中国科学院大学

University of Chinese Academy of Sciences

硕士学位论文

面向时序网络拥塞优化问题的对偶单纯形算法研究

作者姓名：张跃

指导教师：戴彧虹 研究员

中国科学院数学与系统科学研究院

学位类别：理学硕士

学科专业：计算数学

培养单位：中国科学院数学与系统科学研究院

2022 年 6 月

Research on Dual Simplex Algorithm for Time-Based Network Congestion Optimization Problem

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Natural Science

in Computational Mathematics

By

Yue Zhang

Supervisor: Professor Yuhong Dai

Academy of Mathematics and Systems Science,

Chinese Academy of Sciences

June, 2022

中国科学院大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：

日　　期：

中国科学院大学
学位论文授权使用声明

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：　　导师签名：

日　　期：　　日　　期：

摘 要

随着云计算、大数据的发展，网络业务空前繁荣，网络拥塞问题也凸显出来。时序网络拥塞优化问题是此类问题中一个重要问题，在网络调度问题中有极强的现实应用场景。时序网络拥塞优化问题可以建模成为一个线性规划问题，对偶单纯形算法对此问题有较高的求解效率，因此对偶单纯形算法成为解决问题的核心。本文主要研究了基于对偶单纯形算法的线性规划求解器的开发，以及对偶单纯形在时序网络拥塞优化问题上的应用。对于线性规划求解器，实现了不同出基入基规则的算法，并成功求解 Netlib 测试库大多数问题。在求解时序网络拥塞优化问题上，基于已有的预处理方法，提出了与模型相关的预处理方法，并使用对偶单纯形算法求解，在计算效率上取得了优势。而后研究了单纯形算法中重要模块——初始基的选取，在此问题上也是基于已有的选取算法进行两种改进，也在求解时序网络拥塞优化问题计算效率上得到了提升，减少计算时间的同时减少了迭代次数。

关键词：时序网络拥塞优化，对偶单纯形算法，预处理，初始基

Abstract

With the development of cloud computing and big data, network services are booming like never before, and network congestion problems are also highlighted. The time-based network congestion optimization problem is an important problem in this category and has extremely realistic application scenarios in network scheduling problems. The time-based network congestion optimization problem can be modeled as a linear programming problem, and the dual simplex algorithm has a high efficiency in solving this problem. So the study of dual simplex algorithm becomes the core of solving the problem. In this paper, we study the development of a linear programming solver based on the dual simplex algorithm and the application of dual simplex algorithm to the time-based network congestion optimization problem. For the linear programming solver, different pricing and ratio test rules are implemented and most problems of the netlib test library are successfully solved. For solving time-based network congestion optimization problems solved using the pairwise simplex algorithm, model-related presolving methods are proposed based on existing presolving methods, which achieves advantages in terms of computational efficiency. Later, the selection of initial bases, an important module in the simplex algorithm, is studied, and two improvements based on the existing selection algorithm are made in this problem, which also improves the computational efficiency in solving the time-series network congestion optimization problem and reduces the computation time and the number of iterations.

Keywords: Time-based network congestion optimization, Dual simplex algorithm，Presolving, Initial basis

目 录

图形列表

表格列表

符号列表

字符

符号	描述
$\mathbf{R}$	实数集
$\mathbf{R}^n$	n 维向量集
$\mathbf{R}^{m\times n}$	$m \times n$ 维矩阵集
$\mathcal{B}$	基列下表集合
$\mathcal{N}$	非基列下表集合
θ^p	原始变量步长
θ^d	对偶变量步长

算子

符号	描述
Δ	差值算子
∇	梯度算子

缩写

LP	线性规划
DSE	最陡边算法
MGA	最小目标函数与约束梯度角

第 1 章　引言

随着云计算、大数据的发展，网络业务空前繁荣，网络流量呈指数级增长[1]，网络拥塞问题也凸显出来，而网络拥塞就可能导致信息延时、信息丢失等问题，大大降低了用户使用体验。因此，如何进行流量控制，合理分配，减少网络拥塞，给客户一个良好的网络体验成为网络优化的重要问题。Welzl [2] 给出了网络拥塞控制的各种方法，但主要以网络传输的观点来阐述。Pióro 等 [3] 则介绍了与规划相关网络流问题建模与求解。本文所提及的时序网络流拥塞优化问题，就是在现有的网络上如何进行流量分配，从而使成本低、拥塞尽可能的小，本质上是线性规划问题。而对偶单纯形算法是求解此类问题的高效算法之一，因此对偶单纯形算法的研究是求解该问题的重要部分。

本文也是基于对偶单纯形线性规划，阐述在本人在攻读硕士学位期间的三个工作内容：基于对偶单纯形算法的线性规划求解器的开发，以及基于对偶单纯形算法求解时序网络流拥塞问题的预处理方法和初始基选取方法。

1.1　问题背景

1.1.1　时序网络拥塞优化问题

时序网络拥塞优化问题是在一个已知网络上，针对每个业务输出一组路径和分流比，实现最小化网络拥塞面积。实际网络运维中，客户会尽量少的网络调整，以避免网络震荡，影响业务，为了兼顾“少调整”和“少拥塞”，因此需要结合历史和预测的多个时序的流量，来指导调优，减少调整。

时序网络拥塞优化优化问题模型来自于多时段网络流问题，这里依据 Pióro 等 [3]中的描述，

$$
\begin{aligned}
\min \quad & \sum_{t}\sum_{d}\sum_{p} c_{kpt}x_{kpt} \\
\text{s. t.} \quad & \sum_{p} x_{kpt} = y_{kt}, \qquad \forall k \in K, \forall t, \\
& \sum_{k}\sum_{p} \delta_{edp}x_{kpt} \le b_e, \quad \forall e \in E, \forall t.
\end{aligned} \tag{1.1}
$$

其中 $k = 1,2,\ldots,K$ 表示客户需求种类，$t = 1,2,\ldots,N$ 表示时间指标，$p = 1,2,\ldots,P_k$ 表示需求 k 的可选路径，$e = 1,2,\ldots,E$ 表示网络中的边。δ_{ekp} 表示如

果边 e 在路径 p 上，为 1，否则为 0，y_{kt} 表示需求 d 在时间 t 的总量，c_{kpt} 表示时间 t 时需求 k 在路径 p 的单位花费，b_e 为边 e 的容量。变量为 x_{kpt} 表示时间 t 时需求 k 在路径 p 的分配容量。

在实际应用中采用对需求比例进行分配，以保证少调整或者不调整来使问题满足。这里我们以不调整，即 Y_k^p 表示在路径 p 上分配的比例，在此模型上进行修改得到：

$$\begin{aligned} &\min_{\left\{Y_p^k\right\}} \sum_{t=1}^{N} \sum_{k\in K} \sum_{p\in P_k} c_p Y_p^k y_k^t \\ &\text{s. t.} \sum_{(k,p)\in P_e} Y_p^k y_k^t \le b_e, \forall e \in E, \\ &\sum_{p\in Q_k} Y_p^k = 1, \qquad \forall k \in K. \end{aligned} \tag{1.2}$$

更进一步的考虑网络拥塞 z_e^t，即链路使用超过一定数值的量称为拥塞 z_e^t，和最大边链路利用率 u_t，即同一时刻链路的最大利用率称为最大边链路利用率 u_t，以及需求路径需要满足得到模型，

$$\begin{aligned} &\min_{\left\{Y_p^k\right\}} \frac{C}{N} \sum_{t=1}^{N} u_t + \sum_{t} \sum_{e} z_e^t + \sum_{t=1}^{N} \sum_{k\in K} \sum_{p\in P_k} c_p Y_p^k y_k^t \\ &\text{s. t.} \sum_{(k,p)\in P_e} Y_p^k y_k^t \le b_e, && \forall e \in E, \\ &Y_p^k \ge c_0, && \forall k \in K, \forall t, \forall p, \\ &\sum_{p\in Q_k} Y_p^k = 1, && \forall k \in K, \\ &z_e^t \ge 0, && \forall e \in E, \\ &z_e^t \ge \left(\sum_{(k,p)\in P_e} Y_p^k y_k^t \right) - \alpha b_e, && \forall e \in E, \\ &u_t \ge \left(\sum_{(k,p)\in P_e} Y_p^k y_k^t \right) / b_e, && \forall k \in K, \forall e \in E. \end{aligned} \tag{1.3}$$

此时目标函数中增加了拥塞面积成本以及最大链路利用率的成本。

观察时序网络流拥塞优化问题模型特征，发现 $\sum_{(k,p)\in P_e} Y_p^k y_k^t$，出现多次，因此可以用整体替换来减少，同时也减少了 LP 问题的非零元个数，从而减少计算。

最终优化模型为：

$$
\begin{aligned}
\min_{\substack{\{Y_p^k\}\\ \{u_t\},\{z_t^e\}}} \ & \frac{C}{N}\sum_{t=1}^{N} u_t + \sum_{t=1}^{N}\sum_{e\in E} z_t^e + \sum_{t=1}^{N}\sum_{e\in E} c_e x_t^e \\
\text{s.t. } & x_t^e = \sum_{(k,p)\in P_e} Y_p^k y_k^t, && \forall e\in E, \forall t, \\
& x_t^e \le b_e, && \forall e\in E, \forall t, \\
& Y_p^k \ge c_0, && \forall (k,p)\in P_e, e\in E, \forall t, \\
& z_t^e \ge 0,\ z_t^e \ge x_t^e - \alpha b_e, && \forall e\in E, \forall t, \\
& 1 \ge u_t \ge \frac{x_t^e}{b_e}, && \forall e\in E, \forall t, \\
& \sum_{p\in Q_k} Y_p^k = 1, && \forall k\in K, \\
& Y_p^k \ge 0, && \forall p\in Q_k, \forall k\in K.
\end{aligned}
\tag{1.4}
$$

通过建立模型与实验验证得出此类问题的重要突破点在于对问题的处理简化，还有对规划问题的预处理可以大大减少计算时间。因此在此问题上，我们的重点是基于模型的改进和预处理。

1.1.2　对偶单纯形线性规划求解器

自从 1947 年，Dantzig 提出 (原始) 单纯形算法后 [10]，单纯形算法理论与应用的到快速的扩展。同时单纯形算法被应用于实际的问题计算中，但由于当时硬件限制，可计算问题规模都较小。而后 Lemke [12] 提出了对偶单纯形算法，但是在当时对偶方法计算效率不如原始方法。直到 Forrest 等 [14] 提出了用于对偶问题且非常有效的最陡边规则作为出基方法后，对偶单纯形算法效率提升很多。

现如今，原始单纯形、对偶单纯形以及内点法都可以解决大规模线性规划问题，但是 Bixby [13] 中对偶单纯形算法在现实问题中具有更好的表现效果。此外在混合整数线性规划问题上，每进行一次分支定界，相当于需要求解的线性规划问题加了一条约束，而对于对偶单纯形来说，之前问题的解可直接作为新问题的初始可行解直接计算。因此在混合整数线性规划问题上，对偶单纯形算法几乎有着无可替代的作用。在对偶单纯性算法之后的发展中，主流的便是两阶段法。Pan 等 [50] 阐述了自己的一阶段算法，并说明了其计算效果 [21]。而后，Maros [16, 17] 和 Kostina [19] 阐述对偶单纯形中两阶段算法，并展示了计算效果。

由于对偶单纯形算法在现实应用的重要性，基于对偶单纯形算法的线性规划求解器就显得尤为重要。早期计算效率高的求解器几乎被国外垄断，美国有 GUROBI、CPLEX，德国有 SCIP，俄罗斯有 GLPK，芬兰有 LPSOLVE。现在国内一些科技公司也开始积极部署求解器开发工作，如杉树科技的 COPT，阿里巴巴公司的 MindOpt，华为公司的 Optverse，都在单纯形算法中有很好的成果 [18]。基于研究应用，以及对未来可能与 MIP 问题结合，我们尝试探索了对偶单纯形算法求解器的开发。

1.1.3 线性规划预处理方法

众所周知，线性规划预处理对于高效求解大规模线性规划问题是非常重要的 [8, 31, 32, 36]。虽然线性规划软件和计算机的速度已经快得多，但 LP 型号的尺寸也有所增加。此外，线性规划求解用于交互式应用程序和整数规划，在这些应用程序中需要解决许多线性规划子问题。因此，更有效的算法和改进的实现技术仍然非常重要。实际的线性规划模型都是由计算机程序直接或在一个建模系统中生成的。模型生成器是从数学模型结构和模型数据中导出计算机模型。有时需要合并多个模型，或者需要对子模型进行求解。大多数模型生成器的数据分析能力非常有限。因此，模型中通常有很大一部分是冗余的。通过预处理方法去除一些冗余约束，减小问题规模能巨幅提升对问题的求解速度。

近几十年，预处理已经广泛的应用于求解线性规划问题。A. L. Brearley [31]和 Williams [33]在数学规划的系统上讨论了界收紧、删除行和固定变量的情况，而 Andersen 等 [35]发表了线性规划的处理技术，比如能够移除冗余约束的平行行方法。Mszros 等 [36]总结提出了高阶预处理方法，并通过数值实验证明预处理方法的重要性和有效性。Gamrath 等 [37]通过分析变量之前的占优关系，研究了能够将一些变量固定的占优列方法。在已有预处理的方法基础上，我们结合求解模型提出了一些新的预处理方法来加快求解速度。

1.1.4 对偶单纯形算法初始基选取

单纯形算法需要一个表示可行基解的初始顶点，以确保构建一系列可行基解，使其在存在时达到最优值。对于经典的单纯形法来说，找到问题的初始可行解是一个重要问题，有时甚至是不可能的。在这种情况下，一种常用的方法被命名为单纯形算法的第 I 阶段，通过引入人工变量的一阶段算法找到可行基 [50]。

第 I 阶段的解是一个可行解，且可以作为第 II 阶段的起始点。第一阶段的备选方法由 Wolfe [44] 和 Maros [16] 提出。其他人试图改进可行基本解的序列，定义了不同的主元规则。

而后 Luh 等 [46] 和 Paparrizos 等 [47] 使用内点法来寻找一个初始基。在之后单纯形余弦方法 (Simplex-Cosine Method，简称 SCM) 提出了一种通过分析目标函数梯度与约束条件梯度之间的夹角来确定初始顶点的新方法 [49, 51, 56]。这些角是通过余弦来计算的，由于这个原因，该方法被称为单纯形余弦方法。Stojković 等 [53] 利用 SCM 提出一些依据问题约束直接找初始基的方法，但是不保证有效性。Junior 等 [45] 与 Hu [48] 则提出了有效的寻找初始基的方法，本文也是在此基础上进行了改进，在解决时序网络拥塞优化问题上给出了兼顾迭代次数与计算时间的方案。

1.2　预备知识

考虑标准线性规划问题：

$$\begin{aligned} \min \quad & z = \mathbf{c}^{\top}\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \\ & x \geqslant 0. \end{aligned} \tag{1.5}$$

