中国科学院大学
University of Chinese Academy of Sciences

硕士学位论文

混合整数线性规划中冲突图的研究及其应用

作者姓名：张吾帅君

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

2021 年 6 月

Research and Application of Conflict Graph in Mixed Integer Linear Programming

A thesis submitted to the

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Natural Science

in Computational Mathematics

By

Zhang Wushuaijun

Supervisor: Professor Dai Yuhong

Academy of Mathematics and Systems Science, Chinese Academy of Sciences

June, 2021

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。本人完全意识到本声明的法律结果由本人承担。

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大学有关保存和使用学位论文的规定，即中国科学院大学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

混合整数线性规划是指部分决策变量取整数值的线性规划问题, 在实际生活中广泛应用于行程安排、生产排班、通信等方面. 而冲突图作为混合整数线性规划求解器中一种特殊结构, 利用二元变量不可行的信息加速问题求解. 本文主要研究冲突图在探测算法 (probing)、团聚合算法 (clique aggregation) 以及辛烷算法 (octane) 中的应用. 前两个算法是预处理算法, 第三个是启发式算法, 三个算法皆用于处理包含二元变量的一类问题. 本文从算法理论分析, 算法时间复杂度, 算法流程图三个方面分别叙述算法, 并给出数值实验, 验证算法的有效性.

探测算法是基于冲突图对模型变量进行预处理的算法, 通过改变变量的界, 使一些变量被固定到某一个确切值, 或能用其他的变量表示, 因此能减少变量的规模. 本文提出一种基于 gmp 库的大整数数据结构, 在这种数据结构中实现探测算法, 并通过对标准算例库中 27 个算例进行数值实验验证大整数结构的准确性, 分析大整数结构下高精度求解对算法的影响.

不同于探测算法, 团聚合算法是基于冲突图对模型约束进行预处理的算法, 通过冲突图减少约束个数, 从而缩小约束规模. 在一个变量数超六万, 约束数超七百万的机位分配问题实际模型数据集的测试中, SCIP 中团聚合算法无法求得最优解. 本文在邻接矩阵的数据结构中保存冲突图, 并通过对邻接矩阵特征分析扩充团表中的团为极大团, 使实际问题能在两分钟内求得最优解.

辛烷算法是基于纯 0,1 规划问题提出的启发式算法, 从几何的角度出发, 在自定义的一个距离下, 寻找距离当前松弛解最近的整数解. 本文提出了一种基于冲突图提供的冲突信息的新搜索方向, 并与基于目标函数系数的搜索方向通过机位分配问题抽象的模型进行对比, 通过数值实验说明基于冲突图的搜索方向在该模型中效果更好.

关键词： 二元变量, 冲突图, 探测算法, 机位分配, 团聚合, 辛烷算法

Abstract

Mixed integer linear programming refers to the integer value of some decision variables, which is widely used in transportation schedules, production scheduling, and communications. As a special structure in the mixed integer linear programming solver, the conflict graph uses the infeasible information of binary variables to speed up the problem solving. This thesis mainly studies the application of conflict graphs in probing, clique aggregation and octane. The first two presolving algorithms, and the third heuristic algorithm, both deal with a class of problems containing binary variables. It mainly introduces the algorithm from three aspects: algorithm theory analysis, algorithm time complexity, and algorithm flow chart. Numerical experiments are given to verify the effectiveness of the algorithm at the end of each chapter.

As a presolving algorithm, the main effect of probing is to change the bounds of variables. By the bounds of variables change, some variables can be fixed to a certain value, or can be represented by other variables, thus it can reduce the amount of variables. This thesis proposes a large integer data structure based on the gmp library, in which the detection algorithm is implemented. Numerical experiments on 27 examples in the standard example library illustrate the accuracy of the large integer structure and analyze the impact of high accuracy on the algorithm under the large integer structure.

Different from the detection algorithm, the clique aggregation algorithm is a presolving algorithm to deal with constraints. The purpose is to reduce the number of constraints through the conflict graph, thus reducing the scale of constraints. In the test of the actual model data set of aircraft stands assignment problem with the number of variables exceeding 60,000 and the number of constraints exceeding 7 million, the cliques aggregation algorithm in SCIP could not find the optimal solution. In this thesis, the conflict graph is stored in the data structure of the adjacency matrix, and any clique in the clique table is expanded to a maximum clique by analyzing the characteristics of the adjacency matrix, so that the actual industrial problem is successfully solved.

The octane algorithm is a heuristic algorithm that deals with a binary programming

model. From the geometric point of view, the octane algorithm finds the integer solution closest to the current relaxation solution under a custom distance. This thesis proposes a new search direction that relies on the conflict information provided by the conflict graph, and compared with the search direction based on the coefficients of the objective function in the aircraft stands assignment problem, numerical experiments show that the search direction based on the conflict graph is better in aircraft stands assignment model.

Keywords: binary variables, conflict graph, probing, aircraft stands assignment, cliques aggregation, octane

目 录

图形列表

表格列表

第 1 章　引言

混合整数线性规划是近六十年来发展起来的规划论一个分支, 通过实际问题建立部分或者全部变量为整数的线性规划模型, 求解模型得到最优或者近似最优解来帮助决策者更好的做出决策. 本章第一节中先给出一般化的混合整数线性规划模型, 再具体介绍一类特殊的混合整数规划问题, 同时也是本文主要研究的问题之一——指派问题. 求解混合整数线性规划问题离不开专门的求解器, 本章第二节简单介绍现在世界上主流混合整数规划问题求解器的主要组成算法. 本章第三节介绍现在主流求解器中的一种特殊的数据结构, 也是本文研究的重点——冲突图. 本章第四节简单叙述本论文的主要工作.

