上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：约束可行问题的判断和高效算法

学生姓名：魏星
学生学号：5080309543
专　　业：自动化
指导教师：席裕庚
学院（系）：电子信息与电气工程学院自动化系

约束可行问题的判断和高效算法

摘要

线性不等式组约束的可行性及冗余性判断问题是一个经典问题。在控制领域（如化工过程控制，交通控制，电力系统分配等）中， 常常需要处理大量的线性不等式约束，由于不等式的数量巨大，导致求解过程缓慢，从中演变出两个关于线性不等式组约束的问题：第一个问题是可行性判断问题，即能否快速判断一组不等式组是否有可行解，如果可以预先判断出这一组约束没有可行解，则可以避免后续的无用计算；第二个问题是冗余性判断问题，如果一组线性不等式约束有可行解，除了如何找到一个可行的解，还希望能够去除所有的冗余约束，只保留用于求解可行域的所有必要约束，因为在大多数情况下，这一组线性不等式约束中包含了大量冗余。本文在目前线性不等式约束冗余性判断研究结果的基础上，把冗余性问题对偶推广至可行性问题，在可行性问题中得到了一些类似的结论。另外，本文从几何角度考虑线性约束问题，提出了一种新的算法（“切割法”），使用该算法也可以判断线性约束的可行性以及冗余性，并且最后可以得到非常完整的可行域边界信息，这些信息可以在后续的计算中得到利用。

关键词：线性不等式约束，约束可行性，不等式冗余判断，高效算法

FEASIBILITY OF CONSTRAINT PROBLEM AND ITS FAST ALGORITHM

ABSTRACT

The problem of linear constraints and the identification of its feasibility as well as its redundancy is a classical problem. It is always possible to come up with a great deal of linear inequality constraints in the fields of controlling (such as chemical process control, traffic control and the scheduling of electrical transmission in power systems). As the number of inequalities could be very large, the process of solving these inequalities could take a very long time. There are mainly 2 problems concerning linear inequality constraints: The first is to identify a series of constraints as either feasible or unfeasible. If we can find out that the constraints have no solution, then we can avoid the following useless calculations. The second problem is the identification of redundancy, if a series of constraints have a solution, besides finding out a feasible solution, we also expect to remove all the redundant constraints and only keep the necessary ones, because in most cases, a large number of constraints always have many constraints that are redundant. in this paper, based on the current studies of redundant constraint problems, we made a dual expansion of the conclusions on constraint redundancy, and give some similar conclusions on constraint-feasibility. Apart from this, we think about the problem in a geometry point of view, and come up with a new algorithm called “cutting algorithm”, this algorithm can also identify the feasibility and redundancy of constraints, and can give the complete information of the feasible region, which can be useful in later calculation.

Key words: linear inequality constraints , constraint feasibility , redundant inequality judgment , fast algorithm

目 录

第一章 绪论

线性不等式组约束经常出现在控制领域的某些计算环节中，处理大量的线性不等式约束的问题由此产生。如果一组线性不等式约束不存在可行解，那么提前判断出来可以避免后续的无用计算量，而且如果能够知道是哪些不等式约束相互不相容导致的无解，则可以适当放松那些不相容的不等式，使这一组线性不等式约束变为可行。如果一组不等式存在可行域，那么则希望可以找到其可行域内部的一点（即找到一个可行的解），同时把那些冗余的不等式约束全部剔除，以便减少后续的计算量。

线性约束问题在控制领域的许多问题中广泛存在，在约束数量极大的情况下，快速判断可行或不可行以及去除大多数的冗余约束等工作，对于减小计算量、提高计算效率具有重大意义。

本章首先介绍线性不等式约束的相关问题描述（约束不相容、约束冗余的概念等），然后举两个例子介绍线性约束问题在控制领域中的应用，之后介绍目前关于约束冗余性判断的一些研究成果。

1.1 线性约束问题相关概念和问题描述

1.1.1 线性不等式约束问题符号约定

假设问题维数为 N 维，本文关注的是以下一组共 n 个线性不等式约束：

$$a_i^T x \le b_i \quad (i = 1, 2, \ldots, n) \tag{1.1}$$

上式中，x 为 N 维列向量变量，$x = [x_1\ x_2\ \ldots\ x_N]^T$。$a_1, a_2, \ldots a_n$ 均为 N 维列向量，$b_1, b_2, \ldots b_n$ 均为实数。另外，x 的每一个分量都有一定的取值范围：

$$P_i \le x_i \le Q_i \quad (i = 1, 2, \ldots, N) \tag{1.2}$$

其中 P_i 和 Q_i 为变量 x 的第 i 个分量的取值上下界。公式 1.1 和公式 1.2 共同构成了约束问题的表述。

这一组不等式约束的可行域为所有满足式 1.1 和 1.2 的 x 的集合，定义为 R：

$$R = \{x \mid a_i^T x \le b_i,\ i = 1, 2, \ldots, n\} \tag{1.3}$$

再定义集合 R_k，R_k 是在原来的约束中，把第 k 个约束删除后得到的的可行域：

[redacted] （1.4）

1.1.2 约束冗余性问题

本文所指的“冗余的约束”是指这样的约束：把该约束从线性不等式集合中去除之后，求解得到的可行域与求解所有线性约束得到的可行域相等。所以给出不等式“冗余”和“必要”的定义为：

定义 1.1[1] 约束 k （ $a_k^T x \le b_k$ ）是**冗余**的，当且仅当：

$$R = R_k \tag{1.5}$$

定义 1.2[1]约束 k （ $a_k^T x \le b_k$ ）是**必要**的，当且仅当：

$$R \neq R_k \tag{1.6}$$

可以在两维情况下对该问题有一个直观理解。在问题维数是 2 的时候，每一个约束相当于一条线。最后得到的可行域 R 是一个二维平面上的凸多边形，其每一条边就是对应了一个必要的约束。而冗余的约束是指那些没有构成最后可行域凸多边形的线，图 1-1 展示了两维情况下的冗余约束。

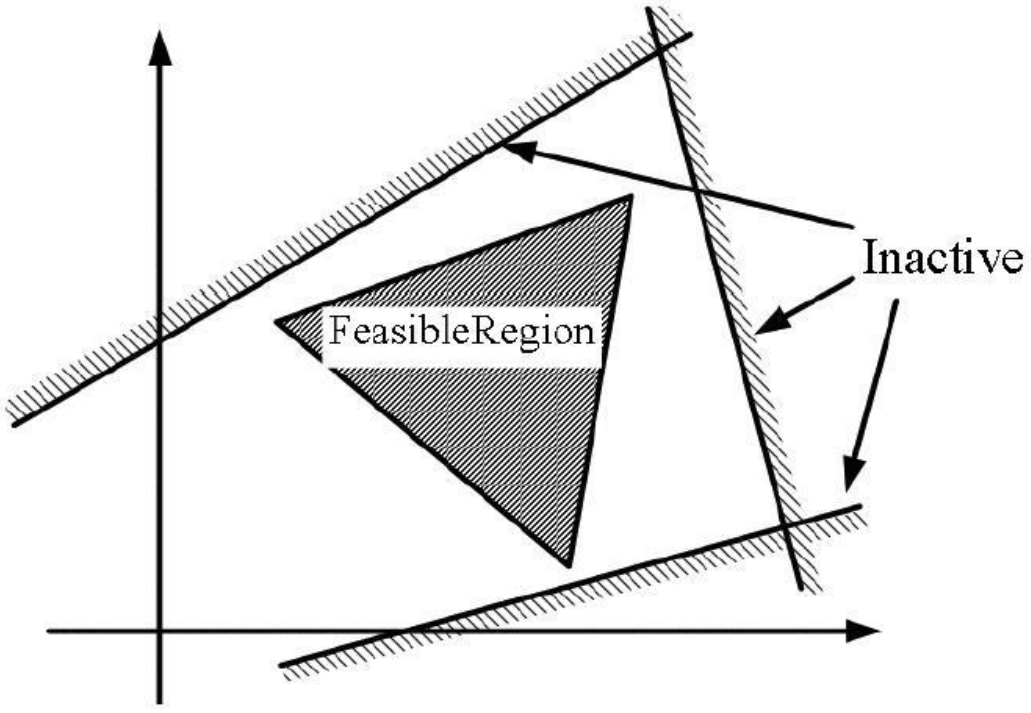

图 1-1 冗余约束示例

1.1.3 约束可行性问题

在某些时候会存在约束相互矛盾的情况，如下例中的约束：

$$\begin{cases} x_1 + x_2 \le 10 \\ -x_1 - x_2 \le -20 \end{cases} \tag{1.7}$$

这两个不等式的可行域之间没有交集，所以会导致最终不存在可行域。所以“约束不相容”的定义为：

定义 1.3[1] 一组线性不等式约束：

$$\mathbf{A}\mathbf{x} \le \mathbf{b}, \quad \mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^{m} \tag{1.8}$$

是**不相容（不可行）的**，当且仅当：

$$R = \varnothing \tag{1.9}$$

需要注意的是，“约束不可行”这样的表述必须是相互的，不可以说某一个约束是不可行的，因为每一个线性不等式都有其可行域，只是不存在同时满足所有约束的可行域。而且如上面的例子所见，不论是去掉第一个约束还是去掉第二个约束，都可以使可行域 R 不再为空集，所以“约束不可行”的概念只有把几个约束放在一起提出才有意义。但是同时，由于某些约束是可以进行放松的，而某些约束是必须严格满足，不能放松的，所以在指定了一部分约束不可放松的情况之下，人们希望能够给出如何放松剩余约束使得问题有可行解的方法。

1.2 约束问题在控制领域的应用背景举例

约束问题在控制领域经常出现，以下列举化工过程和电力系统两个控制领域的约束问题的例子，展示约束冗余性判断和约束可行性判断对这些控制实例的意义。

1.2.1 预测控制在化工过程应用中的约束问题

在化工领域过程控制中，往往对于输入变量和输出变量有很多约束条件。预测控制由于可以做到将输入变量和输出变量的约束条件直接纳入其求解控制律的优化过程中，所以在化工领域中得到了广泛的应用。但是，实际问题中经常出现各种对于输出变量、输入变量甚至中间变量的约束之间相互矛盾的情况（即 1.1.3 节提到的“约束不相容”）。由于这些约束条件无法全部满足，会导致在线预测控制器没有可行解，从而给生产到来负面影响。

如 Shell 公司的典型重油分馏塔[4]控制问题，该模型是一个有 3 个操作变量（塔顶回流量 u_1、侧线抽出量 u_2，塔底再沸加热蒸汽量 u_3），3 个被控变量（塔顶产品组成 y_1，塔侧产品组成 y_2，塔底再沸温度 y_3）的复杂系统。其传递函数矩阵为：

$$G(s)=\begin{bmatrix}\frac{4.05e^{-27s}}{50s+1},\frac{1.77e^{-28s}}{60s+1},\frac{5.88e^{-27s}}{50s+1}\\ \frac{5.39e^{-18s}}{50s+1},\frac{5.72e^{-14s}}{60s+1},\frac{6.9e^{-15s}}{40s+1}\\ \frac{4.38e^{-20s}}{33s+1},\frac{4.42e^{-22s}}{44s+1},\frac{7.2e^{-0s}}{19s+1}\end{bmatrix} \tag{1.10}$$

如果控制的约束要求为：以u_1、u_2、u_3为操作变量控制y_1、y_2、y_3在一定范围内：要求控制y_1、$y_2\in[0.3,0.4]$，$y_3\in[-0.5,-0.4]$。三个操作变量u_1、u_2、u_3的取值范围是$[-0.05,0.05]$。

那么在稳态情况下，系统需要满足的约束有：

$$\begin{cases}4.05u_1+1.77u_2+5.88u_3\le 0.4\\ -4.05u_1-1.77u_2-5.88u_3\le -0.3\\ 5.39u_1+5.72u_2+6.9u_3\le 0.4\\ -5.39u_1-5.72u_2-6.9u_3\le -0.3\\ 4.38u_1+4.42u_2+7.2u_3\le -0.4\\ -4.38u_1-4.42u_2-7.2u_3\le 0.5\\ u_1,u_2,u_3\le 0.05\\ -u_1,-u_2,-u_3\le 0.05\end{cases} \tag{1.11}$$

所以该问题是一个三维、12 个约束的约束问题。

事实上，这个例子中要求的约束优化问题是不可行的，如果可以提前判断该问题不可行，则可以避免不必要的在线求解预测控制器的计算。在约束优化问题没有可行解的情况下，需要对某些约束进行调整，放松一部分约束使整个约束问题有解。所以，此时在判断出来约束问题不相容之后，还希望给出一些调整方案，使问题变为可解。

1.2.2 电力系统分配中的约束问题

随着电力调度精细化要求的提出，短期发电调度从单纯的安全性过渡到安全性与经济性并重，需要以考虑安全约束的优化调度软件为支持，编制发电计划。其中，安全约束机组组合（Security Constrained Unit Commitment，简称 SCUC）问题是发电厂运作者最常处理的一类任务，也是发电方为了最大化效益所经常研究的一类问题[8]。

该问题的符号约定如下：假设一个电力系统包含 I 台发电机组，L 条传输线，K 条负载总线，研究时域被划分为 1,2，…..，T 个时间周期。D_t 表示在第 t 个时间周期系统的总负载；PR_t 表示系统在 t 时刻的备用容量（见后文）需求；$D_{k,t}$ 表示 t 时刻第 k 条负载总线上的负载需求；$p_{i,t}$ 表示第 i 台发电机组在 t 时刻的发电水平；$r_{i,t}$ 表示 t 时刻第 i 台发电机组的旋转备用容量大小；F_l 表示第 l 条传输线的功率传输上限；Γ^U 是一个 L*K 的矩阵，表示总线负载到功率流的转换系数，$\Gamma^U_{l,k}$ 是 Γ^U 的 l 行 k 列元素；$\overline{P_i}$ 和 $\underline{P_i}$ 分别表示第 i 台发电机组的发电

