}, split P_{ij} into 2 parts -P_{ij}^1 and P_{ij}^2.
Find Q and $\mathcal{P}_1, \mathcal{P}_2$. Obtain constraints: $\sum_{l\in P_{ij}} w_l \le \sum_{l\in Q} w_l$, $\sum_{l\in P_{ij}^1} w_l \le \sum_{l\in \mathcal{P}_1} w_l$, $\sum_{l\in P_{ij}^2} w_l \le \sum_{l\in \mathcal{P}_2} w_l$, $l \in \mathcal{L}$.
Have all links in path P_{ij} been selected?
No
Yes
Combine all active constraints with objective function and form an LP model: $min \sum_{l\epsilon\mathcal{L}} |w_l' - w_l|$, and use the simplex algorithm to solve this LP problem.
Is path P_{ij} the shortest path between node i and node j?
No
Yes
Obtain a graph G($\mathcal{N}$,$\mathcal{L}$,W'), STOP

Figure 3.2: The step of the split path method

3.2.3. PSEUDOCODE

Algorithm 1: The split path method based on the simplex method

```
Input  : $G(\mathcal{N},\mathcal{L},W)$, $W$ with elements $w_l$, a prescribed path $P_{ij}$
Output: $G(\mathcal{N},\mathcal{L},W')$, $W'$ with elements $w'_l$

1  find constraints, target (sub)paths $\in \mathscr{P}_0$, corresponding shortest (sub)paths $\in \Omega$
2  for $l \in 1:L$ do
3      $b_l \leftarrow \sum_{l\in\Omega} w_l - \sum_{l\in\mathscr{P}_0} w_l$
4  get $b$
5  for $l \in 1:L$ do
6      if $l \in \mathscr{P}_0 \setminus \Omega$ then $A_l \leftarrow 1$;
7      else if $l \in \Omega \setminus \mathscr{P}_0$ then
8          $A_l \leftarrow -1$
9          else
10             $A_l \leftarrow 0$
11 get $A$
12 $x \leftarrow [X^+;X^-;z]$   // add slack variables
13 $A' \leftarrow [A;-A;I]$, $b' \leftarrow b$, $c' \leftarrow c$
14 $A'x = b'$   // express constraints as equations
15 $x \leftarrow \mathrm{SIMPLEX}(A',b',c')$   // the simplex method
16 return $x = (x_1, x_2, ..., x_n)$
17 if All constraints are satisfied then $x$ is the optimal solution, stop, and
     $w'_l = w_l + X^+_l - X^-_l, l \in \mathscr{L}$;
18 else
19     Repeat above steps.
```

3.2.4. COMPLEXITY

The complexity of Sp determined by the complexity of constraints discovery, the complexity of the simplex method and the complexity of checking step. It also depends on the number of iterations $iter$, but $iter$ is uncertain. Here we ignore $iter$ and only discuss the complexity of one iteration.

For the constraints discovery, its complexity is related to the length of the target path pl and the shortest path algorithm (Dijkstra's shortest path algorithm [19]). The former complexity is $O(1+2*(pl-2))$, when $pl > 1$ or $O(pl)$, when $pl = 1$. Combine them and simplify, its complexity is $O(pl)$. The latter complexity is $O(L + Nlog_2(N))$ [20] (L is the number of links, N is the number of nodes). So the complexity of constraints discovery is $O(pl*(L+Nlog_2(N)))$.

For the simplex method, its complexity is exponential because the worst case requires traversing all 2^n vertices on the boundary to find the optimal solution. The traversal process depends heavily on the particular rules used for the pivot operation [21]. It is still an unsolved problem so far. In the worst case, the complexity of this method is $O(2^n)$ for one iteration.

For the checking part, its complexity is also depend on the number of original constraints. For a single path ISPP, there is one constraint in primal problem. So e only need to compare the weight of this identified path with that of the path obtained by Dijkstra's algorithm. Thus, the complexity is $O(L + Nlog_2(N))$.

3.3. LIMIT CONSTRAINTS METHOD

The limit constraints method (Lc) uses a different way to split the predetermined path to reduce the number of constraints and thus simplify the process of solving a linear programming problem.

3.3.1. ALGORITHM DESCRIPTION

Consider a directed graph $G(\mathcal{N}, \mathcal{L}, W)$, a target path P_{ij} from node i to node j is given.

SELECT ACTIVE CONSTRAINTS

To demonstrate how the limit constraints method chooses constraints, we use a small-sized graph as an example.

On the example graph given in Figure 3.4, suppose that, the solid black line is defined as the prescribed path P_{ij}, $P_{ij} = [i, s, t, j]$. The blue lines are the shortest path between different pairs of nodes.

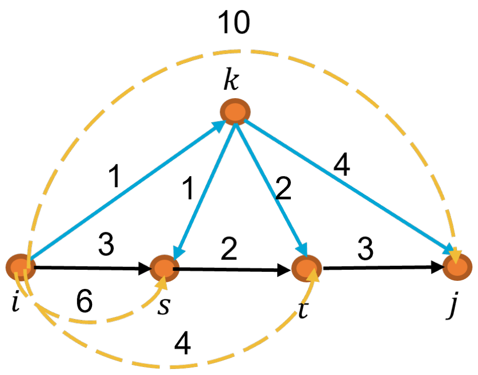


Figure 3.3: An example of graph

Lc compares paths from the source i to every node (s, t, j) in prescribed path P_{ij} with their corresponding shortest paths. The subpaths from node i to each node of P_{ij} are stored in the set $\mathcal{P}_0$ and their corresponding shortest paths are stored in the set $\mathcal{Q}$.

According to this method, constraints can be expressed as the following inequalities:

$$\begin{cases} w_{is} \le w_{ik} + w_{ks} \\ w_{is} + w_{st} \le w_{ik} + w_{kt} \\ w_{is} + w_{st} + w_{tj} \le w_{ik} + w_{kj} \end{cases} \tag{3.17}$$

Compared with Sp, Lc has fewer constraint conditions when the target path is long. The number of constraints is $n_c = pl$, and pl is the path length of the target path.

SOLVE THE LINEAR PROGRAMMING

Combine above inequalities with the objective function after rewritten as the standard form of the simplex method. We can obtain:

$$min \quad {c'}^T v \tag{3.18}$$

$$\text{whereby} \begin{cases} A' v = b' \\ v \geqslant 0 \end{cases}$$

where $v = [X^+; X^-; z]$; $A' = [A, -A, I]$, I is the identity matrix; $b' = [b; 0]$, 0 is the all-zero column vector. The specific solution procedure is the same as that for Sp.

ACCURACY CHECK

The set of solutions v is also approximated, just like the solution of Sp after one iteration. We also choose the BA graph $G(N, 5, 1)$ (with 5 hubs and each time generating one link) to statistic accuracy after one iteration. Table 3.2 illuminates the accuracy in BA graph.

Table 3.2: Accuracy after one iteration in BA graph

Nodes / Graph	10	20	50	100	200	500	1000
BA graph	0.9617	0.9694	0.97348	0.9823	0.9861	0.9848	0.9921

Compared with Sp, the accuracy of Lc after one iteration is lower. It is more obvious in other types of graphs displayed in Appendix C. It still needs a checking step to ensure accuracy. Therefore, we modify this algorithm to further reduce the number of constraints, shorten the running time and ignore the accuracy after one iteration.

MODIFICATION

We select only one violated constraint to approach the optimal solution instead of using all three constraints in equation 3.17. Although more iterations are required, the time for each iteration under one constraint may be shorter than the iterations needed to compute the optimal solution under many constraints.

How to choose a violated constraint is important for the efficiency of the algorithm. We need to find a more binding condition in order to approach the optimal solution faster. We define a parameter E as the total weight difference between the target (sub)paths in set $\mathscr{P}_0$ and its associated shortest (sub)paths in set $\mathscr{Q}$.

$$E = \sum_{l \in \mathscr{Q}} w_l - \sum_{l \in \mathscr{P}_0} w_l \tag{3.19}$$

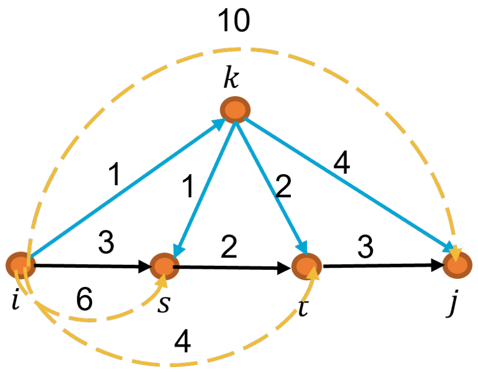


Figure 3.4: An example of graph

According to Figure 3.4, we can obtain

$$E_1 = (w_{ik} + w_{ks}) - w_{is}, E_2 = (w_{ik} + w_{kt}) - (w_{is} + w_{it}), E_3 = (w_{ik} + w_{kj}) - (w_{is} + w_{st} + w_{tj})$$

$$E_1 = (1+1) - 3 = -1, E_2 = (1+2) - (3+2) = -2, E_3 = (1+4) - (3+2+3) = -3$$

We select the constraint whose associated E is the most negative. As a result, the third constraint is picked and $b = E_3$. The number of constraints in each iteration n_c is equal to 1. Then use the simplex method to generate a set of link weights.

Finally, check the solution for optimality. If all original constraints are met, stop. If not, replace the previous constraint with a new constraint whose E is minimal, and repeat algorithm.

3.3.2. FLOW CHART

Figure 3.5 is the flow chart of Lc, which briefly illustrates the entire algorithm.

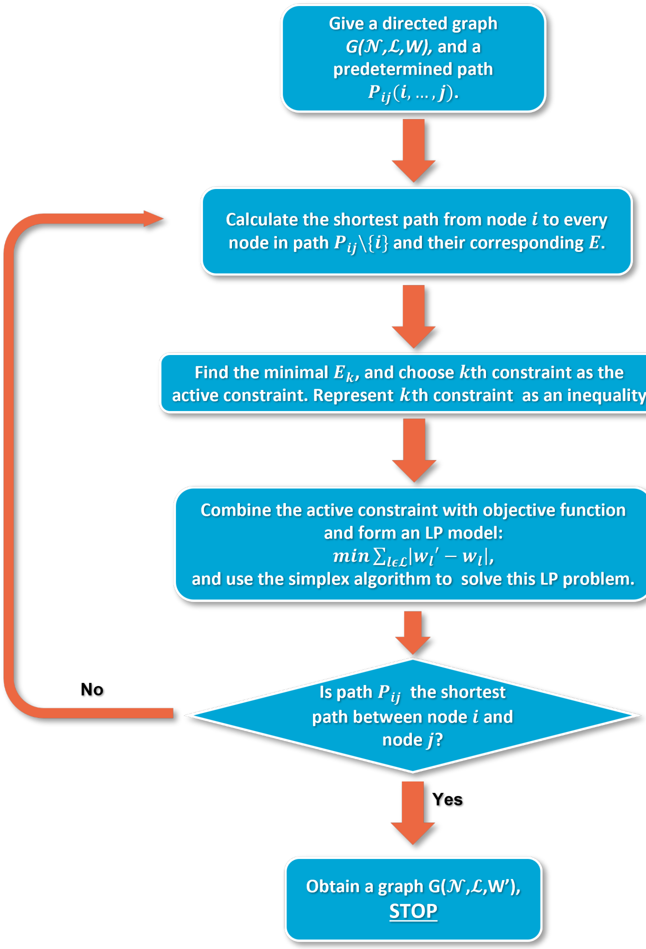

Give a directed graph $G(\mathcal{N},\mathcal{L},W)$, and a predetermined path $P_{ij}(i,\dots,j)$.
Calculate the shortest path from node i to every node in path $P_{ij}\backslash\{i\}$ and their corresponding E.
Find the minimal E_k, and choose kth constraint as the active constraint. Represent kth constraint as an inequality.
Combine the active constraint with objective function and form an LP model: $min\sum_{l\epsilon\mathcal{L}}|w_l' - w_l|$, and use the simplex algorithm to solve this LP problem.
Is path P_{ij} the shortest path between node i and node j?
No
Yes
Obtain a graph G($\mathcal{N}$,$\mathcal{L}$,W'), STOP

Figure 3.5: The step of the limit constraints method

3.3.3. PSEUDOCODE

Algorithm 2: The limit constraints method based on the simplex method

```
Input  : $G(\mathcal{N},\mathcal{L},W)$, $W$ with elements $w_l$, a prescribed path $P_{ij}$
Output: $G(\mathcal{N},\mathcal{L},W')$, $W'$ with elements $w'_l$
1  find constraints, target (sub)paths $\in \mathcal{P}_0$, corresponding shortest (sub)paths
   $\in \mathcal{Q}$
2  $E \leftarrow \sum_{l\in\mathcal{Q}} w_l - \sum_{l\in\mathcal{P}_0} w_l,\ l \in \mathcal{L}$
3  choose the minimum $E_i$, $b \leftarrow E_i$
4  for $l \in 1:L$ do
5      if $l \in \mathcal{P}_0 \setminus \mathcal{Q}$ then $A_l \leftarrow 1$;
6      else if $l \in \mathcal{Q} \setminus \mathcal{P}_0$ then
7          $A_l \leftarrow -1$
8          else
9              $A_l \leftarrow 0$
10 get $A$
11 $x \leftarrow [X^+; X^-; z]$   // add slack variables
12 $A' \leftarrow [A; -A; I]$, $b' \leftarrow b$, $c' \leftarrow c$
13 $A'x = b'$   // express constraints as equations
14 $x \leftarrow \mathrm{SIMPLEX}(A', b', c')$   // the simplex method
15 return $x = (x_1, x_2, ..., x_n)$
16 if All constraints are satisfied then $x$ is the optimal solution, stop, and
   $w'_l = w_l + X^+_l - X^-_l,\ l \in \mathcal{L}$;
17 else
18     Repeat above steps.
```

3.3.4. COMPLEXITY

The complexity of Lc in one iteration is similar to that of Sp. When finding constraints, the complexity is $O(n_c * (L + Nlog_2(N)))$ (n_c is the number of constraints in each iteration, N is the number of nodes, L is the number of links), i.e. $O(pl * (L + Nlog_2(N)))$ (pl is the length of target path). After modification, $n_c = 1$, so the complexity is $O(L + Nlog_2(N))$. The complexity of the simplex algorithm is $O(2^n)$ and the complexity of the checking step is also $O(L + Nlog_2(N))$.

4

SIMULATION AND RESULTS

This chapter presents simulations of four algorithms (Cg, Qp, Sp, Lc; Qp and Cg are benchmarks) for solving the inverse shortest path problem with a single target path (ISPP-S) in different types of directed graphs. Compare and analyze their corresponding experimental results to demonstrate efficiency of our algorithms.