其中，$A \in \mathbf{R}^{m\times n}$，$\mathbf{c}$、$\mathbf{x} \in \mathbf{R}^{n}$。

定义 1.1. 假设 B 是矩阵 A 中任意 m 列组成的 m 阶满秩方阵，称 B 是该问题的一个基。

定义 1.2. $\mathbf{x}$ 中与 B 对应的 m 个变量 $\mathbf{x}_B$ 称为基变量，其余的变量 $\mathbf{x}_N$ 称为非基变量。

定义 1.3. 令所有的非基变量为零，得到的解 $x = \begin{pmatrix} \mathbf{x}_B \\ \mathbf{x}_A \end{pmatrix} = \begin{pmatrix} B^{-1}\mathbf{b} \\ 0 \end{pmatrix}$，称为在基 B 下的基本解；若 $\mathbf{x} \geq 0$，$\mathbf{x}$ 称为基可行解。

定义 1.4. 集合 $S \subset R^n$，若 $\forall \mathbf{x}^1, \mathbf{x}^2 \in S, \forall \lambda_1, \lambda_2 \geq 0$，$\lambda_1 + \lambda_2 = 1$，$\lambda_1\mathbf{x}^1 + \lambda_2\mathbf{x}^2 \in S$，则 S 称为凸集。

定义 1.5. 在凸集 C 中的一个点 $\mathbf{x}$，如果 $\mathbf{x}$ 不是 C 另外两个不同点的凸组合，则称它是 C 的一个顶点。

定理 1.1. 令 $P=\left\{\mathbf{x}\in R^n:A\mathbf{x}=b,\mathbf{x}\geq 0\right\}$，$\mathbf{x}\in P$，则 $\mathbf{x}$ 是 P 的一个顶点当且仅当 $\mathbf{x}$ 的正分量对应 A 中的各列是线性不相关的。

证明. 不仿设 $\mathbf{x}=\left(\mathbf{x}_B,0\right)^T$，其中 $\mathbf{x}_i>0(i=1,2,\cdots,p)$，记 $A=(B,N),\mathbf{x}=\begin{pmatrix}\mathbf{x}_B\\ \mathbf{x}_N\end{pmatrix}$，其中 B 是 A 的前 p 列。则，$A\mathbf{x}=b\Leftrightarrow B\mathbf{x}_B=b$。

“⇒” 用反证法，假设 $\mathbf{x}$ 是 p 的一个顶点，但 B 有一列与其他线性相关，则存在一个非零向量 w，使得 $Bw=0$。令 $\mathbf{x}_B^1=\mathbf{x}_B+\delta w,\mathbf{x}_B^2=\mathbf{x}_B-\delta w$，由于 $\mathbf{x}_B>0$，所以对于充分小的 $\delta>0$，有 $\mathbf{x}_B^1\geq 0,\mathbf{x}_B^2\geq 0$。显然 $B\mathbf{x}_B^1=B\mathbf{x}_B^2=b$。定义: $\mathbf{x}^1=\left(\mathbf{x}_B^1,0\right)^T,\mathbf{x}^2=\left(\mathbf{x}_B^2,0\right)^T$，则 $A\mathbf{x}^1=A\mathbf{x}^2=b$，所以 $\mathbf{x}^1,\mathbf{x}^2\in P$，且 $\mathbf{x}=\frac{1}{2}\mathbf{x}^1+\frac{1}{2}\mathbf{x}^2$，与 $\mathbf{x}$ 是 p 的一个顶点矛盾。

“⇐” 用反证法，假设 $\mathbf{x}$ 不是 p 的一个顶点，则存在 $y^1,y^2\in P,0<\lambda<1$，使得 $\mathbf{x}=\lambda y^1+(1-\lambda)y^2$。因为 $\mathbf{x}_N=0$，所以 $y_N^1=y_N^2=0$。令 $w=\mathbf{x}-y^1$，则 w 为非零向量，且 $Bw_B=B\mathbf{x}_B-By_B^1=b-b=0$，所以 B 中各列线性相关，与假设矛盾。 □

推论 1.2. $\mathbf{x}$ 是 *(1.5)* 的一个基可行解当且仅当 $\mathbf{x}$ 是 P 的一个顶点。

定理 1.3. *(分解定理)* 令 $V=\left\{v^i\in R^n:i\in I\right\}$ 是 P 的所有顶点集合，则 $\forall\mathbf{x}\in P$，有

$$\mathbf{x}=\sum_{i\in I}\lambda_i v^i+\lambda d$$

其中 $\sum_{i\in I}\lambda_i=1,\lambda_i\geq 0,i\in I,\lambda\geq 0$ 且 d 或者是零向量，或者是 P 的一个极方向。

定理 1.4. *(线性规划基本定理)* 对于 *(1.5)*，若 $P\neq\phi$，则 z 或者无下界，或者至少存在 P 的一个顶点，z 其上达到最小值。

证明. 令 $V=\left\{v^i\in P:i\in I\right\}$ 是 P 的所有顶点。$P\neq\phi$ 则 $V\neq\phi$。

根据定理1.3，$\forall\mathbf{x}\in P$ 有 $\mathbf{x}=\sum_{i\in I}\lambda_i v^i+\lambda d$，其中 $\sum_{i\in I}\lambda_i=1,\lambda_i\geq 0,i\in I,\lambda\geq 0$ 且 d 或者是零向量，或者是 P 的一个顶点方向。

(1) 若 P 存在一个顶点方向 d 满足 $c^Td<0$，则 z 必无界。事实上，任给 $\mathbf{x}^0\in P,\left\{\mathbf{x}\in R^n:\mathbf{x}=\mathbf{x}^0+\lambda d,\lambda\geq 0\right\}\subset P$，当 $\lambda\to+\infty$ 时，$c^T\mathbf{x}\to-\infty$，所以 z 无下界。

(2) 否则，设 P 的所有顶点方向 d 满足 $c^T d \geq 0$ (或者不存在顶点方向)。记 $v^{\min} = \arg\min\left\{c^T v^i : i \in I\right\}$ 是目标函数值最小的顶点，且，则 $\forall \mathbf{x} \in P$ 有

$$c^T \mathbf{x} = \sum_{i \in I} \lambda_i c^T v^i + c^T d \geq \sum_{i \in I} \lambda_i c^T v^{\min} = c^T v^{\min}$$

所以 z 在顶点 $v^{\min}$ 上达到最小值。 □

定理 1.5. *(互补松弛性条件)* $\mathbf{x}$ 和 y 分别为原始-对偶可行解，则它们分别是原始-对偶最优解当且仅当 $\forall i, j$ 有:

$$u_i = y_i\left(a_i^T \mathbf{x} - b_i\right) = 0,$$
$$v_j = \left(c_j - A_j^T y\right)\mathbf{x}_j = 0.$$

证明. 显然，对一切 i 和 j 有: $u_i \geq 0, v_j \geq 0$ 。定义

$$u = \sum_i u_i, \quad v = \sum_j v_j,$$

则 $u = 0$ 等价于 $u_i = 0$ (对一切 i)

$$v = 0 \Leftrightarrow v_j = 0 \quad (\text{对一切} \boldsymbol{j})$$

而 $u + v = c^T \mathbf{x} - b^T y$ 所以，$u_i = 0$ (对一切 i) 且 $v_j = 0$ (对一切 j) $\Leftrightarrow u = 0$ 且 $v = 0 \Leftrightarrow u + v = 0 \Leftrightarrow c^T \mathbf{x} - b^T y = 0 \Leftrightarrow \mathbf{x}$ 和 y 是原始-对偶问题最优解。 □

1.3　本文工作

本文在第二章中介绍了对偶单纯形算法的算法框架和具体算法，以及算法实现和数值实验。第三章介绍了基本与处理方法，以及我们提出的基于时序网络拥塞优化问题的预处理方法和实验结果。第四章介绍了一种初始基选取方法，并介绍了改进方案以及实验结果对比。第六章则是对本文内容总结以及展望。

第 2 章　基于对偶单纯形的线性规划求解器

对偶单纯形算法主要是两阶段法，一阶段找出可行基，或者说可行解，而后进入二阶段，在有可行解的前提下，通过不断地出基入基迭代，最终找到最优解。本章主要介绍对偶单纯形算法的两阶段法以及其中其中涉及的出基入基的不同方法。LP 问题的一般形式为：

$$\begin{aligned} &\min\ c_0 + \mathbf{c}^T\mathbf{x} \\ &\text{s. t. } \mathbf{b}_L \le A\mathbf{x} \le \mathbf{b}_U, \\ &\mathbf{l} \le \mathbf{x} \le \mathbf{u}. \end{aligned} \tag{2.1}$$

其中 c_0 为常数，$A \in \mathbf{R}^{m\times n}$，$\mathbf{c}$、$\mathbf{x}$、$\mathbf{l}$、$\mathbf{u} \in \mathbf{R}^n$，$\mathbf{b}_L$、$\mathbf{b}_U \in \mathbf{R}^m$。通常，$\mathbf{b}_L$、$\mathbf{b}_U$ 被称为约束范围，可以为无穷，$\mathbf{l}$、$\mathbf{u}$ 被称为变量的界，可以为无穷。可以看出，一般的线性规划问题，都可以通过添加人工变量建模成如上形式，且 c_0 为固定值，可暂不考虑。计算模型：

$$\begin{aligned} &\min\ \mathbf{c}^T x \\ &\text{s. t. } A\mathbf{x} = \mathbf{b}, \\ &\mathbf{l} \le \mathbf{x} \le \mathbf{u}. \end{aligned} \tag{2.2}$$

其对偶问题为：

$$\begin{aligned} &\max \mathbf{b}^T\mathbf{y} + \sum_{\{j:l_j>-\infty\}} l_j d_j + \sum_{\{j:u_j<\infty\}} u_j d_j \\ &\text{s. t. } A^T\mathbf{y} + \mathbf{d} = \mathbf{c}. \end{aligned} \tag{2.3}$$

其中 $\mathbf{d} = \mathbf{c} - (\mathbf{c}_{\mathcal{B}}^{\top} B^{-1} A)^{\top}$。对偶基本可行解需要满足如下条件：

$$\mathbf{y} = B^{-T}\mathbf{c}_{\mathcal{B}}$$

和

表 2.1 可行原始变量和对偶逻辑变量关系

x_j	fixed	$x_j = l_j$	$x_j = u_j$	free
d_j	d_j free	$d_j \ge 0$	$d_j \le 0$	$d_j = 0$

其中，$\mathcal{B}$ 是基列下标集合，$\mathcal{N}$ 是非基列下标集合，$j \in \mathcal{N}$。

2.1 基本算法

有了基本概念了解后，我们可以考虑对偶单纯形算法，即如何找到最优解。对偶单纯形的中心思想是从一个可行解沿着可行集的边到另一个可行解，从而接近最优值。一般的算法为：

算法 1 一般对偶单纯形算法

步骤一：出基

寻找出基变量 $p \in \mathcal{B}$(具体方法参考2.2节)，且对偶可行。

如果不存在，则达到最优，输出。

步骤二：入基

寻找入基变量 $q \in \mathcal{N}$(具体方法参考2.3节)，且对偶可行。

如果不存在，则无界，输出。

步骤三：更新

更新基与非基：$\mathcal{B} \leftarrow (\mathcal{B}\backslash\{p\}) \cup \{q\}$、$\mathcal{N} \leftarrow (\mathcal{N}\backslash\{q\}) \cup \{p\}$。

更新数值。

返回**步骤一**

此算法是在已有初始对偶可行解的基础上进行，因此如何寻找对偶可行解也是一个问题。经典做法就是分为两阶段：一阶段寻找对偶可行解，二阶段寻找最优解。选择出基入基的过程就是从凸集中的一个极点到另一个极点。

2.1.1 相邻解变换

二阶段是以对偶可行解的基础上进行，假定有对偶可行解，那么我们考虑如何进行迭代使目标函数逐渐增大。令 $\mathcal{B}$ 是基础可行基，则其对应对偶可行解，假设 $p \in \mathcal{B}$ 出基，与之对应当前基矩阵中列数为 r，对应变量且改变 $t \in \mathbf{R}$，且其他约束保持相等，则有：

$$\begin{aligned} t &= a_p^T \bar{\mathbf{y}} - a_p^T \mathbf{y}, \\ a_p^T \bar{\mathbf{y}} - t &= a_p^T \mathbf{y}. \end{aligned} \tag{2.4}$$

$$a_j^T \bar{\mathbf{y}} = a_j^T \mathbf{y} \quad \forall j \in \mathcal{B}\backslash\{p\}. \tag{2.5}$$

由 (2.4) 和 (2.5) 可得到：

$$B^T \bar{\mathbf{y}} - e_r t = B^T \mathbf{y}. \tag{2.6}$$

变换形式得：

$$\begin{aligned}\bar{\mathbf{y}} &= \mathbf{y} + \left(B^T\right)^{-1} e_r t \\ &= \mathbf{y} + \rho_r t.\end{aligned} \tag{2.7}$$

其中 $\rho_r = \left(B^T\right)^{-1} e_r$，可作为 $\bar{\mathbf{y}}$ 的更新因子。相对应的有：

$$\begin{aligned}\bar{d} &= c - A^T \bar{\mathbf{y}} \\ &= c - A^T \left(\mathbf{y} + \rho_r t\right) \\ &= d - A^T \left(B^T\right)^{-1} e_r t \\ &= d - \alpha^r t,\end{aligned} \tag{2.8}$$

其中 $\alpha^r = e_r^T B^{-1} A$，是 $\bar{d}$ 的更新因子。

由 (2.8) 可以得到：

$$\bar{d}_j = d_j = 0, \qquad \forall j \in \mathcal{B}\backslash\{p\}, \tag{2.9a}$$

$$\bar{d}_p = -t, \tag{2.9b}$$

$$\bar{d}_j = d_j - \alpha_j^r t, \qquad \forall j \in \mathcal{N}. \tag{2.9c}$$

这便描述清楚了出基后一些变量的变化情况，接下来便是讨论如何选择出基和入基变量。

2.1.2　选择出基变量

出基入基的本质是为了使目标函数更优，因此我们需要分析出基入基对于目标函数的影响。考虑对偶问题目标函数：

$$\begin{aligned}\bar{Z} &= b^T \bar{y} + \sum_{\substack{j\in\mathcal{J}\\ l_j>-\infty}} l_j \bar{v}_j + \sum_{\substack{j\in\mathcal{J}\\ u_j<\infty}} u_j \bar{w}_j \\ &= b^T \bar{y} + \sum_{\substack{j\in\mathcal{N}\\ d_j\geq 0}} l_j \bar{d}_j + \sum_{\substack{j\in\mathcal{N}\\ d_j\leq 0}} u_j \bar{d}_j - t u_p^{\pm}, \quad \text{其中} u_p^{\pm} = \begin{cases} l_p & t \leq 0 \\ u_p & t \geq 0 \end{cases} \\ &= b^T \left(y + t e_r^T B^{-1}\right) + \sum_{\substack{j\in\mathcal{N}\\ d_j\geq 0}} l_j \left(d_j - t e_r^T B^{-1} a_j\right) + \sum_{\substack{j\in\mathcal{N}\\ d_j\leq 0}} u_j \left(d_j - t e_r^T B^{-1} a_j\right) - t u_p^{\pm} \\ &= Z + t e_r^T B^{-1} b - \sum_{\substack{j\in\mathcal{N}\\ d_j\geq 0}} t e_r^T B^{-1} a_j l_j - \sum_{\substack{j\in\mathcal{N}\\ d_j\leq 0}} t e_r^T B^{-1} a_j u_j - t u_p^{\pm}\end{aligned}$$

$$
\begin{aligned}
&= Z + te_r^T B^{-1}\left(b - A_{\mathcal{N}} x_{\mathcal{N}}\right) - tu_p^{\pm} \\
&= Z + te_r^T x_{\mathcal{B}} - tu_p^{\pm} \\
&= Z + t\left(x_p - u_p^{\pm}\right) \\
&= Z + \Delta Z, \quad \text{其中} \Delta Z = \begin{cases} t\left(x_p - l_p\right), & t \le 0, \\ t\left(x_p - u_p\right), & t \ge 0. \end{cases}
\end{aligned} \tag{2.10}
$$