上下界；$z_{i,t}$ 是离散的决策变量，取值范围是“0”或“1”，表示 t 时刻第 i 台机组是运行还是关闭。

SCUC 是根据研究周期内的各时段系统负荷预测优化机组发电计划，包括机组开停方式和发电出力。这是一个约束优化问题，优化的目标是使系统的总成本最小化（系统总成本=发电成本+开机成本+停机成本）。该优化问题所需要满足的约束主要有四类：

（1） 系统负载平衡（System Load Balance）约束

系统负载平衡约束，是指每一时刻系统各台发电机组的总发电量应当等于这一时刻各条输电总线上的用电需求之和。用公式表示为：

$$\begin{cases} \sum_{i=1}^{I} p_{i,t} = \sum_{k=1}^{K} D_{k,t} = D_t; \\ t = 1,2,.....T \end{cases} \tag{1.12}$$

式（1.12）中的 $p_{i,t}$ 就是控制变量，对应于问题描述中的变量 x 。

（2） 系统旋转备用容量（System Spinning Reserve Requirements）约束

电力系统的总装机容量中，有一部分正在运行并承担着有功出力的容量，这部分叫做工作容量；还有一部分容量是正在运行但是不负担有功出力，或者虽不在运行但随时准备投入运行，这一部分叫做旋转备用容量（Spinning Reserve）。电力系统设置旋转备用容量是为了保障电力系统可以安全稳定运行。

系统旋转备用容量约束，是指每一时刻，每台发电机组的旋转备用容量之和要大于或等于这一时刻的总的旋转备用容量需求。用公式表示为：

$$\sum_{i=1}^{I} P_{i} \geq PR_t, t = 1,2,...T \tag{1.13}$$

（3） 传输线负载能力（Transmission Line Capacity）约束

传输线负载能力约束又称为安全约束（Security Constraint），这组约束的意思是指保证每条传输线路上的功率流小于该条线路的最大功率流上限。对于某一条传输线，功率流超过其允许上限可能会引起极大安全问题，所以这一组约束必须得到保证。这一组约束的公式表达为：

$$\begin{cases} -F_l \leq \sum_{i=1}^{I} \Gamma_{l,i}^{U} p_{i,t} - \sum_{k=1}^{K} \Gamma_{l,k}^{U} D_{k,t} \leq F_l; \\ l = 1,2,.....,L \\ t = 1,2,.....,T \end{cases} \tag{1.14}$$

（4） 发电能力（Generation Capacity）约束

发电能力约束是指每台发电机组的发电量应该在其发电能力（发电功率的上下界，$\overline{P}_i$ 和 $\underline{P}_i$）范围内，这类约束是由发电机组本身决定的固有约束，必须得到满足。公式表述为：

$$\begin{cases} z_{i,t}\underline{P}_i \le p_{i,t} \le z_{i,t}\overline{P}_i; \\ i = 1,2,....., I \\ t = 1,2,....., T \end{cases} \quad (1.15)$$

综合（1）~（4）的约束，电力系统的约束问题是一个高维数、多约束的复杂问题，且每个变量都有一个取值上下界（$\overline{P}_i$、$\underline{P}_i$），比较符合本文对于约束问题的表述。

实际应用中，由于一个实际电力系统可能有几十条输电线路、几十（甚至上百）个发电机组和负载总线，所以涉及的约束数量会非常巨大。参考文献[2]中研究的电力系统，有一个是 IEEE 118 总线系统，包含了 179 条传输线和 54 台发电机组。在如此大量的约束下进行优化求解必然会导致计算量大增，而实际上这些约束的很大部分都是冗余约束，所以，如果能够剔除（部分或全部）冗余的约束，对于减少 SCUC 问题求解的计算量具有重大意义。

1.3 约束问题目前研究成果

关于线性约束问题，目前比较多的研究集中在约束冗余性判断方面。本节对目前世界上关于约束冗余性判断的一些相关研究成果进行一下总结和评价。

关于线性约束的冗余性判断问题，已经有了很多年的研究，得到了一些有用的结论和成熟的方法。如在参考文献[1]中，从数学角度严密的分析了约束问题冗余性的一些判断方法，给出了约束冗余的充分必要条件。利用参考文献[1]提出的方法（基于线性规划），可以对每一个约束进行判断，文献[1]提出的、使用线性规划判断冗余性的思路也是约束问题研究的经典思路。

参考文献[2]在参考文献[1]的基础之上，把充分必要条件放松为充分条件，虽然不能够判别出所有的约束，但是可以避免线性规划的迭代计算，可以快速判断出一部分的冗余约束。这一结果在电力系统优化实践中去除了很大部分的约束，得到了非常好的应用效果。其放松约束的做法很有启发意义。

1.3.1 冗余不等式判别的充要条件

参考文献[1]给出了用线性规划（Linear Programming）表达的判定约束冗余性的充分必要条件。为了表述方便，用简写 LP_k 表示以下线性规划问题：

LP_k：　最小化：　$-a_k^T x$，

受约束于：　$x \in R_k$

（1）　冗余性判断的充要条件

定理 1.1[1]

对于问题描述中的一组不等式，第 k 个约束（$a_k^T x \le b_k$）是冗余的约束，当且仅当线性规划 LP_k 得到的最优解 x^* 满足：$a_k^T x^* \le b_k$。

证明 ：

（充分性）

已知第k 个约束是冗余的，使用反证法证明，假设存在一个$x^* \in R_k$，满足：$a_k^T x^* > b_k$。那么根据定义， $x^* \notin R$，所以$R \neq R_k$。那么由 1.1.2 给出的定义，第k 个约束是必要约束。这一结果与已知条件矛盾，所以假设不成立，所以一定有$a_k^T x^* \leq b_k$。

（必要性）

已知线性规划LP_k得到的最优解x^*满足：$a_k^T x^* \leq b_k$，由于x^*是线性规划LP_k的最优解，对于集合R_k中的任一x，必然有：$a_k^T x \leq a_k^T x^* \leq b_k$。该式表明$R_k$是$R$ 的一个子集（$R_k \in R$），又R 一定是R_k的子集（$R \in R_k$）这一事实，可得$R_k = R$。所以，由 1.1.2 的定义可知约束k 为冗余约束。

证毕

（2） 冗余性判断充要条件的应用

定理 1.1 是冗余性判断问题的理论基础，它也提供了一种冗余性判断问题的方法：对于$k=1,2,3,\ldots,r$，每次都解一个线性规划问题LP_k来判别约束k 是冗余的还是必要的。这种方法虽然可行，但是某些时候（尤其当约束非常多的时候）的效率可能不能令人满意。

参考文献[1]继续讨论了出现退化极点的情况，所谓的“退化点”，又称为“奇异点”，是指同时满足（active，即约束取到等号的情况）了大于 N 个约束的点。对于一般的 N 维问题，可行域的每一个边界点是有 N 个约束决定的，也就是说在可行域的边界点，一般有 N 个约束取到等号（active），如果某一个边界点上有大于 N 个约束取等号，那么这个边界点就称之为“退化极点”或“奇异点”。

图 1-2 所示为一个二维约束问题，最后的可行域R 有四个边界点（点<1>到点<4>）。由图可见点<2>和点<3>都刚好位于两条线上，表示在点<2>和点<3>都是刚好有两个约束 active，而在点<1>和点<4>处则分别有 4 个和 3 个约束 active。所以称点<2>和点<3>属于普通边界点，点<1>和点<4>属于退化极点或奇异点。

参考文献[1]试图减小每次做线性规划的规模，该文章在定理 1.1 的基础上，对于出现退化点的情况给出了一系列结论，在出现退化极点的情况下可以在一定程度上减少线性规划的问题规模。但是实际情况中出现奇异情况的实例并不多，绝大多数情况下约束不会出现奇异情况，而且即使产生了一两个奇异点，使用参考文献[1]的方法也并不能减少太多的计算量。

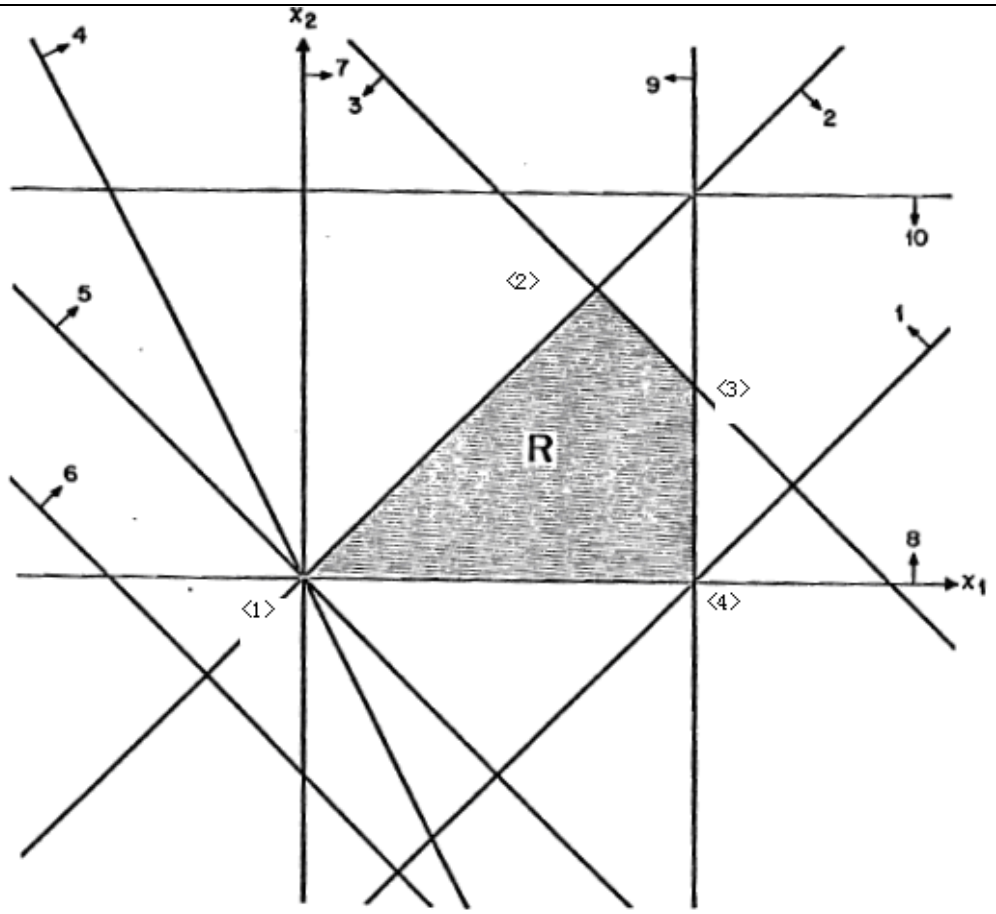

图 1-2 二维情况下的退化极点示例

在此简单介绍一下线性规划的迭代过程。目前比较经典的线性规划算法是使用单纯形（simplex）的表上作业法，表上作业法首先采用 M 法找到一个初始可行基解，之后每一步迭代进行入基出基操作。选取入基出基变量的原则是使得目标函数值增长最大，入基出基操作同时进行旋转变换（矩阵的初等行变换），使得新的基解仍然可行（即仍然是基可行解）。当迭代到目标函数值不能再继续增大的时候，迭代结束，此时的基可行解就是线性规划算得的最优解 x^*。

关于计算效率，单纯形法最坏情况下的时间复杂度是指数级别的（可以构造出让单纯形法的时间复杂度达到指数级别的具体实例），但是实践证明单纯形法在绝大多数情况下的效率还是非常令人满意的。线性规划算法经过了几十年的发展，目前相关算法（如 glpk 线性规划工具包等）在效率上已经做了大量的改进。

另外，由于求解线性规划的算法包含了处理约束不相容的情况，所以使用线性规划算法也可以判定一组约束的可行性问题。只需随便指定一个目标函数，加上所有的约束，看问题有没有解，就可以判断这组约束是否可行。如，求解以下线性规划问题：

$$\begin{cases} \text{最大化 } 0^*x \\ \text{受约束于 } x \in R \end{cases}$$

如果以上问题可以得到一个结果，则说明这一组约束是可行的，否则就表示约束不相容。

1.3.2 冗余不等式判断的充分条件（快速判断算法）

参考文献[1]提供的方法作为冗余性判断问题的基础，虽然可以判断任意一个约束是否冗余，但是由于需要对每一个约束使用线性规划进行一次判断，计算量相对较大，所以人们试图找到一种可以绕过线性规划而剔除冗余约束的算法。2010 年发表的参考文献[2]中，将定理 1.1 的条件进行了放松，得到了冗余性判断的充分条件。虽然该条件不能保证判断出所有的冗余约束，但是在电力系统应用中可以判断出很大部分（85%）的冗余约束，取得了很好的效果。

参考文献[2]是基于电力系统调度分配问题提出的，该问题在 1.2.2 中已经详细介绍，它所涉及的约束包含一个等式约束（$\sum_{i=1}^{I} x_{i,t} = D_t$）和一系列不等式约束。第 k 个不等式约束为：$\sum_{i=1}^{I} a_{k,i} x_{i,t} \le B_{k,t}$（在电力传输系统案例中，这些约束是指 1.2.2 中提到的传输线负载能力约束或安全约束）。

为方便表述，使用简写 $GLP2_k$ 表示以下这种线性规划问题：

GLP2k：　最小化 $-\sum_{i=1}^{I} a_{k,i} x_{i,t}$，

受约束于：$\begin{cases} \sum_{i=1}^{I} x_{i,t} = D_t \\ 0 \le x_{i,t} \le \overline{P_i}, 1 = 1,2,\ldots,I \end{cases}$