4.1. SIMULATION

This section gives a detailed description of experimental subjects, metrics for comparison as well as steps of simulation.

4.1.1. EXPERIMENTAL SUBJECT

We choose four types of graphs introduced in Chapter 1 as experimental subjects. Here we set some parameters to specify the properties of the different graphs in the experiment. The number of nodes and links are important factors that affect the size of the graph [22], so we set $N = 10, 20, 50, 100, 200, 500, 1000$ (N is the number of nodes). Some specific parameters for different types of graphs are as follows:

- **ER graph**: We set the connection possibility $p = 2ln(N)/N$ to construct connected ER graphs. The ER graph can be represented by $G(N, 2ln(N)/N)$.
- **BA graph**: We set $m_0 = 5$, $m_1 = 1$, so the BA graph has 5 hubs, and generates one link between a new node and one of old nodes each time. Generated BA graphs can be denoted by $G(N, 5, 1)$.
- **2D lattice graph**: A 2D lattice graph can be seen as a rectangular with u_0 nodes in its length, u_1 nodes in its width. Its notation is $G(N, u_0 \times u_1)$. So we set a series of 2D lattice graph to $G(10, 5\times 2)$; $G(20, 5\times 4)$; $G(50, 10\times 5)$; $G(100, 10\times 10)$; $G(200, 20\times 10)$; $G(500, 25\times 20)$; $G(1000, 40\times 25)$.
- **Configuration graph**: We set the degree range is $[1, 10]$, so the configuration graph is $G(N, [1, 10])$. The construction method mentioned in section 2.1.3 may generate

self-loops and multi-links. In experiments, all self-loops should be deleted. In the directed graph, we can maintain two links with different directions between a pair of nodes.

We construct these graphs as the undirected graphs. However, since most networks in reality are directed, we start with directed graphs. So except for the direction of links in 2D lattice graph matching the link generation direction, the direction of links is uniformly random.

4.1.2. METRICS FOR COMPARISON

- **Running time**: In complex systems, it is essential to find the ideal solution quickly so that modifications can be made rapidly to keep the system in proper operation. According to the analysis at the beginning of chapter 3, the running time t_r is determined by the time of each iteration t_i and the number of iterations $iter$. The time of each iteration t_i is the sum of constraints discovery time t_d, the optimisation time t_o and the checking time t_k. Fewer constraints mean the shorter optimization time t_o.

- **Adjustment of link weight**: For a fixed graph, no matter what methods are used, the value of the minimum adjustment of link weights is unique. The weighted adjustments of various methods reflect which method's 'optimal solution' is the closest to this real value. The factor that influences the total weighted adjustment is the active constraint.

- **Path length**: Path length is the number of links on a path. A longer path means more time spent on search constraints. The running time may be affected by the path length.

4.1.3. SIMULATION

We wrote these algorithms in Matlab language and run programs on Lenovo Thinkpad 20T8A000CD. Perform 10,000 experiments on each type of graphs mentioned above in QCE cluster. The simulation for each experiment is as follows:

- Randomly generate a directed graph (ER graph, BA graph, 2D-lattice graph, and configuration graph) $G(\mathcal{N},\mathcal{L},W)$ with N nodes, L links and W is a link weight matrix. The elements w_l of the matrix W are numbers uniformly distributed in range $[0,1]$, $l \in \mathcal{L}$.

- Randomly select a path P_{ij}: 1. randomly select a node i as the source; 2. randomly select a integer pl from $[1, N-1]$ as the path length of the target path; 3. select a path whose source is node i, its length is pl and its destination is node j. If path P_{ij} is the shortest between i and j, re-select a path.

- With the graph G and the path P_{ij} as inputs, obtain a new link weight matrix W' with elements w'_l by different algorithms.

- Record the running time of each algorithm and calculate their total weight adjustment $\Delta = \sum_{l\in\mathcal{L}} \left|w'_l - w_l\right|$, where w'_l and w_l are the weight of each link l.

4.2. RESULT

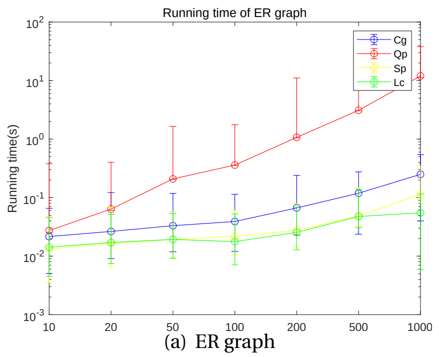
4.2.1. RUNNING TIME

We use error bars to plot the running time of different methods. The X-axis represent the number of nodes in graph and the Y-axis is the logarithmic form of running time.

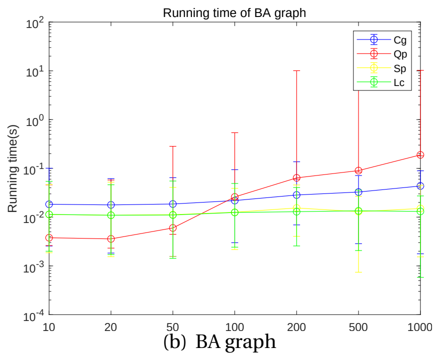
Figure 4.1 indicates that running times of Sp and Lc are generally shorter than other methods. They are more efficient than Qp in ER, 2D lattice and configuration graphs. In graphs with 1000 nodes, Lc has better performance than Sp. This is because the number of active constraints of Sp is much more than that of Lc in large-sized graphs.

However, the running time of Cg is longer than that of Sp and Lc; this is because Cg uses the dual LP problem. For the ISPP-S, compared to the number of variables (the number of links), the number of constraints is still relatively small. After transforming the original problem to the dual problem, the number of constraints is more than variables, which increases the complexity of optimization and t_o is longer.

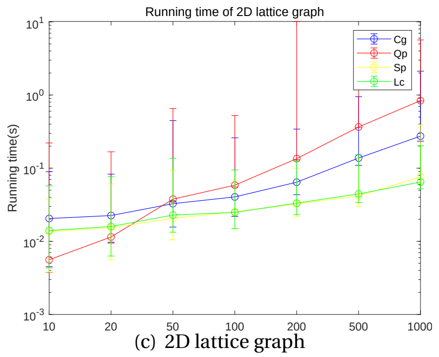
The running time of Qp is short in BA graphs with fewer nodes ($N < 100$), and also in graphs with relatively few nodes ($N = 10$). The reason is that Qp needs to balance the weight of paths by increasing or decreasing link weights by the same value, which is relatively easy to achieve in graphs with a simple structure.



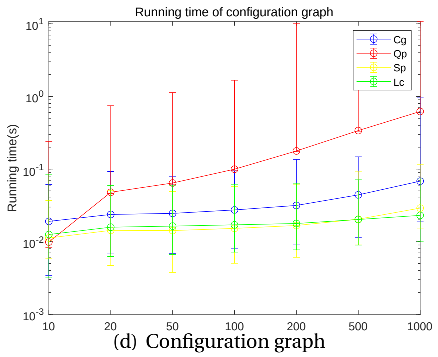
(a) ER graph



(b) BA graph



(c) 2D lattice graph



(d) Configuration graph

Figure 4.1: The running time of four algorithms solving ISPP-S in four different types of directed graphs. X-axis represent the number of nodes N; Y-axis is the logarithmic form of the running time. Blue line-Cg; red line-Qp; yellow line-Sp; green line-Lc.

4.2.2. ADJUSTMENT OF LINK WEIGHT

Error bars are used to display the total adjustment of link weights. The X-axis represent the number of nodes in graph and the Y-axis is the sum of link weight adjustment.

Figure 4.2 illustrates that the total weight adjustment of Cg, Sp and Lc is generally similar in all kinds of small-size graphs ($N < 100$). Especially in BA graphs, the weight adjustment of each method tends to be equal in most cases, this is because the structure of BA graphs is relatively simpler.

Adjustment of Qp changes most. This is because the active set method requests the weight of links involved in active constraints to balance, which means all links should increase or decrease the same value simultaneously. So the step length has multiple decimal places and the link cannot change its weight flexibly. Links with negative weights generated in the iteration also need additional adjustments, which increases the weight adjustment.

Compared with Qp, the total weight adjustment of other methods (Cg, Sp, Lc) are close. With the size of graphs growing, their adjustments tend to be unequal. The reason is the different steplength and step direction under distinct active constraints and the steplength actually is the weight adjustment in each iteration.

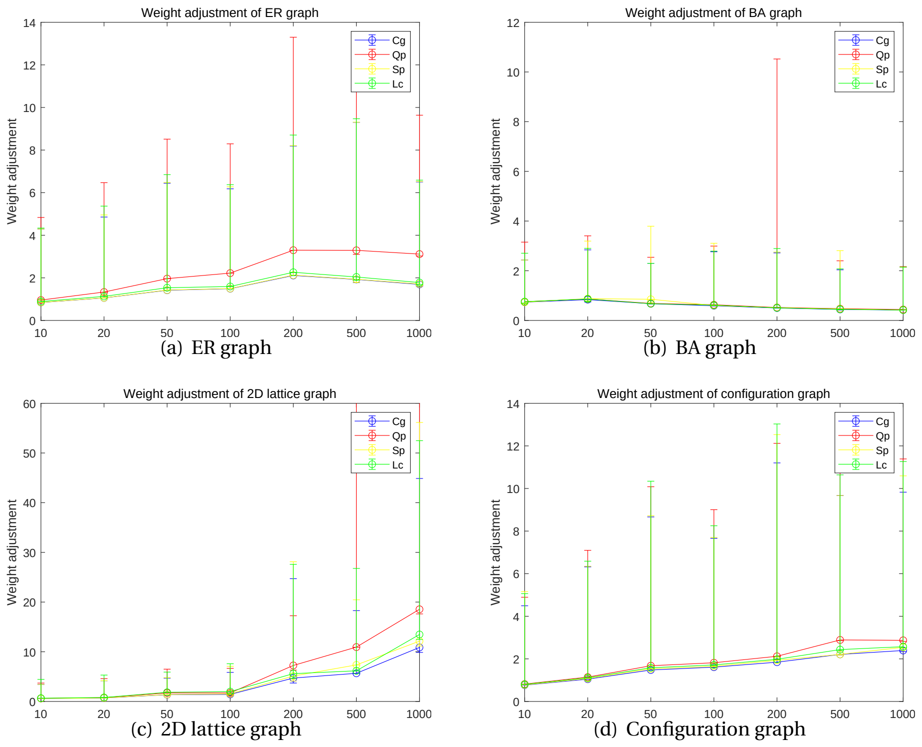


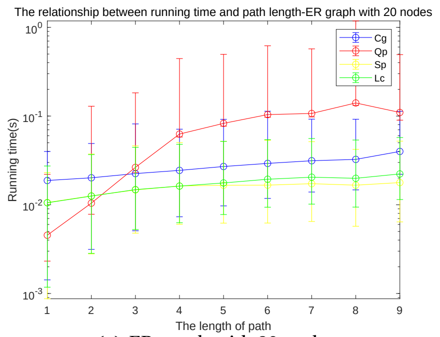
Figure 4.2: The weight adjustment of four algorithms solving ISPP-S in four different types of directed graphs. X-axis represent the number of nodes N; Y-axis is the sum of link weight adjustment. Blue line-Cg; red line-Qp; yellow line-Sp; green line-Lc.

4.2.3. PATH LENGTH

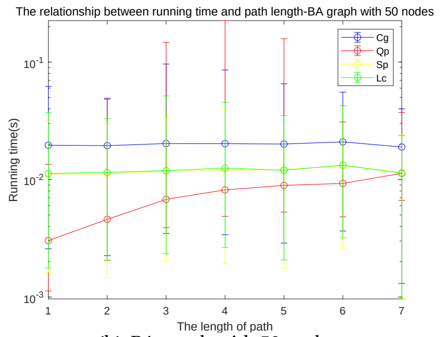
We use error bars to display the relationship between running time and path length. The X-axis represent the path length pl; Y-axis is the logarithmic form of the running time.

Figure 4.3 shows that the running times of Cg, Sp and Lc are mostly independent of the path length. The total running time is determined by the time per iteration t_i and the number of iterations $iter$. Due to the fewer constraints of ISPP-S, the optimal solution can be obtained without too many iterations in these three methods and the total running time is mainly affected by the time of each iteration t_i. The time t_i includes the time of constraints discovery t_d, the time of finding optimal solution t_o and the time of checking. Although the time t_d depends on the path length in Sp and Lc, it is much shorter than the time to solve the LP model t_o and can be negligible. So the running times of these three methods are not significantly impacted by the path length.

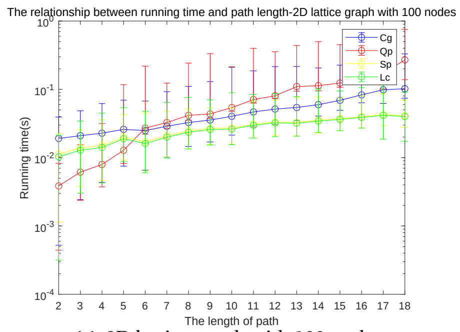
However, the running time of Qp is positively correlated with the path length. Compared with the above three methods, Qp requires more iterations to obtain the optimal solution, which leads to an increase in the total running time. After each iteration, the number of constraints in the active set increases, and the running time to find the optimal solution t_o also increases.



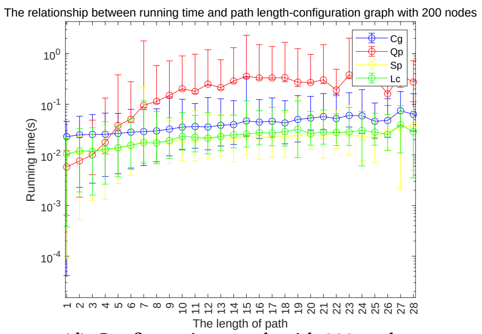
(a) ER graph with 20 nodes



(b) BA graph with 50 nodes



(c) 2D lattice graph with 100 nodes



(d) Configuration graph with 200 nodes

Figure 4.3: The relationship between running time and path length in (a) directed ER graph with 20 nodes, (b) directed BA graph with 50 nodes, (c) directed 2D lattice graph with 100 nodes and (d) directed configuration graph with 200 nodes. X-axis represent the path length pl; Y-axis is the logarithmic form of the running time. Blue line-Cg; red line-Qp; yellow line-Sp; green line-Lc.

4.3. SUMMARY

For ISPP-S in different types of directed graphs, Sp and Lc work more efficiently than Qp and Cg.

- The running time of Sp and Lc are shorter than that of Qp and Cg in ER graphs, 2D lattice graphs, configuration graphs and large-sized BA graphs($N > 100$). Lc has a better performance than Sp in graphs with complicated structures (both N and L are large).
- The total weight adjustment of Sp and Lc is equal to Cg in most cases and sometimes a little larger than it, but smaller than Qp.
- The effect of path length on the running time of Sp and Lc is minimal. These two methods can efficiently handle a target path of any length.