1.1　混合整数规划模型以及应用

1.1.1　一般混合整数规划模型

混合整数规划模型可以写成如下形式:

定义 1.1.　给定一个矩阵 $A \in \mathbb{R}^{m\times n}$, 向量 $b \in \mathbb{R}^m, c \in \mathbb{R}^n$, 子集 $I \in N = \{1, ..., n\}$, 混合整数规划问题 MIP= (A, b, c, I) 求如下问题的最优解:

$$\text{(MIP)} \qquad c^* = \min\{cx : Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}, \forall j \in I\} \tag{1.1}$$

任意在集合 $X_{MIP} = \{x \in \mathbb{R}^n | Ax \leq b, x_j \in \mathbb{Z}, \forall j \in I\}$ 中的向量称为混合整数规划问题 (MIP) 的可行解. 若在可行解集合中的某一可行解 x^* 使得目标函数值满足 $c^* = c^T x^*$, x^* 称为混合整数规划问题的最优解.

通常情况下, 对于变量 $x \in \mathbb{R}^n$, 模型对变量的范围会做出限定, $l_j \leq x_j \leq u_j$, 其中 $l_j, u_j \in \mathbb{R} \cup \{\pm\infty\}, \forall j \in N$.

1.1.2　指派模型

上一节中介绍了混合整数规划模型的一般形式, 而本文主要研究的是一类特殊的问题——指派问题 [1]. 在生活中, 经常会遇到这样的一种情况, 例如某个单位需要完成 n 项任务, 恰好有 n 个人能承担这些任务, 但是由于每个人擅长的方向不同, 每个人完成任务的成本也是不相同的 (例如时间, 花费等). 于是会产生这样

一种情况, 怎么把不同的 n 个任务分配给 n 个员工使得公司收益最大 (或成本花费最小).

对于这一类形式简单的指派问题, 有 n 个工人分别做 n 项任务, 任意一项任务都只能分配给一个工人, 且每一项任务对于任意一个工人都有一定的花费, 应该如何给 n 个工人分配任务, 模型如下:

$$\begin{aligned} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ s.t. \quad & \sum_{j=1}^{n} x_{ij} = 1, \forall i \in \{1,...,n\} \\ & \sum_{i=1}^{n} x_{ij} = 1, \forall j \in \{1,...,n\} \\ & x_{ij} \in \{0,1\}, \forall i,j \in \{1,...,n\} \end{aligned}$$

文章第四节讨论的机位分配问题就与指派模型十分类似.

混合整数规划的这类优化问题现已经成为建模、解决实际规划的问题中的一种非常强大的工具, 并被广泛应用在各个领域, 包括行程安排、项目规划、公共交通、航空运输、通信、经济金融、时刻表、矿业和林业等等, 在社会生产生活中有着巨大作用. 因为本文讨论的模型仅包括机位分配问题, 故其他模型不再一一介绍.

1.2 混合整数规划求解器发展历史和主要算法

1.2.1 混合整数规划求解器发展历史

最早的混合整数规划研究可追溯至上个世纪五十年代.1954 年, 线性规划之父 George Dantzig[1]教授、网络流优化专家 Fulkerson[2]、Johnson 开始对旅行商问题[3]进行研究, 建立了整数规划模型, 并通过手动计算, 求解了 49 个城市的旅行商问题, 引起了巨大的轰动. 从此之后, 整数规划得到了蓬勃发展. 早期的整数规划求解器开发始于二十世纪七十年代, 有 FMPS、UMPIRE、MPSX、MPS III、APEX、

[1]乔治 · 伯纳德 · 丹齐格, 著名数学家, 1947 年提出创造了简捷法, 又称单纯形法 (Simplex Method), 被称为线性规划之父.

[2]Delbert Fulkerson, Ford-Fulkerson 算法提出者, 该算法是求解网络流问题的经典算法.

[3]旅行商问题 (Travelling Salesman Problem): 给定一系列城市和每两个城市之间的距离, 求解访问每一座城市一次并回到起始城市的最短回路. 它是组合优化中的一个 NP-Hard 问题, 在运筹学和理论计算机科学中非常重要.

MPSX/370 等.

然而, 由于混合整数规划问题是一个 NP-Hard 问题, 实际应用中常常需要使用各式各样的技巧去求解不同的整数规划问题. 因此, 成熟的整数规划求解器需要拥有处理多种整数规划问题的能力, 这就导致整数规划求解器的开发极其困难. 目前, 仅有少数几个发达国家拥有自己的整数规划求解器, 如美国有 GUROBI[4]、CPLEX[5]、SAS[6]、MATLAB、CBC[7]、SYMPHONY, 德国有 SCIP[8], 俄罗斯有 MIPCL 和 GLPK, 英国有 XPRESS(后被美国 FICO 公司收购), 芬兰有 LP-SOLVE. 国内最早的整数规划求解器是中国科学院数学与系统科学院于 2018 年 3 月发布的 CMIP.

1.2.2　分支定界算法

分支定界算法 [2] 是解决优化问题的一种非常通用且广泛使用的方法. 其想法是将原问题依次划为分解空间较小的子问题, 直到各个子问题求得最优解, 所有子问题中最好的解必然是全局最优.