比较 $GLP2_k$ 与 LP_k 的定义，不难发现，$GLP2_k$ 的优化目标函数与 LP_k 相同，但是 $GLP2_k$ 在受限制的约束中去除了所有不等式约束（在电力系统中指安全约束），只保留一个等式约束和各个变量的取值范围约束，所以 $GLP2_k$ 是一个比 LP_k 放松了的线性规划问题。

（1） 冗余性判断的充分条件

定理 1.2[2]

如果求解 $GLP2_k$ 线性规划得到的最优结果 $GLP2_k^*$ 满足：

$$GLP2_k^* \le B_{k,t}$$

则约束 k（$\sum_{i=1}^{I} a_{k,i} x_{i,t} \le B_{k,t}$）是冗余约束。

证明

设 LP_k 的最优解为 x_1^*，$GLP2_k$ 的最优解为 x_2^*。

通过比较 $GLP2_k$ 和 LP_k 的约束，可以发现任一 LP_k 的可行解 x 一定满足 $GLP2_k$ 的约束（因为 $GLP2_k$ 要求的约束是 LP_k 要求约束的子集）。所以 LP_k 的可行域是 $GLP2_k$ 可行域的子集。所以 x_1^* 一定属于 $GLP2_k$ 的可行域。而 $GLP2_k$ 和 LP_k 优化的目标函数相同，所以，优化得到的最优函数值 LP_k^* 一定不大于 $GLP2_k^*$。

所以，如果 $GLP2_k \le B_{k,t}$，则一定有 $LP_k \le B_{k,t}$，由定理 2.1 可知，此时约束一定为冗余约束。

证毕

上述定理提供了松弛了的一个冗余判断方法。在电力系统应用的实际物理含义，是指如果在只满足系统负载平衡约束（见 1.2.2）的情况下，$GLP2_k^*$ 是传输线上可能达到的最大功率流。如果这个最大功率流还达不到线路限制的最大功率流，那么这一项安全约束就自然而然是冗余的。但是该定理还是依赖于求解线性规划问题（尽管比 LP_k^* 的线性规划问题简单许多），还需要进行迭代求解。

（2） 冗余性判断的快速算法

参考文献[2]进一步给出了以下定理，可以不用线性规划而求解得到 $GLP2_k^*$，进而判断出约束的冗余性。

定理 1.3[2]

定义 $i_1, i_2, \ldots i_I$ 是 1,2,…,I 的一个排列，这个排列满足：

$$c_{j,i_1} \ge c_{j,i_2} \ge \cdots \ge c_{j,i_I}$$

那么如果存在一个 $k(1 \le k \le I)$ 满足：

$$\begin{cases} \sum_{m=1}^{k-1} \overline{P}_{im} \le D_t \le \sum_{m=1}^{k} \overline{P}_{im}. \\ \sum_{m=1}^{k-1} (a_{j,im} - a_{j,ik}) \overline{P}_{im} + a_{j,ik} D_t \le B_{j,t} \end{cases} \tag{1.16}$$

则约束 k（$\sum_{i=1}^{I} a_{j,i} x_{i,t} \le B_{j,t}$）是冗余约束。

（证明见参考文献[2]，由于证明较长，此处证明略。）

该定理提供了快速判断不等式约束冗余性快速算法，由于这个定理只使用了一次排序

（对约束的系数进行排序）和简单的向量相乘计算，就可以判断出约束的冗余性，所以速度比线性规划的迭代要快很多。

根据参考文献[2]在电力系统中的测试结果，使用该快速算法，超过 85%的安全约束被判断为冗余，这 85%的安全约束都可以从该问题中剔除，该结果在大规模电力传输系统的传输规划问题中大大减少了传输规划所用的机器时间（根据测试结果，最佳的测试情况下机器时间减少了两倍），这一结果在应用中具有重要意义。

1.4 第一章小结

约束问题在控制领域广泛出现，目前关于如何去除冗余约束已经得到了一些较好的研究成果，既有使用线性规划的冗余充要条件，又有避免线性规划计算的冗余充分条件。在下面的章节中，我们把约束冗余问题进行对偶推广，在约束可行性问题上给出类似的结论。之后，再从几何的角度思考约束问题，提出一种新的切割算法，并对其算法的效率、算法优缺点进行分析。

第二章 对于目前研究成果的对偶推广

由于约束的冗余性问题其实与约束可行性问题有一定的对偶关系，所以第一章提到的一些结论在约束的可行性问题上可以有相应的推广形式，得到类似的结论。本章首先讨论约束冗余性问题和约束可行性问题的对偶关系，然后给出第二章的结论在约束可行性问题上的推广结论，随后再对充分条件的快速算法进行进一步分析。

2.1 冗余性问题与可行性问题的对偶关系

对于一组线性不等式，如果把第k个约束去除，得到的可行域为R_k。那么根据 1.1 中提出的约束冗余的定义，第k个约束冗余的含义是指加入约束k之后，新的可行域R与原可行域R_k完全相等。也就是说约束 k 的加入没有改变原来的可行域R_k，或者说原来的可行域R_k全部处于第k个约束的可行域范围内。

图 2-1 展示了冗余约束的一个例子，假设原来不加入约束k时，问题的可行域R_k是一个四边形区域，如果加入的约束是k（冗余约束），可见原来的整个可行域R_k都处于约束k的可行域之内。而如果假如的约束是k'，则原来的可行域R_k有一部分在约束k'的可行域之外，所以约束k'不是冗余约束。

而对于约束不可行的情况，是指所有约束的可行域是空集。假设把约束k去除后，剩余的约束存在可行域R_k，那么如果假如约束k''，可见整个R_k都在约束 k”的可行域之外，所以约束不相容。

注意到图 2-1 中的约束k和约束k''都属于“没有和原可行域R_k相交”的约束，只不过他们的方向相反，如果把约束 k 的方向取反，就和约束k''一样，整个原可行域R_k都处于约束k的可行域之外，所以这一组约束变为不相容。

由以上的分析可见，只要改变一个冗余约束的符号，这一组约束就由相容变为了不相容，所以约束的冗余性和可行性有一定的对偶关系，可以把第一章的结论进行对偶推广。

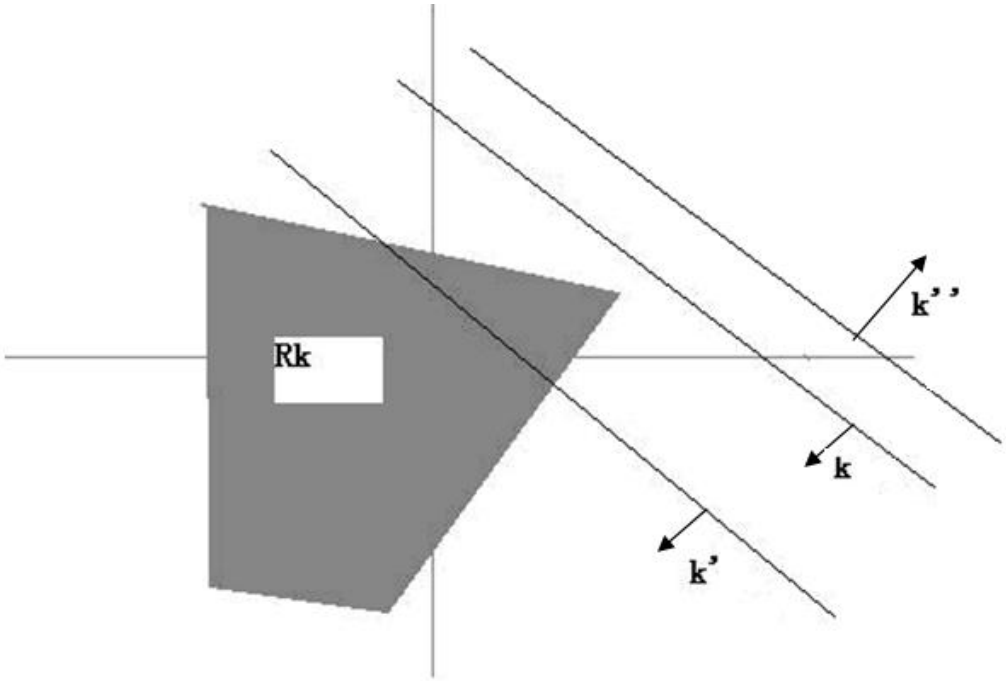

图 2-1 冗余约束、必要约束、不相容约束示例

2.2 冗余充要条件和充分条件的对偶推广

基于 2.1 的分析，可以把 1.3 的两个冗余性判别条件在可行性分析上进行推广。

2.2.1 可行性判断的充要条件

由定理 1.1，推广出可行性判断的充要条件：

定理 2.1

如果一组线性不等式组 1,2，…，k -1 有可行域 R_k，加入第 k 个约束（$a_k^T x \le b_k$）后，这一组约束不相容，当且仅当线性规划 LP_k' 得到的最优解 $LP_k^* = a_k^T x^*$ 满足：$LP_k^* > b_k$。

其中 LP_k' 是以下线性规划：

LP'k 最小化： $a_k^T x$，

受约束于： $x \in R_k$

证明

（充分性）

已知没有约束 k 时，存在可行域 R_k，加入约束 k 之后，这一组不等式变为不可行。

约束变为不可行就是说整个原来的可行域 R_k 都处于约束 k 的可行域之外，即对任意的 $x \in R_k$，都有 $a_{k_k}^T x \geq b_k$，$x^* \in R_k$，所以显然 x^* 也满足 $a_{k_k}^T x^* \geq b_k$。

（必要性）

已知线性规划 LP'$_k$ 得到的最优解 x^* 满足：$a_k^T x^* > b_k$。

由得 LP'$_k$ 的定义可知，对任意的 $x \in R_k$，都有 $a_k^T x \geq a_k^T x^*$，又 $a_k^T x^* > b_k$，所以对任意的 $x \in R_k$，都有 $a_k^T x > b_k$，所以可行域 R_k 内的任意一点都不满足约束 k，所以这一组约束不相容。

证毕

虽然使用线性规划算法可以直接判断一组约束是否有可行域，但是定理 2.1 给出的是从迭代的角度考虑解决方法。假设原来的约束存在可行域，新加入一个约束，可以使用定理 2.1 来判断加入这个新加入的约束是否会导致问题无解。

2.2.2 可行性判断的充分条件

定理 1.2 是把定理 1.1 放松而得到的，同理，也可给出可行性判断的充分条件。定理 2.2 是约束不可行的充分条件（逆命题为约束可行的必要条件）。

定理 2.2

一组线性不等式组 1,2，…,k-1，加入第 k 个约束（$a_k^T x \leq b_k$）。若线性规划 $GLP2'_k$ 得到的最优结果 $GLP2'^*_k$ 满足：$GLP2'^*_k > b_k$，则加入约束 k 后不等式组无可行解。

其中，$GLP2'_k$ 定义为：

GLP2'$_k$ 最小化：$a_k^T x$，

受约束于：$\begin{cases} a_1^T x \geq b_1 \\ \cdots \\ a_{k-1}^T x \geq b_{k-1} \end{cases}$

证明

因为 $GLP2'_k$ 的优化目标与 LP_k' 相同，且 $GLP2'_k$ 受限于的约束比 LP_k' 少，所以 LP_k' 的可行域 R_k 是 $GLP2'_k$ 可行域 R_{k2} 的子集。所以 LP_k' 的最优点 $x^* \in R_{k2}$。所以 $GLP2'^*_k \leq LP'^*_k$。

所以 $LP_k^* > b_k$，根据定理 2.1，约束没有可行解。

证毕

由于$GLP2_k'$和$GLP2_k$只是优化的目标函数不一样（符号相反），所以定理 2.2 也可以使用类似于定理 1.3 的快速算法进行判别，避开线性规划。

2.3 关于充分条件（快速判断算法）的进一步讨论

参考文献[2]的结果非常有使用价值，它的思想也值得进一步分析。

对于 1.1.1 提出的标准问题，$GLP2$ 相当于是只保留每个变量的上下界（$Q_i \le x_i \le P_i$）的约束。而在二维或三维情况下，这样的 x 的每个分量上下界构成的可行域实际上是一个长方形或长方体。

所以不难理解，实际上参考文献[2]的方法是在一个由变量各个分量取值范围构成的（高维的）方形区域（$Q_i \le x_i \le P_i$）中寻找目标函数的最大值，而且显然这样的最大值会取在方形区域的某个顶点上。所以最后得到的$GLP2$ 的最优解 x^* 的每个分量 x_i 一定取 Q_i 或 P_i。正是由于 x 的每个分量只可能有两种取值，才使得可以找到一种方法避免求解线性规划问题。

对于问题描述中的不等式约束，由于没有电力系统中的等式约束（系统负载平衡约束），所以可以进一步简化，不用排序操作，直接由约束系数得到$GLP2$ 的最优解。

定理 2.3

对于一个 N 维的$GLP2$ 线性规划问题，其最优解 x^* 为：

$\vec{x}=[x_1 x_2 \cdots x_N]$，其中 x^* 的每个分量为：

$$x_i=\begin{cases} Q_i, & 若a_{k,i}<0 \\ P_i, & 若a_{k,i}>0 \end{cases} \qquad (2.1)$$

证明

只要证明按照定理 2.3 得到的 x^* 可以使得 $a^T{}_k x$ 最大。

线性规划问题最优解一定是在可行域的顶点，所以可能的最优解一定每个分量都取 Q_i 或 P_i。所以只要证明把 x^* 的任一分量 x_i 改掉后都会使目标函数值减少即可。