5
Extensions

Chapter 2, 3, and 4 mainly research the inverse shortest path problem with only one prescribed path (ISPP-S). We expand the number of the target paths from one to more than one and attempt to adapt these algorithms to deal with the inverse shortest path problem with multiple target paths (ISPP-M) and the inverse shortest path problem with target paths in a spanning tree (ISPP-T).

Given that the above experiments used directed graphs, we should also think about whether our methods (Sp and Lc) can be used for undirected graphs.

5.1. ISPP-M

ISPP-M requests that various target paths P_i $(i = 1, 2, ..., n,\ n$ is a positive integer greater than 1) in a graph $G(\mathcal{N}, \mathcal{L}, W)$ become the shortest path between their corresponding starting and ending node pairs (s_i, t_i). This problem can be expressed as

$$\min \sum_{l \in \mathcal{L}} |w'_l - w_l| \tag{5.1}$$

$$\text{whereby} \begin{cases} P_i \text{ is the shortest path from } s_i \text{ to } t_i \text{ with } W', i = 1, 2, ..., n \\ w_l, w'_l \geq 0, l \in \mathcal{L} \end{cases}$$

Figure 5.1 shows an ISPP-M with 5 prescribed paths $(n = 5)$ in a graph.

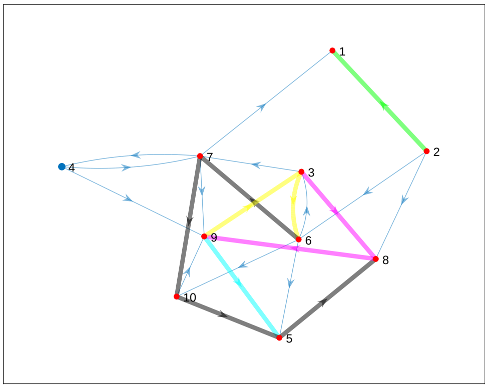


Figure 5.1: The ISPP-M with 5 identified paths

5.1.1. MODIFICATION

Compared to the ISPP-S, the ISPP-M has n prescribed paths and more constraints. We can decompose this problem into n subproblems. Each subproblem requests one target path P_i to become the shortest one between corresponding node pairs (s_i, t_i).

The way to choose active constraints for each target path is the same as mentioned in section 3.2 and 3.3.

But, assuming that path P_1 has become the shortest path between node s_1 and node t_1, when we solve the second subproblem whose target path is P_2, the weight of links on path P_1 may be increased to make path P_2 be the shortest one between node pairs (s_2, t_2). In such a situation, path P_1 may not be the shortest path between node s_1 and node t_1.

Considering this issue, we combine each set of constraints corresponding to each target path. In this way, all predetermined paths can simultaneously become shorter than the original shortest path between their associated sources and destinations. However, the solution here is still an approximation. Algorithms need to loop more times to find the optimal solution, which takes a lot of time.

There are two ways to avoid this situation:

- Maintain previous constraints and merge new constraints with them in the next iteration.
- Only reduce the weight of links on target paths to satisfy all constraints [23]. The weights of links on previous target paths are not affected when processing other prescribed paths. But this method will increase the total adjustment of link weight.

When choosing the first approach, the number of constraints will increase after each iteration. The time for optimization t_o will become longer, so we can solve its dual problem to avoid complex calculations.

For **the split path method**, we choose the first approach and find its optimal solution by solving its dual problem. Add new constraints to the original ones rather than simply replacing the old ones with the new ones. The approach for selecting constraints is to split the target path, which is the same as the method mentioned in section 3.2.

For **the limit constraints method**, we select the second approach and set the upbound of increasing vector as 0. So other links that are not part of the current target path will not be influenced. The old constraints will be dropped after each iteration.

5.1.2. Simulation

The simulation for ISPP-M is similar to ISPP-S. The difference is that the input of algorithms is 5 paths, so it is unnecessary to investigate the path length here. We run programs on the Lenovo Thinkpad 20T8A000CD and perform 1,000 experiments on each type of graphs mentioned in section 4.1.1 in QCE cluster. The simulation for each experiment is as follows:

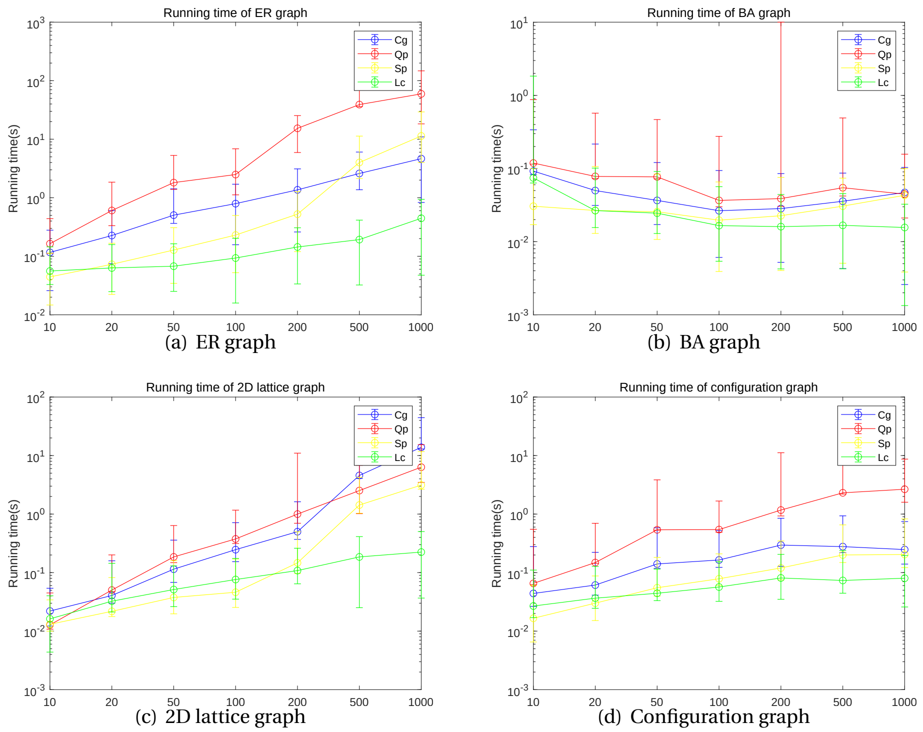
- Randomly generate a directed graph (ER graph, BA graph, 2D-lattice graph, and configuration graph) $G(\mathcal{N},\mathcal{L},W)$ with N nodes, L links and W is a link weight matrix. The elements w_l of the matrix W are numbers uniformly distributed in (0,1), $l \in \mathcal{L}$.
- Randomly select 5 paths $P_i,(i = 1,2,...,5)$. Request that at least one of the target paths not be the shortest between its corresponding source and destination.
- With the graph G and 5 paths P_i as inputs, obtain a new link weight matrix W' with elements w'_l by different algorithms.
- Record the running time of each algorithm and calculate each algorithm's total weight adjustment $\Delta = \sum_{l\in\mathcal{L}} \left|w'_l - w_l\right|$, where w'_l and w_l are the weight of each link l.

5.1.3. Results

Running time

We use error bars to plot the running time of different methods. The X-axis represents the number of nodes in a graph and the Y-axis is the logarithmic form of running time.

The running time of solving ISPP-M is as shown in Figure 5.2. In small-sized graphs ($N \leq 20$), Sp's running time is usually the shortest. With the increasing number of nodes in graphs, Lc performs better. The reason is that as the size of graphs grows, new constraints need to be merged with the old ones in Sp, so the number of active constraints will increase. But the number of active constraints in Lc will remain the same. More constraints mean longer optimization time, so Lc's optimization time t_o is shorter than Sp's.



(a) ER graph (b) BA graph

(c) 2D lattice graph (d) Configuration graph

Figure 5.2: The running time of four algorithms solving ISPP-M with 5 target paths in four different types of directed graphs. X-axis represents the number of nodes N; Y-axis is the logarithmic form of the running time. Blue line-Cg; red line-Qp; yellow line-Sp; green line-Lc.

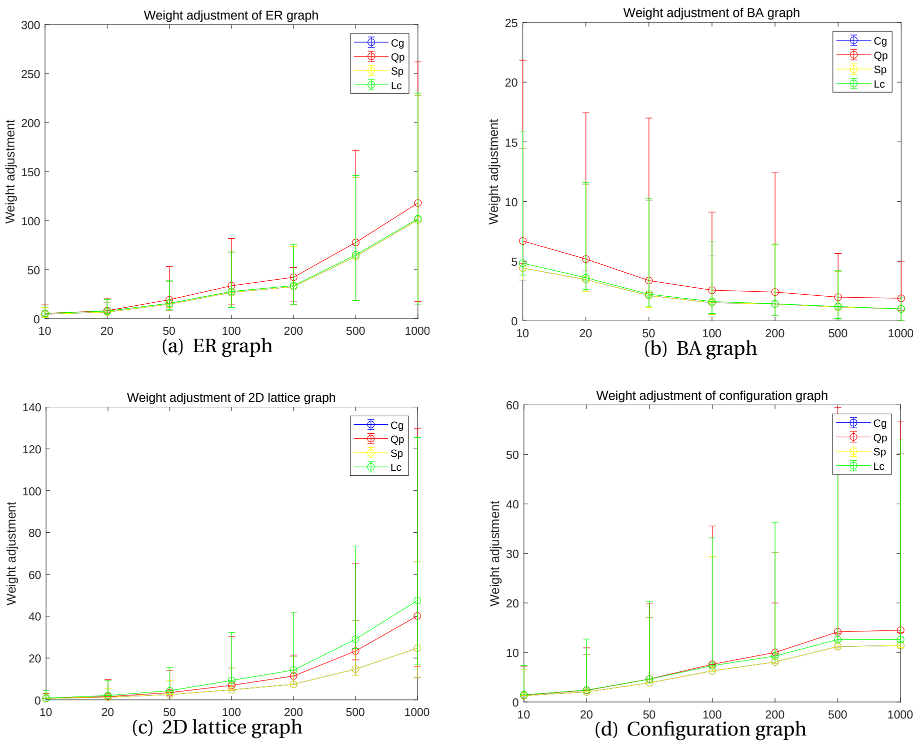
ADJUSTMENT OF WIGHT

Error bars are used to display the total adjustment of link weights. The X-axis represents the number of nodes in graph and the Y-axis is the sum of link weight adjustment.

From the previous analysis, we know that the difference in weight adjustment is influenced by the steplength and step direction in each iteration.

Combining Figure 5.2 and 5.3, we find that the weight adjustment of Cg and Sp is the smallest, although their running time is a little longer. This is because these two methods still maintain the old constraints and the set of constraints becomes more binding with the addition of the new constraints. The weight adjustment in each iteration is more accurate.

Lc only decreases the weight of the target path, which may lead to a larger weight adjustment (steplength) in each iteration. It sacrifices the total weight adjustment to obtain a shorter running time. However, the increase in link weight adjustment is within an acceptable range.



(a) ER graph (b) BA graph (c) 2D lattice graph (d) Configuration graph

Figure 5.3: The weight adjustment of four algorithms solving ISPP-M with 5 target paths in four different types of directed graphs. X-axis represents the number of nodes N; Y-axis is the sum of link weight adjustment. Blue line-Cg; red line-Qp; yellow line-Sp; green line-Lc.

5.1.4. SUMMARY

For ISPP-M in different types of directed graphs,

- Lc can solve this problem within a short time. Sp runs fast in relatively small-sized graphs and its running time becomes longer in graphs with complicated structures.
- The total weight adjustment Δ of Lc is larger than others' in 2D lattice graphs. In other graphs, Lc's weight adjustment is relatively larger but smaller than Qp's.

5.2. ISPP-T

ISPP-T is a special variant of ISPP-M. Find the spanning tree T of a graph G such that T becomes the shortest path tree of graph G, i.e., the weight of path from root r to any other nodes t_i in the spanning tree is the shortest from r to t_i in the original graph G [24]. Figure 5.4 shows an example of ISPP-T in the 2D lattice graph. The solid black lines and red nodes compose a spanning tree of the lattice graph. The tree's root r is node 1.

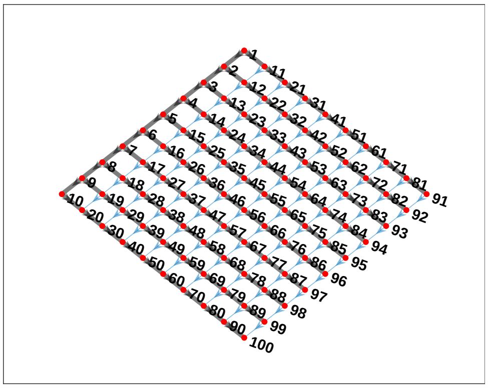


Figure 5.4: The spanning tree of 2D lattice graph with 100 nodes

This problem can be expressed as follows,

$$\min \sum_{l\in\mathscr{L}} |w_l' - w_l| \tag{5.2}$$

$$\text{whereby}\begin{cases} T \text{ is the shortest path tree of graph } G \\ w_l, w_l' \geq 0, l \in \mathscr{L} \end{cases}$$

5.2.1. MODIFICATION

ISPP-T can apply improved algorithm Lc for ISPP-M in section 5.1. Because this problem needs every subpath with source r to be the shortest, the way to search for constraints in Lc is perfectly suited to this problem. But it is unnecessary to split target paths using Sp, so we delete the process of splitting the path, Sp has become a totally new method which utilizes all constraints and the simplex method to solve the linear programming problem. We name it the all constraints method (Ac).

5.2.2. SIMULATION

The simulation of ISPP-T is similar to that of the above problems. But the input, prescribed paths, should be changed to a spanning tree that is not the shortest path tree of graph G. We run programs on Lenovo Thinkpad 20T8A000CD and perform 1,000 experiments on each type of graphs mentioned in section 4.1.1 using QCE cluster. The simulation for each experiment is as follows:

- Randomly generate a directed graph (ER graph, BA graph, 2D-lattice graph, and configuration graph) $G(\mathscr{N},\mathscr{L},W)$ with N nodes, L links and W is a link weight matrix. The elements w_l of the matrix W are numbers uniformly distributed in (0,1), $l \in \mathscr{L}$.
- Find the spanning tree T of the graph.
- With the graph G and the spanning tree T as inputs, obtain a new link weight matrix W' with elements w_l' by different algorithms.

- Record the running time t_r of each algorithm and calculate each algorithm's total weight adjustment $\Delta = \sum_{l \in \mathscr{L}} |w'_l - w_l|$, where w'_l and w_l are the weight of each link l.