因此需要选择出基列 p 使 $\Delta Z > 0$，解才会提升目标函数值。因此选择的 p 需要满足 $x_p < l_p$，$t \le 0$ 或者 $x_p > u_p$，$t \ge 0$。当所有的原始变量 x 都处于可行区域内，这目标函数无法提升，即达到最优值。

2.1.3 选择入基变量

在前一节中，放松约束的对偶逻辑始终保持对偶可行。由 (2.9) 其他基本约束保持相等，只需要考虑非基本对偶逻辑变量来满足对偶可行性条件。此外，我们可以忽略与固定原始变量相关的对偶逻辑变量，因为它们永远不可能是对偶不可行时。为了保持对偶可行，入基列 q 的对偶逻辑变量 d_q 变为 0，同时需要使 $\Delta Z > 0$。由 (2.9c):

$$
\bar{d}_j = d_j - \alpha_j^r t \quad \forall j \in \mathcal{N}. \tag{2.11}
$$

当 $x_j = l_j$ 时，为保持对偶可行，$\bar{d}_j = d_j - \alpha_j^r t \ge 0$，有：

$$
t \ge \frac{d_j}{\alpha_j^r} \quad \text{当} \alpha_j^r < 0 \quad \text{或者} \quad t \le \frac{d_j}{\alpha_j^r} \quad \text{当} \alpha_j^r > 0. \tag{2.12}
$$

当 $x_j = u_j$ 时，为保持对偶可行，$\bar{d}_j = d_j - \alpha_j^r t \le 0$，有：

$$
t \ge \frac{d_j}{\alpha_j^r} \quad \text{当} \alpha_j^r > 0 \quad \text{或者} \quad t \le \frac{d_j}{\alpha_j^r} \quad \text{当} \alpha_j^r < 0. \tag{2.13}
$$

当 x_j 是自由变量时，为保持对偶可行，$\bar{d}_j = d_j - \alpha_j^r t = 0$，有：

$$
t = \frac{d_j}{\alpha_j^r} = 0. \tag{2.14}
$$

总结，当 $t \ge 0$，令集合

$$
\mathcal{F}^+ = \left\{ j : j \in \mathcal{N}, x_j \text{自由} \ \vee \left(x_j = l_j, \alpha_j^r > 0\right) \vee \left(x_j = u_j, \alpha_j^r < 0\right) \right\} \tag{2.15}
$$

若集合为空集，问题无界。若非空，则 q 取其中一个即可。

当 $t \leq 0$，令集合

$$\mathcal{F}^+ = \left\{ j : j \in \mathcal{N}, x_j \text{自由} \ \vee \left(x_j = l_j, \alpha_j^r < 0 \right) \vee \left(x_j = u_j, \alpha_j^r > 0 \right) \right\} \tag{2.16}$$

若集合为空集，问题无界。若非空，则 q 取其中一个即可。如此循环迭代，目标函数增大直到找到最优或者发现问题无界。

2.1.4　基变换后更新

在选择出基入基列之后，基发生了变化，相对应的原始变量的值以及对偶逻辑变量都发生了改变。假设出基列为 p，进基列为 q，对偶变量变化步长 $t = \theta^d$，则对偶逻辑变量：

$$\bar{d}_p = -\theta^d, \tag{2.17a}$$

$$\bar{d}_q = 0, \tag{2.17b}$$

$$\bar{d}_j = d_j = 0, \qquad \forall j \in \mathcal{B} \backslash \{p\}, \tag{2.17c}$$

$$\bar{d}_j = d_j - \theta^d \alpha_j^r, \qquad \forall j \in \mathcal{N} \backslash \{q\}. \tag{2.17d}$$

假设原始变量变化步长 $t = \theta^p$，则原始变量：

$$\bar{x}_p = \begin{cases} l_p, & x_p < l_p, \\ u_p, & x_p > u_p, \end{cases} \tag{2.18a}$$

$$\bar{x}_q = x_q + \theta^p, \tag{2.18b}$$

$$\bar{x}_j = x_j, \qquad \forall j \in \mathcal{N} \backslash \{q\}, \tag{2.18c}$$

$$\bar{x}_j = x_j - \theta^p \alpha_q^i, \qquad \forall j \in \mathcal{B} \backslash \{p\}. \tag{2.18d}$$

由 (2.18a) 和 (2.18d) 得到：

$$\begin{aligned} &\text{当} x_p < l_p, \quad \theta^p = \frac{x_p - l_p}{\alpha_q^p}, \\ &\text{当} x_p > u_p, \quad \theta^p = \frac{x_p - u_p}{\alpha_q^p}. \end{aligned} \tag{2.19}$$

至此，对偶单纯形算法中需要用到的变量都在选择出基入基后完成更新。

算法 2 对偶单纯形算法

1: 计算 $\tilde{b} = b - A_{\mathcal{N}} x_{\mathcal{N}}$ 和求解 $B x_{\mathcal{B}} = \tilde{b}$ 得到 $x_{\mathcal{B}}$；
2: 求解 $B^T y = c_{\mathcal{B}}$ 而后计算 $d_{\mathcal{N}} = c_{\mathcal{N}} - A_{\mathcal{N}}^T y$；
3: **while** 出基列 p 存在 **do**
4: 求解 $B^T \rho_p = e_p$ 得到 ρ_p，计算 $\alpha^p = A_{\mathcal{N}}^T \rho_p$；
5: 选择出基列 q；
6: 求解 $B\alpha_q = a_q$ 得到 α_q；
7: 更新 d，$d_p = -\theta^d$，$d_q = 0$，$d_j = d_j - \theta^d \alpha_j^r, \forall j \in \mathcal{N}$；
8: 更新 x，$x_{\mathcal{B}} = x_{\mathcal{B}} - \theta^P \alpha_q$，$x_q = x_q + \theta^P$；
9: 更新基矩阵 B；
10: **end while**

其中求解方程组使用 LU 分解方法 [23, 24]，使得 $B = LU$，其中 L 为下三角矩阵，U 为上三角矩阵，在求解 $Bx = y$ 时，即求解 $LUx = y$。先求解 $L\tau = y$ 再求解 $Ux = \tau$ 即可得到原方程的解。在出基入基之后，基矩阵 B 需要更新，与之对应的 L 与 U 也需要更新，详细步骤参考 [25, 26]

2.2 出基规则

在对偶单纯形法的出基中，我们必须在所有原始不可行的变量中选择一个变量来保留基。在几何术语中，这对应于沿着由当前基本解定义的顶点发出的对偶多面体的一条边选择搜索方向，使目标函数值下降。出基规则是如何选取出基列，使得尽可能好的改善目标函数从而减少迭代次数。下面介绍几种常见的出基规则。

2.2.1 Dantzig 规则

由 (2.10) 可知目标函数的表达式为：

$$\bar{Z} = Z + t\left(x_j - u_j^{\pm}\right).$$

Dantzig 规则 [10] 在此表达式上，考虑单位步长变化量最大，即选取出基列 p 满足

$$p \in \arg\max |x_j - u_j^{\pm}|. \tag{2.20}$$

2.2.2　最陡边规则

最陡边规则 (*dual steepest edge pricing*，DSE) 的基本思想是确定与双梯度形成最锐角的边方向，在这个意义上是最陡的。如下将遵循 Forrest 等 [14]的描述：

最陡边规则则是考虑选择目标函数上升最快的方向，$\bar{y} = y - t\rho_i$ 其中 $\rho_i = \boldsymbol{B}^{-T} e_i$，$-\rho_i$ 为 i 出基的边方向，使边方向与目标函数方向夹角最小，从而使目标函数下降最快。选取出基列 p 使得

$$p \in \underset{i\in\{1,\ldots,m\}}{\arg\max} \left\{ \frac{|x_i - u_i^{\pm}|}{\|\rho_i\|} \right\}.$$

为了方便计算，取

$$p \in \underset{i\in\{1,\ldots,m\}}{\arg\max} \left\{ \frac{|x_i - u_i^{\pm}|^2}{\|\beta_i\|} \right\}.$$

其中 $\beta_i = \rho_i^T \rho_i = \|\rho_i\|^2$。假设 p 出基，q 入基，β 更新方式为：

$$\bar{\beta}_p = \bar{\rho}_r^T \bar{\rho}_p = \left(\frac{1}{\alpha_q^p} \right)^2 \beta_p, \tag{2.21}$$

$$\begin{aligned} \bar{\beta}_i &= \bar{\rho}_i^T \bar{\rho}_i \\ &= \left(\rho_i^T - \frac{\alpha_q^i}{\alpha_q^p} \rho_p^T \right) \left(\rho_i - \frac{\alpha_q^i}{\alpha_q^p} \rho_r \right) \\ &= \beta_i - 2\frac{\alpha_q^i}{\alpha_q^p} \tau_i + \left(\frac{\alpha_q^i}{\alpha_q^p} \right)^2 \beta_p. \end{aligned} \tag{2.22}$$

其中 $\tau = \boldsymbol{B}^{-1} \rho_p$。

算法 3 最陡边算法 (DSE)

1: **for** $i \le m$ **do**
2: 　　选取 p，使得 $\frac{|x_p - u_p^{\pm}|^2}{\|\beta_p\|} \le \frac{|x_j - u_j^{\pm}|^2}{\|\beta_i\|}$，作为出基列；
3: **end for**
4: 按照 (2.21) 和 (2.22) 更新 β。

2.2.3　近似最陡边规则

近似最陡边时最陡边的近似，效果相近，计算量大大减少，主要是近似最陡边中的 β。为了方便计算，一般在初始时 h_i 都取 1。近似最陡边的选取规则为

$$p \in \arg\max_{j\in\mathcal{B}} \left\{ \frac{|x_p - u_p^{\pm}|}{h_j} \right\}. \tag{2.23}$$

选出出基列后，对 h 的更新方式为

$$\begin{aligned}\bar{h}_p &= \max\left\{1, \|\hat{\alpha}^p\| / \alpha_q^p\right\}, \\ \bar{h}_j &= \max\left\{h_j, |\alpha_q^j / \alpha_q^p| \, \|\hat{\alpha}^p\|\right\}, \ j \in \mathcal{B}, \ j \neq p.\end{aligned} \tag{2.24}$$

用 h 近似的表示 β，在减少更新计算量的同时，少解一次求解 (2.22) 中的 τ 的方程。

2.3 入基规则

出基列选取完成后，选取入基规则是在保证对偶变量可行的前提下选择合适的步长。最简单的方法是保证所有的变量都在可行范围内，此方法成为简易入基规则，也叫短步长规则。而另外一种就是，对某些变量来说，我们可以使其短暂的不可行，使目标函数增加更多，而后通过 (2.27) 使这些变量重新变得可行。这种方法被称为有上下界变换的入基规则，也称长步长规则。

2.3.1 简易入基规则

由 (2.17d) 更新对偶变量：

$$\bar{d}_j = d_j - \alpha_j^p t, \quad \forall j \in \mathcal{N}.$$

为了所有变量保持可行，当 $t \geq 0$ 时，记集合

$$\mathcal{F}^+ = \left\{ j : j \in \mathcal{N}, x_j \text{ 自由 } \vee \left(x_j = l_j, \alpha_j^r > 0\right) \vee \left(x_j = u_j, \alpha_j^r < 0\right) \right\},$$

此时入基列选取为

$$q \in \arg\min_{j \in \mathcal{F}^+} \left\{ \frac{d_j}{\alpha_j^r} \right\}, \quad \theta^d = \frac{d_q}{\alpha_j^q}. \tag{2.25}$$

当 $t \leq 0$，记集合

$$\mathcal{F}^- = \left\{ j : j \in \mathcal{N}, x_j \text{ 自由 } \vee \left(x_j = l_j, \alpha_j^r < 0\right) \vee \left(x_j = u_j, \alpha_j^r > 0\right) \right\},$$

此时入基列选取为

$$q \in \arg\max_{j \in \mathcal{F}^-} \left\{ \frac{d_j}{\alpha_j^r} \right\}, \quad \theta^d = \frac{d_q}{\alpha_j^q}. \tag{2.26}$$

2.3.2　带有边界转换的入基规则

与简易入基规则不同的是，不必保持所有变量对偶可行，对于同时拥有上下界的可暂时使其不可行，之后进行上下界变换使其重新可行。这样在一次迭代中有更长的步长，减少迭代次数。对于有上下界的且对偶不可行变量，可通过如下方法变换上下界，使其对偶可行：

$$\begin{aligned} x_B &:= x_B - \sum_{j\in T^+} u_j\alpha_j + \sum_{j\in T^-} u_j\alpha_j \\ &= x_B - B^{-1}\left(\sum_{j\in T^+} u_j a_j - \sum_{j\in T^-} u_j a_j\right) \\ &= x_B - B^{-1}\tilde{a}. \end{aligned} \tag{2.27}$$

其中 $T^+ = \{j : x_j = u_j \text{且} d_j > 0\}$，$T^- = \{j : x_j = l_j \text{且} d_j < 0\}$。

我们考虑 $x_p > u_p$，$t > 0$ 情况：由 (2.10) 目标函数关于 t 斜率为：

$$\delta_1 = x_p - u_p.$$

当没有 d_j 改变正负，

$$0 \le t \le \theta_1^d, \quad q_1 \in \arg\min_{j\in\mathcal{Q}_1^+}\left\{\frac{d_j}{\alpha_j^p}\right\},\ \theta_1^D = \frac{d_{q_1}}{\alpha_{q_1}^r} \text{ 且 } \mathcal{Q}_1^+ = \mathcal{F}^+. \tag{2.28}$$