若 $a_{k,i}>0$，则 $x_i=P_i$，则 $a_{ki}x_i=a_{ki}P_i$。若把 x_i 改为 $x_i'=Q_i$，则 $a_{ki}x_i'=a_{ki}Q_i$，因为 $a_{k,i}>0$，必然有 $a_{ki}x_i>a_{ki}x_i'$，所以 $a_k^T x^* \ge a_k^T x^{*'}$，修改后的点 $x^{*'}$ 的目标函数值小于 x^* 的函数值。同理，若 $a_{k,i}<0$，改变 x_i 也一定会使规划目标函数值变小。

所以 x^* 处的目标函数值在所有边界点中最大，所以 x^* 是使目标函数取最大值的最优

解。

证毕

使用定理 2.3 的方法可以进一步简化快速判别法的计算，不需要进行排序操作就可以直接得到 GLP2 的结果。

根据以上的分析，直观上理解，快速算法能够判断出的是那些可行域包含全部方形区域的约束（判别为冗余约束）或者方形域全部位于可行域之外的约束（判断为约束不相容），这两类约束都属于没有与可行域“相交”的约束。如图 2-2 所示，约束 k 属于冗余约束，约束 k’加入将导致问题问可行解，约束 k”虽然冗余，但是由于它与方形域相交，使用快速判别法不能判断它的冗余性。

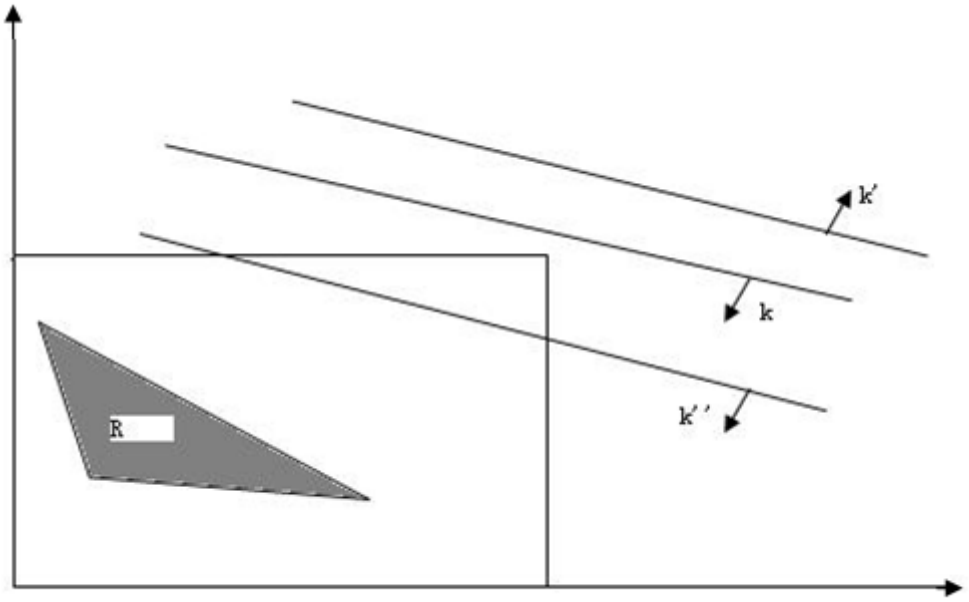

图 2-2 使用快速判别法可以判别出的约束

从另一方面讲，在应用了快速判别法之后，所有没有“切割”到方形区域的约束都会被判别出。剩余的约束一定是与方形区域相交的，那么最后得到的可行域一定是位于方形域内部。所以，应用了快速判别法之后，可以确定一个初始可行域——就是本文所说的方形区域。这个方形域也是下一章介绍的切割法的初始迭代区域。

2.4 第二章小结

本章分析了约束冗余问题和约束可行问题之间的对偶关系，并由此将 1.3 中关于约束冗余判断的结论进行了推广，得到了约束可行性问题判断的充要条件和充分条件。还对参考文献[2]中的快速判别算法结合本文讨论的问题进行了改进，使得可以不必排序快速找到 *GLP*2 的最优解。另外，利用快速判别算法后，可以得到一个方形的初始可行域，该方形区域也是下一章切割算法的初始可行域。

第三章 切割法简介

本章提出一种从几何角度出发考虑问题的新的算法，即“切割法”。该方法在初始可行域（2.3 提到的、由变量的各分量上下界构成的方形区域）基础之上，依次添加约束对可行域进行切割，最后得到可行域的所有边界点信息。

本章先介绍切割法的基本原理，然后给出切割算法的基本算法流程。

3.1 切割法原理

切割法是一个从几何角度考虑约束问题的思路，这是一个构造性的迭代过程。

线性规划得到的可行域是一个 N 维凸集（N 维欧氏空间的凸多面体）。切割法把每加入一个约束之前的可行域看作一个 N 维的凸多面体，每次且已经保存了上一步可行域的所有边界顶点信息。把新加入的约束看作一个 N-1 维的超平面，加入一个新约束就相当于用超平面“切割”。在新加入的约束之后，如果原来的可行域所有边界点都在新约束可行域内或可行域外（即没有切到），则新约束为冗余约束或约束不相容；如果原来可行域的顶点分居新约束可行域的内外，则原来的可行域被切割，这样的切割把可行域外的顶点去除，加入了新生成的顶点。

在 N 维的约束问题中，N 维空间的一个点的坐标是有 N 个坐标分量。同时， N 维空间的一个点也可以通过求解 N 个线性无关的方程得到。在约束问题中，每一个约束都是含有 N 个变量的线性不等式，而可行域的边界上的点就是这些不等式中的某些取等号的情形。

先不考虑奇异点的情况，可行域边界的顶点处有 N 各不等式取等号，所以可行域边界的顶点是由 N 个约束“相交”构成的（比如，在三维、无奇异点情况下，一个三维物体的每一个顶点都是由 3 个面相交构成）。所以有以下结论：

结论 3.1

N 维空间的可行域顶点，是由 N 个 N-1 维超平面（N 个约束不等式）相交构成。

由此，对于可行域的每一个顶点，除了可以用 N 个坐标表示之外，还可以用在该点相交的 N 个约束的编号来唯一确定。知道了这 N 个编号，如果想要确定这个顶点的坐标，只需要把对应的 N 个约束去等号，变为 N 个方程，然后联立即可求出这个顶点的坐标。**在切割法中，除了保存可行域顶点的坐标之外，还保存了每一个顶点所涉及的约束的序号。**

在一个约束（$a_k^T x \le b_k$）加入后，把上一步的所有可行域顶点 x_i 分为两侧：一些顶点在新约束的可行域内部，另一些顶点在新约束外部，新约束的超平面切割到了上一步的可行域，所以可行域将发生变化：在新约束可行域内部的顶点仍然是可行域的顶点；而在新约束可行域外部的顶点将被去除。另外，新约束的超平面切割上一步可行域会产生新点，关键是如何确定这些新产生的可行域顶点。

从直观（三维）上理解，如果原来的两个可行域顶点之间有一条“棱”，那么新加入的

约束切割到这条“棱”就会产生一个新的顶点。

那么如何确定一条“棱”？抽象理解，根据结论 3.1，N 个约束可以唯一确定一个顶点，如果放松一个自由度，N-1 个约束方程联立会得到一组解，这些解将会处于 N 维空间的一条线上（一个自由度），所以有以下结论：

结论 3.2

如果 N 维空间的两个顶点涉及的约束有 N-1 个是相同的，则这两个点之间的连线是可行域凸多面体的一条棱。

在可行域凸多面体的棱上的点都涉及 N-1 个约束，那么再加上新加的约束，正好构成 N 个约束，这 N 个约束进行联立即可确定一个新点，这个新点就是加入的新约束切割原来可行域而得到的。

所以，切割法的原理为：

新加入的约束将上一步的边界点分为两类。在新约束可行域内部的顶点保存，可行域外部的顶点舍弃。然后，依次考察每个在可行域内的顶点和每个在可行域外的顶点，如果一对顶点，一个在新约束可行域内，一个在新约束可行域外，且两者涉及的约束有 N-1 个是相同的，则把这 N-1 个约束与新加约束进行联立，得到的就是新的可行域。

以上就是切割法的基本原理，主要在于保留每个可行域顶点所涉及的约束用于产生新的可行域顶点。切割法在几何上有非常明确的意义，所以比较容易理解。

3.2 切割法算法流程

下面给出规范化的切割法处理流程，注意这里给出的只是切割法最基本最原始的算法流程，没有做任何细节上的优化，关于切割法的进一步讨论放在第四章。

3.2.1 切割法算法流程

符号说明

切割法是一个迭代过程，假设上一步结束时，得到的可行域边界点的集合是 dots{}，dotCons{i}保存了边界点 i 所涉及的约束的序号。每次加入新约束后，把 dots{}中的点放入两个集合：可行域内的顶点放入集合 Fdots{}，可行域外的顶点放入集合 NFdots{}。

切割法算法流程

（1） 可行域初始化：

初始的可行域即为前面提到的“方形区域”。对于 N 维的约束问题，初始方形区域包含有 2^N 个顶点（X 的 N 个分量的每一个分量 x_i 都可以取 Q_i 或 P_i），2N 和个约束（对于 N 个分量的每一个分量 x_i，都对应两个取值约束：$x_i \le Q_i$，$x_i \le P_i$）。

初始化的可行域主要包括对集合 dots{}和 dotCons{}的数据的初始化。即给出初始顶点的坐标（保存于集合 dots{}中）和给出每个顶点涉及的是哪几个约束（保存于集合 dotCons{}

中）。

对于维数 N 较大的情况，这些初始化的工作可能比较耗时，但是这样的信息可以离线计算好，以后对于同样的问题可以不必每次重复初始化。

（2） 加入新约束 k: $a_k^T x \le b_k$：

对于集合 dots{}中的每一个顶点 i（dots{i}），把顶点 i 的坐标（记作 X_i）带入该约束系数，计算顶点 i 对应的数值 $b_i' = a_k^T X_i$。

（3） 把顶点分为两类：

根据上一步计算得到的 b_i' 把顶点集合 dots{}分为两部分：

若 $b_i' < b_k$，则把顶点 i 放入可行点集合 Fdots{}；若 $b_i' \ge b_k$，则把顶点 i 放入不可行点集合 NFdots{}。（这里假设不会出现奇异情况，关于奇异情况的讨论见 3.2.2。）

如果 NFdots{}为空，说明这个新约束冗余，跳到步骤（6）；

如果 Fdots{}为空，说明这一组约束是不相容的，算法迭代结束，跳至（8）。

（4） 对于集合 Fdots{}和 NFdots{}的每一对组合进行检查：

对于可行点集合 Fdots{}中的每一个顶点 Fdots{i}，依次与不可行点集合 NFdots{}中的每一个顶点进行比较（比较两者涉及的约束）。

如果 Fdots{i}所涉及的约束（dotCons{ Fdots{i}}）与 NFdots{j}所涉及的约束（dotCons{ NFdots{j}}）有 N-1 个是相同的约束，则说明 Fdots{i}和 NFdots{j}之间有一条棱相连，此处会切割产生一个新点：

用 Fdots{i}和 NFdots{j}之间相同的 N-1 个约束加上新约束构成 N 个 N 维方程组，求解出新边界点的坐标 X_{new}，而新边界点涉及的约束（dotCons{new}）就是这 N-1 个约束加上新加入的约束。

然后把所有生成的新可行域边界顶点放入临时集合 newdots{}。

（5） 更新 dots 信息：

在集合 dots{}中，去掉 NFdots{}中的点，加入新产生的点 newdots{}。

（6） 进入下一次迭代：

如果约束还未加完，则加入下一个约束，跳至步骤（2）；若已经加完全部约束，跳至下一步。

（7） 得到所有必要约束

把所有约束加完后，得到的集合 dots{}即为可行域的所有顶点集合，而对应的所有必要约束可以从 dotCons{}中得到，这些约束就是去除了所有冗余之后剩下的所有必要约束。这时，dots{}和 dotCons{}也给出了关于这一组约束的可行域的所有信息。