5.2.3. Results

In ISPP-T with a lot of constraints, it is hard for Qp to find the optimal solution. We set the total adjustment Δ to infinity when infeasible solution cases occur. Additionally, Cg needs to introduce slack variables, which may exceed the default memory of Matlab in some graphs (ER graphs with more than 200 nodes). These cases will be ignored.

Running time

We use error bars to plot the running time of different methods. The X-axis represents the number of nodes in graphs and the Y-axis is the logarithmic form of running time.

Figure 5.5 shows the running time of different methods taken on different graphs. From this figure, Lc and Ac run relatively fast. Ac has better performance in ER graphs and configuration graphs. This is because fewer active constraints are not sufficient to find the optimal solution quickly, Lc needs more iterations, which leads to an increase in its total running time.

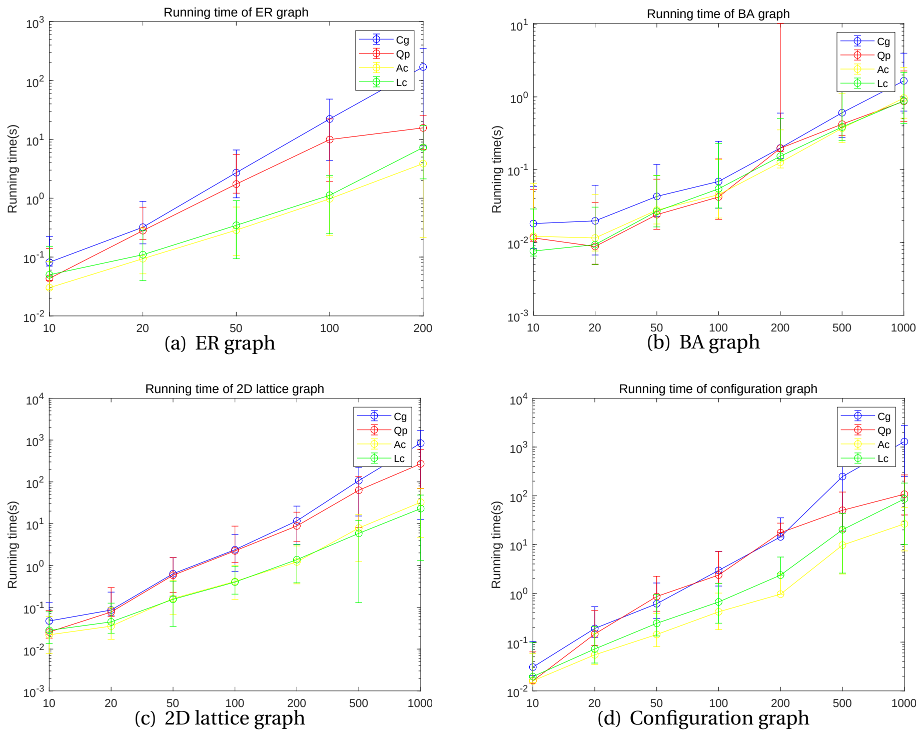


Figure 5.5: The running time of four algorithms solving ISPP-T in four different types of directed graphs. X-axis represent the number of nodes N; Y-axis is the logarithmic form of the running time. Blue line-Cg; red line-Qp; yellow line-Ac; green line-Lc.

ADJUSTMENT OF WIGHT

Error bars are used to display the total adjustment of link weights. The X-axis represent the number of nodes in graph and the Y-axis is the sum of link weight adjustment.

Figure 5.6 demonstrates that the weight adjustment of Cg and Ac is the smallest, while that of Lc is the largest. The reason is the same as mentioned in ISPP-M, Lc's adjustment of each iteration Δ_i is a little larger. As the number of iterations of Lc increases, the total adjustment Δ increases significantly. In addition, Qp cannot deal with ISPP-T in the large-sized networks.

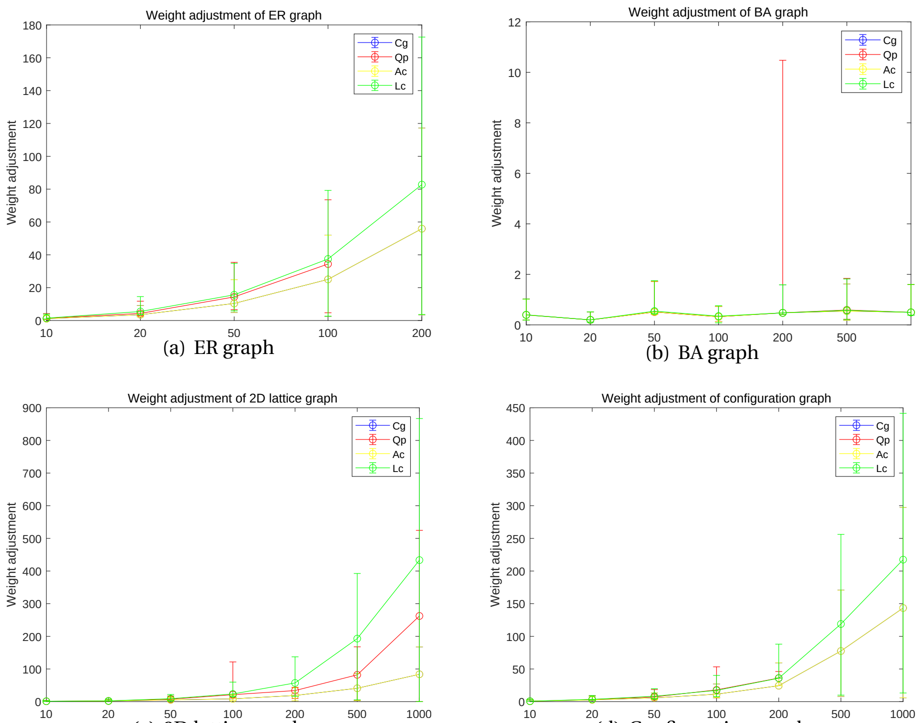


Figure 5.6: The weight adjustment of four algorithms solving ISPP-T in four different types of directed graphs. X-axis represents the number of nodes N; Y-axis is the sum of link weight adjustment. Blue line-Cg; red line-Qp; yellow line-Ac; green line-Lc.

5.2.4. SUMMARY

For ISPP-T, the running time of Ac and Lc are both shorter, and Ac has a better performance in relatively complicated graphs (ER graphs and configuration graphs). The weight adjustment of Lc far exceeds that of other methods, especially in large-sized graphs.

5.3. UNDIRECTED GRAPH

Compared with directed graphs, the undirected graph with the same number of nodes has more paths, and the length of these paths tends to be longer. This means that, the ISPP in undirected graphs has more constraints.

5.3.1. MODIFICATION

In the process of searching constraints, we need to find the shortest path with the same origin and destination as the target path and compare their weights. In undirected graphs, the existence of negative weight links makes it more difficult to find the shortest path. With negative weights in undirected graphs, Dijkstra's shortest path algorithm is no longer applicable [25]. Bellman–ford algorithm [26] can handle negative links in directed graphs but not undirected graphs.

However, Cg, Sp and Lc have set bounds to avoid generating negative links and Dijkstra's shortest path algorithm [19] is still available. Therefore, there is no need to modify these methods.

In Qp, the unsuitable steplength may lead to generating negative link weights (see section 2.3.1 for specific reasons). So we need to employ a technique that can calculate the shortest path in an undirected graph containing negative link weights. We can calculate the weight of all paths between the same source and destination as the target path. Compare them and obtain the shortest one.

5.3.2. SIMULATION

The simulation is the same as the previous simulations. But the experimental subjects are the undirected graphs. We need to set the link direction of the experimental subjects mentioned in section 4.1.1 to undirected. When dealing with ISPP-S, the number of predefined paths is 1; when solving ISPP-M, the number of identified paths is 5; in ISPP-T, the constraint is to make a spanning tree T become the shortest path tree. We use Matlab to perform 1,000 experiments in QCE cluster.

5.3.3. RESULTS

Because Qp needs to traverse all paths to find the shortest path, it may exceed Matlab's default memory. For Qp, we only show its results obtained from the graphs that can be handled. The infeasible solution case will be ignored. Because the optimization process of the algorithms has not changed, the total weight adjustment is the same as previous results. Here we only investigate the influence on the running time.

RUNNING TIME

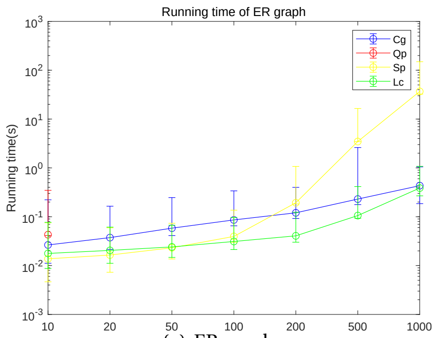
We use error bars to plot the running time of different methods. The X-axis represents the number of nodes in a graph and the Y-axis is the logarithmic form of running time.

ISPP-S

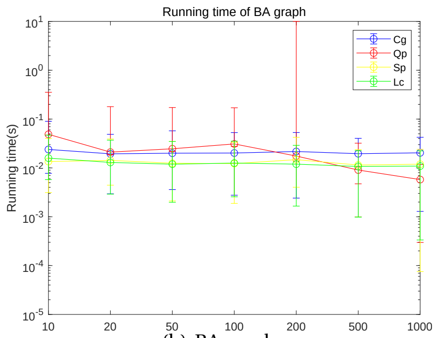
Figure 5.7 shows the running time of algorithms for solving ISPP-S in undirected graphs, where the line ends means that the running time is too long and is considered infinite.

Sp performs well in graphs with fewer nodes. With the increasing number of nodes, the path tends to be long and the number of constraints increases. The optimization time will become longer. Compared to Sp, the running time of Lc is stable, the reason is that there is one constraint in every iteration.

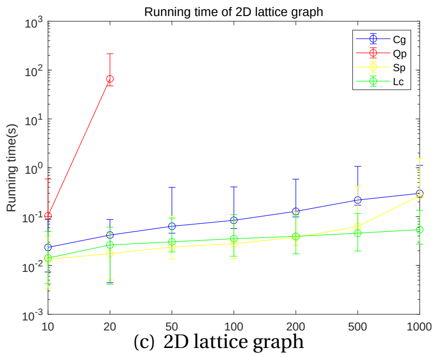
Qp spends a lot of time when the graph structure tends to be complex. This is because there are negative link weights during the iteration. So Qp needs a long time calculating the weight of paths.



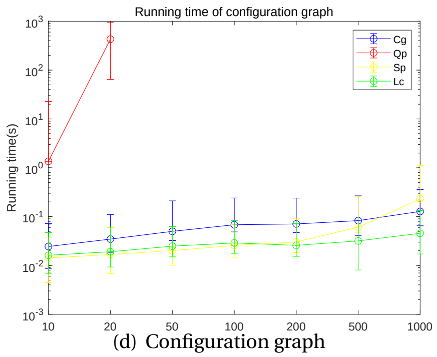
(a) ER graph



(b) BA graph



(c) 2D lattice graph



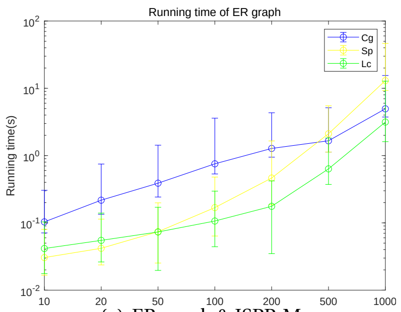
(d) Configuration graph

Figure 5.7: The running time of four algorithms solving ISPP-S in four different types of undirected graphs. X-axis represents the number of nodes N; Y-axis is the logarithmic form of the running time. Blue line-Cg; red line-Qp; yellow line-Sp; green line-Lc.

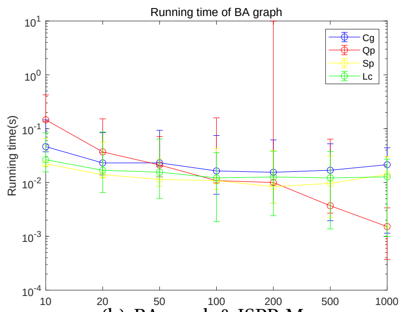
ISPP-M & ISPP-T

For ISPP-M, Figure 5.8 displays that, Qp can only deal with BA graphs as well as small-sized graphs; Sp and Lc still perform better. As the size of graph grows and the number of constraints increases, Lc's running time asymptotically approaches that of Cg due to a large number of iterations. Sp's running time exceeds that of Cg because of a larger size of active constraints set after every iteration.

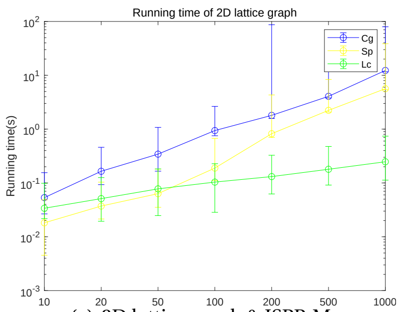
For ISPP-T, the running time of Lc is longer than Ac and they exceed Cg in large-sized graphs. Column generation algorithm is gradually showing its advantages.



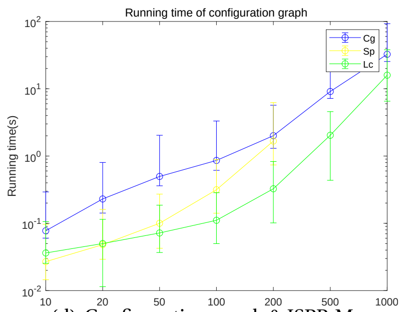
(a) ER graph & ISPP-M



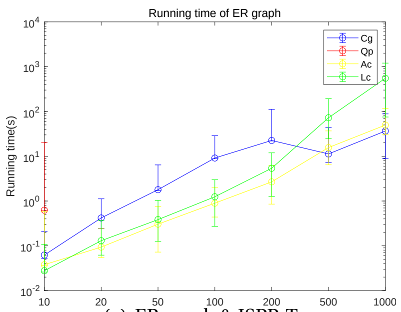
(b) BA graph & ISPP-M



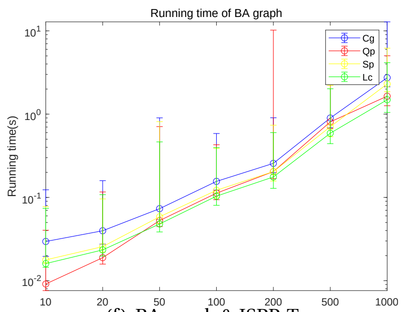
(c) 2D lattice graph & ISPP-M



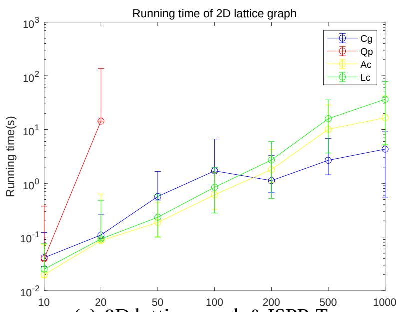
(d) Configuration graph & ISPP-M



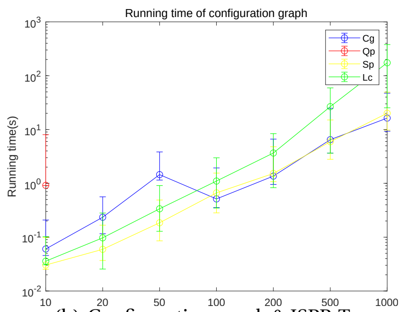
(e) ER graph & ISPP-T



(f) BA graph & ISPP-T



(g) 2D lattice graph & ISPP-T



(h) Configuration graph & ISPP-T

Figure 5.8: The running time of four algorithms solving ISPP-M (a)-(d) and ISPP-T (e)-(h) in four different types of undirected graphs. X-axis represents the number of nodes N; Y-axis is the logarithmic form of the running time. Blue line-Cg; red line-Qp; yellow line-Sp/Ac; green line-Lc.