目标函数改变量为 $\Delta Z_1 = \theta_1^D\delta_1$。此时与简易入基规则选取相同。当 t 继续增大，目标函数关于 t 的斜率为 δ_2。如果 $\alpha_{q_1}^r > 0$ 且 $x_{q_1}: l_{q_1} \to u_{q_1}$，

$$\delta_2 = \delta_1 - \left(u_{q_1} - l_{q_1}\right)\alpha_{q_1}^r.$$

如果 $\alpha_{q_1}^r < 0$ 且 $x_{q_1}: u_{q_1} \to l_{q_1}$，

$$\delta_2 = \delta_1 - \left(l_{q_1} - u_{q_1}\right)\alpha_{q_1}^r.$$

当新的斜率 $\delta_2 > 0$，表明此时目标函数还在上升中，步长可以持续叠加。

$$\theta_2^d = \frac{d_{q_2}}{\alpha_{q_2}^r}, \quad q_2 \in \arg\min_{j\in\mathcal{Q}_2^+}\left\{\frac{d_j}{\alpha_j^r}\right\}, \quad \mathcal{Q}_2^+ = \mathcal{Q}_1^+\backslash\{q_1\}. \tag{2.29}$$

目标函数改变量为 $\Delta Z_2 = \left(\theta_2^d - \theta_1^d\right)\delta_2$。重复进行直到 $\delta_k < 0$ 或者无界，本次迭代结束。当 $x_p < l_p$，$t < 0$，与上面描述同理。由此选取出比简易入基规则更长的步长。

算法 4 带有边界转换的入基规则

1: 当 $x_p < l_p$，令 $\tilde{\alpha}^r := -\alpha^r$；当 $x_p > u_p$ 令 $\tilde{\alpha}^r := \alpha^r$；

2: $\mathcal{Q} := \left\{ j : j \in \mathcal{N}, x_j \text{ 自由 或者} \left(x_j = l_j, \tilde{\alpha}_j^r > 0 \right) \text{ 或者} \left(x_j = u_j, \tilde{\alpha}_j^r < 0 \right) \right\}$；

3: **while** $\mathcal{Q}$ 非空且 $\delta \geq 0$ **do**

4: 选取 q，使得 $q \in \arg\min_{j \in \mathcal{Q}} \left\{ \frac{d_j}{\tilde{\alpha}_j^r} \right\}$；

5: $\mathcal{Q} := \mathcal{Q} \setminus \{q\}$；

6: $\delta := \delta - (u_q - l_q)|\alpha_q^r|$；

7: **end while**

8: **if** $\mathcal{Q}$ 为空集 **then**

9: 问题无界。

10: **else**

11: $\theta^d = \frac{d_q}{\alpha_q^r}$。

12: **end if**

2.4 对偶单纯形一阶段

在前面的算法描述中，我们以一个对偶可行解为初始进行迭代。通常来说对偶可行解是无法直接获取的，那么我们就需要在二阶段之前寻找对偶可行解，这便是一阶段的主要作用。可以首先任意取一组解，使其从对偶不可行逐步迭代为对偶可行。

对偶不可行主要存在两种情形：原始变量处于下界，其对偶逻辑变量小于 0；原始变量处于上界，其对偶逻辑变量大于 0。对于有上下界的且对偶不可行变量，可通过如下方法变换上下界 (2.27)，使其对偶可行。在此变换之后，对偶不可行的两种情况为：

$P = \left\{ j : d_j > 0 \text{，} x_j \text{固定} \vee x_j = u_j \right\}$ 和 $M = \left\{ j : d_j < 0 \text{，} x_j \text{固定} \vee x_j = l_j \right\}$。

总的对偶不可行可用函数表示为：

$$f = \sum_{j \in M} d_j - \sum_{j \in P} d_j. \tag{2.30}$$

由集合定义知 $\mathbf{f} \leq 0$ 恒成立，当 $\mathbf{f} = 0$ 时，就达到了对偶可行。寻找对偶可行解，就转化为优化问题。同样的假设选择 p 出基，步长为 t，可以得到：

$$d_j^{(p)}(t) = d_j - t\alpha_{pj}. \tag{2.31}$$

那么目标函数为：

$$f^{(p)}(t)=\sum_{j\in M}d_j^{(p)}(t)-\sum_{j\in P}d_j^{(p)}(t)=f^{(p)}(0)-t\left(\sum_{j\in M}\alpha_{pj}-\sum_{j\in P}\alpha_{pj}\right).$$

目标函数变化量为：

$$\Delta f=f(t)-f(0)=-t\left(\sum_{j\in M}\alpha_{pj}-\sum_{j\in P}\alpha_{pj}\right). \tag{2.32}$$

记 $v_p=\sum_{j\in M}\alpha_{pj}-\sum_{j\in P}\alpha_{pj}$，则需要 $-tv_p>0$。对于 v 的计算可用一次方程组求解即可，

$$v=\sum_{j\in M}\alpha_j-\sum_{j\in P}\alpha_j=B^{-1}\left(\sum_{j\in M}a_j-\sum_{j\in P}a_j\right)=B^{-1}\tilde{a}. \tag{2.33}$$

出基入基后 (假设进基为 q)，

$$\bar{d}_j=d_j-\frac{d_q}{\alpha_{pq}}\alpha_{pj},\quad \forall j\in\mathcal{N}.$$

分析可知，当 $t\geq 0$，$d_j<0$, $\alpha_{pj}<0$ 或者 $d_j\geq 0$, $\alpha_{pj}>0$；当 $t\leq 0$，$d_j<0$, $\alpha_{pj}>0$ 或者 $d_j\geq 0$, $\alpha_{pj}<0$。在出基入基选择以及更新可与前面方法相同使用。由此不断迭代可找到对偶可行解或者发现对偶不可行。

算法 5 对偶单纯形 I 阶段算法

1: 当 $x_p<l_p$，令 $\tilde{\alpha}^r:=-\alpha^r$；当 $x_p>u_p$ 令 $\tilde{\alpha}^r:=\alpha^r$；

2: $\mathcal{Q}:=\left\{j:j\in\mathcal{N},x_j\text{ 自由 }\text{或者}\left(x_j=l_j,\tilde{\alpha}_j^r>0\right)\text{ 或者}\left(x_j=u_j,\tilde{\alpha}_j^r<0\right)\right\}$；

3: **while** $\mathcal{Q}$ 非空且 $\delta\geq 0$ **do**

4: 　选取 q，使得 $q\in\arg\min_{j\in\mathcal{Q}}\left\{\frac{d_j}{\tilde{\alpha}_j^r}\right\}$；

5: 　$\mathcal{Q}:=\mathcal{Q}\setminus\{q\}$；

6: 　$\delta:=\delta-(u_q-l_q)|\alpha_q^r|$；

7: **end while**

8: **if** $\mathcal{Q}$ 为空集 **then**

9: 　问题无界;

10: **else**

11: 　$\theta^d=\frac{d_q}{\alpha_q^r}$。

12: **end if**

2.5 数值实验

本次实验基于我们开发的线性规划求解器 CZLP 中对偶单纯形算法部分，分别测试了本章节描述的出基入基算法，来验证此求解器的可行性以及是否达到预期。测试问题来自 LP 问题测试库 Netlib[30]。实验所用 CPU 型号为 Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz，系统为 Ubuntu 20.04.1 LTS。

实验结果如下表，其中 Row 表示 LP 问题的行数，Col 表示列数，NNZ 表示非零元个数，Dantzig 表示 Dantzig 方法求解的迭代次数，DSE 表示最陡边方法的迭代次数，diff 表示两种方法迭代次数差值，其中正数表示最陡边方案更优，Rel 表示相对差值。

表 2.2 Dantzig 与 DSE 迭代次数对比

Table 2.2 Comparison of iteration times between Dantzig and DSE

Name	Row	Col	NNZ	Dantzig	DSE	Diff	Rel
25fv47	822	1571	10400	12529	2965	9564	0.763
80bau3b	2263	9799	21002	7988	5566	2422	0.303
adlittle	57	97	383	108	88	20	0.185
afiro	28	32	83	27	25	2	0.074
agg2	517	302	4284	799	623	176	0.220
agg3	517	302	4300	713	626	87	0.122
bandm	306	472	2494	914	745	169	0.185
beaconfd	174	262	3375	107	102	5	0.047
blend	75	83	491	148	119	29	0.196
bnl1	644	1175	5121	1513	912	601	0.397
bnl2	2325	3489	13999	2872	1841	1031	0.359
boeing1	441	384	3819	1088	659	429	0.394
boeing2	186	143	1283	225	198	27	0.120
bore3d	234	315	1429	277	237	40	0.144
brandy	221	249	2148	583	281	302	0.518
capri	272	353	1767	404	381	23	0.057
czprob	930	3523	10669	3268	2969	299	0.091

表 2.2 Dantzig 与 DSE 迭代次数对比 (续表)

Name	Row	Col	NNZ	Dantzig	DSE	Diff	Rel
d2q06c	2172	5167	32417	30493	6418	24075	0.790
d6cube	416	6184	37704	711	539	172	0.242
degen2	445	534	3978	1562	1175	387	0.248
degen3	1504	1818	24646	9773	6270	3503	0.358
e226	224	282	2578	571	386	185	0.324
etamacro	401	688	2409	1045	634	411	0.393
fffff800	525	854	6227	1262	1035	227	0.180
finnis	498	614	2310	448	380	68	0.152
fit1d	25	1026	13404	973	489	484	0.497
fit1p	628	1677	9868	3164	801	2363	0.747
fit2d	26	10500	129018	8591	4601	3990	0.464
fit2p	3001	13525	50284	30000	7198	22802	0.760
forplan	163	421	4564	269	215	54	0.201
ganges	1310	1681	6912	1598	1438	160	0.100
gfrd-pnc	617	1092	2377	641	636	5	0.008
grow7	141	301	2612	167	245	-78	-0.467
israel	175	142	2269	360	187	173	0.481
kb2	44	41	286	65	52	13	0.200
lotfi	154	308	1078	314	274	40	0.127
modszk1	688	1620	3168	686	681	5	0.007
nesm	751	2923	13680	6815	3701	3114	0.457
pilot4	411	1000	5141	2788	1154	1634	0.586
pilot-we	723	2789	9126	8136	2895	5241	0.644
recipe	92	180	663	46	50	-4	-0.087
sc105	106	103	280	243	238	5	0.021
sc205	206	203	551	453	453	0	0.000
sc50a	51	48	130	55	56	-1	-0.018
sc50b	51	48	118	49	48	1	0.020

表 2.2 Dantzig 与 DSE 迭代次数对比 (续表)

Name	Row	Col	NNZ	Dantzig	DSE	Diff	Rel
scagr25	472	500	1554	863	640	223	0.258
scagr7	130	140	420	252	196	56	0.222
scfxm1	331	457	2589	732	606	126	0.172
scfxm2	661	914	5183	2728	2211	517	0.190
scfxm3	991	1371	7777	4017	3358	659	0.164
scorpion	389	358	1426	367	330	37	0.101
scrs8	491	1169	3182	867	821	46	0.053
scsd1	78	760	2388	106	95	11	0.104
scsd6	148	1350	4316	394	260	134	0.340
scsd8	398	2750	8584	1617	954	663	0.410
sctap1	301	480	1692	334	316	18	0.054
sctap2	1091	1880	6714	847	767	80	0.094
sctap3	1481	2480	8874	1084	1027	57	0.053
seba	523	1028	4367	440	439	1	0.002
share1b	118	225	1151	431	346	85	0.197
share2b	97	79	694	134	94	40	0.299
shell	537	1775	3556	676	562	114	0.169
ship04l	403	2118	6332	395	406	-11	-0.028
ship04s	403	1458	4352	399	400	-1	-0.003
ship08l	779	4283	12802	676	718	-42	-0.062
ship08s	779	2387	7114	635	672	-37	-0.058
ship12l	1152	5427	16170	1173	1213	-40	-0.034
ship12s	1152	2763	8178	1087	1087	0	0.000
sierra	1228	2036	7302	652	591	61	0.094
stair	357	467	3856	506	399	107	0.211
standata	360	1075	3031	93	72	21	0.226
standgub	362	1184	3139	93	72	21	0.226
standmps	468	1075	3679	220	215	5	0.023

表 2.2 Dantzig 与 DSE 迭代次数对比 (续表)

Name	Row	Col	NNZ	Dantzig	DSE	Diff	Rel
stocfor1	118	111	447	127	122	5	0.039
stocfor2	2158	2031	8343	3065	2912	153	0.050
truss	1001	8806	27836	11775	20311	-8536	-0.725
tuff	334	587	4520	469	291	178	0.380
vtp-base	199	203	908	250	187	63	0.252
wood1p	245	2594	70215	299	377	-78	-0.261
woodw	1099	8405	37474	2660	1342	1318	0.495

由实验数据知，我们的求解器初步求解了 netlib 中 80 个问题。在对比出基方法时可以看出，最陡边方法在绝大所数问题中占优。特别是在迭代次数比较多的问题上，减小迭代次数更多。而最陡边方法迭代次数不占优的例子中大多相差不大。通过已有实验结果可以得出，最陡边方法相比较于 Dantzig 方法有普遍优势。

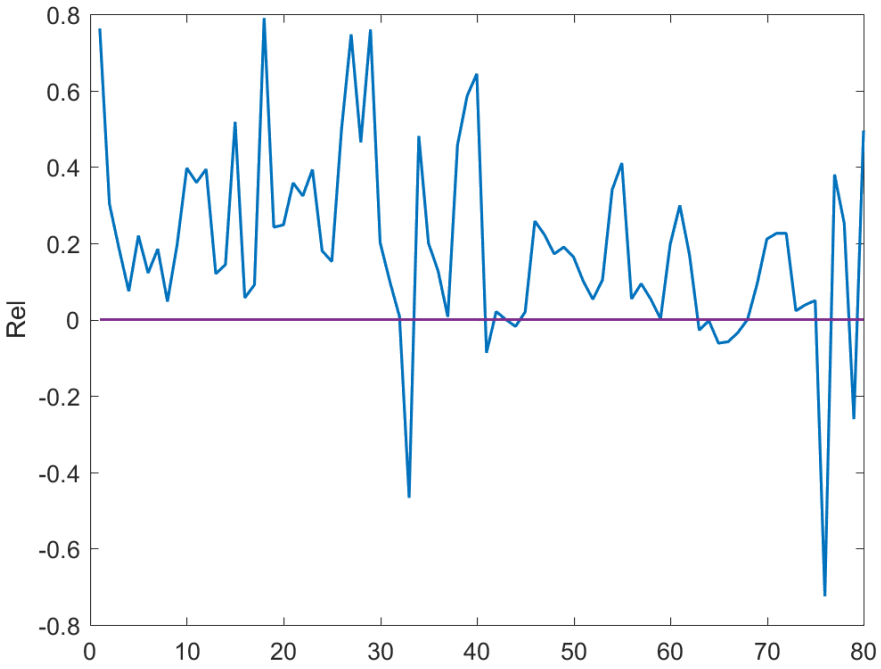

图 2.1 迭代次数降低相对比率

Figure 2.1 Relative ratio of iterations reduction

2.6 小结

本章介绍了对偶单纯形算法以及不同出基入基方案，并详述了方法推理。在此基础上，初步开发了线性规划求解器 CZLP，并使用了不同方法求解在 netlib 测试库中成功求解了大多数问题，初步完成目标。