（8） 结束

切割算法的流程图见图 3-1。

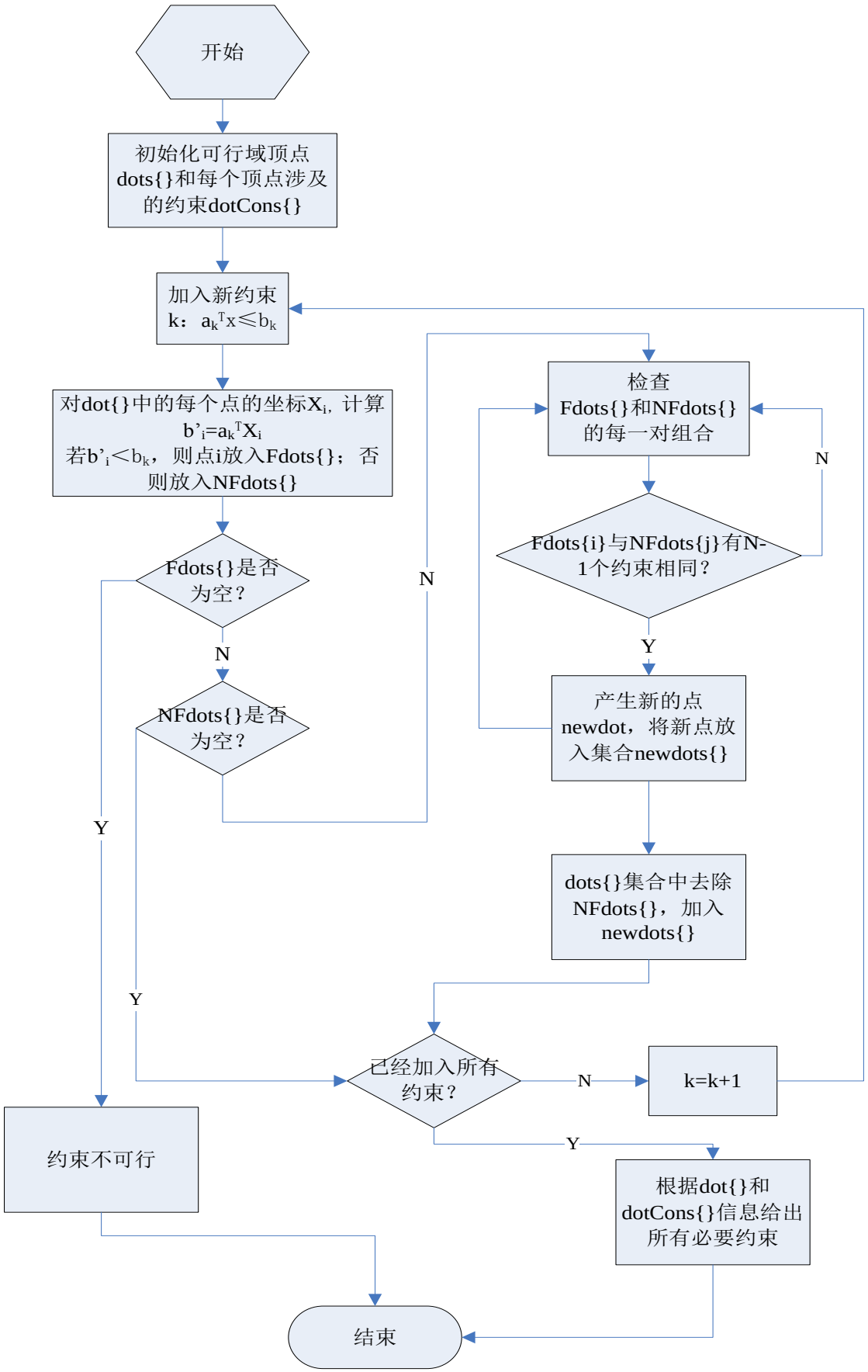

图 3-1 切割法算法流程图

3.2.2 切割法计算示例

本节用一个三维示例展示切割法一步迭代（一次切割）的计算过程。

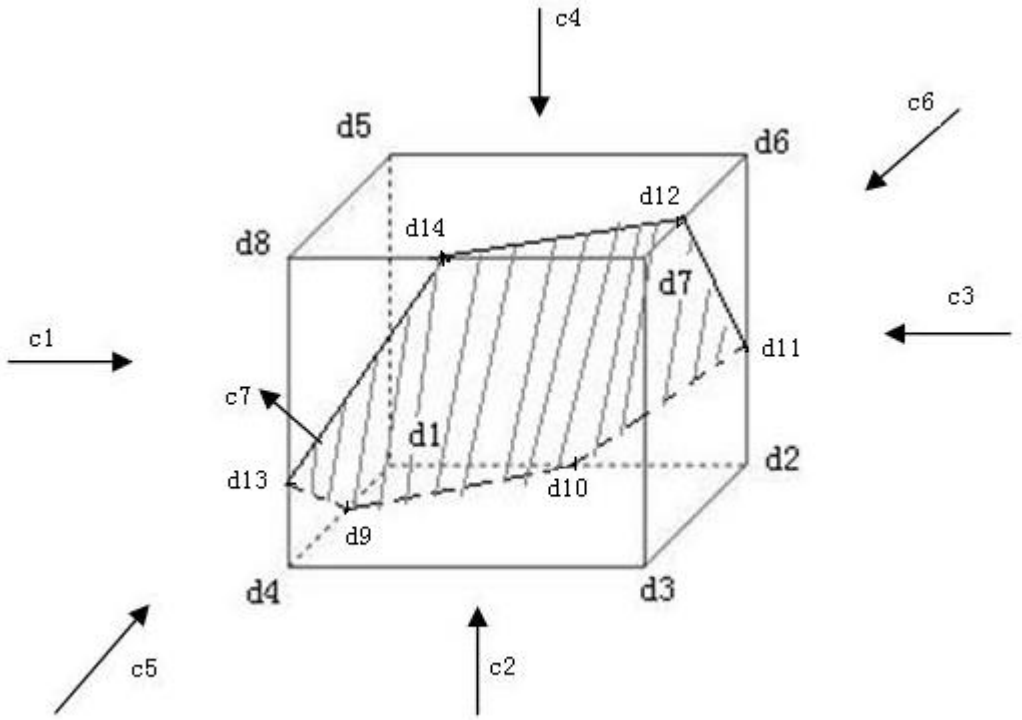

图 3-2 切割法示例

图 3-2 展示的是一个方形区域（方形区域的边界面为 c1~c6，边界点为 d1~d8）被一个约束（c7）切割时的情形，根据图中的序号，列出初始的信息表：

表 3-1 切割法信息表

	d1	d2	d3	d4	d5	d6	d7	d8
c1	√			√	√			√
c2	√	√	√	√				
c3		√	√			√	√	
c4					√	√	√	√
c5			√	√			√	√
c6	√	√			√	√		
加入约束 c7	√	×	×	×	√	√	×	√

表 3-1 的前六行表示了上一步迭代结束时剩余的可行域边界点，以及每个边界点涉及的约束，三维情况，每一格边界点涉及 3 个约束（以打钩标记）。

现在约束 7 加入，把表 3-1 中的点 d8~d10 分成两部分，d1、d5、d6、d8 处于约束 c7

的可行域内部（Fdots{}），用加粗的对号表示，d2、d3、d4、d7 处于可行域外部（NFdots{}），用加粗的叉号表示。

然后，一一检查 Fdots{}与 NFdots{}的每一对组合，产生新的边界点：

d1 和 d2 有两个约束 c2、c6 相同，→（c2，c6，c7）产生新点 d9；

d1 和 d4 有两个约束 c1、c2 相同，→（c1，c2，c7）产生新点 d10；

d6 和 d2 有两个约束 c3、c6 相同，→（c3，c6，c7）产生新点 d11；

d6 和 d7 有两个约束 c3、c4 相同，→（c3，c4，c7）产生新点 d12；

d8 和 d4 有两个约束 c1、c5 相同，→（c1，c5，c7）产生新点 d13；

d8 和 d7 有两个约束 c4、c5 相同，→（c4，c5，c7）产生新点 d14；

所以，这次切割产生了 6 个新的边界点 d9~d14。

这次切割后，将表格 3-1 更新：去除不可行点（NFdots{}）的那些列，加入新产生的顶点，更新后的表格为表 3-2：

表 3-2 经过一次切割之后的表

	d1	d5	d6	d8	d9	d10	d11	d12	d13	d14
c1	√	√		√	√				√	
c2	√				√	√				
c3			√				√	√		
c4		√	√	√				√		√
c5				√					√	√
c6	√	√	√			√	√			
c7					√	√	√	√	√	√

表 3-2 就是经过一次切割后得到的可行域信息表，再加入新的约束时，只需要在这个表的基础上按照如上的步骤进行计算、更新操作即可。

得到了最终的可行域信息表后，可以根据可行域边界点涉及的约束，求出所有的必要约束。

3.2.3 用切割法得到可行域的解析表示

由于切割法最终得到的是可行域的所有顶点的坐标，利用这些坐标就可以表示可行域内部任一点。事实上，用切割法得到的可行域顶点的凸组合可以表示可行域内部任意一点。假设可行域有 m 个顶点， 每个顶点坐标为 X_i，则可行域内部任一点的坐标表示为：

$$x=\sum_{i=1}^{m}\lambda_i * X_i, i=1,2,.....,m$$
$$0\le\lambda_i\le 1, \qquad (3.1)$$
$$\sum_{i=1}^{m}\lambda_i=1$$

所以，利用切割法，我们可以给出可行域的解析表达式，这样就是得到了关于可行域的最完整的信息。

3.2.4 奇异情况的处理

之前 3.1 和 3.2 的讨论是假定约束没有出现奇异的情况，假设不会产生奇异点（一个边界点涉及的约束大于 N 个就称之为奇异点）。在切割时发生奇异情况是指对于新加入的约束 k，某一点的坐标 X_i 刚好满足约束 k。

虽然在实际应用中，可能很少会遇到奇异情况发生，但是为了保证切割算法理论上的严密性，本节简单讨论一下出现奇异情况的处理。

产生奇异点是因为新加入的约束刚好切割到原来的边界点，此时奇异点仍然是可行域的顶点，并不会产生新的点，只是这一点所涉及的约束大于 N 个。

如果使用 3.2 中的程序，出现奇异的情况会导致产生两个（或多个）坐标相同的点，这两个点所涉及的约束各为 N 个，但实际这两个点的约束取并集才是这一个奇异点的完整的约束信息。另外，如果一个点已经是奇异点，在下一次切割的时候仍然刚好又刚好落在新约束上，那么可能它原来涉及的约束信息可能有的会变为冗余信息，而使用 3.2 的程序不会消除这样的冗余，导致最后给出的必要约束中有些实际上是冗余的。

使用 3.2 的程序，最后得到的可能不完全是必要约束，同时最后得到的边界点的数量会出现问题，而且一个奇异点用两个点表示可能会影响下一步的迭代。所以，对于奇异点要进行一下特别处理，处理方式如下：

加入一个约束后，对于 $a_k^T x \le b_k$ 的点，将其放入集合 Sdots{}中（这样就防止了产生与奇异点坐标相同的新边界点）。在处理完普通点（Fdots{}和 NFdots{}产生新点）之后，单独处理奇异点。

对于每一个 Sdots{}中的点，将其涉及的约束与目前所有的必要约束取交集（目前所有的必要约束可以从几何 dots{}得到），然后将取交集得到的结果再加上新约束，构成奇异点新的涉及的约束信息。

使用这样的方法，在新约束刚好切割到原来边界点时，不会增加新的边界点，同时也不会造成最后给出的约束中出现冗余约束的情况。不过，在实际问题中会很少遇到新约束切割到原来边界点的情况。

3.3 第三章小结

本章提出了切割算法， 该算法的核心就是利用 N 个（线性无关的）约束确定 N 维空间可行域的一个顶点，并且记录每一个顶点涉及的约束。切割法有较为直观的几何意义，其计算过程也比较简单。但是，本章提出的切割法可能在运行效率上不尽如人意，下一章主要讨论关于切割法性能的分析和一些改进的措施。

第四章 切割法的进一步讨论

第四章给出的切割算法是没有经过优化的，但其每一步的意义也是最清楚的。本章首先分析讨论切割法的运行时间复杂度，给出一些为了改进切割法的效率而采取的措施，然后讨论如何利用切割法得到的关于可行域边界的丰富信息为后续的计算提供帮助。

4.1 切割法的运行效率

4.1.1 切割法运行效率分析

切割算法的复杂度分析如下（假设问题是 N 维问题，共有 n 个约束，每一步迭代时可行域的顶点个数为 m）。

从图 4-1 可见，每一步迭代时，主要有三部分计算：

（1） 加入新约束，对所有的顶点进行计算

进行 m 次向量乘法操作；

（2） 将新约束可行域内外的点两两进行比较

假设可行域内有 m_1 个点，可行域外有 m_2 个点，则要进行 m_1*m_2 次求交际操作；

（3） 产生新的边界点

假设下一次迭代时的可行域顶点数为 m’，则这一次产生的新顶点数为 $m'-m_1$。所以要进行 $m'-m_1$ 次解方程组操作（矩阵求逆）。

以上三部分计算，（1）和（3）虽然是矩阵向量运算，但是可以认为其计算次数的复杂度为 O（m），而（2）虽然只涉及比较运算，但其复杂度为 O（m^2）。在维数较高、约束较多的情况下，可行域有可能包括很多顶点，此时若 m 很大，会造成切割法效率降低。之后的算例也印证了这样的分析。

4.1.2 切割法运行效率的改进措施

（1） 通过预处理优化约束加入次序

考虑图 4-1 中的情况，在依次加入了约束 1~6 后，可行域 R 是一个六边形，此时加入约束 7，约束 7 的加入使可行域变为了四边形，约束 1、2、3 都是冗余约束，所以之前计算的可行域的由 1、2、3 构成的顶点全部没有意义（不会出现最后得到的可行域边界中）。