算法流程图如下:

算法 1 分支定界算法

输入: 最小化原问题 R

输出: 目标函数最优值 c^*, 最优解 x^*, 如果目标函数无解或者为无穷也输出

1: 初始化 $\mathcal{L} = R, \hat{c} = \infty$;
2: **if** $\mathcal{L} = \varnothing$ **then**
3:　　停止, 返回 $x^* = \hat{x}, c^* = \hat{c}$;
4: **end if**
5: 选择子问题 $Q \in \mathcal{L}, \mathcal{L} = \mathcal{L} \backslash \{Q\}$;
6: 求解子问题 Q 的松弛解 Q_{relax};
7: **if** Q_{relax} 无解 **then**
8:　　$\check{c} = \infty$;
9: **else**

[4]Gurobi 是由美国 Gurobi 公司开发的新一代大规模数学规划优化器, 现在主流商业求解器之一.

[5]IBM 研发的一款运筹优化工具包, 与 Gurobi 同为功能强大的商业求解器.

[6]全称 STATISTICAL ANALYSIS SYSTEM, 是全球最大的私营软件公司之一, 是由美国北卡罗来纳州立大学 1966 年开发的统计分析软件.

[7]Coin-or Branch and Cut 是用 C ++ 编写的开源混合整数编程求解器, 带有命令行工具的 C ++ 库, 是 COIN-OR(运营研究社区的开源) 项目的一部分.

[8]Solving Constraint Integer Programs, 起源于 ZIB(Zuse Institute Berlin), 由博士生 Tobias Achterberg 奠定整个框架, 目前世界上最好的开源求解器之一.

10: $\check{x}$ 为子问题 Q 的最优解,$\check{c}$ 为子问题 Q 的最优值;
11: **end if**
12: **if** $\check{c} \geq \hat{c}$ **then**
13: 回到第 2 步;
14: **end if**
15: **if** $\check{x}$ 是 R 的可行解 **then**
16: $\hat{x} = \check{x}, \hat{c} = \check{c}$, 回到第 2 步;
17: **end if**
18: 把子问题 Q 分解成 $Q = Q_1 \cup ... \cup Q_k$, $\mathcal{L} = \mathcal{L} \cup \{Q_1 \cup ... \cup Q_k\}$, 回到第 2 步;

步骤 12 中的边界的目的是避免对 R 的所有可行解进行枚举, 通常 R 的可行解是指数级别的. 因此为了加速分支定界算法, 给原问题寻找一个优秀的上界下界是必须的. 下界通常可以通过求解子问题的松弛解获得, 而上界, 从算法步骤 15 发现, 可以通过原问题的一个混合整数可行解获得, 在主流求解器中会用启发式的方法尝试去获得一个可行解.

步骤 5 中的节点选择和步骤 18 中的分支方案确定了分支定界算法的重要决策, 针对实际生活中给定的不同问题, 不同的选择策略会影响算法的求解过程. 主要体现在两个方面: 一是否能在分支定界早期就发现一个较好的可行解, 由可行解决定的原问题上限会影响分支定界算法的剪支过程, 从而影响算法效率; 二是 $\mathcal{L}$ 中子问题下限的增长速度.

对于一个有效的分支定界算法来说, 步骤 6 中解决的松弛类型十分重要. 合理的松弛必须满足两个相互排斥的要求: 易于求解, 可产生强对偶边界. 一般在求解器中采用单纯性算法 [3] 求解当前子问题的线性松弛解.

1.2.3 预处理算法

预处理算法 [4] 是求解器中必不可少的算法组模块, 可以修改混合整数线性规划模型的结构使模型更接近可行解的凸包. 较强的模型通常具有更严格的对偶边界, 这使分支定界过程更有效. 因此, 预处理算法可以加速求解过程, 并能够及早发现问题不可行.

由于在建模过程中可能存在一些冗余的数据, 这些数据会使得求解器求解过程变慢, 最直观的体现, 因为系数矩阵 A 太大, 使得分支定界求解线性规划松弛解的过程很慢. 预处理算法有三个目标:

1. 通过删除不相关的信息（例如冗余约束或已经固定的变量）来减小模型的大小.

2. 通过利用一些整数变量的信息来加强模型的线性松弛度, 例如收紧变量的界限或改善约束中的系数.

3. 提取来自模型的信息（例如隐含图和冲突图）保存下来为以后的分支定界或者割平面做准备.

1975 年 Brearley[5] 等学者通过固定变量, 删除多余的行, 用简单的边界替换等预处理方法来减小问题的规模. 1994 年, Savelsbergh[6] 提出的预处理框架细分了预处理算法, 把算法依照时间复杂度分为简单算法和复杂算法. 近来, Gamrath[7] 等学者于 2015 年在 SCIP 中发展了三种预处理算法, 以及 Achterberg[8] 等学者于 2016 年在 GUROBI 中补充了一些新的预处理方法.

1.2.4　割平面算法

割平面算法 [9] 的主要思想是, 在松弛后的解空间中去掉不是可行解的那一部分, 使得最后的解空间尽可能的接近可行集的凸包.

割平面这一领域的基本工作是 Gomory[9][10–12] 教授完成的, 他证明了带有有理数据的整数规划问题可以通过只使用割平面方法在不进行分支定界算法的情况下以有限的步数求解, 并提出一个算法寻找整数规划问题中的割平面, 但是这样的方法在实际计算中表现的并不是很好.

随着 1999 年 CPLEX 6.5 的发布, 通过结合分支定界的方法, 割平面算法取得显著的效果. 据 CPLEX 报告说, 从 CPLEX 6.0 到 CPLEX 6.5, 求解速度提高了 22.3 倍, 其中割平面为这一成功做出了重要贡献.