第 3 章　时序网络拥塞优化问题的预处理方法

时序网络拥塞优化问题的构建基于不同时间的状态，在自身网络规模上再加上时间段因素导致问题规模巨幅增大。在解决大规模线性规划问题上，预处理方法非常重要和有效。因此研究预处理方法对于简化时序网络拥塞优化问题和提高求解效率非常重要。本章节介绍了一些已有的预处理方法及其原理，同时在在研究时序网络拥塞优化问题本身性质上提出了特有的预处理方法。

3.1　预处理方法介绍

预处理的过程就是简化问题的系数矩阵，并可能需要重复此过程。在每个化简过程中要进行适当处理，减小系数矩阵的规模兼顾减少非零元数量来减少计算所需的储存空间。此模型问题规模庞大，因此首先考虑其预处理。下面介绍一些简单预处理，复杂预处理。

3.1.1　简单预处理

3.1.1.1　不可行性

在系数矩阵 A 中，如果某个变量的上界小于它的下界或约束条件左右两边出现矛盾，此时问题不可行。

$$\exists\, j : l_j > u_j,$$

或

$$\exists\, i : \sum_{j\in M_i} a_{ij}l_j + \sum_{j\in P_i} a_{ij}u_j < b_i \vee \sum_{j\in P_i} a_{ij}l_j + \sum_{j\in M_i} a_{ij}u_j > b_i. \tag{3.1}$$

其中，$P_i = \{j : a_{ij} > 0\}, M_i = \{j : a_{ij} < 0\}$，$l_j$ 为第 j 个变量的下界，u_j 为第 j 个变量的上界。如果有以上情况发生则可以判断出问题模型无解，则不用对其进行求解。

3.1.1.2　空行和空列

在系数矩阵 A 中，如果某行或某列的系数全为零，则称之为空行或空列。

$$\exists\, i : a_{ij} = 0 \quad \forall\, j, \text{ 或 } \exists\, j : a_{ij} = 0 \quad \forall\, i.$$

如果是空行或空列则可以将该行系数或列系数从矩阵 A 中删除，并修正右端向量 b。

3.1.1.3 固定变量

在系数矩阵 A 中，如果某个变量的上界等于下界，则可称为固定变量。

$$\exists\, j : l_j = u_j.$$

如果有这种情况发生则记录该变量的值，并删除变量所对应的列系数，修正右端向量 b。

3.1.1.4 单独行

在系数矩阵 A 中，如果某行有且只有一个变量不为 0，则可称为单独行。

$$\exists\,(i,k) : a_{ij} = 0,\ \forall j \neq k, a_{ik} \neq 0,$$

此时可固定约束 i 的变量 x_j 为 b_i/a_{ik}。一次单独行的变量固定可能导致出现新的单独行。我们的预处理过程首先计算每个约束中的非零数。然后，创建一个包含所有单独行约束的列表。这个列表中的约束用于固定变量，每个约束的非零元个数都会被更新，并且新的单独行约束会被添加到列表中。

3.1.2 复杂预处理

3.1.2.1 单独列

在系数矩阵 A 中，如果某列只有一个非零元，其他为 0，则称此列为单独列。即，

$$\exists\,(i,k) : a_{ij} = 0, a_{ik} \neq 0, \forall\, j, \quad j \neq k.$$

如果 x_k 是自由变量，则称此列为自由单独列。对于自由单独列，可直接删除该列，并用公式

$$x_k = \frac{b_i - \sum_{p=1,p\neq k}^{n} a_{ip}x_p}{a_{ik}}, \tag{3.2}$$

记录该列对应的变化，用于求解完成后还原原始解。

如果该列不是自由单独列，则需要判断此列能否成为自由单独列。如果是潜在的自由列，也可以采用同样的处理方法。设该列唯一的非零元是 a_{ik}。

如果 $a_{ij} > 0$:

$$L = (b_i - \sum_{j\in P_i} a_{ij}u_j - \sum_{j\in M_i} a_{ij}l_j)/a_{ik},$$

$$U = (b_i - \sum_{j \in P_i} a_{ij} l_j - \sum_{j \in M_i} a_{ij} u_j)/a_{ik}).$$

如果 $a_{ij} > 0$:

$$L = (b_i - \sum_{j \in P_i} a_{ij} l_j - \sum_{j \in M_i} a_{ij} u_j)/a_{ik},$$
$$U = (b_i - \sum_{j \in P_i} a_{ij} u_j - \sum_{j \in M_i} a_{ij} l_j)/a_{ik}).$$

其中，$P_i = \{j : a_{ij} > 0, j \neq k\}, M_i = \{j : a_{ij} < 0, j \neq k\}$，$l_j$ 为第 j 个变量的下界，u_j 为第 j 个变量的上界。

如果 a_{ik} 所对应的变量范围在 L 和 U 之间，即 $L \leq l_i \leq x_k \leq u_i \leq U$，则说明该列是一个潜在的自由单独列。

3.1.2.2　强制列

首先找到所有的单独列，并利用它们计算最优拉格朗日乘子 y^* 的隐含上下限。令 $\bar{y}_i$ 和 $\hat{y}_i$ 是所有这些界的最大值和最小值，也就是

$$\bar{y}_i \leq y_i^* \leq \hat{y}_i, \quad \forall i.$$

这些拉格朗日乘数的边界可以用来固定一些原始变量。定义

$$e_j = \sum_{i \in P_j} a_{ij} \bar{y}_i + \sum_{i \in M_j} a_{ij} \hat{y}_i, \tag{3.3}$$

$$d_j = \sum_{i \in P_j} a_{ij} \hat{y}_i + \sum_{i \in M_j} a_{ij} \bar{y}_i. \tag{3.4}$$

其中，$P_i = \{j : a_{ij} > 0\}, M_i = \{j : a_{ij} < 0\}$，则强制列可定义为：

$$\exists j \notin S : (c_j - e_j = 0 \wedge u_j = \infty) \vee (c_j - d_j = 0 \wedge l_j = -\infty).$$

若 $c_j - e_j = 0$ 且 $u_j = \infty$，则

$$y_k^* = \begin{cases} \bar{y}_k, & a_{kj} > 0, \\ \hat{y}_k, & a_{kj} < 0. \end{cases} \tag{3.5}$$

当 $a_{kj} \neq 0$ 时，如果从目标函数中减去 y_k^* 乘以第 k 个约束条件，第 k 个约束条件可以从问题中去除。如果 $a_{kj} \neq 0$，则

$$y_k^* \leq \frac{c_j - e_j}{a_{kj}} + \bar{y}_k, \quad a_{kj} > 0,$$

或

$$y_k^* \geq \frac{c_j - e_j}{a_{kj}} + \hat{y}_k, \quad a_{kj} < 0.$$

利用由上式生成的拉格朗日乘数 y^* 的新界可以重新计算 e_j 和 d_i。

3.1.2.3 强制行

在系数矩阵 A 中，如果某行约束条件最大值或最小值等于右端项的值，则称为强制行。

$$\exists\, i : \sum_{j \in M_i} a_{ij} l_j + \sum_{j \in P_i} a_{ij} u_j = b_i, \tag{3.6}$$

或

$$\exists\, i : \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in M_i} a_{ij} u_j = b_i. \tag{3.7}$$

其中，$P_i = \{j : a_{ij} > 0\}, M_i = \{j : a_{ij} < 0\}$，$l_j$ 为第 j 个变量的下界，u_j 为第 j 个变量的上界。若满足公式 (3.6)，该行正系数所对应的变量取上界，负系数所对应的变量值取下界。记录这些固定的变量的值，并采取与固定变量相同的方法。同理，若满足公式 (3.7)，则该行正系数所对应的变量取下界，负系数所对应的变量值取上界。记录这些固定的变量的值，并采取与固定变量相同的方法。

3.1.2.4 比例列

在系数矩阵 A 中，如果有两列非零元位置相同且对应成比例，则这两列为比例列。

$$\exists\, (j,k) : a_{ij} = r a_{ik}, \quad \forall i \wedge j \neq k.$$

其中 r 为比例系数。在简化比例列时，选其中一列为基准列，计算目标函数值差异

$$s = c_b - r c_u. \tag{3.8}$$

其中 c_b 为基准列系数所对应的目标函数值系数，c_u 为非基准列系数所对应的目标函数值系数，r 为非基准列系数与基准列系数的比值。如果 $s < 0$ 则选取另一列为基准列。

设第 k 列和第 j 列为比例列，j 列为基准列，则 $a_j = r a_k$，即

$$a_j x_j + a_k x_k = a_k (r x_j + x_k).$$

两列所对应的目标函数系数

$$\begin{aligned} c_j x_j + c_k x_k &= r c_k x_j - r c_k x_j + c_j x_j + c_k x_k \\ &= c_k (r x_j + x_k) + (c_j - r c_k) x_j. \end{aligned} \tag{3.9}$$

令 $x_k = r x_j + x_k$，为两列所对应变量合并的新变量，若 $r > 0$，则 x_k 的界限为 $r l_j + l_k \le x_k \le r u_j + u_k$；若 $r < 0$，则 x_k 的界限为 $r u_j + l_k \le x_k \le r l_j + u_k$。

考虑指标差异 s 的不同取值情况，当 $c_j - r c_k = 0$ 时，可直接删除非基准列，用新的变量 x_k 来代替基准列变量。当 $c_j - r c_k > 0$ 时，为了使目标值最小，取 $x_j = l_j$。通过交换基准列和非基准列来保持 $s > 0$，从而使选取的非基准变量总是取其下界值。

综上所述，我们能够将比例列对应的两个变量合并为一个新变量，并记录其幻化，在求解结束后对还原它们真实的值，只需求解以下方程

$$\begin{aligned} x_k &= r x_j + x_k, \\ l_j &\le x_j \le u_j, \\ l_k &\le x_k \le u_k. \end{aligned} \tag{3.10}$$

当 $s \neq 0$ 时，非基准变量的值可以固定在其下界。

3.1.2.5　比例行

在系数矩阵 A 中，如果某两行非零元位置相同且对应成比例，则这两行为比例行。

$$\exists\,(j,k):\ a_{ij} = r a_{kj}, b_i = r b_k, \quad \forall\, j \wedge i \neq k,$$

其中 r 为比例系数。如果某两行为比例行，若 $b_i = r b_k$，则可删除 i 或 k 行；若 $b_i \neq r b_k$，则 LP 模型无解。

3.2　基于模型的预处理

3.2.1　固定路径分流比

对于原问题，通过数值实验可以发现，对于某些任务只有唯一路径可供选择，定义有唯一路径的任务集合为：

$$K_0 := \{k : k\text{的路径唯一}\},$$

则此需求的分流比唯一且确定：

$$Y_p^k = 1, \qquad \forall k \in K_0. \tag{3.11}$$

由此固定了某些单一路径变量，从而直接减少模型规模。

3.2.2 界缩紧

经过固定变量后，边的容量约束 b_e 去除固定变量的容量后记为 $b_e^{'}$，对于每条边的约束为：

$$\sum_{(k,p)\in P_e, k\notin K_0} Y_p^k y_k^t \le b_e^{'}, \qquad \forall e \in E, \forall t = 1, \dots, N,$$

已知模型中约束 $Y_p^k \ge \frac{1}{16}, \forall (k,p) \in P_e, e \in E, \forall t = 1, \dots, N$，对于某一边 e_0，其剩余最大容量为：

$$R_{max} = b_{e_0}^{'} - \sum_{(k,p)\in P_{e_0}, k\notin K_0} \frac{1}{16} y_k^t. \tag{3.12}$$

1 如果 $R_{max} < 0$，问题无可行解；

2 如果 $R_{max} = 0$，则 $Y_p^k = \frac{1}{16}, \forall (k,p) \in P_{e_0}$，再次固定变量;

3 如果 $R_{max} > 0$，则对于经过此边任务 k 的路径的分流比最大为 $\frac{R_{max}}{y_k^t} + \frac{1}{16}$，即：

$$Y_p^k \le \frac{R_{max}}{y_k^t} + \frac{1}{16}, \tag{3.13}$$

得到变量的一个上界。

在计算中，循环对每条边做这样的计算，就可以的到每个分流比的上界，从而尽可能的缩紧变量的界。

3.2.3 冗余行

经过界缩紧之后，变量的约束更加精确，记变量 Y_p^k 的上界为 $U_{Y_p^k}$，如果

$$\sum_{(k,p)\in P_e} U_{Y_p^k} y_k^t \le b_e,$$

则约束 $x_t^e \le b_e$ 为冗余约束，可以直接去除。更进一步，如果

$$\sum_{(k,p)\in P_e} U_{Y_p^k} y_k^t \le \alpha b_e,$$

表明此边链路不会拥塞，$z_t^e \ge 0, z_t^e \ge x_t^e - \alpha b_e$ 为冗余约束，且可以得到 $z_t^e = 0$。

3.2.4　时序块消除

时序网络拥塞优化模型，系数矩阵为：

$$\begin{bmatrix} \{Y_p^k\} & \{x_t^e\} & \{z_t^e\} & \{u_t\} \\ \hline A & -I & 0 & 0 \\ 0 & -I & I & 0 \\ 0 & -I & 0 & E \\ B & 0 & 0 & 0 \end{bmatrix}$$

考虑以下简易问题，

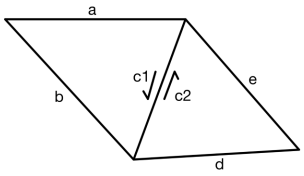

图 3.1 简易问题网络图

Figure 3.1 Network graph of a simple problem

$K := \{1,2\}, E := \{a,b,c_1,c_2,d,e\},$

$Path := \{\langle 1\rangle\{a,e\}, \langle 2\rangle\{a,c_1,d\}, \langle 3\rangle\{b,d\}, \langle 4\rangle\{b,c_2,e\}\},$

$P_1 := \{\langle 1\rangle, \langle 2\rangle\}, P_2 := \{\langle 1\rangle, \langle 3\rangle, \langle 4\rangle\}.$

Y_p^k 的系数矩阵可以写为，

$$\begin{array}{c|ccc|cc|c} & Y^1_{\langle 1\rangle} & Y^1_{\langle 2\rangle} & & Y^2_{\langle 1\rangle} & Y^2_{\langle 3\rangle} & Y^2_{\langle 4\rangle} & \cdots \\ \hline a: & y_1^{t_1} & y_1^{t_1} & & y_2^{t_1} & & & \\ b: & & & & & y_2^{t_1} & y_2^{t_1} & \\ c_1: & & y_1^{t_1} & & & & & \\ c_2: & & & & & & y_2^{t_1} & \\ d: & & y_1^{t_1} & & & y_2^{t_1} & & \\ e: & y_1^{t_1} & & & y_1^{t_1} & & y_2^{t_1} & \\ \hline a: & y_1^{t_2} & y_1^{t_2} & & y_2^{t_2} & & & \\ b: & & & & & y_2^{t_2} & y_2^{t_2} & \\ c_1: & & y_1^{t_1} & & & & & \\ c_2: & & & & & & y_2^{t_1} & \\ d: & & y_1^{t_2} & & & y_2^{t_2} & & \\ e: & y_1^{t_2} & & & y_1^{t_2} & & y_2^{t_2} & \end{array} .$$