如果能够先加入约束 4、5、6、7，那么当加入约束 1、2、3 时，约束 1、2、3 就会被判断为冗余约束，从而不会进行切割法的计算。

从这个简单例子中可以看到，约束的加入次序可能会影响切割法的效率，如果能够对约束加入的次序做一定调整，则有可能提高切割法的运行效率。

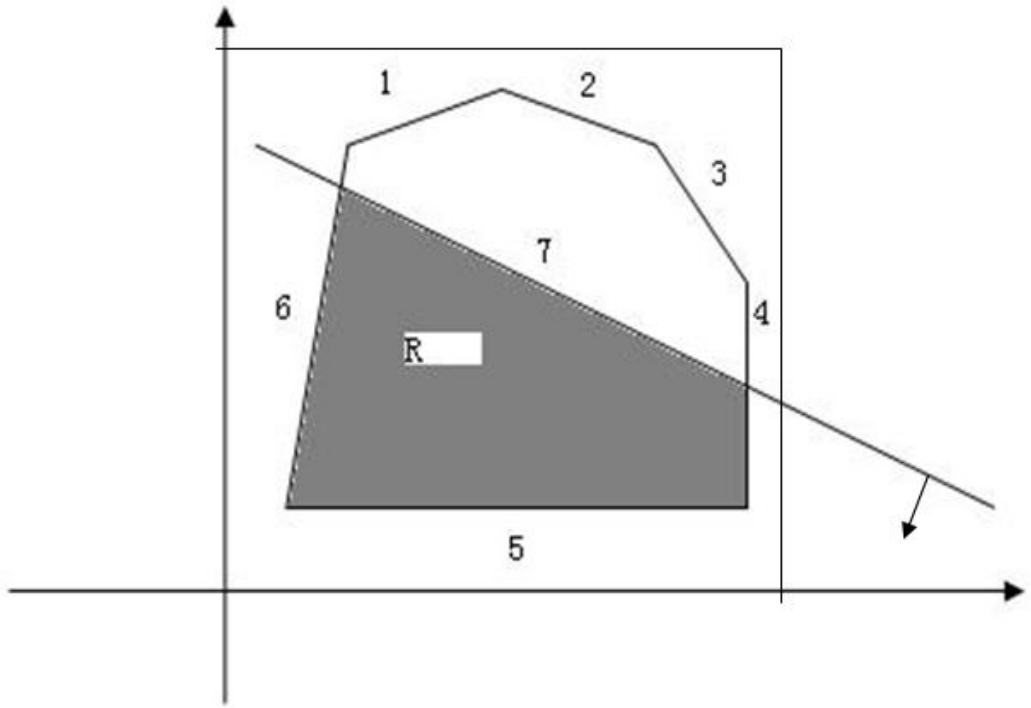

图 4-1 约束加入的次序对于计算效率的影响

从图 4-1 的例子中可见，相对于约束 1、2、3，约束 7 切入方形区域的程度更“深”，约束 1、2、3 则切入方形区域较“浅”。由此得到启发，如果优先加入切入方形域较深的约束，则后加入的约束可能更多的会被判别为冗余约束。

那么，如何定义各个约束切入方形域的深浅？

在第二章的冗余判断快速算法中，可以很快求出在方形区域内的 $a_k^T x$ 最大值，且已经讨论过，如果 $a_k^T x \le b_k$，则判断约束 k 为冗余，而若 $a_k^T x > b_k$ 则不能判断。我们可以利用这里计算得到的 $a_k^T x$ 来判断切入的深浅。

使用解析几何中点到平面的距离公式：

$$d = \frac{a_k^T x - b_k}{\| a_k \|} \tag{4.1}$$

用公式 4.1 得到的距离 d 就是 $GLP2$ 的最优解 x^* 这一点距离相应约束超平面的距离，从以上的分析可见，使用这样的距离作为描述约束切入方形域的深浅的判据是比较合理的，而且由于公式中的分子（$a_k^T x - b_k$）已经在快速判别算法中计算出，所以进行这样预处理不会造成过多的附加计算。

所以，利用快速判别法得到的数据（$a_k^T x - b_k$），对于所有没有被快速判别法判别出的约束，按照 $\frac{a_k^T x - b_k}{\| a_k \|}$ 的大小进行排序，$\frac{a_k^T x - b_k}{\| a_k \|}$ 数值较大的首先加入，数值较小的后面加

入。

使用这样的预处理方法对约束进行重新排序，使冗余约束放在相对靠后的位置加入，从而尽量避免了一些无谓的计算。

后来的实验数据证明，使用这样重新排序约束的与处理办法，可以大幅降低切割法的计算时间，很多约束由于后来加入，可以直接被判出是冗余的（图 3-1 流程图中的第二个判断），避免了求解无用的顶点。而使用这样的预处理方法，只涉及一次排序操作，附加代价很小，对于优化切割法的速度效果明显。

使用快速判别法预处理的程序流程见图 4-2，注意，这里同时使用了约束冗余和约束不可行两个快速判别法，如果在快速判别结算发现约束不可行，则直接结束。

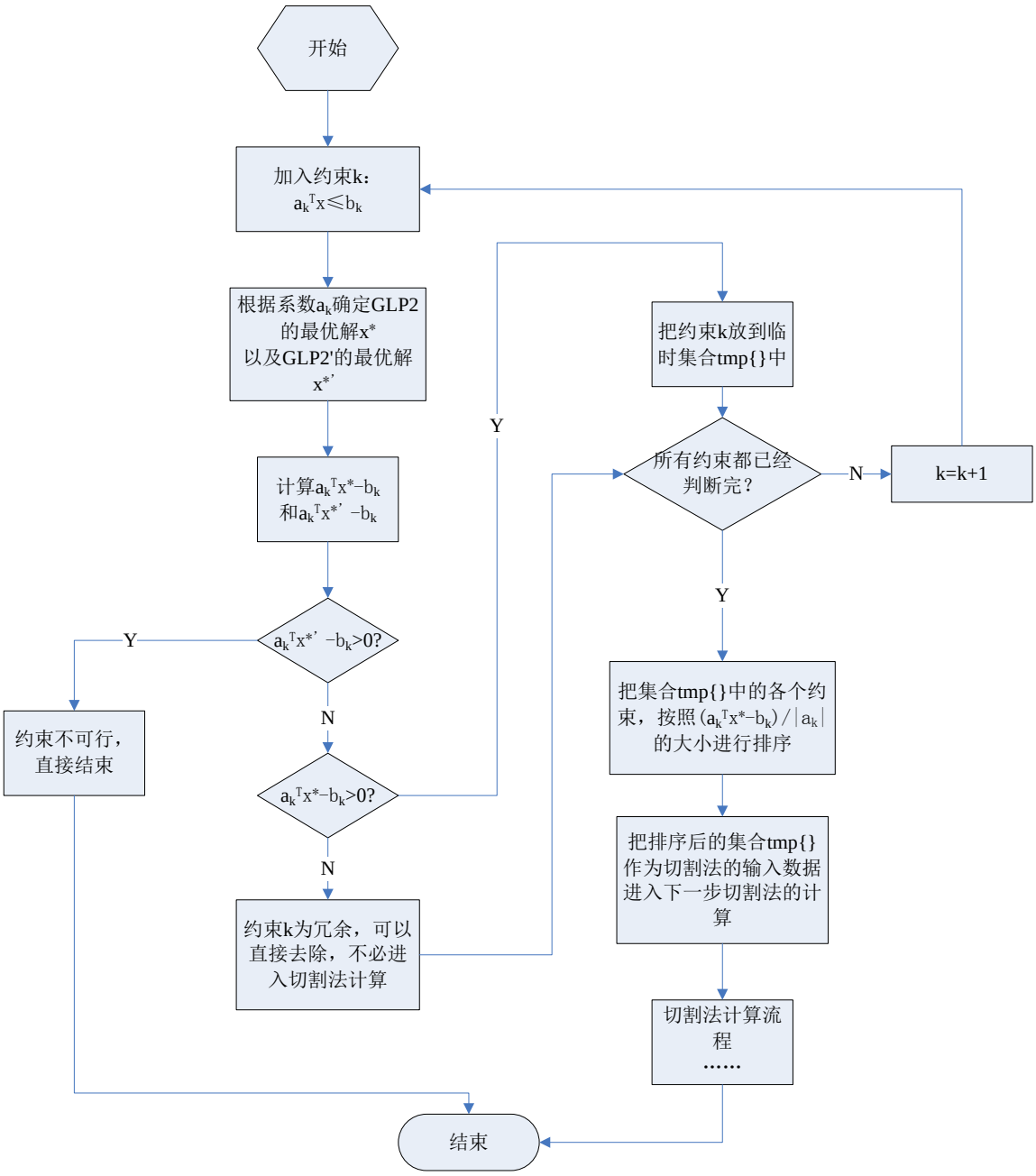

图 4-2 使用快速判别法以及对约束加入顺序进行优化

（2） 对于求交集操作的优化

根据 4.1.1 的分析，影响切割法效率的主要是 O（m^2）次的集合求交集操作。注意到这样的求交集操作是对两个顶点所涉及的约束（每一点涉及均为 N 个）求交集，同时，一旦两点涉及的约束有大于 2 个不一样，就已经判断出两点不会有连线，那么就不用继续比较，可以直接结束。

由于这样的操作在计算中大量重复，同时这样的求交集操作在大于两个元素不相同的时候可以直接结束，所以有希望简化这样的求交集操作。

首先，可以通过程序保证所有点所涉及的约束 dotCons{i}均为排序之后的：在初始化 dotCons{}时对 dotCons{}的每一行排序；之后在产生新边界点 newdot 时，是把求交集得到的 N-1 个约束首先放到 newdot 的约束信息中，最后再放入新约束的编号，所以一定每一个点涉及的约束信息都为递增排序好了的。

在保证 dotCons{i}均为排序这一点之后，可以通过一定的技巧减少求交集操作的计算量，下面是求两个向量（设为 v1 和 v2）交集的操作流程：

a) 设置指针 p1 和 p2，分别指向 v1 和 v2 的元素。

初始 p1=p2=1（指向 v1 和 v2 的首个元素）。

b) 比较 v1(p1)和 v2(p2)是否相等：

若 v1(p1)=v2(p2)，则 p1、p2 同时加 1，同时返回结果向量 intv{}加入 v1(p1)；

若 v1(p1)>v2(p2)，则 p2 加 1，p1 保持不变；

若 v1(p1)<v2(p2)，则 p1 加 1，p2 保持不变。

c) 比较 p1 和 p2:

若|p1-p2|<2，则继续调至（b）循环；

若|p1-p2|≥2，则说明 v1 和 v2 必定没有 N-1 个元素相同，直接结束。

d) 如果没有在（c）中结束，则循环至 v1（v2）的最后一个元素退出，返回它们共有的 N-1 个数字 intv{}。

注意以上的求交集操作只适合我们提到的这种比较特殊的情况：v1、v2 均只有 N 个元素；v1、v2 的元素都已经按升序排好；只要 v1 和 v2 有大于 2 个元素不相同就没有必要继续求交集。

算例测试显示，使用专门的求交集程序，可以使程序的运行速度提高约 10%（在一个算例中，计算时间从 0.53s 减少到 0.48s）。

（3） 增加可行域顶点的邻居信息

尽管使用了（1）和（2）中的方法后，切割法的计算效率得到了显著优化，但是没有从根本上解决在寻找产生新边界点的约束时的 O（m^2）的时间复杂度。这样的复杂度会导致某些时候点数较多时程序运行时间显著增加。

为了降低寻找新边界点的复杂度，对于每一个点，增加其邻居节点（和该点有连线的其他节点）信息：把每一个点 i 的邻居节点序号保存在数据 dotNeib{i}中。

这样做的好处是，在把边界点分成 Fdots{}和 NFdots{}两个集合以后，不必配对寻找每一个 Fdots{i}和 NFdots{j}，只要对于每一个 Fdots{}的点，看看它的邻居点有哪些是在集合 NFdots{}里的，即可确定该点与哪一个点相连产生新点。在这一步里所消耗的时间复杂度是 O（m）。

但是这种尝试没有成功，原因在于，对于新生成的点的邻居信息初始化工作。对于新产生的点，它的一个邻居可以确定是 Fdots{}中的对应点，而它的其他邻居则都将会是这一次切割产生的新顶点中的某些点。所以为了得到新顶点的邻居信息，需要把此次切割产生的所有新顶点两两进行比较。这里，邻居信息的初始化工作消耗的时间复杂度依然是 O（m^2），

在高维情况下，一次切割有可能产生几十个甚至几百个新的可行域顶点，所以仍然会有成千上万次的求交集运算。而如果不给出新顶点的全部邻居节点数据（或者新顶点的邻居信息数据不完整），则会影响下一步计算，可能会导致最后得到的可行域缺少顶点等问题。

实际算例测试结果也表明，加入邻居点信息并不能增加切割法的计算效率，有时甚至会使效率变差。

4.2 切割法运行算例和效率评价

4.2.1 一个切割法高维算例

这里给出一个 10 维、80 个约束的算例数据。十维变量每个分量的取值范围都是[0,8]（这里共 10 个变量、20 个约束，编号记为 1～20）。80 个约束（编号为 21~100）不包含每个分量的取值范围（20 个约束），这些约束可以写成矩阵形式：

$$Ax \le b \tag{4.1}$$

其中 A 为 80*10 的矩阵，b 为 80 行列向量，A、b 的数据分别为：

A=[

5.04,4.28,2.17,2.97,5.89,9.34,3.47,4.88,9.22,2.91,1.09;
4.92,9.54,7.18,4.61,4.88,5.38,9.76,5.15,6.86,7.00,2.75;
8.29,5.14,6.64,1.21,7.56,0.97,9.29,5.82,6.71,7.83,2.51;
3.97,7.51,0.47,2.42,0.73,6.71,5.32,7.37,2.93,9.92,4.11;
3.12,6.81,2.07,2.60,0.94,5.74,6.12,7.34,3.39,4.45,9.68;
4.91,2.41,5.42,7.79,0.95,1.09,1.68,2.58,5.79,6.13,7.00;
8.34,1.00,1.94,5.73,1.55,4.65,9.10,8.24,0.16,7.12,4.78;
8.66,6.43,9.26,9.51,0.69,7.99,0.62,5.93,7.69,3.88,1.34;
4.70,0.39,2.74,2.99,7.03,3.86,5.71,2.36,1.66,7.09,4.36;
2.17,1.01,4.61,1.35,6.68,4.02,5.65,5.21,1.01,6.57,4.46;
7.96,6.94,8.25,0.77,4.76,1.66,0.19,7.93,3.61,0.07,3.58;
8.26,9.76,0.69,6.79,6.64,7.87,9.74,2.07,4.17,9.11,5.38;
6.42,2.11,4.24,3.84,4.94,5.01,6.30,7.63,8.12,8.89,9.37;
9.06,8.51,8.92,1.14,6.78,4.42,7.77,0.28,3.45,0.35,8.55;
8.32,9.78,9.42,0.90,3.41,0.67,7.84,2.53,4.56,4.52,3.13;
4.57,1.85,9.50,4.00,7.80,3.39,9.55,2.62,9.37,9.51,8.25;
5.34,1.98,1.52,3.51,7.26,5.57,5.10,6.70,2.32,6.43,1.04;
4.61,9.28,7.13,7.22,4.64,9.83,1.54,6.21,0.23,2.90,4.09;
4.96,9.81,4.66,6.73,8.75,8.49,8.44,3.41,7.05,6.44,3.35;
5.71,2.02,1.49,0.83,9.30,4.33,7.16,8.57,5.07,4.42,2.46;
5.48,2.85,2.65,7.95,3.86,9.60,7.38,3.41,7.70,4.30,2.73;
0.57,0.97,8.60,1.69,5.90,5.05,5.57,9.16,6.73,1.74,9.32;
8.69,2.24,3.89,9.76,9.82,2.03,4.27,3.68,4.73,6.25,3.88;
1.28,6.59,3.33,4.70,0.48,3.94,8.84,0.32,6.58,2.95,0.36;
3.56,0.20,1.93,3.76,2.81,7.32,8.83,2.43,5.39,7.88,5.34;
6.82,3.91,1.73,8.58,8.02,7.22,5.78,5.12,7.88,1.95,3.82;
0.13,5.91,9.75,3.40,0.13,8.80,5.83,1.29,1.02,1.52,1.11;