5.3.4. SUMMARY

- The running time of Lc for solving ISPP-S and ISPP-M is relatively short. It takes longer time in ISPP-T and exceed the time of Cg in large-sized undirected graphs.
- When dealing with ISPP-S, the running time of Sp is longer than others in large ER graphs. After modification, its performance has been improved when solving ISPP-M. When Ac handling ISPP-T in large-sized undirected graphs, it performs worse than Cg.
- Cg can efficiently handle the ISPP-T problem in undirected graphs with complex structures.

6 EMPIRICAL APPLICATIONS

The inverse shortest path problem (ISPP) can be applied to resource reallocation and route reprogramming in empirical networks. This chapter mainly introduces four networks in real life (US airlines network [27], Protein interaction network [28], University email network [29] and European roads network [29]) and how to apply four methods (Qp, Cg, Sp, Lc) to solve relevant ISPP in these networks. The basic statistical properties of these four networks are shown in Table 6.1.

Table 6.1: Basic statistical properties of empirical networks

Network	number of nodes (N)	number of links (L)
US airlines network	332	2.1k
Protein interaction network	1.8k	4.4k
University email network	1.1k	5.5k
European roads network	1.2k	1.4k

6.1. US AIRLINES NETWORK

The U.S airline network [27] can be seen as an undirected graph, which describes airlines between different US airports. Airports are represented by nodes, airlines are represented by links, and traffic flow on each airline is the link weight.

6.1.1. PHYSICAL MEANING

Assuming that, here is a route from DTW airport in Detroit to Miami's FLL airport via ATL airport in Atlanta. Due to flight flow control, ATL airport cannot accommodate many airplanes. Flights would need to be adjusted to reduce traffic flow on the route via ATL. This route can choose a new transit city (keep the same origin DTW and destination FLL as before). Based on the physical meaning, we can map traffic flow problem to ISPP with multiple target paths (ISPP-M). All routes via ATL can be considered as the target paths.

6.1.2. SIMULATION

Figure 6.1 shows the US airline network. The simulation is as follows:

- Give the US airlines network with the link weight matrix W whose elements are $w_l, l \in \mathcal{L}$.
- Select a node representing the "ATL airport" (yellow point in Figure 6.1). Target paths (black solid lines in Figure 6.1) are all paths containing this node and their path length are equal to 2.
- With the graph G and the target paths as inputs, obtain a new link weight matrix W' with elements w'_l by different algorithms.
- Record the running time of each algorithm and calculate each algorithm's total weight adjustment $\Delta = \sum_{l\in\mathcal{L}} |w'_l - w_l|$, where w'_l and w_l are the weight of each link l.

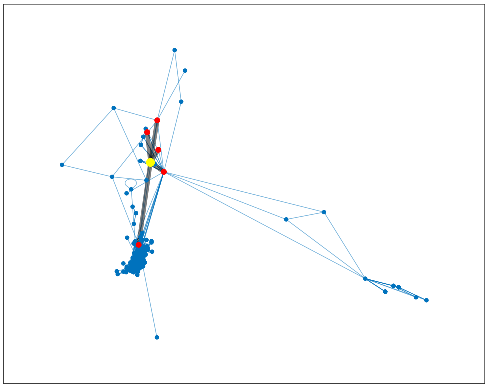


Figure 6.1: US airline network

6.1.3. RESULTS

Table 6.2 indicates their running time and total adjustment of link weights. Lc takes the shortest time to deal with this problem. The weight adjustments of Cg, Sp and Lc are the same, all lower than that of Qp.

Table 6.2: US airline network

Method / Result	Column generation	Quadratic programming	Split path	Limit constraints
Running time(s)	0.181613	3.501054	0.974586	0.040649
Weight adjustment	27.6254	38.9748	27.6254	27.6254

6.2. PROTEIN INTERACTION NETWORK

The protein interaction network [28] is considered to be directed. The nodes of it are proteins; links are the channel for transmitting signals. The link weight can be defined as transmission time, which determines which product can be generated.

6.2.1. PHYSICAL MEANING

One of the critical functions of proteins is to generate chemical products for human metabolic processes. The transmission time of signal will decide the product generated [30]. We plot Figure 6.2 to explain more clearly how the protein works.

There are four proteins in Figure 6.2. If protein A sends a signal to protein B, it needs to be via another protein, C or D, as a medium. There are two routes (A-C-B or A-D-B) available for this signal, and it tends to choose the shortest one. This signal will change after it passes through medium D, and the changed signal will stimulate protein B to produce another product that is different from the one we want to obtain. In order to ensure that protein B will generate the target product, the transmission time on route A-C-B must be shorter than that on A-D-B. We can adjust the signal delivery time through external intervention. This kind of problem can be considered as ISPP-S. The route for signal transmission is the path in the network. We predetermine a path and request that it should be the shortest in order to obtain the target chemical product.

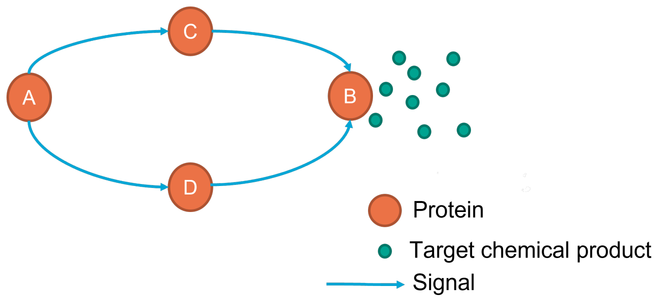


Figure 6.2: Protein interaction network

6.2.2. SIMULATION

The protein network can be simplified as Figure 6.3. The simulation is as follows:

- Give the protein interaction network with the link weight matrix W whose elements are w_l, $l \in \mathscr{L}$.
- Select a specific target path based on a target product to be generated (black solid lines in Figure 6.2).
- With the graph G and this target path as inputs, obtain a new link weight matrix W' with elements w'_l by different algorithms.
- Record the running time of each algorithm and calculate each algorithm's total weight adjustment $\Delta = \sum_{l \in \mathscr{L}} |w'_l - w_l|$, where w'_l and w_l are the weight of each link l.

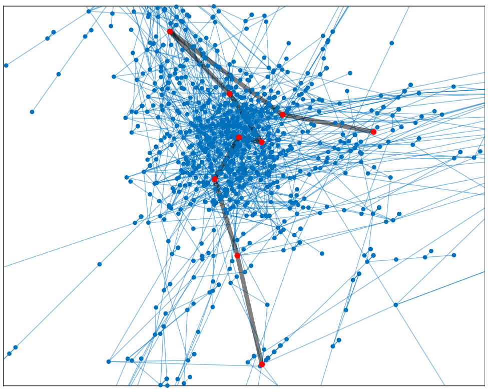


Figure 6.3: Protein interaction network

6.2.3. RESULTS

Table 6.3 shows that Sp and Lc are more effectively used in this network, their running times are the shortest.

Table 6.3: Protein interaction network

Method / Result	Column generation	Quadratic programming	Split path	Limit constraints
Running time(s)	0.8662	0.0341	0.0152	0.0134
Weight adjustment	5	5	5	5

6.3. UNIVERSITY EMAIL NETWORK

The university email network can be considered a directed network. The nodes in this network are universities, the links represent email sending and receiving, and the link weights indicate the different delay times when sending or receiving emails.

6.3.1. PHYSICAL MEANING

The university email network is similar to the routing network. Some emails about academic communication and data sharing are urgent and need to be sent within a shorter period of time. A guideline for sending and receiving emails is as follows: The delay in direct contact between the two universities must not exceed the delay in forwarding through the other university, which ensures efficient communication. The communication rule can be expressed as an ISPP-S problem. The delay of directed communication should be the shortest, meaning that this target path has minimal weight compared with other paths between the same node pair.

6.3.2. SIMULATION

Figure 6.4 is a university email network. The simulation is as follows:

- Give the university Email network with the link weight matrix W whose elements are w_l, $l \in \mathscr{L}$.

- Select two universities and set the link connecting them as the target path (black solid lines in Figure 6.4).
- With the graph G and this target path as inputs, obtain a new link weight matrix W' with elements w'_l by different algorithms.
- Record the running time of each algorithm and calculate each algorithm's total weight adjustment $\Delta = \sum_{l\in\mathscr{L}} \left|w'_l - w_l\right|$, where w'_l and w_l are the weight of each link l.

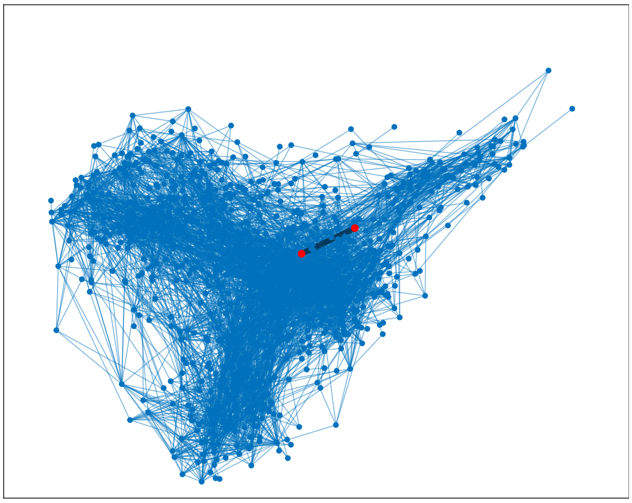


Figure 6.4: University email network

6.3.3. RESULTS

Table 6.4 illustrates that Cg (long running time) and Qp (no solution) are not suitable for handling one short target path in a large network.

Table 6.4: University email network

Method / Result	Column generation	Quadratic programming	Split path	Limit constraints
Running time(s)	21.387559	0.349255	0.086833	0.082319
Weight adjustment	0.0198	inf	0.0198	0.0198

6.4. EUROPEAN ROADS NETWORK

In this undirected network, we address cost allocation. The network's nodes are European cities; links are the roads connecting them; and the link weight is the investment in road construction.

6.4.1. PHYSICAL MEANING

Before planning the construction of roads, each local government needs to dedicate some finances to the paths across its territory. Due to the different financial strengths of each country, some routes are shorter and do cost little but have a significant investment. Therefore, a thorough professional analysis and survey will be conducted to select routes and reduce their investment and distribute it to other routes with the same origins and destinations. But other routes cannot be allocated too much money, otherwise it would not be fair to some countries that provide significant funding. This problem can be seen as an ISPP-S problem. Reassigning the investment on each route is redistributing the link weights on each path.

6.4.2. SIMULATION

Figure 6.5 shows the network and the identified path. The simulation is as follows:

- Give the European roads network with the link weight matrix W whose elements are w_l, $l \in \mathscr{L}$.
- Select an over-invested route as the target path (black solid lines in Figure 6.5).
- With the graph G and this target path as inputs, obtain a new link weight matrix W' with elements w'_l by different algorithms.
- Record the running time of each algorithm and calculate each algorithm's total weight adjustment $\Delta = \sum_{l\in\mathscr{L}} \left|w'_l - w_l\right|$, where w'_l and w_l are the weight of each link l.

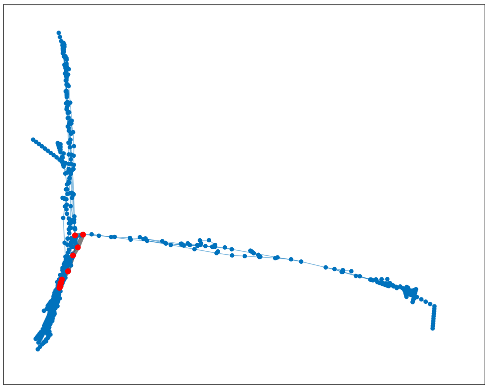


Figure 6.5: European roads network

6.4.3. RESULTS

Table 6.5 depicts that Sp is the most suitable method to deal with this type of problems in this network (short running time and minimal weight adjustment).

Table 6.5: European roads network

Method / Result	Column generation	Quadratic programming	Split path	Limit constraints
Running time(s)	0.3734	0.0017	0.0120	0.0205
Weight adjustment	18	inf	18	21

6.5. SUMMARY

According to the above results, the ISPP in empirical networks can be easily solved by Sp and Lc. Their running time is far shorter than that of Cg and Qp, especially when solving ISPP-S. Their associated adjustment of link weight is also relatively small. However, Qp is not really suitable for handling the large-sized complex network. It is possible to fail to find the optimal solution. Cg takes a long time to deal with problems where the number of variables far exceeds the constraints, but this type of problem is common in real-world complex networks.

7 CONCLUSION AND FUTURE WORK

This chapter reviews the primary objectives of this thesis, provides an overview of the completed research, and makes some suggestions for future work.

7.1. CONCLUSION

The main objective of this thesis is to propose new algorithms to modify a path into the shortest path, which is also known as the inverse shortest path problem (ISPP).

This thesis begins with the properties and metrics of graph theory and random graph constructions. We introduce ISPP, provide an overview of prior research on ISPP, and explain and evaluate two existing algorithms, the quadratic programming method (Qp) and the column generation method (Cg) in Chapter 2. These two methods are as the benchmarks in following experiments.

We have proposed two algorithms, the split path method (Sp) and the limit constraints method (Lc), to solve ISPP with one target path (ISPP-S) in directed graphs in Chapter 3. Then in Chapter 5 we have modified these two methods and made them can deal with ISPP with multiple target paths (ISPP-M) and with target paths in a spanning tree (ISPP-T) in directed graphs. Moreover, Sp and Lc can be used to handle the undirected graphs without any modification. In Chapter 6, we applied these algorithms (Qp, Cg, Sp, Lc) to solve ISPP in the real networks.