利用分块矩阵可表示为:

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1T} & A_{1T+1} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2T} & A_{2T+1} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \\ A_{T1} & A_{T2} & \cdots & A_{TT} & A_{TT+1} & \cdots & A_{TK} \end{bmatrix}.$$

由于分块矩阵的上下快之间的非零元位置完全相同，可以将块矩阵看做一个整体在上下相加减，且不会引入新的非零元。经过上下块消除可以得到：

$$\begin{bmatrix} A'_{11} & 0 & \cdots & 0 & A'_{1T+1} & \cdots & A'_{1K} \\ 0 & A'_{22} & \cdots & 0 & A'_{2T+1} & \cdots & A'_{2K} \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \\ 0 & 0 & \cdots & A'_{TT} & A'_{TT+1} & \cdots & A'_{TK} \end{bmatrix}.$$

此时原问题主要的非零元部分的非零元数目可以巨幅减小。

3.2.5 同路径三角化消除

同样以图3.1为例，任务 1，2 都经过路径 ⟨1⟩，可以发现在列上的非零元位置相同，可知列变换不会引入新的非零元。从左向右看在每个时间段的列成比例。可以把某一时间的列看做一个整体，进行列运算。与3.2.4节进行类似的消去，只是对列进行消除，同时进行变量变换。

	$Y^1_{\langle 1\rangle}$	$Y^1_{\langle 2\rangle}$	$Y^2_{\langle 1\rangle}$	$Y^2_{\langle 3\rangle}$	$Y^2_{\langle 4\rangle}$	$\cdots$
$a:$	$y_1^{t_1}$	$y_1^{t_1}$	$y_2^{t_1}$			
$b:$				$y_2^{t_1}$	$y_2^{t_1}$	
$c_1:$		$y_1^{t_1}$				
$c_2:$					$y_2^{t_1}$	
$d:$		$y_1^{t_1}$		$y_2^{t_1}$		
$e:$	$y_1^{t_1}$		$y_2^{t_1}$		$y_2^{t_1}$	
$a:$	$y_1^{t_2}$	$y_1^{t_2}$	$y_2^{t_2}$			
$b:$				$y_2^{t_2}$	$y_2^{t_2}$	
$c_1:$		$y_1^{t_1}$				
$c_2:$					$y_2^{t_1}$	
$d:$		$y_1^{t_2}$		$y_2^{t_2}$		
$e:$	$y_1^{t_2}$		$y_2^{t_2}$		$y_2^{t_2}$	

例 3.1. A_{row} 表示时序块消除后矩阵，A_{col} 表示同路径列消除后矩阵。

$$A=\left[\begin{array}{ccccc}
40 & 40 & 60 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 \\
0 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 \\
0 & 40 & 0 & 60 & 0 \\
40 & 0 & 60 & 0 & 60 \\
\hline
50 & 50 & 60 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 \\
0 & 50 & 0 & 60 & 0 \\
50 & 0 & 60 & 0 & 60 \\
\hline
45 & 45 & 55 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 45 & 0 & 55 & 0 \\
45 & 0 & 55 & 0 & 55 \\
\hline
50 & 50 & 55 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 50 & 0 & 55 & 0 \\
50 & 0 & 55 & 0 & 55
\end{array}\right] A_{row}=\left[\begin{array}{ccccc}
40 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 40 & 0 & 0 & 0 \\
40 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & -15 & 0 & 0 \\
0 & 0 & 0 & -15 & -15 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -15 \\
0 & 0 & 0 & -15 & 0 \\
0 & 0 & -15 & 0 & -15 \\
\hline
45 & 45 & 55 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 45 & 0 & 55 & 0 \\
45 & 0 & 55 & 0 & 55 \\
\hline
50 & 50 & 55 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 50 & 0 & 55 & 0 \\
50 & 0 & 55 & 0 & 55
\end{array}\right]$$

$$A=\left[\begin{array}{ccccc}
40 & 40 & 60 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 \\
0 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 \\
0 & 40 & 0 & 60 & 0 \\
40 & 0 & 60 & 0 & 60 \\
\hline
50 & 50 & 60 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 \\
0 & 50 & 0 & 60 & 0 \\
50 & 0 & 60 & 0 & 60 \\
\hline
45 & 45 & 55 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 45 & 0 & 55 & 0 \\
45 & 0 & 55 & 0 & 55 \\
\hline
50 & 50 & 55 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 50 & 0 & 55 & 0 \\
50 & 0 & 55 & 0 & 55
\end{array}\right] A_{col}=\left[\begin{array}{ccccc}
40 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 \\
0 & 40 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 \\
0 & 40 & 0 & 60 & 0 \\
40 & 0 & 0 & 0 & 60 \\
\hline
0 & 50 & -15 & 0 & 0 \\
0 & 0 & 0 & 60 & 60 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 60 \\
0 & 50 & 0 & 60 & 0 \\
0 & 0 & -15 & 0 & 60 \\
\hline
3.33 & 45 & -12.5 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 45 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 45 & 0 & 55 & 0 \\
3.33 & 0 & -12.5 & 0 & 55 \\
\hline
-16.67 & 50 & -20 & 0 & 0 \\
0 & 0 & 0 & 55 & 55 \\
0 & 50 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 55 \\
0 & 50 & 0 & 55 & 0 \\
-16.67 & 0 & -20 & 0 & 55
\end{array}\right]$$

3.3 数值实验

本实验预处理测试基于开源求解器 HiGHS，分别测试了 HiGHS 原方案以及加入新的预处理方案后再使用 HiGHS 求解。实验所用 CPU 型号为 Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz，系统为 Ubuntu 20.04.1 LTS。下表给出了问题的信息其中 Name 表示问题名称，Link 表示网络图边的数量，Path 表示业务总的路径数，数字越大，规模越大。

表 3.1 测试问题信息

Table 3.1 Informations of test problems

Name	Link	Demand	Path
N600	2400	1000	1658
N800	3200	2000	3107
N1000	4000	4000	5621
N1200	4800	8000	11429
N1400	5600	10000	12489
N1600	6400	12000	14642
N1800	7200	15999	19113
N2000	8000	16000	18696

表 3.2 HiGHS 求解结果

Table 3.2 Result solved by HiGHS

Name	Row	Col	NNZ	Pre Row	Pre Col	Pre NNZ	iter	Pre Time	Tol Time
N600	217000	145690	666790	41740	23532	206999	7594	0.92	1.61
N800	290000	195140	1083860	63276	36210	340676	12161	1.63	2.83
N1000	364000	245659	1711429	65914	44453	380477	15876	2.60	4.18
N1200	440000	299480	3090710	40783	28300	284114	10632	5.96	7.62
N1400	514000	348538	3482548	26708	14677	209199	6897	5.52	7.24
N1600	580000	398696	4100216	25746	14235	203412	6709	6.56	8.56
N1800	663999	451180	5164780	28065	18405	179617	12183	9.04	11.65
N2000	736000	498758	5255108	27508	14663	151505	7322	9.06	11.66

实验数数值结果在表3.2和3.3，在实际计算中取 $c_0 = \frac{1}{16}$，Row 表示生成问题的行数，Col 表示生成问题的列数，NNZ 表示非零元个数，PreRow 、PreCol、

PreNNZ 表示预处理后的行数、列数以及非零元个数。计算时间对比如图3.2和图3.3。

表 3.3 预处理 +HiGHS 求解结果

Table 3.3 Result solved by presolving + HiGHS

Name	Row	Col	NNZ	Pre Row	Pre Col	Pre NNZ	iter	Pre Time	Tol Time
N600	217000	145690	666790	41734	23529	206987	7598	0.73	1.39
N800	290000	195140	1083860	63131	36210	340676	13810	1.20	2.41
N1000	364000	245659	1711429	65658	44325	379965	15832	2.15	3.53
N1200	440000	299480	3090710	44353	30965	318329	10798	3.37	4.64
N1400	514000	348538	3482548	27153	14894	213856	6923	2.28	3.33
N1600	580000	398696	4100216	26594	14658	210375	6702	3.08	4.53
N1800	663999	451180	5164780	29834	19732	195914	12475	3.30	4.81
N2000	736000	498758	5255108	30022	16189	170801	7491	3.02	4.45

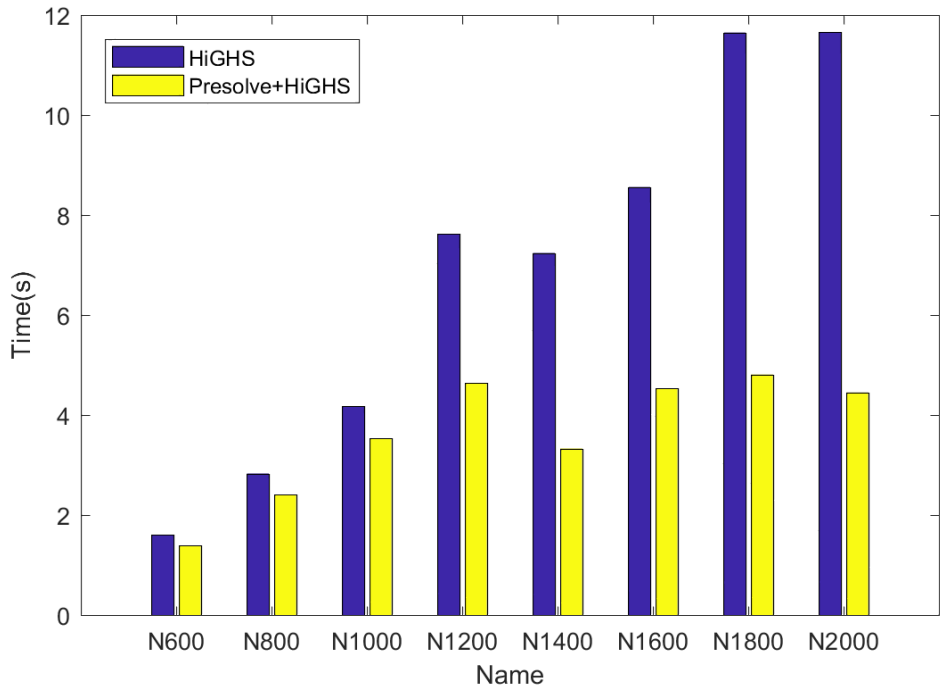

图 3.2 时间对比

Figure 3.2 Comparison of time

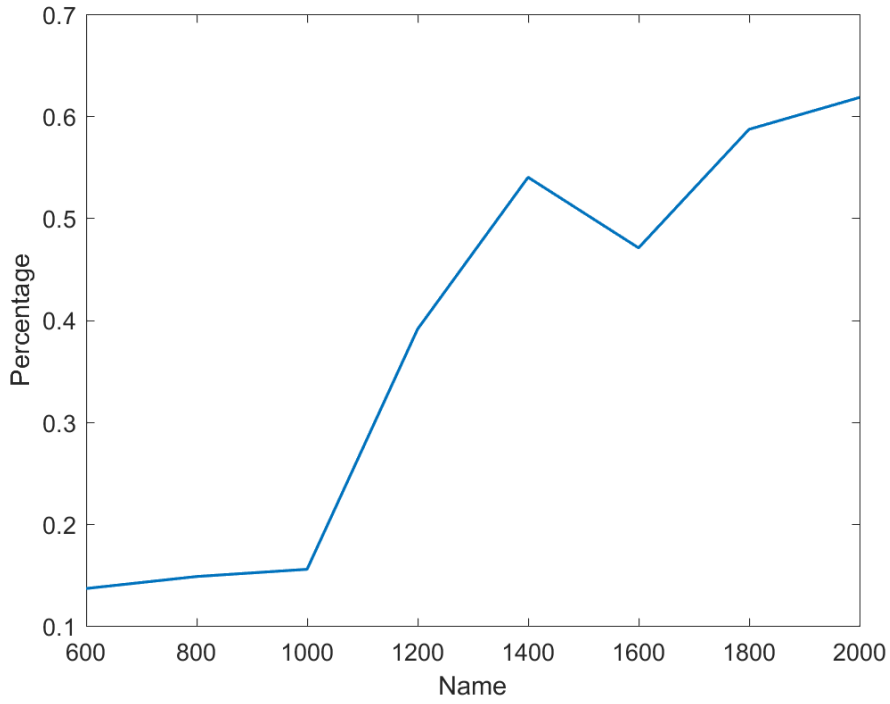

图 3.3 时间提升比率

Figure 3.3 Improvement ratio of time

由数值结果发现两者的迭代次数相近，但是加了基于模型的预处理之后，计算时间下降，求解速率增加。实验结果表明，我们添加的基于模型的预处理能有效降低矩阵的稀疏性，并提升计算效率。

3.4 小结

本章中介绍了时序网络拥塞优化问题以及预处理方法，在模型上通过引入新的变量减少了非零元数目。同时在解决问题中发现预处理对该类型问题的重要性，而后介绍了一些预处理方法，并依据模型提出基于模型的预处理方法，在计算效果上得到了显著提升。

第 4 章　对偶单纯形算法初始基选取及应用

在单纯形算法中一个好的初始基选取，可以大大减少计算的迭代次数，即出基入基的次数，从而节省计算量。最一般的方法是直接以单位矩阵作为基，即直接添加逻辑基。而后进行迭代计算。

4.1　最优解条件

考虑标准线性规划的对偶问题：

$$\begin{aligned} &\max\ f(\mathbf{y}) = \mathbf{b}^T\mathbf{y} \\ &\text{s.t.}\ \ A^T\mathbf{y} \le \mathbf{c}. \end{aligned} \tag{4.1}$$

为了确定 $\mathbf{y}$ 是否为最优解，假设 $\mathbf{d}$ 是 $f(\mathbf{y})$ 的一个可行方向 [9]，即满足:

$$f(\mathbf{y}+\mathbf{d}) > f(\mathbf{y}). \tag{4.2}$$

一阶泰勒展开得到：

$$\nabla f(\mathbf{y})^T\mathbf{d} > 0. \tag{4.3}$$

$\mathbf{d}$ 相对于 $\mathbf{y}$ 是一个微小变动，我们可以认为 $\mathbf{y}+\mathbf{d}$ 依然可行。记 $g_j(\mathbf{y}) = \mathbf{a}_j^T\mathbf{y}$，其中 $\mathbf{a}_j$ 表示矩阵 A 的第 j 列。可得：

$$g_j(\mathbf{y}+\mathbf{d}) = g_j(\mathbf{y}) + \nabla g_j(\mathbf{y})^T\mathbf{d} \le c_j. \tag{4.4}$$