2.82,0.65,0.17,2.01,4.26,6.62,8.47,6.18,7.73,6.60,4.92;
9.82,7.66,3.80,2.91,7.64,3.30,3.86,0.71,2.96,6.19,2.93;
8.28,9.11,9.81,7.38,6.21,7.88,8.86,7.30,9.15,5.22,2.91;
2.08,7.97,2.30,3.62,0.92,8.11,2.90,2.91,6.65,1.41,3.14;
9.92,0.88,9.02,4.27,5.07,1.03,1.33,0.81,4.53,8.53,7.60;
6.77,7.29,7.98,6.67,5.31,1.53,3.47,1.66,6.77,4.21,8.32;
2.70,7.10,7.22,6.34,2.59,0.66,7.01,6.35,3.65,5.18,0.11;
4.81,5.30,8.39,0.56,2.84,3.16,3.92,9.49,0.71,3.99,6.94;
4.63,5.73,4.28,8.16,7.60,0.28,9.75,5.48,2.89,1.40,0.99;
8.69,2.11,8.89,4.45,5.29,0.73,9.63,2.20,6.68,9.80,2.24;
3.11,4.44,7.12,1.28,2.73,4.87,6.81,5.89,7.68,0.77,1.60;
9.12,3.53,6.32,1.55,3.75,3.19,1.16,0.07,6.45,2.05,4.59;
5.23,3.21,5.64,8.68,9.79,2.13,4.28,0.12,2.66,6.00,9.59;
0.25,4.00,8.93,0.97,2.08,2.53,6.59,5.77,9.19,4.70,0.84;
9.28,7.75,2.46,1.15,6.38,9.02,1.56,6.69,4.15,7.88,8.91;
0.73,7.16,0.62,0.98,5.83,5.26,6.53,7.32,5.03,4.08,4.48;
8.19,5.68,7.03,1.46,7.64,7.70,6.18,6.75,0.56,5.88,2.19;
8.83,9.25,6.70,4.54,4.21,1.90,0.41,7.13,7.71,7.89,9.80;
3.09,0.93,7.40,3.79,1.03,5.02,0.63,8.13,8.07,0.09,8.76;
1.79,8.09,2.73,6.81,2.46,4.28,4.96,1.54,9.70,3.58,7.53;
9.53,9.96,6.16,1.98,0.82,0.54,7.88,2.40,5.29,5.32,9.51;
0.65,3.02,6.09,4.25,3.46,9.22,4.30,1.01,6.81,6.87,2.38;
5.14,2.15,1.48,7.38,1.65,2.51,5.74,4.33,2.93,6.29,3.51;
9.00,8.04,7.23,1.87,8.49,6.72,2.66,6.07,5.57,4.63,4.93;
5.02,8.24,2.24,0.68,2.68,2.08,8.28,1.67,4.06,1.54,7.91;
8.63,0.80,7.84,9.99,5.45,1.12,5.60,6.50,7.02,8.56,2.93;
9.09,3.74,2.39,4.58,2.92,7.36,5.53,2.27,5.41,8.87,2.41;

7.84,8.99,7.75,7.04,1.62,7.11,4.38,0.28,6.91,6.82,0.88;	**4.09,3.26,7.54,9.93,4.94,8.40,8.51,9.89,7.21,7.70,0.27;**
2.60,3.58,1.19,9.83,3.95,4.04,0.46,7.17,6.12,3.93,8.01;	3.19,6.18,1.54,6.01,4.97,6.28,5.96,9.45,3.47,2.16,2.05;
0.33,9.56,3.10,4.52,8.35,4.51,6.89,1.36,1.87,9.97,1.91;	1.02,4.14,0.44,8.60,9.87,4.92,8.56,9.13,3.70,0.39,8.75;
0.38,1.33,8.98,7.92,9.03,1.37,6.60,4.57,6.21,0.03,6.38;	5.02,8.28,9.19,3.42,7.73,7.16,0.27,7.91,9.87,5.20,7.34;
2.47,0.81,6.22,2.72,1.21,3.92,7.93,6.88,5.79,2.47,2.16;	1.88,6.69,9.60,3.34,0.97,5.76,6.33,6.26,8.20,1.06,0.28;
6.82,1.51,3.78,8.31,0.06,0.75,9.68,3.97,6.18,2.35,9.13;	8.03,1.17,3.33,1.66,8.18,4.58,7.52,1.03,0.76,4.24,3.26;
2.53,4.05,2.09,5.07,7.51,1.76,2.78,6.40,9.12,2.57,0.15;	2.49,0.90,2.27,0.10,1.20,5.97,0.91,9.99,3.08,3.97,1.90;
4.17,5.47,1.09,7.34,5.34,2.96,6.70,3.97,6.54,5.14,9.78;	3.58,9.10,0.63,9.50,1.25,7.00,1.00,9.22,2.73,2.79,6.30;
5.00,9.92,4.98,0.36,3.05,5.71,6.13,3.63,9.11,5.61,6.75;	4.24,2.96,4.68,4.56,3.05,7.05,5.29,2.73,9.82,9.00,1.78;
0.71,6.78,0.52,4.70,1.13,7.04,2.09,4.45,2.77,5.54,5.23;	1.63,4.02,5.66,9.91,1.20,1.04,6.76,0.93,5.15,1.49,0.33;
3.10,2.66,9.13,8.23,8.38,1.68,2.65,6.79,2.47,9.47,4.89;	7.78,2.51,5.52,0.47,0.70,3.84,9.71,4.28,2.53,6.94,3.69;
7.61,6.38,3.10,3.07,8.16,2.47,1.50,9.22,4.40,9.21,4.12;	7.71,1.11,3.54,1.52,9.38,9.55,2.45,0.06,8.33,2.05,4.89;
3.27,3.10,3.92,4.11,1.86,8.09,3.47,8.02,7.40,2.67,6.62;	3.14,9.79,0.47,8.67,8.48,8.24,3.52,8.55,5.06,7.01,3.37;

]

b=

[1.09,2.75,2.51,4.11,9.68,7,4.78,1.34,4.36,4.46,3.58,5.38,9.37,8.55,3.13,8.25,1.04,4.09,3.35,2.46,2.73,9.32,3.88,0.36,5.34,3.82,1.11,4.92,2.93,2.91,3.14,7.6,8.32,**0.11**,6.94,0.99,2.24,1.6,4.59,9.59,0.84,8.91,4.48,2.19,9.8,8.76,7.53,9.51,2.38,3.51,4.93,7.91,2.93,2.41,0.88,8.01,1.91,6.38,2.16,9.13,**0.15**,9.78,6.75,5.23,4.89,4.12,6.62,**0.27**,2.05,8.75,7.34,0.28,3.26,1.9,6.3,1.78,0.33,3.69,4.89,3.37]^T

经过预处理后，约束（序号 21～100）加入的顺序为（其中加粗的是最终判别为必要的约束）：

[50,22,39,**88**,71,33,46,82,37,41,64,**54**,96,23,75,89,91,74,100,21,53,73,32,28,43,87,58,86,70,83,36,65,**81**,25,57,49,62,56,92,40,69,85,38,29,30,79,63,67,26,35,45,60,48,24,61,77,76,98,51,55,44,84,27,42,68,93,34,90,59,72,97,78,95,80,31,52,99,66,47,94]

可见必要约束都放在叫靠前位置加入，从而后面的约束直接被判别为冗余，减少了计算量。如果按照约束的编号放入，则约束 88 将会很靠后放入，前面会有大量无用计算。

经过切割法计算，得到必要约束为 1，2，3，4，5，6，7，8，9，10，88，54，81 共 13 个。其中约束 1～10 是保证每个变量大于 0 的约束，所以 80 个约束有 3 个必要约束，即为 A 和 b 中加粗的部分，这一结果与用线性规划方法求得的结果一致。而且可以得知，可行域是一个有 48 个顶点构成的 10 维图形，这些顶点的坐标为（每行是一个顶点的坐标值）：

dots=[

0, 0, 0, 0, 0, 0, 0, 0, 0, 0

7.3293e-18, 0, -7.9518e-18, 0, 0, 3.2143e-02, 0, 0, 0, 0

4.0741e-02, 0, 0, 0, 0, 0, 0, 0, 0, 0

4.5230e-18, 0, 1.5235e-02, 0, 0, 0, 0, 0, 0, 0

4.5230e-18, 1.7205e-18, 0, 1.7350e-02, 0, 0, 0, 0, 0, 0

0, -3.4398e-18, 0, 0, 0, 0, 0, 1.7323e-02, 0, 0

0, 1.5493e-02, 0, 0, 0, 0, 0, 0, 0, 0

0, -3.4398e-18, 0, 0, 0, 0, 1.5692e-02, 0, 0, 0

0, 0, 0, 0, 0, 0, 0, 0, 0, 2.1236e-02

3.7326e-02, -1.5315e-18, 0, 0, 0, 1.3969e-02, 0, 0, 0, 0

-3.0357e-18, 0, 1.3396e-02, 0, 0, 2.0118e-02, 0, 0, 0, 0

9.0087e-18, 5.7670e-18, 2.0210e-18, 1.5969e-02, 0, 1.3265e-02, 0, 0, 0, 0

-1.6181e-18, 7.2098e-18, 3.4353e-18, 3.6578e-18, 0, 1.3385e-02, 0, 1.5932e-02, 0, 0

0, 1.2973e-02, 5.7693e-18, 3.4725e-18, 0, 2.7108e-02, 0, 0, 0, 0

-5.8525e-19, 2.8886e-18, 4.5622e-18, 3.6441e-18, 0, 1.7958e-02, 1.4001e-02, 0, 0, 0

-1.5616e-18, 7.2098e-18, 1.5906e-18, 1.1869e-18, 0, 1.4353e-02, 0, 0, 0, 1.9407e-02

6.5825e-18, 0, 0, 0, 2.2181e-18, 0, 0, 0, 1.6447e-02, 0

0, 0, 0, 0, 1.9973e-02, 0, 0, 0, 0, 0

-1.7682e-19, 2.3826e-18, 3.6541e-20, 0, 3.2041e-18, 2.1604e-02, 0, 0, 1.2278e-02, 0

-2.6475e-19, 2.4746e-18, 0, 0, 1.4429e-02, 2.3657e-02, 0, 0, 0, 0

2.9611e-02, 0, -1.1141e-18, 0, 0, 0, 0, 0, 8.2329e-03, 0

3.1885e-02, -2.0020e-18, 0, 0, 9.2318e-03, 0, 0, 0, 0, 0

-4.2637e-18, 0, 7.8275e-03, 0, 0, 0, 0, 0, 1.4654e-02, 0

-3.2352e-18, 6.5780e-18, 8.9655e-03, 4.0557e-18, 1.7478e-02, 0, 0, 0, 0, 0

-3.9760e-18, -4.0365e-18, 0, 1.1591e-02, 1.2662e-18, 0, 0, 0, 1.0004e-02, 0

-2.7343e-18, 6.0785e-18, -3.7396e-18, 1.2691e-02, 1.1406e-02, 0, 0, 0, 0, 0

-4.1449e-18, -4.1410e-18, -7.0338e-19, 3.7513e-18, -3.1125e-19, 0, 0, 1.3189e-02, 7.1921e-03, 0

-3.8479e-18, 5.4696e-18, -3.0584e-18, 3.5736e-18, 7.9871e-03, 0, 0, 1.4065e-02, 0, 0

-1.8824e-18, 9.1196e-03, 0, 0, 1.6055e-18, 0, 0, 0, 1.2398e-02, 0

-2.5984e-18, 1.0217e-02, -4.7750e-18, 0, 1.4464e-02, 0, 0, 0, 0, 0

-1.2084e-17, -2.1485e-19, 2.6741e-18, -1.2985e-18, 3.5193e-18, 0, 8.4727e-03, 0, 1.3865e-02, 0

-1.1399e-17, 8.7473e-18, -3.0584e-18, 3.6892e-18, 1.6409e-02, 0, 9.6292e-03, 0, 0, 0

-1.2342e-17, 2.2477e-19, -1.3897e-18, -1.4718e-19, -6.3295e-20, 0, 0, 0, 1.3056e-02, 1.2036e-02

-1.0844e-17, 8.1043e-18, -4.0549e-18, 5.2198e-18, 1.5329e-02, 0, 0, 0, 0, 1.3571e-02

2.9945e-02, 2.5857e-19, -1.4695e-18, 0, -1.3965e-18, 1.2675e-02, 0, 0, 5.6943e-03, 0