Corresponding simulations and results (running time, total weight adjustment as well as path length) are shown in Chapter 4, 5, 6 respectively. There are two metrics that we concentrate on: running time and the adjustment of link weights. They can provide some insight into the effectiveness of our algorithms. We can draw a few inferences from the study of all experimental results.

For **the split path method**, this approach works most effectively to solve ISPP-S in small-sized directed graphs. Its running time is short, and the overall weight adjustment is modest. After modification, this method can be used to handle ISPP-M. Sp still performs well in directed graphs with few nodes, running fastest and adjusting fewer weights. But this method is not applicable to solving ISPP-T. So we utilize all violated constraints instead of choosing constraints by splitting target path. Sp becomes another method, the all constraints method (Ac). Ac can take the shortest time to solve ISPP-T in directed graphs and the value of changing the weight is the smallest. This method is also feasible in undirected graphs, and similar results can be obtained.

The reason why Sp is suitable for small networks is that, it uses more constraints during each iteration than the other methods. In relatively small-sized graphs, the number of constraints remains within a reasonable range and the complexity of optimization is still lower. The optimization time t_o will not be affected significantly. With this premise, more constraints make it possible to obtain the optimal solution in only one iteration, which decreases the total running time.

For **the limit constraint method**, it seems to be the fastest of our general research methods to solve ISPP-S and ISPP-M in large-sized directed graphs. Its running time is generally the shortest and significantly shorter than other methods in graphs with relatively complex structures (like configuration graph and ER graph). It is relatively effective to solve ISPP-T. The results in undirected graphs are very similar to those in directed graphs. But the total weight adjustment of Lc is larger than that of other methods, which is related to the number of iterations.

Lc is more effective in large-sized complex networks, because the number of constraints in each iteration is fixed and smaller. In every iteration of Lc, the old constraints are totally replaced by new ones. While other methods merge new constraints with old ones, the size of the active constraint set will grow after each iteration.

In summary, for the ISPP with ***relatively few constraints*** (ISPP in small-sized networks with simple structures or ISPP-S), the number of iterations $iter$ has a greater impact on the total running time. Increasing the number of active constraints in a reasonable range allows the optimal solution to be obtained in few iterations, so the total running time will be short and the adjustment of link weight is relatively small. So **Sp** is more efficient. For the ISPP with ***relatively more constraints*** (ISPP in large-sized networks with complicated structures or ISPP-M), the entire running time is more heavily influenced by the time of optimization t_o. We can restrict the number of active constraints in each iteration to obtain the optimal solution rapidly. So **Lc** is more effective. However, this approach would come at the cost of increasing the weight adjustment. For the ISPP with ***a large number of constraints*** (ISPP-T), Ac is more efficient in directed graphs and small-sized undirected graphs. When solving ISPP-T in large undirected graphs, Cg shows its advantages.

In addition, when Lc and Sp are applied to the empirical networks, they are effective for solving ISPP-S (with a large number of links and fewer con·‘straints). This type of problem is common in the real-world complex systems.

7.2. FUTURE WORK

Although Lc and Sp are efficient in most cases, they perform worse than Cg when solving ISPP-T in large-sized undirected graphs. Additionally, when dealing with complicated networks, Lc's overall weight adjustment has a tendency to be bigger. So, the following future work is suggested:

- Further improve the algorithm to enhance the speed of solving ISPP-T in undirected graphs.
- Find the balance between running time and the total weight adjustment such that minimal weight adjustment can be achieved in a relatively short running time.

Bibliography

[1] M. Newman, *Networks: An Introduction*. Oxford University Press, 03 2010.

[2] D. Burton, “On the inverse shortest path problem,” 1993.

[3] J. Zhang, Z. Ma, and C. Yang, “A column generation method for inverse shortest path problems,” *Zeitschrift für Operations Research*, vol. 41, no. 3, pp. 347–358, 1995.

[4] S. Xu and J. Zhang *Japan Journal of Industrial and Applied Mathematics*, vol. 12, pp. 47–59, feb 1995.

[5] J. Zhou, F. Yang, and K. Wang, “An inverse shortest path problem on an uncertain graph,” *Journal of Networks*, vol. 9, pp. 2353–2359, 09 2014.

[6] M. Call, *Inverse shortest path routing problems in the design of ip networks*. PhD thesis, Linköping University Electronic Press, 2010.

[7] P. Van Mieghem, *Graph spectra for complex networks*. Cambridge University Press, 2010.

[8] J. M. Hernández and P. Van Mieghem, “Classification of graph metrics,” *Delft University of Technology: Mekelweg, The Netherlands*, pp. 1–20, 2011.

[9] P. Zhang and G. Chartrand, *Introduction to graph theory*. Tata McGraw-Hill, 2006.

[10] P. Erdős, A. Rényi, *et al.*, “On the evolution of random graphs,” *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[11] M. Newman, “The configuration model,” in *Networks*, Oxford University Press, 07 2018.

[12] D. Burton and P. L. Toint, “On an instance of the inverse shortest paths problem,” *Mathematical Programming*, vol. 53, no. 1, pp. 45–61, 1992.

[13] A. Faragó, Áron Szentesi, and B. Szviatovszki, “Inverse optimization in high-speed networks,” *Discrete Applied Mathematics*, vol. 129, no. 1, pp. 83–98, 2003. Algorithmic Aspects of Communication.

[14] M. Karimi, M. Aman, and A. Dolati, “Inverse multi-objective shortest path problem under the bottleneck type weighted hamming distance,” in *Topics in Theoretical Computer Science* (M. R. Mousavi and J. Sgall, eds.), (Cham), pp. 34–40, Springer International Publishing, 2017.

[15] D. Goldfarb and A. U. Idnani, “A numerically stable dual method for solving strictly convex quadratic programs,” *Mathematical Programming*, vol. 27, pp. 1–33, 1983.

[16] S. Misra, L. Roald, and Y. Ng, “Learning for constrained optimization: Identifying optimal active constraint sets,” 2018.

[17] G. Nemhauser, “Column generation for linear and integer programming,” *Documenta Mathematica*, 01 2012.

[18] G. B. Dantzig, *Origins of the Simplex Method*, p. 141–151. New York, NY, USA: Association for Computing Machinery, 1990.

[19] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[20] M. Barbehenn, “A note on the complexity of dijkstra’s algorithm for graphs with weighted vertices,” *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263–, 1998.

[21] J. A. Kelner and D. A. Spielman, “A randomized polynomial-time simplex algorithm for linear programming,” in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pp. 51–60, 2006.

[22] B. C. Van Wijk, C. J. Stam, and A. Daffertshofer, “Comparing brain networks of different size and connectivity density using graph theory,” *PloS one*, vol. 5, no. 10, p. e13701, 2010.

[23] B. Fortz and M. Thorup, “Internet traffic engineering by optimizing ospf weights,” in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 2, pp. 519–528 vol.2, 2000.

[24] P. Van Mieghem, “A tree realization of a distance matrix: the inverse shortest path problem with a demand matrix generated by a tree,” *Delft University of Technology*, vol. 1, no. 12, 2021.

[25] J. B. Singh, “Investigation of bellman-ford algorithm, dijkstra’s algorithm for suitability of sp,” *International Journal of Engineering Development and Research*, vol. 6, pp. 755–758, 2018.

[26] R. Bellman, “On a routing problem,” *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[27] V. Colizza, R. Pastor-Satorras, and A. Vespignani, “Reaction–diffusion processes and metapopulation models in heterogeneous networks,” *Nature Physics*, vol. 3, no. 4, pp. 276–282, 2007.

[28] H. Jeong, S. Mason, A. Barabasi, and Z. Oltvai, “Lethality and centrality in protein networks,” *arXiv preprint cond-mat/0105306*, 2001.

[29] R. A. Rossi and N. K. Ahmed, “The network data repository with interactive graph analytics and visualization,” in *AAAI*, 2015.

[30] B.-Q. Li, T. Huang, L. Liu, Y.-D. Cai, and K.-C. Chou, “Identification of colorectal cancer related genes with mrmr and shortest path in protein-protein interaction network,” *PloS one*, vol. 7, no. 4, p. e33393, 2012.

A

APPENDIX

This appendix presents all involving notations in graph theory and abbreviations in the thesis.

A.1. INVOLVING NOTATIONS IN GRAPH THEORY

Table A.1: Notations in graph theory

Mathematical notation	Purpose
G	graph or network
$\mathcal{N}$	node set
$\mathcal{L}$	link set
N	number of nodes
L	number of links
A	adjacency matrix
a	element of adjacency matrix
W	link weight matrix
w	link weight
P	path
T	the shortest path tree
r	root of the tree
d	degree
$Pr(d=k)$	degree distribution
B	betweenness
γ	power law exponent of degree distribution
$G(N,p)$	ER graph
$G(N,m_0,m_1)$	BA graph
$G(N,n_0 \times n_1)$	2D lattice graph
$G(N,[1,K])$	Configuration graph

A.2. ABBREVIATIONS IN THE THESIS

Table A.2: List of abbreviations

Abbreviations	Full name
ISPP	Inverse shortest path problem
ISPP-S	Inverse shortest path problem with a single prescribed path
ISPP-M	Inverse shortest path problem with multiple prescribed paths
ISPP-T	Inverse shortest path problem with target paths in a spanning tree
ER graph	Erdős–Rényi graph
BA graph	Barabási–Albert graph
GI	Goldfarb-Idnani
LP	Linear programming
Qp	Quadratic programming method
Cg	Column generation method
Sp	Split path method
Lc	Limit constraints method
Ac	All constraints method

B
Appendix

This appendix presents the flowchart of the framework for solving ISPP, the quadratic programming method (Qp) and the column generation method (Cg). All of them specific show how to solve ISPP.

B.1. FLOWCHART OF QP

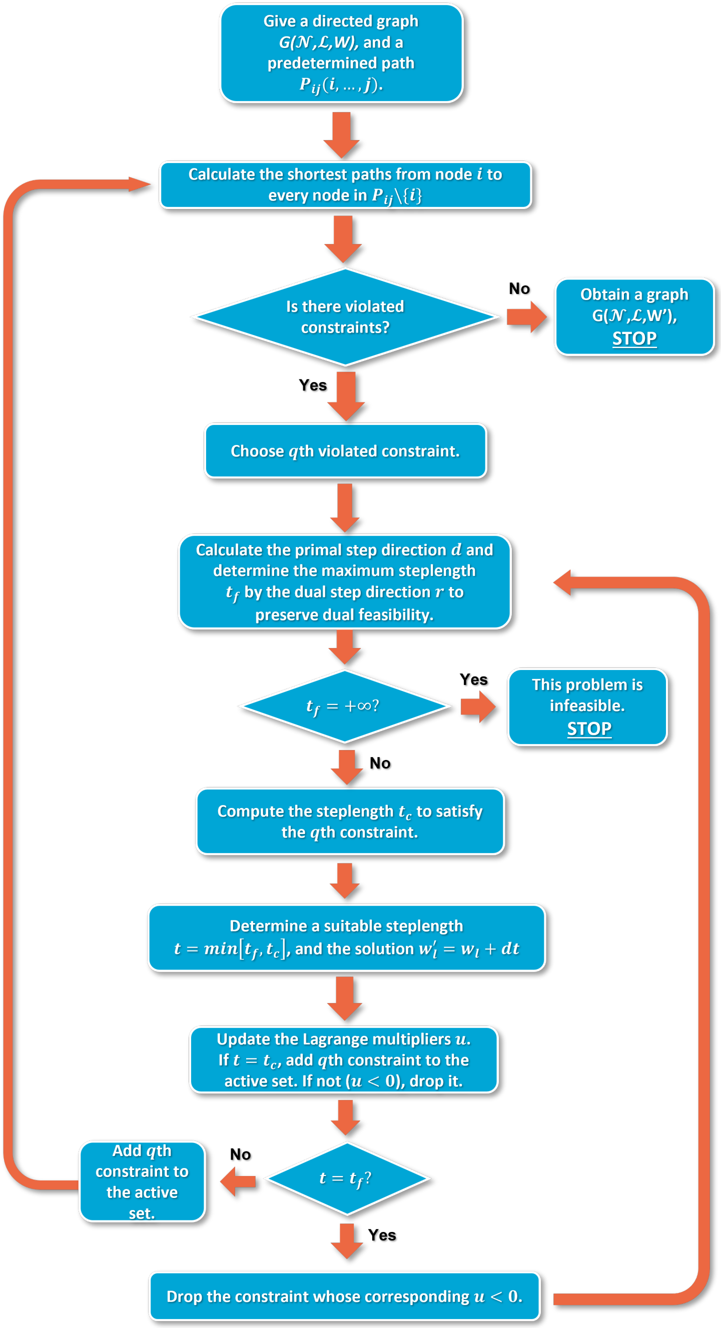

Give a directed graph $G(\mathcal{N},\mathcal{L},W)$, and a predetermined path $P_{ij}(i,\ldots,j)$.
Calculate the shortest paths from node i to every node in $P_{ij}\backslash\{i\}$
Is there violated constraints?
No
Obtain a graph $G(\mathcal{N},\mathcal{L},W')$, STOP
Yes
Choose qth violated constraint.
Calculate the primal step direction d and determine the maximum steplength t_f by the dual step direction r to preserve dual feasibility.
$t_f = +\infty$?
Yes
This problem is infeasible. STOP
No
Compute the steplength t_c to satisfy the qth constraint.
Determine a suitable steplength $t = min[t_f, t_c]$, and the solution $w_l' = w_l + dt$
Update the Lagrange multipliers u. If $t = t_c$, add qth constraint to the active set. If not ($u < 0$), drop it.
$t = t_f$?
No
Add qth constraint to the active set.
Yes
Drop the constraint whose corresponding $u < 0$.