1.2.5　启发式算法

求解分支定界树的过程通常是一个很复杂的过程, 需要耗费大量的时间去求得一个最优解, 所以需要一些启发式的方法寻找一个可行解, 给原问题提供一个上界, 从而加速问题求解. 在求解器中加入启发式算法 [13] 有如下三点好处:

1. 通常来说用户不需要求得一个最优解, 一个高质量的可行解也能满足用

[9](1929-), 割平面算法的发明者, 开创了使用单纯性法求解整数规划的先河. 同时也为列生成算法做出重大贡献, 提出了著名的 Gilmore-Gomory Formulation.

户的需求, 使用一些启发式算法的方法可以快速得到一个可行解.

2. 如果通过启发式算法得到了一个可行解, 可以帮助之后的分支定界过程剪支.

3. 通过启发式方法检查模型在求解过程中是否发生了错误.

1.3 冲突图

冲突图 [14, 15] 是存在于求解器中的一种特殊的结构, 表示二元变量之间的一种逻辑关系. 更准确的来说, 冲突图以二元变量 x 和它的补变量 $1-x$ 作为冲突图的顶点, 冲突图任意两个顶点的边表示这两个顶点至多有一个可以取值为 1. 因为其结构的特殊性, 充分利用冲突图结构的信息可以加快求解器求解一些特殊问题的速度, 所以这样的特殊结构也逐渐发展成求解器中不可缺少的一部分.

同样在求解一般问题的混合整数线性规划中, 如果原问题中包含二元变量, 则可以利用二元变量潜在的一些逻辑上的冲突信息构造冲突图, 利用冲突图在之后的分支定界搜索树中剪支, 从而缩短问题的求解时间.

1.3.1 冲突图背景与研究意义

分支定界算法将原问题划分成若干个子问题, 然后递归地迭代求解每个子问题, 直到可以求得每个子问题的最优解或者得出原问题不可行的结论而终止. 早期的混合整数规划求解器如 CPLEX, LINGO, SIP[16] 或 XPRESS 仅仅只是在分支定界过程中丢弃了不可行或者目标函数值超出界的子问题, 并未继续关注这些子问题内部的一些特征.

相对于标准的混合整数规划问题从可行性的方向考虑, 可满足性问题求解器试从不可行的子问题中学习, 这是 Marques-Silva 和 Sakallah[17] 提出的一个想法. 分析不可行问题中的二元变量之间的关系, 生成冲突信息, 而这些隐含的冲突信息可以帮助修剪分支定界搜索树. 而且这些信息还使求解器能够应用所谓的非时间顺序回溯. 顺带一提, 可满足性问题也可以通过分支定界分解方案来解决, 如文献 [18, 19] 中所提, 最早的算法是 Davis[10], Putnam[11], Logemann 和 Loveland 提出

[10]计算机科学发展史上的先驱人物, 其《可计算性与不可解性》一书被誉为计算机科学领域极少数真正的经典著作之一.

[11]美国当代著名哲学家, 逻辑学家、科学哲学家. 在洛杉矶加利福尼亚大学获哲学博士学位, 曾任教于普林斯顿大学和马萨诸塞理工学院, 后任哈佛大学哲学教授和 W.B. 皮尔逊讲座现代数学与数理逻辑教授.

的 DPLL[12]算法.

冲突分析的思想是找出当前子问题不可行的一些信息, 并在以后的搜索中利用这些信息. 例如在可满足性问题中, 假设固定了某一些变量, 通过这些变量固定后作出一系列推导发现原问题不可行了, 那么这些变量构成的集合中至少有一个变量的取值必须取之前固定的值的补值, 或者说逻辑上相反的值. 把类似这样的信息加入其他子问题中, 通过域传播的方法, 修剪分支定界搜索树.

Padberg[20] 提出在混合整数线性规划中使用图的结构来保存二元变量的冲突信息, 他主要基于图的结构来研究集合配置多面体 (set packing polyhedron) 的一些特征, 把集合配置多面体的面分成两组, 一组是团, 另一组是无弦奇圈 (the odd cycles without chords), 其中团的结构就是早期的冲突图.

随后, Johnson 和 Nemhauser[21] 于 1992 年提出一种算法在背包约束下探测冲突关系. 他们还指出, 通过冲突关系可以找到一些团不等式 (clique inequalities) 来改善混合整数规划模型的线性松弛解.

Hoffman 和 Padberg[22] 于 1993 年使用图的结构来解决航空公司机组人员调度问题, 其中图的构造与 Padberg 于 1973 年提出的类似. 他们提出了一种分支割平面 (branch-and-cut) 的方法使用图来在预处理中生成团和奇圈不等式, 且基于集合划分约束来扩充团. 他们的计算结果表明, 在分支中通过这种方法寻找到的割平面甚至可以解决航空公司机组人员调度问题的大规模实例.

为了进一步优化混合整数线性规划的模型, Savelsbergh[6] 于 1994 年提出了加入探测技术的预处理框架. 他的工作区分了简单预处理方法和困难预处理方法, 并为实现探测算法构造隐含图. 通过一些隐含的逻辑关系利用探测算法消除变量, 生成图和一些割不等式. 实验结果证明了这些工作在减少原问题上下界的差上的有效性.

Bixby 和 Lee[23] 在 1998 年用分支割平面算法中使用冲突图中的信息来解决卡车调度问题 (truck dispatching scheduling problem). 其中构造冲突图的方法与 Johnson 和 Nemhauser 相同, 通过对背包约束冲突信息的分析构造冲突图. 通过松弛解中不为整数变量构成的子图产生团和奇圈不等式, 通过贪婪的启发式算法产生割平面. 实验结果表明在解决某些困难的问题中, CPU 时间有明显下降.

Borndorfer[24] 于 1998 年通过图和分支割平面的方法去解决集合分离问题.