3.1505e-02, 1.7547e-19, -9.5110e-19, 0, 6.2888e-03, 1.3105e-02, 0, 0, 0, 0

3.1624e-18, -2.9512e-19, 7.9314e-03, 4.1663e-18, 1.5248e-18, 1.4941e-02, 0, 0, 1.1746e-02, 0

1.5642e-18, 0, 8.8339e-03, 4.2580e-18, 1.3733e-02, 1.6137e-02, 0, 0, 0, 0

-2.0970e-18, 8.0470e-19, -1.5320e-18, 1.1711e-02, -1.8379e-18, 1.1709e-02, 0, 0, 7.6771e-03, 0

-1.7316e-18, 7.9973e-19, -1.5135e-18, 1.2549e-02, 8.6353e-03, 1.2229e-02, 0, 0, 0, 0

1.9937e-18,-5.9885e-19, 8.6261e-19, -3.9240e-19, -6.4201e-19, 1.2414e-02, 1.9292e-18,1.3334e-02, 4.6943e-03,0

2.7260e-18, 7.9973e-19, 2.3485e-19, 2.9244e-19, 5.1373e-03, 1.2754e-02, 2.1092e-18, 1.3902e-02, 0, 0

-2.2346e-19, 9.2940e-03, -2.3046e-18, 1.1400e-18, -2.0493e-18, 2.1527e-02, 0, 0, 8.1658e-03, 0

1.7568e-19, 1.0004e-02, -2.1414e-18, 0, 9.2269e-03, 2.2834e-02, 0, 0, 0, 0

-2.9994e-19, -4.6307e-19, -1.4199e-18, 3.6050e-19, -3.4347e-18, 1.3879e-02, 8.5772e-03, 0, 1.1154e-02, 0

7.4664e-18, 7.9973e-19, -4.0960e-18, 6.3023e-19, 1.2967e-02, 1.4894e-02, 9.4987e-03, 0, 0, 0

2.8785e-18, -4.6920e-19, 7.7406e-19, 6.3423e-19, -2.6514e-18, 1.1768e-02, 0, 0, 1.0749e-02, 1.2162e-02

1.7932e-18, 7.9973e-19, -2.6458e-18, 1.1213e-18, 1.2447e-02, 1.2525e-02, 2.1092e-18, 0, 0, 1.3416e-02

]

利用以上顶点的坐标信息，结合公式（3.1）就可以得到可行域的解析表达式。

4.2.2 算例测试结果

用 matlab 编程实现了切割法的算法流程，采用 4.1.2 中提到的措施增加其计算效率，并使用了一些算例进行测试检验。为了与切割法的计算效率进行比较，还另外用 matlab 编程实现了单纯性表上作业法（M 法）的程序。[7]

采用一些算例进行计算，切割法和单纯形法的计算速度比较如表 4-1。

从表 4-1 可以看出，在所试验的算例情况下，切割法的速度均优于单纯形算法。但是，

仔细观察，发现切割法的速度还是不太稳定。比如在十维、80 个约束的情况下，最好情况（0.48s）比最差情况（1.11s）快了一倍，而单纯形法在同样的规模下的计算时间却相对稳定。这说明切割法的运行效率与具体的约束情况密切相关，而且，在没有使用 4.1.2 中优化约束加入顺序之前，高维算例的计算时间最坏有十几秒，可见优化约束加入顺序的操作十分必要。

表 4-1 切割法运行速度测试结果

算例编号	维数	必要约束数/总约束数	单纯形法	切割法(优化约束加入次序后)	切割法(未优化约束加入次序)
1	3	5/28	0.074s	0.027s	0.031s
2	3	3/7	0.021s	0.016s	0.029s
3	3	6/6	0.020s	0.019s	0.019s
4	10	3/80	1.12s	0.48s	2.48s
5	10	3/80	1.21s	1.11s	2.36s
6	10	5/80	1.56s	0.67s	12.41s

统计各步骤所占时间，也发现切割法的运行瓶颈就是在寻找新边界点时进行的求交集运算。在某些时候，这种运算占据了超过 90%的时间，进行了成千上万次的求交集运算，而其中的大部分计算实际上没有得到 N-1 个相同的约束。所以，在维数高、约束数量多的情况下，约束可行域的边界点会有很多，而如果没有优化约束加入的次序，某次切割时约束可行域顶点在新约束可行域内外各有几十个，则会造成求交集运算量大增，从而减慢切割法的计算。所以，切割法的运行效率并不很稳定，其计算速度与具体约束以及约束加入的先后顺序有很大关系。

而采用线性规划（单纯形法），每一次都选择一个使得目标函数值增加最多的方向，从一个顶点（基可行解）移动到另一个顶点，并不需要遍历所有的可行域顶点即可找到线性规划的最优解，这样的操作与约束的具体数值关系不大。

4.3 切割法优缺点评价

切割法的缺点有两点：

第一点，算法是一个构造性过程，需要在初始可行域的基础上依次增加一个约束，不能够一次判定一组约束，而且下一步计算结果正确的前提是上一步得到正确的结果；

第二点，由于要两两比较可行域内外的边界点，切割法的运行效率可能很不稳定，在同样的问题规模下，运行的效率和具体的约束情况以及约束的加入次序有很大关系（如表 4-1 的算例 4、5）。

与此同时，切割法也有两个明显的优点：

第一，是切割法在计算结束后可以得到关于可行域非常完整的信息，包括可行域顶点的坐标、包含的约束，这些信息可以在后续的计算中发挥作用；

第二，切割法是一个迭代过程，下一步可以利用上一步的结果，而不必重复计算，单纯形法则每加入一个约束都要再次计算。

4.4 切割法的应用展望

由于切割法最后得到了关于可行域的很详细的信息，可以使用这些信息进行一些应用。

（1） 对于不相容约束提出可参考的松弛方案

对于线性规划算法，在判断一组约束是不相容的之后，只能得到这一组约束是不相容的结论。关于哪些约束不相容、如何放松这些约束等信息并没有给出。而若使用切割法，几何上已经完全掌握了可行域的信息，这样就可以为约束的放松方案提出参考。

比如，在加入约束 k 之后，所有顶点的 $a_k^T x-b_k$ 均为正值，则加入约束 k 后约束会变为不可行，此时，可以通过松弛约束使可行域出现。设所有顶点的 $a_k^T x-b_k$ 中的最小值是 Δ（>0），则要使得约束相容，需要增大 b_k 使某些点位于约束 k 的可行域内部，所以，b_k 至少要增大 Δ，变为 $b_k'=b_k+\Delta$，才能保证有原可行域的顶点位于约束 k 的可行域内部，约束问题才有解。

另外，对于一组约束，可能有些约束是不能够放松的，而对于另外一些约束，放松一些并不会造成太大影响。一个例子就是城市交通控制中，限定城市中心的车辆排队长度不能超过道路长度的 40%，这个约束如果再放松会导致市中心交通状况恶化；而对于郊区道路，由于其重要性不大，把其约束从排队长度小于道路长度 40%放松成 60%也不会造成很大损失。所以在约束不相容的情况下，应当优先放松那些重要性较小的约束。此时，可以先把必须满足的约束加入，之后再加入重要性较小的约束，这样如果出现了约束不相容情况，进行放松的就是那些不重要的约束。

（2） 得到可行域内部一点

在很多情况下，人们并不一定需要得到最优的结果，往往只需要一个符合要求的可行解即可。线性规划方法得到的是可行域的边界点，在此边界点，会有 N 个约束取等号（active）。虽然问题描述中，所有的约束都是小于等于的，但很多时候约束要求的是严格不等号，不能出现等号的情况。而且选取可行域的一个顶点作为可行解，在系统参数变化、模型误差等因素的影响下，选取的这一点容易变为可行域之外的点，从而造成问题。

而采用切割法后，知道了可行域的所有的顶点，而可行域又是一个凸的几何体，所以可以通过这些可行域顶点的凸组合来表示可行域内部的任意一点（公式 3.1），这样就可以选取可行域内部某点而不是可行域边界上的点作为可行解。比如，任取两个不在同一约束超平面上的点，选择这两点的中点作为可行解，则在这个可行解处所有约束都是取严格的不等号，用这样的点作为可行解就不会出现上面提到的问题。

（3） 计算可行域的体积

在（2）中提到，可以选择可行域内部某一点作为可行解，但是，控制问题中同样有很多属于约束优化问题。需要在一定的约束下使得性能指标达到最优（或满意），如果问题的

可行区域过小，则可能在此可行域内得到的最优结果的控制效果也不好，会出现输出超调过大、震荡明显等问题。

所以，可行域的体积越大，说明进行优化求解的选择空间越大，控制效果就会越好。如果可行域的体积小于一定值，虽然这一组约束是相容的，也要选择主动放松某些约束，牺牲某些不重要的约束条件再做优化，以达到满意的控制效果。若使用的是线性规划方法，得到约束相容后不会考虑可行域是否太小，从而可能达不到太好的控制效果。

从另一方面，如果发现优化得到的解不能有很好的控制效果，也可以相应放松约束，利用可行域顶点信息，选择一个使可行域扩张尽量大的约束进行放松。

4.5 第四章小结

本章对于切割法的运行效率进行了深入讨论，分析表明切割法效率的瓶颈在于产生可行域新顶点时的检查操作。为了提高切割法效率，进行了一系列的改进措施，其中，优化约束的加入顺序对于切割法的效率有显著提高。虽然测试的算例表明切割法的速度不差于单纯形法，但是也可以看出切割法效率并不稳定，与具体约束密切相关。另外，由于切割法可以提供非常详细的可行域信息，可以考虑充分利用这样的信息，以减少约束优化问题的计算量。

第五章 结论

约束可行性问题和冗余性问题在控制系统中应用十分广泛，其判别问题被广泛研究。使用线性规划的方法判断冗余是该类问题的经典方法，而参考文献[2]提出的充分条件可以避免线性规划的迭代过程，虽然只是约束冗余的充分条件，但是在电力系统应用中可以判断出相当多的冗余约束，其放松约束条件的做法很有启发性。

本文指出了约束冗余性判断问题和可行性判断问题具有某种对偶关系，进而把关于冗余判断的充要条件和充分条件推广至约束可行性问题中。仿照参考文献[2]的方法，提出了约束不可行快速判别的充分条件，且对于问题描述中的情况进一步简化了参考文献[2]的方法，可以不必排序，直接由约束系数的符号得到相应的 GLP2 的最优解 x^* 。

之后，又从快速判别算法留下的初始可行方形区域出发，从几何角度提出了切割算法，通过每个约束依次切割可行域的方法，得到最后的完整可行域信息（给出可行域的解析表达式）。切割法几何意义直观，算法流程也不难理解。虽然算例显示切割法速度优于单纯性算法，但其效率不很稳定，与具体约束数据和约束的加入顺序密切相关。

利用切割算法的结果，可以掌握关于可行域的非常完整的信息，这些信息在松弛不相容约束、得到可行域内部一点以及主动扩大可行域范围等方面可以派上用场。

参考文献

[1] R. J. Caron et al, Classification of linear constraints as redundant or necessary[R], Windsor Mathematics Report，WMR85-09, 1985

[2] Q. Zhai et al, Fast Identification of Inactive Security Constraints in SCUC Problems[C], IEEE Trans. on Power Systems, 25(4):1946-1954, 2010

[3] 葛荣荣，基于非合作博弈的异构目标生产调度研究，第五章[D]上海交通大学硕士论文，2007

[4] 张惜岭，王书斌，罗雄麟，化工过程约束优化控制的可行性分析及约束处理[J]，化工学报，2011

[5] 张惜岭，王书斌，罗雄麟，过程预测控制中约束可行性研究与在线调整[J]，化工学报，2012

[6] 李利利，姚建国，耿健，徐帆，SCUC/SCED 问题分析[J]，江苏电机工程，2010

[7] 甘应爱，田丰，等，运筹学[M]，清华大学出版社，2005

[8] Y. Fu and M. Shahidehpour, Fast SCUC for large-scale power systems[C]，IEEE Trans. Power Syst., vol. 22, no. 4, pp.2144–2151, Nov.2007

[9] J. J. Shaw, A direct method for security constraints unit commitment[C]，IEEE Trans. Power Syst., vol. 10, no. 3, pp. 1329–1339, Aug. 1995.

[10] Robert G. Bland, New Finite Pivoting Rules for the Simplex Method[M]，Mathematics of Operations Research 2 (1977) 103-107

[11] M.I. Norbis and J.M. Smith, A multi-objective multi-level heristic for dynamic resource constrained scheduling problems [J], European Journal of Operational Research, 1988, 33, 30-41

[12] J. F. Benders, Partition procedure for solving mixed-variables programming problems [J], Numer. Math., vol. 4, no. 3, pp. 238–252, 1962.

谢辞

我非常有幸可以在席裕庚老师的指导下进行毕业设计，这也是我第一次参与一个科研探索项目，我很兴奋，也很享受整个毕业设计的过程。

约束问题是一个经典问题，同时也比较抽象，很有挑战性，席老师在整个过程中给了我很大帮助，我们经常进行讨论，在这样的讨论中我渐渐了解了这个问题，并进入问题研究的轨道。在提出切割算法的过程中，我经常遇到各种各样问题，有时甚至感觉没有希望，但是在席老师的帮助下，终于提出了规范化切割法算法流程。虽然这个算法有明显的几何意义，但是提出它也花费了一番功夫。

我和席老师每周进行一次关于约束问题进展的讨论，每一次讨论都让我受益良多。席老师严谨的态度、开阔的思路和遇到困难时的执着等品质给我很深刻的印象！在此，我非常感谢席老师在毕业设计过程中对我的帮助和指导！同时，我也要感谢复杂系统控制实验室的师兄们给予我的帮助！