Figure B.1: The steps of QP

B.2. FLOWCHART OF CG

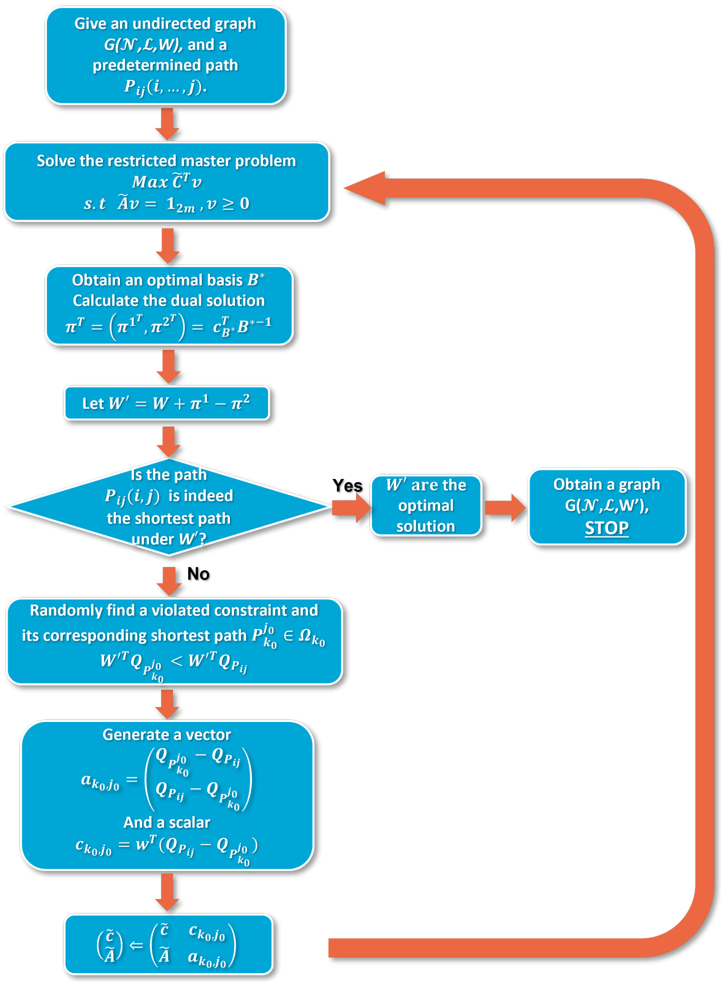

Give an undirected graph $G(\mathcal{N},\mathcal{L},W)$, and a predetermined path $P_{ij}(i,\dots,j)$.
Solve the restricted master problem $Max\ \tilde{C}^T v$ $s.t\ \ \tilde{A}v = 1_{2m}, v \geq 0$
Obtain an optimal basis B^* Calculate the dual solution $\pi^T = \left(\pi^{1^T}, \pi^{2^T}\right) = c_{B^*}^T B^{*-1}$
Let $W' = W + \pi^1 - \pi^2$
Is the path $P_{ij}(i,j)$ is indeed the shortest path under W'?
Yes
W' are the optimal solution
Obtain a graph G($\mathcal{N}$,$\mathcal{L}$,W'), STOP
No
Randomly find a violated constraint and its corresponding shortest path $P_{k_0}^{j_0} \in \Omega_{k_0}$ $W'^T Q_{P_{k_0}^{j_0}} < W'^T Q_{P_{ij}}$
Generate a vector $a_{k_0,j_0} = \begin{pmatrix} Q_{P_{k_0}^{j_0}} - Q_{P_{ij}} \\ Q_{P_{ij}} - Q_{P_{k_0}^{j_0}} \end{pmatrix}$ And a scalar $c_{k_0,j_0} = w^T(Q_{P_{ij}} - Q_{P_{k_0}^{j_0}})$
$\begin{pmatrix} \tilde{c} \\ \tilde{A} \end{pmatrix} \Leftarrow \begin{pmatrix} \tilde{c} & c_{k_0,j_0} \\ \tilde{A} & a_{k_0,j_0} \end{pmatrix}$

Figure B.2: The steps of Cg

B.3. ALGORITHMIC FRAMEWORK FOR SOLVING ISPP

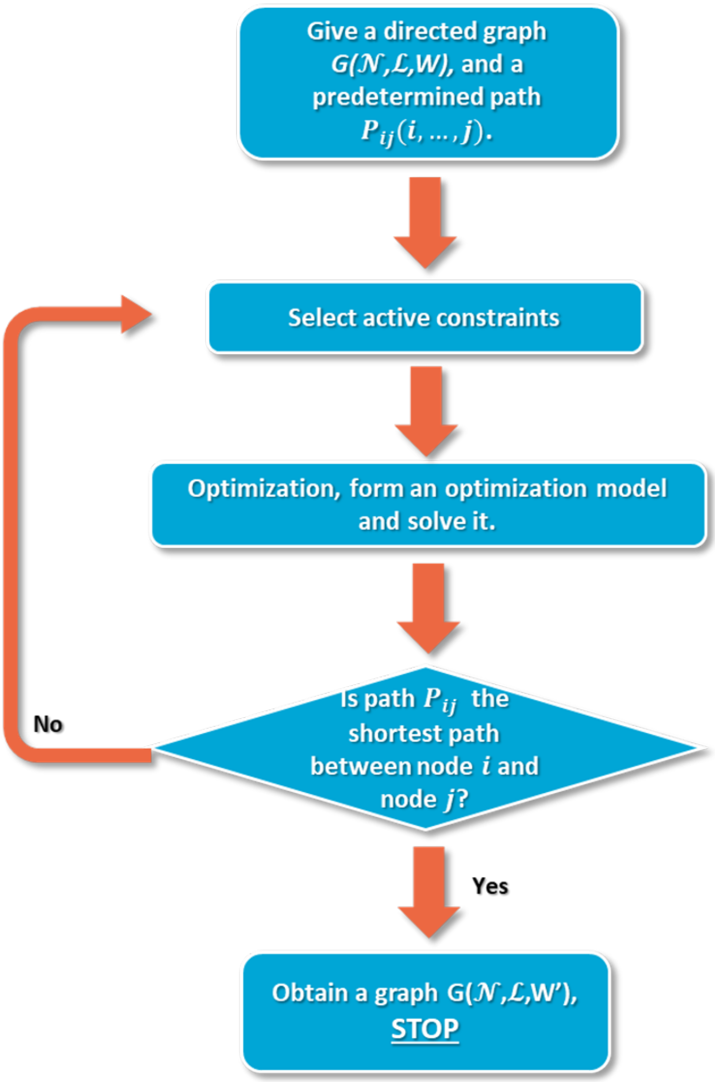

Give a directed graph $G(\mathcal{N},\mathcal{L},W)$, and a predetermined path $P_{ij}(i,\dots,j)$.
Select active constraints
Optimization, form an optimization model and solve it.
Is path P_{ij} the shortest path between node i and node j?
No
Yes
Obtain a graph $G(\mathcal{N},\mathcal{L},W')$, STOP

Figure B.3: Algorithmic framework for solving ISPP

C
APPENDIX

This appendix displays the accuracy of Sp and former Lc in different types of graphs.

C.1. ACCURACY OF ER GRAPH

The connection possibility of ER graphs p equals to $2log(N)/N$. N is the number of nodes.

Table C.1: Accuracy after on iteration in ER graph

Method \ N	10	20	50	100	200	500	1000
Split path	0.9764	0.9385	0.9075	0.8714	0.8470	0.8242	0.6477
Former Limit constraints	0.9086	0.8777	0.7089	0.5516	0.3863	0.2228	0.1171

C.2. ACCURACY OF 2D LATTICE GRAPH

The structure of 2D lattice graphs is $G(10,5\times 2)$; $G(20,5\times 4)$; $G(50,10\times 5)$; $G(100,10\times 10)$; $G(200,20\times 10)$; $G(500,25\times 20)$; $G(1000,40\times 25)$ respectively. N is the number of nodes.

Table C.2: Accuracy after on iteration in 2D lattice graph

Method \ N	10	20	50	100	200	500	1000
Split path	1	0.9758	0.9116	0.8195	0.7063	0.5515	0.4208
Former Limit constraints	0.9993	0.9831	0.9082	0.7781	0.5867	0.3606	0.2487

C.3. ACCURACY OF CONFIGURATION GRAPH

The degree range of configuration graphs is $[1, 10]$. N is the number of nodes.

Table C.3: Accuracy after on iteration in configuration graph

Method \ N	10	20	50	100	200	500	1000
Split path	0.9769	0.9647	0.9508	0.9398	0.9337	0.9081	0.8989
Former Limit constraints	0.8684	0.7532	0.5542	0.4413	0.3614	0.2879	0.2534

D

APPENDIX

D.1. CONSTRUCTION OF BA GRAPH

```
function [G]=BA(N,m0,m1)
A = sparse(N,N);
for i=1:m0
    for j= (i+1):m0
        A(i,j)= round(rand());
    end
end
for new = m0+1:N
    Degree = sum(A(1:new-1,1:new-1));
    DegreeInterval(1) = Degree(1);
    for i=2:new-1
        DegreeInterval(i) = Degree(i)+DegreeInterval(i-1);
    end
    AllDegree = sum(sum(A(1:new-1,1:new-1)));
    for i = 1:m1
        while 1
            RandDegree = fix(AllDegree*rand()+1);
            Ans = find(RandDegree <= DegreeInterval(1:new-1));
            old = Ans(1);
            if A(new,old) == 0
                A(new,old) = 1;
                break;
            end
        end
    end
end
Degree = sum(A);
UniDegree = unique(Degree);
for i = 1:length(UniDegree)
```

```
    DegreeNum(i) = length(find(Degree==UniDegree(i)));
end
G=graph(A,'upper');
m=numedges(G);
f=rand(1,m);
G.Edges.Weight=f.';
end
```

D.2. Construction of ER graph

```
function [G]=ER(N,p)
A = zeros(N,N);
C = zeros(1,N);
for i=1:N
    for j=i+1:N
        b=rand();
        if b<=p
            A(i,j)=1;
        else
            A(i,j)=0;
            C(1,i)=1;C(1,j)=1;
        end
    end
end
G=graph(A,'upper');
m=numedges(G);
f=rand(1,m);
G.Edges.Weight=f.';
end
```

D.3. Construction of 2D lattice graph

```
function [G]=lattice(N)
if N==10
    n=5;
elseif N==20
    n=5;
elseif N==50
    n=10;
elseif N==200
    n=20;
elseif N==500
    n=25;
elseif N==1000
    n=40;
elseif N==100
    n=10;
end
A = delsq(numgrid('S',n+2));
G = graph(A,'omitselfloops');
```

```
A=full(adjacency(G));
A=A(1:N,1:N);
G=graph(triu(A),'upper');
m=numedges(G);
f=rand(1,m);
G.Edges.Weight=f.';
```

D.4. CONSTRUCTION OF CONFIGURATION GRAPH

```
 function [G]=configurationD(N,deg)
Node_ofDegree=randi(deg,1,N);
d=sum(Node_ofDegree);
if mod(d,2)==0
else
    [~,Q]=max(Node_ofDegree);
    Node_ofDegree(Q)= Node_ofDegree(Q)-1;
    d=d-1;
end
s=[];
t=[];
while 1
    x=find(Node_ofDegree>=1);
    u=randi(length(x),2,1);
   B=x(u(1));
   A=x(u(2));
s=[s,B];
t=[t,A];
if A~=B
Node_ofDegree(A)=Node_ofDegree(A)-1;
Node_ofDegree(B)=Node_ofDegree(B)-1;
else
   Node_ofDegree(A)=Node_ofDegree(A)-1;
end
if Node_ofDegree==zeros(1,length(Node_ofDegree))
    break
end
end

G=graph(s,t);
A=adjacency(G);
m=numedges(G);
f=rand(1,m);
G.Edges.Weight=f.';
 end
```

E
Appendix

E.1. The column generation method

```
function [G,count]=CG(G,MP)
h=0;
CX=G.Edges.Weight;
m=numedges(G);
e=table2array(G.Edges);
ee=0.00001;
PE=[];
w=G.Edges.Weight;
count=0;
for i=1:length(e(:,1))
    PE=[PE,findedge(G,e(i,1),e(i,2))];
end
A_1=sparse([zeros(m);-eye(m)]);
A_2=sparse([eye(m);zeros(m)]);
A_3=sparse([zeros(m);eye(m)]);
A_tilde=[A_1,A_2,A_3];
y_1=sparse(zeros(m,2*m).');
y_2=sparse(ones(m,2*m).');
y_3=sparse(ones(m,2*m).');
Y=[y_1,y_2,y_3].';
c_1=sparse(-w.');
c_2=sparse(zeros(1,m));
c_3=sparse(zeros(1,m));
c_tilde=[c_1,c_2,c_3];
% constraint shortest path
flag=1;
while flag==1
    count=count+1;
    f=ones(2*m,1);
    a=-A_tilde.';
    b=-c_tilde;
    aeq=[];
    beq=[];
    lb=zeros(2*m,1);
    ub=[inf(1,m),w.'].';
    options = optimoptions('linprog','Display','none');
    pi=linprog(f,a,b,aeq,beq,lb,ub,options).';
    if isempty(pi)
        G.Edges.Weight=w;
        break
    else
        pi1=pi(1,1:m);
        pi2=pi(1,m+1:2*m);
        w_tilde=(w.'+pi1-pi2).'; %w_tilde=-cj
        G.Edges.Weight=w_tilde;


        DD1=[];
        D1=[];
        for i=1:length(MP)
            P=cell2mat(MP(i));
            pe=[];
            for j=2:length(P)

                pe=[pe,findedge(G,P(j-1),P(j))];
            end
            dd1=sum(G.Edges.Weight(pe));
            DD1=[DD1,dd1];
            try
                [~,d1,omegae]=shortestpath(G,P(1),P(end));
            catch
                h=1;
                break
            end

            D1=[D1,d1];
            if d1<dd1
                peo=pe;
                omegaeo=omegae;
            end
        end
        if D1==DD1
            G.Edges.Weight=w_tilde;
            break
        else
            G.Edges.Weight=w;

            QP1=ismember(PE,peo).';
            u1=ismember(PE,omegaeo).';%u is the QP_kj
            QP_1=u1;
            c1=w.'*(QP1-QP_1);
            c_tilde=[c_tilde,c1];
            a1=[QP_1-QP1;QP1-QP_1];
            A_tilde=[A_tilde,a1];
            flag=1;
        end
    end
    if h==1
        G.Edges.Weight=inf(m,1);
        break
    end
end
end
```