当 $g_j(\mathbf{y}) < \mathbf{c}_j$，任意的方向 $\mathbf{d}$ 满足 (4.3) 和 (4.4)。此时不是最优，只有 $\nabla f(\mathbf{y}) = 0$，是最优解。

当 $g_j(\mathbf{y}) = \mathbf{c}_j$，则 $\mathbf{d}$ 需要满足：

$$\nabla g_j(\mathbf{y})^T\mathbf{d} \le 0. \tag{4.5}$$

可以得出如果 $\mathbf{y}^*$ 是最大值解，(4.3) 和 (4.5) 不能同时满足，即

$$(\nabla f(\mathbf{y}^*)^T\mathbf{d})(\nabla g_j(\mathbf{y}^*)^T\mathbf{d}) > 0. \tag{4.6}$$

在没有其他约束时，几何上的描述为 $\nabla f(\mathbf{y}^*)$ 和 $\nabla g_j(\mathbf{y}^*)$ 在任意 $\mathbf{d}$ 上的投影方向相同，

$$\nabla f\left(\mathbf{y}^*\right) - \mu_{\mathrm{j}}^*\nabla g_j\left(\mathbf{y}^*\right) = 0, \mu_j^* \ge 0. \tag{4.7}$$

通过以上分析可以发现，当 $\mu_j^* > 0$，此不等式约束是积极约束，可作为一个基；当 $\mu_j^* = 0$，为非积极约束。

在实际问题中，约束的个数是多个，因此 **d** 的方向并不是任意的。此时目标函数的梯度应为积极约束梯度的正向线性组合。(4.7) 需要变为

$$\nabla f\left(\mathbf{y}^*\right) = \sum \mu_{\mathrm{j}}^* \nabla g_j\left(\mathbf{y}^*\right), \mu_j^* \geq 0. \tag{4.8}$$

4.2 梯度最小夹角基选取

在基于函数梯度的线性问题数学模型的最优性条件下，观察到约束的梯度和目标函数的角度可以提供约束的信息，我们可以知道一个约束是积极的还是非积极的，这个前提被用来构造单纯余弦法。由 (4.6) 知，当 $\nabla f(\mathbf{y})$ 和 $\nabla g_j(\mathbf{y})$ 的夹角尽可能小的时候，其同时在 **d** 上的投影方向相同的可能性越大，如图4.1所示，x^* 表示最优解，θ_i 表示约束梯度与目标函数梯度之间夹角。

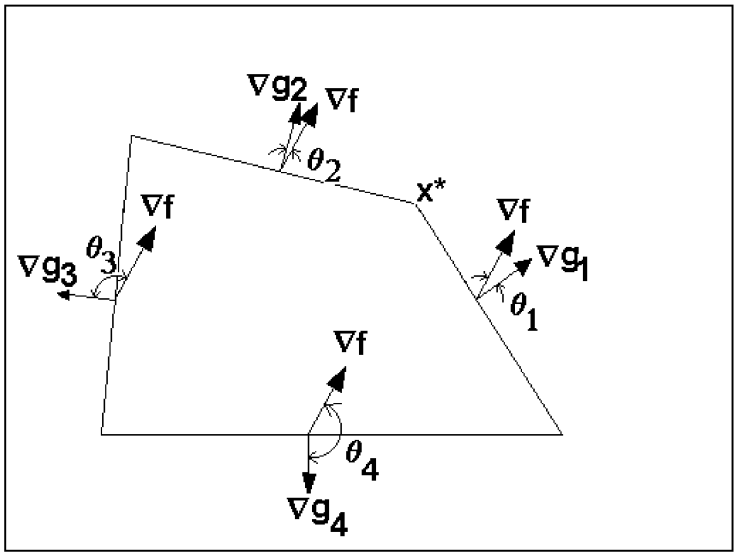

图 4.1 目标函数梯度与约束梯度夹角示意图

Figure 4.1 Graph of angle between objective function gradient and constraint gradient

与目标函数约束梯度夹角小约束梯度组成的基不一定是最优基，是不可能仅使用此信息来确定最佳值的。尽管如此，约束形成最小角度的顶点可以作为起

始点一样使用。为了确定两个向量之间的夹角，使用了向量的点积公式，

$$\begin{aligned}\cos\theta &= \frac{(\nabla f(\mathbf{y}))^{\top}\nabla g_j(\mathbf{y})}{\|\nabla f(\mathbf{y})\|\|\nabla g_j(\mathbf{y})\|} \\ &= \frac{b^{\top}a_j}{\|b\|\|a_j\|}\end{aligned} \tag{4.9}$$

其中 θ 表示向量 $\nabla f(\mathbf{y})$ 和 $\nabla g_j(\mathbf{y})$ 的夹角。算法6选择夹角最小足够基个数的变量作为基，来开始单纯形算法，在一定程度上可能减少迭代次数。

算法 6 梯度最小夹角角基选取 (*minium gradients angle* ,MGA)

1: **for** $i = 1, 2, \cdots, m$ **do**
2: $J = 1, 2, 3, \cdots, n$;
3: 选取 j_0，使得 $j_0 \in \arg\max\limits_{j\in J}\left\{\frac{b^{\top}a_j}{\|b\|\|a_j\|}\right\}$，作为基列数;
4: $J := J \setminus \{j_0\}$;
5: **end for**

例 4.1. 考虑如下线性规划问题，

$$\begin{aligned}\min\quad & 4x_1 + 6x_2 + 36x_3 + 38.5x_4 - x_5 \\ \text{s.t.}\quad & -x_1 + 3x_2 + 6x_3 + 5.5x_4 - x_5 \geq 6, \\ & 2x_1 - x_2 + 6x_3 + 7x_4 - x_5 \geq 5, \\ & x_1, x_2, x_3, x_4, x_5 \geq 0.\end{aligned} \tag{4.10}$$

其对偶问题为：

$$\begin{aligned}\max\quad & 6y_1 + 5y_2 \\ \text{s.t.}\quad & -1y_1 + 2y_2 \leq 4, \\ & 3y_1 - y_2 \leq 6, \\ & 6y_1 + 6y_2 \leq 36, \\ & 5.5y_1 + 7y_2 \leq 38.5, \\ & -y_1 - y_2 \leq -1, \\ & y_1, y_2 \geq 0.\end{aligned} \tag{4.11}$$

对偶问题的可行域如图4.2阴影部分所示，虚线部分指目标函数与各个约束的梯度方向。基于以上选择夹角最小的方法，对偶问题的第三个和第四个约束为基约束。对应到原问题，即选择第三列、第四列作为原问题的基。此时问题虽然没有达到最优，但与最优值很接近，只需按照单纯形算法再迭代一次即可。

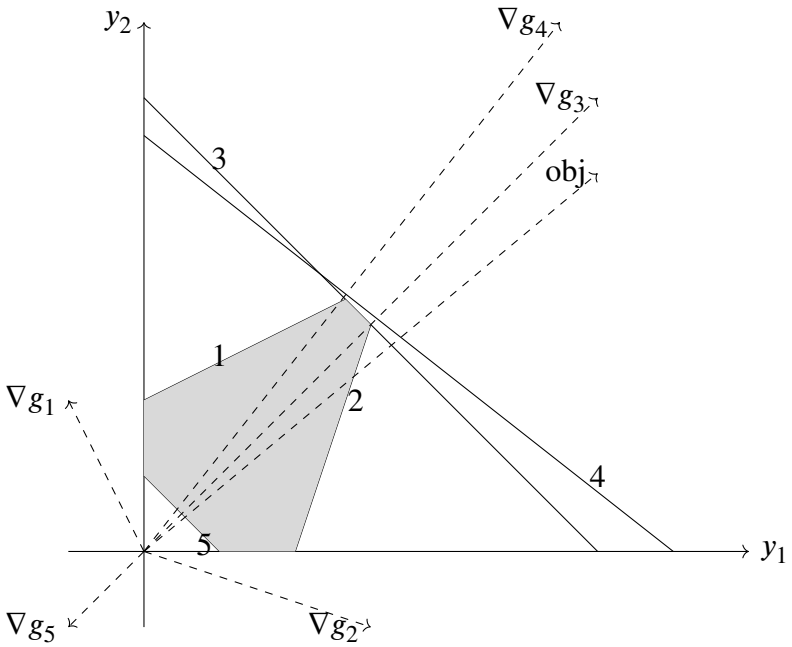

图 4.2 对偶模型可行域与梯度示意图

Figure 4.2 Feasible region and gradient diagram of dual model

4.3 算法改进

由对偶单纯形出基入基规则知，当原始变量全部可行时，问题达到最优。由 (2.18d) 知基变量中的原始变量的值在不断变化，且由 (2.27) 知，在非基列中有上下界的变量可以保持可行性。因此，在选择基的时候，对于原始变量为固定变量或者同时具有上下界的，便可以不作为基。

算法 7 梯度最小夹角角基选取改进 I(MGAI)

1: **for** $i = 1, 2, \cdots, m$ **do**

2: $J := \{j | x_j$非固定变量或者同时有上下界$\}$;

3: 选取 j_0，使得 $j_0 \in \arg\max_{j \in J} \left\{ \frac{b^\top a_j}{\|b\| \|a_j\|} \right\}$，作为基列数;

4: $J := J \setminus \{j_0\}$;

5: **end for**

经过数值实验发现算法6和算法7的基选取规则，相比较于 HiGHS 本身算法，在迭代次数都巨幅减少。特别是基于改进的算法7，在迭代次数减少更多，但是计算时间也有上涨。

经过分析，选择的基矩阵的稀疏性和条件树不太好，导致了在矩阵相关计算

时花费的时间较多，特别是在解方程时耗费大量时间。因此，我们考虑增加基列稀疏性的权重来使计算时间下降，由此得到新的选基算法。

算法 8 梯度最小夹角角基选取改进 II(MGAII)

1: **for** $i \leq m$ **do**

2: $J := \{j | x_j$非固定变量或者同时有上下界$\}$;

3: 选取 j_0，使得 $j_0 \in \arg\max_{j \in J} \left\{ \frac{(b^{\top} a_j)^2}{\|b\|^2 \|a_j\|^2} + \alpha(1 - \frac{nnz_j}{m}) \right\}$，作为基列数;

4: $J := J \setminus \{j_0\}$;

5: $i := i + 1$;

6: **end for**

其中 α 表示稀疏度权重，实际计算中选取 0.48，nnz_j 表示第 j 列的非零元数目。

4.4　数值实验

本次实验分别测试了基于算法6 的基选取规则以及改进的两种算法，来验证其算法的可行性和效率。实验数据来自第三章中实际问题的数据。实验所用 CPU 型号为 Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz，系统为 Ubuntu 20.04.1 LTS。表4.1给出了测试问题信息，其中 Name 表示问题名称，Link 表示网络图边的数量，Path 表示业务总的路径数，Row 表示生成问题的行数，Col 表示生成问题的列数，NNZ 表示非零元个数，PreRow 、PreCol、PreNNZ 表示预处理后的行数、列数以及非零元个数。

表 4.1 测试问题信息

Table 4.1 Information test problems

Name	Link	Demand	Path	Row	Col	NNZ	PreRow	PreCol	PreNNZ
N600	2400	6000	30000	222000	174030	5974200	222000	172622	5694002
N800	3200	8000	40000	296000	232030	8267140	296000	230107	7872877
N1000	4000	10000	50000	370000	290030	10635980	369768	287295	10106963
N1200	4800	12000	60000	444000	348030	13036980	443382	344460	12399522
N1400	5600	14000	70000	518000	406030	15501670	517174	401744	14741518
N1600	6400	16000	80000	592000	464030	17994920	590940	459079	17126919
N1800	7200	18000	90000	666000	522030	20509650	663280	514957	19518107
N2000	8000	20000	100000	740000	580030	23087050	736514	571574	21939638

本实验算法测试都是基于开源求解器 HiGHS，并使用对偶单纯形算法求解问题。我们进行四组测试，分别测试了 HiGHS 原方案、基于算法6 的基选取规则以及改进的算法，时间单位为秒。

表 4.2 数值结果对比

Table 4.2 Comparison of numerical results

	HiGHS		MGA		MGAI		MGAII	
Name	Iter	Time	Iter	Time	Iter	Time	Iter	Time
N600	93360	21.64	26320	25.90	20336	25.11	25888	25.22
N800	122899	33.93	34323	41.86	26303	39.64	59759	34.49
N1000	153224	50.94	41957	63.95	32007	63.80	79809	48.11
N1200	179197	66.62	45927	78.95	34016	79.20	100458	58.67
N1400	209234	88.73	53830	108.83	39836	104.39	115356	74.77
N1600	238904	105.02	62283	130.60	46295	133.10	128556	91.75
N1800	266106	134.14	67453	169.30	49493	161.79	143043	111.74
N2000	298930	169.31	79372	219.35	59374	219.96	194047	159.20
汇总	1561854	670.3284	411465	838.7482	307660	827.00	846916	603.96

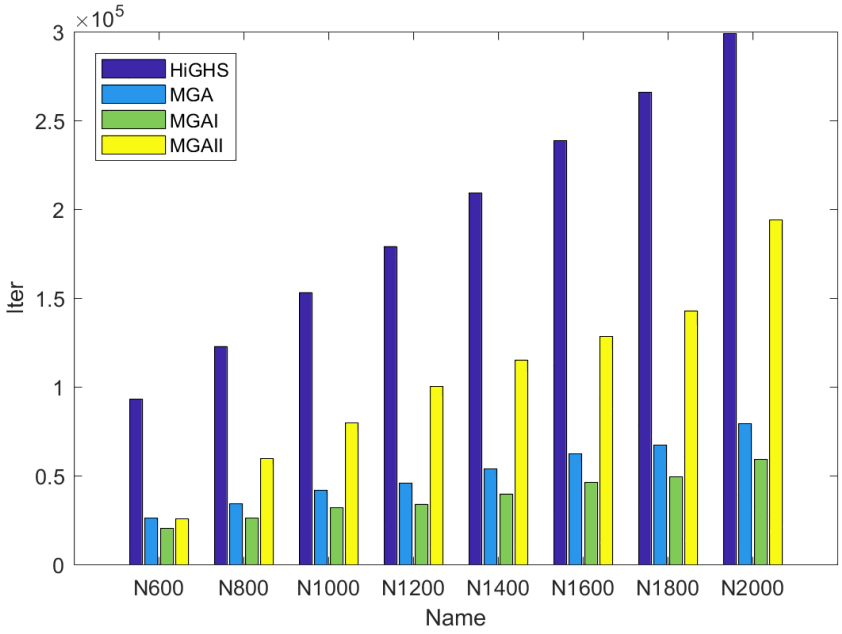

图 4.3 迭代次数对比

Figure 4.3 Comparison of iterations

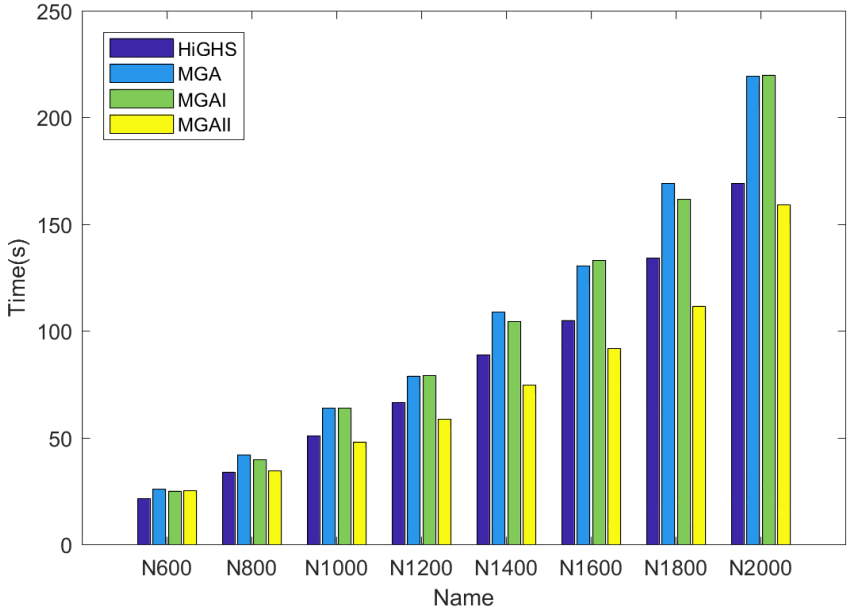

图 4.4 求解时间对比

Figure 4.4 Comparison of solving time

由数值结果可知，虽然采用 MGA 算法可以减少迭代次数，而改进算法 MGAI 进一步减少了迭代次数，但是在计算时间上不占优。故此增加稀疏度的权重，来降低计算时间同时减少了迭代次数。由图4.3和图4.4，MGAII 在减少迭代次数的同时降低了计算时间。