[12]一种完备的、以回溯为基础的算法, 用于解决在合取范式 (CNF) 中命题逻辑的布尔可满足性问题.

在他的工作中通过三个步骤寻找团，以及使用 Dijkstra 算法在辅助二部图中搜索最短路来寻找奇圈不等式. 实验结果表明他的算法能处理超大规模的集合分离问题.

与冲突图相关的工作还有, Santos[25] 等学者于 2016 年使用冲突图解决护士排班问题 (nurse rostering problem), Araujo[26] 等人在一些资源受限的项目调度问题 (resource constrained project scheduling problem) 中通过一些特殊的问题约束得出四种类型的冲突等. 因此, 冲突图已经发展成解决混合整数线性规划问题中的重要结构.

1.3.2 构造冲突图

1.3.2.1 特殊约束构造冲突图

混合整数规划中有三类特殊约束, 分别为集合划分约束 (set partitioning constraints), 集合配置约束 (set packing constraints), 集合覆盖约束 (set covering constraints). 集合划分约束和集合配置约束是指必须从一组给定的变量集中选择一个或者最多选择一个变量的取值为 1. 这两类约束在几种应用中非常普遍, 例如对图形着色问题建模. 它们形式如下:

$$\sum_{j \in S} x_j = 1 \qquad (set\ partitioning)$$

$$\sum_{j \in S} x_j \leq 1 \qquad (set\ packing)$$

其中 $x_j \in \{0,1\}, \forall j \in S$; S 为模型中所有二元变量构成的集合的子集.

集合覆盖约束是变量选择约束的第三种类型, 即要求从一组二元变量中至少选择一个. 它可以表示为:

$$\sum_{j \in S} x_j \geq 1 \qquad (set\ covering)$$

其中 $x_j \in \{0,1\}, \forall j \in S$; S 为模型中所有二元变量构成的集合的子集.

对于集合划分约束和集合配置约束中的所有变量, 都不可以同时取值为 1 变量取值为 1, 因此这两种约束中的所有变量都应该保存在冲突图中.

1.3.2.2 团结构与冲突图

从集合划分约束和集合配置约束的形式看, 这两类约束至多有一个变量取值为 1, 具有这种特性的变量构成的集合称作团. 例如, 上一节中的集合 S 称为一个

团, 即满足至多有一个变量可以取值为 1 的集合称为一个团. 更形象的看, 团可以用图来表示, 假设有三个约束如下:

$$x_1 + x_2 + x_3 \leq 1 \tag{1}$$

$$x_3 + x_4 \leq 1 \tag{2}$$

$$x_4 + x_5 + x_6 \leq 1 \tag{3}$$

用点来表示变量, 用边来表示两个变量无法同时取 1 的值, 则上述三个不等式构成的图如下:

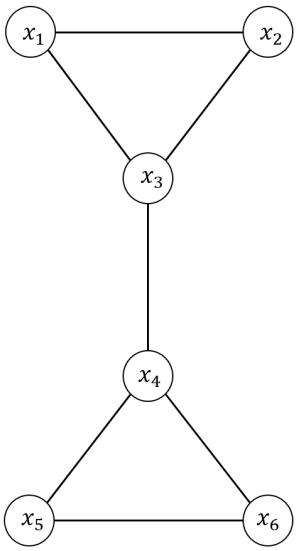


图 1.1 团和完全图

从图上来看, 变量集合 $\{x_1, x_2, x_3\}$, $\{x_4, x_5, x_6\}$, $\{x_3, x_4\}$ 分别构成三个完全图, 即团结构用图来表示, 一个团就是一个完全图.

1.3.2.3　0, 1 背包约束构造冲突图

在集合配置约束中至多有一个变量取值为 1, 即若同时有两个变量取值为 1, 就会导致原问题不可行, 这一类关系称为第一种冲突. 而集合划分约束一定会有一个变量的取值为 1, 所有变量取 0 的值或者两个以上的变量取 1 的值也会导致原问题不可行, 这一类关系称为第二种冲突. 上述小节可以知道, 冲突关系可以用图的形式来表示, 这两类冲突关系构成的图称为冲突图.

在集合配置约束和集合划分约束中, 很容易分析出冲突关系, 但是对于更加

一般的约束如何找到其中二元变量的冲突关系呢? 先来看另外一种比较特殊的约束:

$$\sum_{j\in S} a_j x_j \leq b$$

其中 $b \geq 0$, $a_j \geq 0$, S 为模型中所有二元变量构成的集合的子集.

从形式上看, 这种约束是标准的 0, 1 背包约束, 这一类约束在整数规划问题中十分常见. 通过 0, 1 背包约束构造冲突图的算法是 Johnson 和 Nemhauser[21] 于 1992 年提出, 下面简单叙述算法如何实现.

第一步是按照变量系数非降序排序, 然后考虑约束两个系数最大的变量 (这个过程对应算法流程图的第 2 行和第 3 行) 与约束右端项的比较, 检查约束中是否有团. 如果比较后并没有违反约束的右端项, 则这条约束中就不会发生冲突, 算法完成. 此步骤对于检查不包含团的约束至关重要, 可以节省构建冲突图的时间.

第二步, 如果约束中存在冲突, 搜索最小的 k 使得 $a_{j_k} + a_{j_{k+1}} > b$ 成立. 则对于 $j_k, j_{k+1}, ..., j_n$ 下标对应的变量都会发生冲突, 可以加入冲突图中, 这个过程对应算法的 5, 6 行.