E.2. THE QUADRATIC PROGRAMMING METHOD

```
function [G]=QP(G,MP)
v=0;
u=0;
N1=[];
A_ind=[];
A=[];
r=0;
f=0;
epsilon=0.0001;
m=numedges(G);
c=G.Edges.Weight;
Aip=[];
Ain=[];
flag_1=1;
count=0;
h=0;
%% step 1
while flag_1==1
    flag_1=0;
    CIY=[];
    E=[];
    ipv=[];
    inv=[];
    c=real(c);
    G.Edges.Weight=c;
    count1=0;
    if ~isempty(find(isnan(c)==1))
        h=1;
        break
    else
        for iu=1:length(MP)
            P=cell2mat(MP(iu));
            PE=[];
            for i=2:length(P)
                PE=[PE,findedge(G,P(i-1),P(i))];
            end
            for node=2:length(P)
                try
                    [~,~,edgepath]=shortestpath(G,P(1),P(node),"Method","auto");
                catch
                    h=1;
                    break
                end
                [~,i_index_sp_vertex,i_index_p_vertex] = intersect(edgepath,
                PE(1:node-1));
                IP=edgepath;
                IP(i_index_sp_vertex)=[];
                IPV=0;
                for i=1:length(IP)
                    IPV=IPV+G.Edges.Weight(IP(i));
                end
                IN=PE(1:node-1);
                IN(i_index_p_vertex)=[];
                INV=0;
                for i=1:length(IN)
                    INV=INV+G.Edges.Weight(IN(i));
                end
                e=IPV-INV;
                E=[E,e];
                count1=count1+1;
                ipv{count1}=IP;
                inv{count1}=IN;
            end
            if h==1
                break
            end
        end
        if h==1
            break
        end
        CIY_ind = find(c<-epsilon);
        for vo=1:length(CIY_ind)
            IY=zeros(m,1);
            IY(CIY_ind(vo))=c(CIY_ind(vo));
            CIY=[CIY,IY];
        end

        E=[E,c.'];
        uE=find(E>-epsilon&E<0);
        E(uE)=0;
        uc=find(c>-epsilon&c<0);
        c(uc)=0;
        ip=[ipv,num2cell(1:m)];
        in=[inv,cell(1,m)];
        %% step 2
        if E<-epsilon==zeros(1,length(E))
            G.Edges.Weight=c;
            break
        else
            [IqV,IqI]=min(E);
            nq=zeros(m,1);
            if IqI<=count1
                Iqp=cell2mat(ip(IqI));
                Iqn=cell2mat(in(IqI));
                nq(Iqp)=1;
                nq(Iqn)=-1;
                Eq = IqV;
                Iq=nq.*c;
            else
                Iqp=IqI-count1;
                nq(Iqp)=1;
                Iqn =[];
                Iq=zeros(m,1);
                Eq = IqV;
                Iq=nq.*c;
            end
        end
        if v==0
            alpha=sqrt(length(Iqp)+length(Iqn));
            % go to step5
            flag_1=5;
        else
            u=[u;0];
            %% step 3a
            N1=[N1,nQ];
            if v==1
                R=alpha;
                %go to the step 4
            else
                alpha=sqrt(norm(nq,2)^2-norm(z,2)^2);

                R=[R,z;zeros(1,size(R,2)),alpha];
            end
            flag_1=4;
            %% step 4
            while flag_1==4
                y=[];
                for j = 1:v
                    Ijp=cell2mat(Aip(j));
                    Ijn=cell2mat(Ain(j));
                y(j)=length(intersect(Ijp,Iqp))+length(intersect(Ijn,Iqn))
                -length(intersect(Ijp,Iqn))-length(intersect(Ijn,Iqp));
            end
            y=y.';
            if isempty(y)
                break
            else
                z=pinv(R.')*y;
                r=pinv(R)*z;
                alpha=sqrt(norm(nq)^2-norm(z)^2);
                flag_1=5;
            end
        end
    end
    %% step 5
    while flag_1==5
        flag_1=0;
        S=[];
        tf1=[];
        for xj=1:v
            if r(xj)>0
                S=[S,j];
                tf1=[tf1,u(xj)/r(xj)];
            end
        end
        if isempty(S)
            tf = inf;
        else
            [tf,l]=min(tf1);
        end
        %% step 6
        if alpha<= epsilon
            %% step 7b
            if tf==inf
                break
            else
                u = u+tf.*[-r;1];
                A(:,l)=[];
                A_ind(l)=[];
                Aip(l)=[];
                Ain(l)=[];
                v=v-1;
                u(l)=[];
                %go to step 3b
                %% step 3b
                N1(:,l)=[];
                R(:,l)=[];
                T=R(l:end,l:end);% Givens rotation
                [~,n1]=size(T);
```

```
                    R2=Givens_rotation(T);
                    R=[R(1:l-1,:);zeros(n1,l-1),R2];
                    y=[];
                    for j = 1:v
                        Ijp=cell2mat(Aip(j));
                        Ijn=cell2mat(Ain(j));
                        y(j)=length(intersect(Ijp,Iqp))+length(intersect(Ijn,Iqn))
                        -length(intersect(Ijp,Iqn))-length(intersect(Ijn,Iqp));
                    end
                    y=y.';
                    if isempty(y)
                        break
                    else
                        z=pinv(R.')*y;
                        r=pinv(R)*z;
                        alpha=sqrt(norm(nq)^2-norm(z)^2);
                        flag_1=5;
                    end
                end
            end
            if flag_1==5
            else
                %lemma3
                Y=CIY_ind;
                X=[cell2mat(Aip),cell2mat(Ain)];
                X=setdiff(X,Y);
                X=unique(X);
                V_allelements=[];
                V_matrix=[];
                islandV_index=[];
                for col_index=1:size(A,2)
                    a_island=[cell2mat(Aip),cell2mat(Ain)];
                    if length(a_island)>1
                        V_allelements=[V_allelements;a_island];
                        V_matrix=[V_matrix,A(:,col_index)];
                        islandV_index=[islandV_index,col_index];
                    end
                end
                X=unique(V_allelements);
                X=setdiff(X,Y);
                d=zeros(m,1);
                for i=1:m
                    if find(Iqp==i,1)%a(i) is the index of edge
                        delta1=1;
                    else
                        delta1=0;
                    end
                    if find(Iqn==i,1)%a(i) is the index of edge
                        delta2=1;
                    else
                        delta2=0;
                    end
                    if find(X==i,1)%a(i) is the index of edge
                        delta3=1;
                    else
                        delta3=0;
                    end
                    if delta3==0
                        d(i)=delta1-delta2;
                    elseif ~isempty(V_matrix)
                        Ip_i_index_in_V=find(V_matrix(i,:)>0);
                        Ip_i = islandV_index(Ip_i_index_in_V);
                        In_i_index_in_V=find(V_matrix(i,:)<0);
                        In_i =[islandV_index(In_i_index_in_V)];
                        d(i)=delta1-delta2+delta3*(sum(r(In_i))-sum(r(Ip_i)));
                    else
                        d(i)=delta1-delta2;
                    end
                end
                if ~isempty(V_matrix)
                    if length(find(nq~=0))==1
                        q=find(nq~=0);
                        Ip_q_index_in_V=find(V_matrix(q,:)>0);
                        Ip_q = islandV_index(Ip_q_index_in_V);
                        In_q_index_in_V=find(V_matrix(q,:)<0);
                        In_q = islandV_index(In_q_index_in_V);
                        dTnq=1+sum(r(In_q))-sum(r(Ip_q));
                    else
                        dTnq=0;
                        for i = 1:length(Iqp)
                            Ip_q_index_in_V=find(V_matrix(Iqp(i),:)>0);
                            Ip_q = islandV_index(Ip_q_index_in_V);
                            In_q_index_in_V=find(V_matrix(Iqp(i),:)<0);
                            In_q = islandV_index(In_q_index_in_V);
                            dTnq=dTnq+(1+sum(r(In_q))-sum(r(Ip_q)) );
                        end
                        for i = 1:length(Iqn)
                            Ip_q_index_in_V=find(V_matrix(Iqn(i),:)>0);
                            Ip_q = islandV_index(Ip_q_index_in_V);
                            In_q_index_in_V=find(V_matrix(Iqn(i),:)<0);
                            In_q = islandV_index(In_q_index_in_V);
                            dTnq=dTnq+(1+sum(r(Ip_q))-sum(r(In_q)) );
                        end
                    end
                else
                    if length(nq)==1
                        dTnq=1;
                    else
                    dTnq=length(Iqp)+length(Iqn);
                end
            end
            tc=-Eq/dTnq;
            %% step 7a
            t=min([tf,tc]);
            c=c+t*d;
            f=f+t*(0.5*t+u(size(A_ind,2)+1))*dTnq;
            if v==0
                u=u+t;
            else
                u=u+t.*[-r;1];%
            end
            if t==tc
                A=[A,Iq];
                A_ind=[A_ind,IqI];
                Aip=[Aip,ip(IqI)];
                Ain=[Ain,in(IqI)];
                v=v+1;
                nQ=nq;
                flag_1=1;
                %go to step 1
            else
                A(:,l)=[];
                A_ind(l)=[];
                Aip(l)=[];
                Ain(l)=[];
                v=v-1;
                u(l)=[];
                %go to step 3b
                %% step 3b
                N1(:,l)=[];
                R(:,l)=[];
                T=R(l:end,l:end);% Givens rotation
                [~,n1]=size(T);
                R2=Givens_rotation(T);
                R=[R(1:l-1,:);zeros(n1,l-1),R2];
                %% step 4
                y=[];
                for j = 1:v
                    Ijp=cell2mat(Aip(j));
                    Ijn=cell2mat(Ain(j));
                    y(j)=length(intersect(Ijp,Iqp))+length(intersect(Ijn,Iqn))
                    -length(intersect(Ijp,Iqn))-length(intersect(Ijn,Iqp));
                end
                y=y.';
                if isempty(y)
                    break
                else
```

```
                        z=pinv(R.')*y;
                        r=pinv(R)*z;
                        alpha=sqrt(norm(nq)^2-norm(z)^2);
                        flag_1=5;
                    end
                end
            end
        end
    end
end

if h==1
    G.Edges.Weight=inf(m,1);
end
end
```

E.3. THE SPLIT PATH METHOD

```
function [G,count]=SP(G,MP)
h=0;
m=numedges(G);
pe=[];
flag=1;
count=0;
CX=G.Edges.Weight;
epsilon=0.0000001;
while flag==1
    A1=[];
    B1=[];
    c=G.Edges.Weight.';
    for v=1:length(MP)
        P=cell2mat(MP(v));
        pe=[];
        for i=2:length(P)
            pe=[pe,findedge(G,P(i-1),P(i))];
        end
        cd1=sum(G.Edges.Weight(pe));
        try
            [~,cd2,PE]=shortestpath(G,P(1),P(end));
        catch
            h=1;
            break
        end
        if cd1==cd2
        else
            if length(pe)==1
                ap=zeros(1,m);
                an=zeros(1,m);
                ap(pe)=1;
                an(PE)=-1;
                A=ap+an;
                B=cd2-cd1;
                A1=[A1;A];
                B1=[B1;B];
            end
            if length(pe)>=2
                for u=1:length(pe)
                    [~,~,Pe1]=shortestpath(G,P(1),P(u+1));
                    sp=zeros(1,m);
                    sn=zeros(1,m);
                    sp(pe(:,1:u))=1;
                    sn(Pe1)=-1;
                    S=sp+sn;
                    bp=c(Pe1);
                    bn=-c(pe(:,1:u));
                    B=sum(bp)+sum(bn);
                    A1=[A1;S];
                    B1=[B1;B];
                    [~,~,Pe2]=shortestpath(G,P(u),P(end));
                    sp=zeros(1,m);
                    sn=zeros(1,m);
                    sp(pe(:,u:end))=1;
                    sn(Pe2)=-1;
                    S=sp+sn;
                    bp=c(Pe2);
                    bn=-c(pe(:,u:end));
                    B=sum(bp)+sum(bn);
                    A1=[A1;S];
                    B1=[B1;B];
                end
            end
        end
    end
    uE=find(B1>-epsilon&B1<0);
    B1(uE)=0;
    if B1<-epsilon==zeros(1,length(B1))
        G.Edges.Weight=c.';
        break
    else
        uu=find(c<0);
        c(uu)=0;
        count=count+1;
        f=ones(2*m,1).';
        A2=[A1,-A1];
        b2=B1;
        lb=zeros(2*m,1);
        ub=[inf(1,m),c].';
        Aeq=[];
        beq=[];
        options = optimoptions('linprog','display','none');
        x=linprog(f,A2,b2,Aeq,beq,lb,ub,options);


        if isempty(x)
            G.Edges.Weight=c.';
        else
            X=x(1:m,:)-x(m+1:end,:)+c.';
            G.Edges.Weight=X;
        end
        DD1=[];
        D1=[];
        for i=1:length(MP)
            P=cell2mat(MP(i));
            pe=[];
            for j=2:length(P)
                pe=[pe,findedge(G,P(j-1),P(j))];
            end
            dd1=sum(G.Edges.Weight(pe));
            DD1=[DD1,dd1];
            try
                [~,d1]=shortestpath(G,P(1),P(end));
            catch
                h=1;
                break
            end
            D1=[D1,d1];
        end

        if D1==DD1
            break
        end
        if h==1
            G.Edges.Weight=inf(m,1);
            break
        end
    end
end
end
```

E.4. THE LIMIT CONSTRAINTS METHOD

```
function [G,count]=LC(G,MP)
h=0;
CX=G.Edges.Weight;
m=numedges(G);
count=0;
flag=1;

A1=[];
```

```
B1=[];
epsilon=0.000001;
while flag==1
    EP=[];
    Pe=[];
    E=[];
    COUNT=0;
    c=G.Edges.Weight.';
    for u=1:length(MP)
        P=cell2mat(MP(u));
        pe1=[];
        for v=2:length(P)
            pe1=[pe1,findedge(G,P(v-1),P(v))];
        end
        for node=2:length(P)
            pe=pe1(1:node-1);
            cd1=sum(c(pe));
            try
                [~,cd2,edgepath]=shortestpath(G,P(1),P(node),'Method','auto');
            catch
                h=1;
                break
            end

            e=cd2-cd1;
            if e<0
                COUNT=COUNT+1;
                E=[E,e];
                Pe{COUNT}=pe;
                EP{COUNT}=edgepath;
            end
        end
    end
    if isempty(E)
        break
    else
        [Eq,IqI]=min(E);
        ap=zeros(1,m);
        an=zeros(1,m);
        ap(cell2mat(Pe(IqI)))=1;
        an(cell2mat(EP(IqI)))=-1;
        A=ap+an;
        A1=A;
        B1=Eq;

    end
    uu=find(c<0);
    c(uu)=0;

    if B1>-epsilon&B1<0
        G.Edges.Weight=c.';
        break
    else

        f=ones(2*m,1).';
        A2=[A1,-A1];
        b2=B1;
        lb=zeros(2*m,1);
        ub=[inf(1,m),c].';
        Aeq=[];
        beq=[];
        options = optimoptions('linprog','Algorithm','interior-point');
        x=linprog(f,A2,b2,Aeq,beq,lb,ub,options);
        count=count+1;
        if isempty(x)
            G.Edges.Weight=c.';
        else
            X=x(1:m,:)-x(m+1:end,:)+c.';
            G.Edges.Weight=X;
        end


        DD1=[];
        D1=[];
        for i=1:length(MP)
            P=cell2mat(MP(i));
            pe=[];
            for j=2:length(P)
                pe=[pe,findedge(G,P(j-1),P(j))];
            end
            dd1=sum(G.Edges.Weight(pe));
            DD1=[DD1,dd1];
            try
                [~,d1]=shortestpath(G,P(1),P(end));
            catch
                h=1;
                break
            end
            D1=[D1,d1];
        end

        if D1==DD1
            e4=count;
            break
        end
        if h==1
            G.Edges.Weight=inf(m,1);
            break
        end
    end
end
end
```