4.5　小结

本章节中介绍了单纯形算法中的一种初始基选取算法 MGA，实现并与求解器 HiGHS 进行了对比。MGA 确实能减少迭代次数，但是却大幅增加了计算时间。通过对 MGA 改进得到的 MGAII 算法，在与 HiGHS 对比中，在迭代次数与时间上都有领先。求解时序网络拥塞优化问题的效率再次提升。

第 5 章　总结与展望

本文研究了对偶单纯形算法，以解决时序网络拥塞优化问题，在解决实际问题上提出了切实有效的处理方法以及算法。

首先，我们开发了基于对偶单纯形算法的线性规划求解器，此类求解器在现实应用上特别是求解整数规划问题上有及其重要作用。在充分学习解线性规划理论知识后，了解单纯形算法线性规划求解器的基本结构后，开发了一个初始版本线性规划求解器，并实现了多种出基入基规则。在测试库例子中也能求解大部分问题。

其次，对于实际应用问题，基于时序的网络拥塞优化问题，根据其问题特征，首先在问题建模上进行了一定的优化，使问题更加稀疏。而后在求解过程中发现预处理在此类问题上的重要作用，在研究传统预处理的基础上，提出了基于模型的预处理方法固定路径分流比、界缩紧、去除冗余行、时序块消除、同路径三角化消除等，通过计算对比，提升了问题求解速度。

最后，还是基于时序的网络拥塞优化问题的求解，也是单纯形算法的一个重要方面，初始基的选取问题。在此问题上提出了两种基于梯度最小夹角的改进方案，第一种改进并能巨幅减少迭代次数，第二种在减少迭代次数的基础上，加入了稀疏度的考量，减少了求解时间，在实际问题中也有很好的效果。

当然本文研究内容问题还有可以提升方面，其中对偶单纯形算法求解器还需要添加更多模块以及更快的求解更多问题。在基于时序的网络拥塞优化问题求解上，还可以研究问题本身性质给出更好的求解方案，如研究其解的性质，从问题的解来探索初始基选取等。

参考文献

[1] 中国互联网络信息中心. 第 49 次中国互联网络发展状况统计报告 [Z]. 2022.

[2] Welzl M. Network congestion control: managing internet traffic [M]. John Wiley & Sons, 2005.

[3] Pióro M, Medhi D. Routing, flow, and capacity design in communication and computer networks [M]. Elsevier, 2004.

[4] Maros I. Computational techniques of the simplex method: volume 61 [M]. Springer Science & Business Media, 2002.

[5] Bertsimas D, Tsitsiklis J N. Introduction to linear optimization: volume 6 [M]. Athena Scientific Belmont, MA, 1997.

[6] Wright S, Nocedal J, et al. Numerical optimization [J]. Springer Science, 1999, 35(67-68): 7.

[7] Ping-Qi P. Linear programming computation [M]. Springer, 2014.

[8] Karwan M H, Lotfi V, Telgen J, et al. Redundancy in mathematical programming: A state-of-the-art survey: volume 206 [M]. Springer Science & Business Media, 2012.

[9] 袁亚湘. 非线性优化计算方法 [M]. 科学出版社, 2008.

[10] Dantzig G. Programming in a linear structure. 1948 [J]. Comptroller, United States Air Force, Washington DC.

[11] Dantzig G B, Orden A, Wolfe P, et al. The generalized simplex method for minimizing a linear form under linear inequality restraints [J]. Pacific Journal of Mathematics, 1955, 5(2): 183-195.

[12] Lemke C E. The dual method of solving the linear programming problem [J]. Naval Research Logistics Quarterly, 1954, 1(1): 36-47.

[13] Bixby R E. Solving real-world linear programs: A decade and more of progress [J]. Operations Research, 2002, 50(1): 3-15.

[14] Forrest J J, Goldfarb D. Steepest-edge simplex algorithms for linear programming [J]. Mathematical Programming, 1992, 57(1): 341-374.

[15] Orchard-Hays W. History of the development of lp solvers [J]. Interfaces, 1990, 20(4): 61-73.

[16] Maros I. A piecewise linear dual phase-1 algorithm for the simplex method [J]. Computational Optimization and Applications, 2003, 26(1): 63-81.

[17] Maros I. A generalized dual phase-2 simplex algorithm [J]. European Journal of Operational Research, 2003, 149(1): 1-16.

[18] [EB/OL]. http://plato.asu.edu/ftp/lpsimp.html.

[19] Kostina E. The long step rule in the bounded-variable dual simplex method: numerical experiments [J]. Mathematical Methods of Operations Research, 2002, 55(3): 413-429.

[20] Pan P Q. Practical finite pivoting rules for the simplex method [J]. Operations-Research-Spektrum, 1990, 12(4): 219-225.

[21] Pan P Q. The most-obtuse-angle row pivot rule for achieving dual feasibility: a computational study [J]. European Journal of Operational Research, 1997, 101(1): 164-176.

[22] Koberstein A. The dual simplex method, techniques for a fast and stable implementation [J]. Unpublished doctoral thesis, Universität Paderborn, Paderborn, Germany, 2005.

[23] Markowitz H M. The elimination form of the inverse and its application to linear programming [J]. Management Science, 1957, 3(3): 255-269.

[24] Suhl U H, Suhl L M. Computing sparse lu factorizations for large-scale linear programming bases [J]. ORSA Journal on Computing, 1990, 2(4): 325-335.

[25] Forrest J, Tomlin J A. Updated triangular factors of the basis to maintain sparsity in the product form simplex method [J]. Mathematical Programming, 1972, 2(1): 263-278.

[26] Leena, M., SuhlUwe, et al. A fast lu update for linear programming [J]. Annals of Operations Research, 1993, 43(1): 33-47.

[27] Huangfu Q. High performance simplex solver [J]. 2013.

[28] Harris P M. Pivot selection methods of the devex lp code [J]. Mathematical programming, 1973, 5(1): 1-28.

[29] Pan P Q. A revised dual projective pivot algorithm for linear programming [J]. SIAM Journal on Optimization, 2005, 16(1): 49-68.

[30] Netlib test problems [EB/OL]. http://www.netlib.org/lp/data/.

[31] A. L. Brearley H P W, G. Mitra. Analysis of mathematical programming problems prior to applying the simplex algorithm [J]. Mathematical Programming, 1975, 8(1): 54-83.

[32] Gondzio J. Presolve analysis of linear programs prior to applying an interior point method [J]. Informs Journal on Computing, 1997, 9(1): 169-170.

[33] Williams H P. A reduction procedure for linear and integer programming models [M]// Redundancy in Mathematical Programming. Springer, 1983: 87-107.

[34] Babayev D A, Mardanov S S. Reducing the number of variables in integer and linear programming problems [J]. Computational Optimization and Applications, 1994, 3(2): 99-109.

[35] Andersen E D, Andersen K D. Presolving in linear programming [J]. Mathematical Programming, 1995, 71(2): 221-245.

[36] Mszros C, Suhl U. Advanced preprocessing techniques for linear and quadratic programming [J]. OR Spectrum, 2003, 4(25): 575-595.

[37] Gamrath G, Koch T, Martin A, et al. Progress in presolving for mixed integer programming [J]. Mathematical Programming Computation, 2015, 7(4): 367-398.

[38] Bixby R E, Wagner D K. A note on detecting simple redundancies in linear systems [J]. Operations Research Letters, 1987, 6(1): 15-17.

[39] Bixby R E. Progress in linear programming [J]. Informs Journal on Computing, 1993, 6(1): 15-22.

[40] Mészáros C, Suhl U H. Advanced preprocessing techniques for linear and quadratic programming [J]. OR Spectrum, 2003, 25(4): 575-595.

[41] Adler I, Karmarkar N, Resende M G, et al. Data structures and programming techniques for the implementation of karmarkar's algorithm [J]. ORSA Journal on Computing, 1989, 1(2): 84-106.

[42] Chang S F, McCormick S T. A hierarchical algorithm for making sparse matrices sparser [J]. Mathematical Programming, 1992, 56(1): 1-30.

[43] Lustig I J, Marsten R E, Shanno D F. Computational experience with a primal-dual interior point method for linear programming [J]. Linear Algebra and Its Applications, 1991, 152: 191-222.

[44] Wolfe P. The composite simplex algorithm [J]. Siam Review, 1965, 7(1): 42-54.

[45] Junior H V, Lins M P E. An improved initial basis for the simplex algorithm [J]. Computers & Operations Research, 2005, 32(8): 1983-1993.

[46] Luh H, Tsaih R. An efficient search direction for linear programming problems [J]. Computers & Operations Research, 2002, 29(2): 195-203.

[47] Paparrizos K, Samaras N, Stephanides G. A new efficient primal dual simplex algorithm [J]. Computers & Operations Research, 2003, 30(9): 1383-1399.

[48] Hu J F. A note on "an improved initial basis for the simplex algorithm" [J]. Computers & operations research, 2007, 34(11): 3397-3401.

[49] Corley H, Rosenberger J, Yeh W C, et al. The cosine simplex algorithm [J]. The International Journal of Advanced Manufacturing Technology, 2006, 27(9): 1047-1050.

[50] Pan P Q, Pan Y. A phase-1 approach for the generalized simplex algorithm [J]. Computers & Mathematics with Applications, 2001, 42(10-11): 1455-1464.

[51] Trigos F, Frausto-Solis J, Lopez R. A simplex cosine method for solving the klee-minty cube [J]. Advances in Simulation System Theory and Systems Engineering, 2002, 70: 27-32.

[52] Murshed M M, Sarwar B M, Sattar M A, et al. A new polynomial algorithm for linear programming problem [J]. 1993.

[53] Stojković N V, Stanimirović P S. Two direct methods in linear programming [J]. European Journal of Operational Research, 2001, 131(2): 417-439.

[54] Khan N, Inayatullah S, Imtiaz M, et al. New artificial-free phase 1 simplex method [J]. arXiv preprint arXiv:1304.2107, 2013.

[55] Arsham H. A big-m free solution algorithm for general linear programs [J]. International Journal of Pure and Applied Mathematics, 2006, 32(4): 549.

[56] Yeh W C, Corley H. A simple direct cosine simplex algorithm [J]. Applied mathematics and Computation, 2009, 214(1): 178-186.

致 谢

时光荏苒，攻读硕士的三年即将结束，在此三年间，我得到了身边很多人的帮助。值此论文完成时，向三年来年来关心和帮助我的老师、同门师兄弟姐妹、同学、朋友和家人表示由衷感谢！

首先，我要向我的本科加硕士导师戴彧虹研究员表示最诚挚的谢意，感谢戴老师多年来给与我生活上的关心，科研学习上的指导。在戴老师的指引下，我踏入了优化的领域，感受到了数学在实际应用中的魅力。您对前沿问题的见解以及对科研的执着，对我启发颇深。感谢您在一次次的讨论班中对我学习科研的谆谆教导，让我开拓视野的同时在更认识到自己的不足。您这多年来的指导，必将称为我人生中的宝贵财富。

感谢课题组的袁亚湘院士，您对问题的具象化见解使我印象深刻，您总能把理论问题讲的通俗易通。感谢您在讨论班上对我工作的宝贵建议。同时感谢您在科研之余，组织课题组爬山等娱乐活动，为大家创造了愉快的科研环境。

感谢课题组的刘歆研究员，有幸上了您的凸分析课程，令人印象深刻收获满满。在讨论班上，您提的问题让我获益匪浅。感谢课题组的刘亚锋副研究员，在讨论班上，您提的问题让我拓展视野。您对科研严谨的态度值得我们学习。感谢课题组的史斌副研究员，马俊杰助理研究员在讨论班上的提问和指导。

感谢陈伟坤师兄、陈亮师兄、张瑞进师兄对我科研的指导和帮助。感谢王小玉师姐、吴宇宸师兄、肖纳川师兄、陈雅丹师姐、赵浩天师兄、刘为师兄、黄磊师兄、姜博鸥师姐、王磊师兄、汪思维师兄、陈硕、谢鹏程、赵成、胡雨宽、武哲宇、章煜海师弟、胡雨婷师妹、吴鹏举师弟、王薪潼师妹、张亦师弟、刘上琳师妹、汤宇杨师弟、王子岳师弟、李冠达师弟、郭钰琦师妹，以及博士后刘泽显师兄、戴金雨师兄、王姝师姐、李成梁师兄对我的帮助。感谢李翔同学、张文豪同学，在科研之余互相鼓励。感谢我的答辩秘书马平川同学。

此外，特别感谢王圣超在我们共同开发的单纯形算法求解器 CZLP 中 LU 分解部分代码的开发以及裴骞和王兆维师弟在 LU 更新部分代码的开发。

最后感谢我的父母家人，感谢你们一直以来对我学业的支持，为我无私的付出。感谢你们对我的包容和照顾。最后，希望疫情阴霾早日散去，愿我们所有人

未来可期。

作者简历及攻读学位期间发表的学术论文与研究成果

作者简历：

张跃，男，河南信阳人，1995 年生。

2015.9–2019.6 就读于中国科学院大学，获得学士学位。

2019.9–2022.6 就读于中国科学院数学与系统科学研究院，攻读硕士学位。