第三步, 需要继续寻找约束中未被发现的冲突关系, 算法流程图 7 行后对应这个过程, 当满足 $a_{j_i} + a_{j_f} > b$, 这个条件后, 变量 x_{j_i} 与 x_{j_f} 到 x_{j_n} 都不能同时取 1 的值.

下面给出 0, 1 背包约束的团探测算法:

算法 2 0, 1 背包约束团探测算法

输入: $\sum_{j\in S} a_j x_j \leq b$

输出: 更新冲突图 CG(conflict graph)

1: 把变量下标按变量前面系数非降序排列, 即 $S = j_1, j_2, ..., j_n$, $a_{j_1} \leq a_{j_2} \leq ... \leq a_{j_n}$;
2: **if** $a_{j_{n-1}} + a_{j_n} \leq b$ **then**
3: return;
4: **end if**
5: 找到最小的 k 满足 $a_{j_k} + a_{j_{k+1}} > b$;
6: $\{x_{j_k}, ..., x_{j_n}\}$ 加入冲突图;
7: **for** $i = k-1, ..., 1$ **do**
8: 找到最小的 f 满足 $a_{j_i} + a_{j_f} > b$;
9: **if** f 存在 **then**
10: $\{x_{j_i}, x_{j_f}, ..., x_{j_n}\}$ 加入冲突图;

11: **else**

12: break;

13: **end if**

14: **end for**

时间复杂度分析: 从程序框图上看, 算法第 5 步以及第 7 步的循环中遍历一遍所有变量, 因此算法的时间复杂度为 O(n).

1.3.2.4　线性约束约束构造冲突图

给定线性约束如下:

$$\underline{\beta} \leq \sum_{j \in S} a_j x_j \leq \bar{\beta} \tag{1}$$

其中 S 为任意变量构成的集合, 因为只有二元变量才存在冲突关系, 不失一般性假设 S 存在一个二元变量子集 B_S, 且 $a_j > 0$.

在线性约束中构造冲突图的方法启发于 Johnson 和 Nemhauser 在背包约束中构造冲突图的方法. 通过线性约束的形式做一些简单的推导能得出如下关系:

$$\begin{aligned} \underline{\beta} - \max\{\sum_{j \in S \setminus B_S} a_j x_j | l_j \leq x_j \leq u_j\} \leq \sum_{j \in B_S} a_j x_j \\ \leq \bar{\beta} - \min\{\sum_{j \in S \setminus B_S} a_j x_j | l_j \leq x_j \leq u_j\} \end{aligned} \tag{2}$$

先考虑 (2) 右边

$$\sum_{j \in B_S} a_j x_j \leq \bar{\beta} - \min\{\sum_{j \in S \setminus B_S} a_j x_j | l_j \leq x_j \leq u_j\} \tag{3}$$

这就是一个 0, 1 背包约束的形式, 通过 0, 1 背包约束探测团的算法就可以解决. 再考虑 (2) 左边

$$\underline{\beta} - \max\{\sum_{j \in S \setminus B_S} a_j x_j | l_j \leq x_j \leq u_j\} \leq \sum_{j \in B_S} a_j x_j \tag{4}$$

如果对 $\forall x_j \in B_S$, 用 $y_j = 1 - x_j$ 来代替 x_j 代入 (4), 有:

$$\sum_{j \in B_S} a_j y_j \leq \max\{\sum_{j \in S \setminus B_S} a_j x_j | l_j \leq x_j \leq u_j\} + \sum_{j \in B_S} a_j - \underline{\beta} \tag{5}$$

通过这样的方法把不等式左端也换成了一个 0, 1 背包约束的形式, 同样只需要通过 0, 1 背包约束探测团的算法就能解决.

下面给出线性约束的团探测算法, 仅需要稍微修改之前的关于 0, 1 背包约束团探测的算法就可以得到:

算法 3 线性约束团探测算法

输入: $\underline{\beta} \le \sum_{j\in S} a_j x_j \le \bar{\beta}$

输出: 更新冲突图 CG(conflict graph)

1: 计算 $b = \bar{\beta} - \min\{\sum_{j\in S\setminus B_S} a_j x_j | l_j \le x_j \le u_j\}$ 构造新的不等式 $\sum_{j\in B_S} a_j x_j \le b = \bar{\beta} - \min\{\sum_{j\in S\setminus B_S} a_j x_j | l_j \le x_j \le u_j\}$;

2: 把变量下标按变量前面系数非降序排列, 即 $B_S = j_1, j_2, ..., j_n, a_{j_1} \le a_{j_1} \le ... \le a_{j_n}$;

3: **if** $a_{j_{n-1}} + a_{j_n} \le b$ **then**

4: return;

5: **end if**

6: 找到最小的 k 满足 $a_{j_k} + a_{j_{k+1}} > b$;

7: $\{x_{j_k}, ..., x_{j_n}\}$ 加入冲突图;

8: **for** $i = k-1, ..., 1$ **do**

9: 找到最小的 f 满足 $a_{j_i} + a_{j_f} > b$;

10: **if** f 存在 **then**

11: $\{x_{j_i}, x_{j_f}, ..., x_{j_n}\}$ 加入冲突图;

12: **else**

13: break;

14: **end if**

15: **end for**

16: 计算 $b = \max\{\sum_{j\in S\setminus B_S} a_j x_j | l_j \le x_j \le u_j\} + \sum_{j\in B_S} a_j - \underline{\beta}$, 用 $y_j = 1 - x_j$ 取代 x_j, 构造新的不等式 $\sum_{j\in B_S} a_j y_j \le b = \max\{\sum_{j\in S\setminus B_S} a_j x_j | l_j \le x_j \le u_j\} + \sum_{j\in B_S} a_j - \underline{\beta}$;

17: 把变量下标按变量前面系数非降序排列, 即 $B_S = j_1, j_2, ..., j_n, a_{j_1} \le a_{j_1} \le ... \le a_{j_n}$;

18: **if** $a_{j_{n-1}} + a_{j_n} \le b$ **then**

19: return;

20: **end if**

21: 找到最小的 k 满足 $a_{j_k} + a_{j_{k+1}} > b$;

22: $\{y_{j_k}, ..., y_{j_n}\}$ 加入冲突图;

23: **for** $i = k-1, ..., 1$ **do**

24: 找到最小的 f 满足 $a_{j_i} + a_{j_f} > b$;

25:　　**if** f 存在 **then**

26:　　　　$\{y_{j_i}, y_{j_f}, ..., y_{j_n}\}$ 加入冲突图;

27:　　**else**

28:　　　　break;

29:　　**end if**

30: **end for**

时间复杂度分析: 从程序框图上看, 算法第 1 步以及第 16 步的时间复杂度为 O(n), 又调用了两次 0, 1 背包约束探测团的算法, 故时间复杂度依旧为 O(n).

1.4　本论文主要工作

首先, 对于求解器中的预处理算法——探测算法 [4](probing), 本文把算法拆分成四个主要的过程：域传播算法, 隐含图算法, 冲突图算法, 探测算法. 在论文第二章分别介绍各个算法目的, 算法的流程框图, 以及算法的效果. 通过对标准算例库选择出的部分算例数值实验证明了算法在求解器中的有效性.

在探测算法的数值实验中发现域传播算法在某个算例中出现数值不稳定情况, 因此依靠 gmp 库构造一种以大整数为基础的数据结构解决发现的数值不稳定的问题, 通过数值实验来比较 SCIP 中保持数值稳定性的方法和大整数的数据结构保持数值稳定性的方法的优劣.

探测算法主要处理包含二元变量的特殊问题, 这类问题在实际生活中有很多应用, 如互斥计划问题, 指派问题, 背包问题等. 在论文第三章将讨论基于冲突图的一个预处理方法——团聚合算法 [4](clique aggregation), 对于求解此类问题有十分优异的表现. 然而 SCIP 实现的团聚合算法无法求解基于实际工业生产中给出的一个机位分配问题, 因此本文采用邻接矩阵保存数据, 设计新的算法流程, 重新实现团聚合算法, 使问题成功求解.

对于解决只包括二元变量的问题还有一种比较特殊的启发式方法——辛烷算法 [13, 27](octane). 论文第四章会从算法的几何背景开始介绍, 梳理算法实现的关键步骤, 进一步提出一种基于冲突图的新搜索方向, 并用于机位分配模型上, 给出了相关的数值实验.

第 2 章　冲突图在探测算法中的应用

本章节所有算法中输入的混合整数规划模型有 n 个变量, m 条约束, 并用 n, m 两个参数来表示算法时间复杂度. 算法中涉及到一些数值稳定性会在下一章叙述. 本文把探测算法分四个部分, 冲突图部分已经在上一章介绍, 下面分别介绍域传播, 隐含图和探测算法.

2.1　域传播

域传播 [4] 通过分支定界树中子问题上的所有约束以及约束中变量的当前变量界来收紧模型中每个变量的变量界. 在混合整数线性规划中, 域传播算法通常归类为一种节点预处理 (node preprocessing) 方法.

2.1.1　积极界

给出如下所示的线性约束:

$$\underline{\beta} \leq a^T x \leq \bar{\beta}$$

其中把 $\underline{\beta}$ 记作 lhs(letf hand side), $\bar{\beta}$ 记作 rhs(right hand side), 且 $\underline{\beta}, \bar{\beta} \in \mathbb{R} \cup \{\pm\infty\}$, $a \in \mathbb{R}^n$ 称为此约束的系数. 下面给出域传播算法中十分重要的定义:

定义 2.1.　给定一条线性约束 $\underline{\beta} \leq a^T x \leq \bar{\beta}$, 记

$$\underline{\alpha} := \min\{a^T x | \tilde{l} \leq x \leq \tilde{u}\}, \bar{\alpha} := \max\{a^T x | \tilde{l} \leq x \leq \tilde{u}\}$$

其中 $\underline{\alpha}$ 叫做这条约束的积极下界, $\bar{\alpha}$ 叫做这条约束的积极上界.

定义 2.2.　给定一条线性约束 $\underline{\beta} \leq a^T x \leq \bar{\beta}$, 记

$$\underline{\alpha}_j := \min\{a^T x - a_j x_j | \tilde{l} \leq x \leq \tilde{u}\}, \bar{\alpha}_j := \max\{a^T x - a_j x_j | \tilde{l} \leq x \leq \tilde{u}\}$$

称为除了 x_j 之外 $a^T x$ 中包含的所有变量的积极界余量.

通过上述给出的积极界和积极界余量的定义能发现如下五条性质:

1. 如果 $\underline{\beta} \leq \underline{\alpha}$, 那么这条约束的 lhs 可以用 $-\infty$ 来代替, 不会改变所需求解的问题的可行性.

2. 如果 $\bar{\beta} \geq \bar{\alpha}$, 同样这条约束的 rhs 可以用 ∞ 来代替.

3. 如果 $\underline{\beta} \leq \underline{\alpha}$ 且 $\bar{\beta} \geq \bar{\alpha}$, 这条约束是一条冗余约束, 在模型中删除这条约束不会改变所需求解的问题的可行性.

4. 如果 $\underline{\beta} > \bar{\alpha}$ 或者 $\bar{\beta} < \underline{\alpha}$, 则当前问题无可行解.

5. 对 $\forall j = 1, ..., n$, 有:

$$\frac{\underline{\beta} - \bar{\alpha}_j}{a_j} \leq x_j \leq \frac{\bar{\beta} - \underline{\alpha}_j}{a_j}, a_j > 0$$

$$\frac{\bar{\beta} - \underline{\alpha}_j}{a_j} \leq x_j \leq \frac{\underline{\beta} - \bar{\alpha}_j}{a_j}, a_j < 0$$

通过以上五点可以发现, 如果一条约束的积极界发生改变, 那么这条约束中的变量的变量界可能发生变化, 甚至这条约束的 lhs, rhs 可能会变成无穷.

2.1.2 算法流程图

通过对积极界的分析, 知如果一个变量的变量界发生了改变, 那么包含这个变量的所有约束积极界会发生变化, 这些约束的积极界发生变化又会导致这些约束中其他变量的变量界发生变化, 这样一个连锁反应的过程称为域传播.

下面是域传播算法的流程图:

算法 4 域传播算法

输入: 线性约束 $\underline{\beta} \leq a^T x \leq \bar{\beta}$, 这条约束中变量的变量界 $\tilde{l} \leq x \leq \tilde{u}$, 约束的积极界 $\underline{\alpha}, \bar{\alpha}$

输出: 加紧后的 x 的变量界

1: 如果此约束已经进行过域传播了, 则跳出;
2: 标记这条约束已经进行域传播;
3: 如果 $\underline{\beta} > \bar{\alpha}$ 或者 $\underline{\alpha} > \bar{\beta}$, 则当前问题无解;
4: 如果 $\underline{\beta} \leq \underline{\alpha}$ 且 $\bar{\alpha} \leq \bar{\beta}$, 则此条约束删除;
5: **for** $j = 1, .., n$ **do**
6: **if** $a_j > 0$ **then**
7: 计算积极界余量 $\underline{\alpha}_j = \underline{\alpha} - a_j \tilde{l}_j, \bar{\alpha}_j = \bar{\alpha} - a_j \tilde{u}_j$;
8: **if** $\underline{\alpha}_j > -\infty$ 且 $\bar{\beta} < +\infty$ **then**
9: $\tilde{u}_j^{new} = \frac{\bar{\beta} - \underline{\alpha}_j}{a_j}$;
10: **if** x_j 是整数变量 **then**
11: $\tilde{u}_j^{new} = \lfloor \tilde{u}_j^{new} \rfloor$;

12: **end if**
13: **if** $\tilde{u}_j^{new} < \tilde{u}_j$ **then**
14: $\tilde{u}_j = \tilde{u}_j^{new}$;
15: **end if**
16: **end if**
17: **if** $\bar{\alpha}_j < +\infty$ 且 $\underline{\beta} > -\infty$ **then**
18: $\tilde{l}_j^{new} = \frac{\underline{\beta} - \bar{\alpha}_j}{a_j}$;
19: **if** x_j 是整数变量 **then**
20: $\tilde{l}_j^{new} = \lceil \tilde{l}_j^{new} \rceil$;
21: **end if**
22: **if** $\tilde{l}_j^{new} > \tilde{l}_j$ **then**
23: $\tilde{l}_j = \tilde{l}_j^{new}$;
24: **end if**
25: **end if**
26: **end if**
27: **if** $a_j < 0$ **then**
28: 计算积极界余量 $\underline{\alpha}_j = \underline{\alpha} - a_j \tilde{u}_j, \bar{\alpha}_j = \bar{\alpha} - a_j \tilde{l}_j$;
29: **if** $\underline{\alpha}_j > -\infty$ 且 $\bar{\beta} < +\infty$ **then**
30: $\tilde{l}_j^{new} = \frac{\bar{\beta} - \underline{\alpha}_j}{a_j}$;
31: **if** x_j 是整数变量 **then**
32: $\tilde{l}_j^{new} = \lceil \tilde{l}_j^{new} \rceil$;
33: **end if**
34: **if** $\tilde{l}_j^{new} > \tilde{l}_j$ **then**
35: $\tilde{l}_j = \tilde{l}_j^{new}$;
36: **end if**
37: **end if**
38: **if** $\bar{\alpha}_j < +\infty$ 且 $\underline{\beta} > -\infty$ **then**
39: $\tilde{u}_j^{new} = \frac{\underline{\beta} - \bar{\alpha}_j}{a_j}$;
40: **if** x_j 是整数变量 **then**
41: $\tilde{u}_j^{new} = \lfloor \tilde{u}_j^{new} \rfloor$;
42: **end if**
43: **if** $\tilde{u}_j^{new} < \tilde{u}_j$ **then**
44: $\tilde{u}_j = \tilde{u}_j^{new}$;
45: **end if**
46: **end if**

47: **end if**

48: **end for**

域传播算法的时间复杂度: 对于原模型中的任意约束, 在执行域传播算法的时候, 只需要把约束中的所有变量遍历一遍, 即对于一条约束而言, 算法的时间复杂度为 O(n).

2.1.3 终止条件

考虑域传播中的一种极端的情况:

$$0.1 \leq x - y \leq 0.9$$

$$x, y \in \{1, ..., 1000\}$$

如果先缩紧变量 y 的界, 通过域传播算法可以得到 $1 \leq y \leq 999$, 再缩紧 x 的界可以得到 $1 \leq x \leq 999$, 以此类推下去, 需要运行 1000 此域传播算法才能得出这样的情况不存在可行解. 相当于做了大量无用计算才得出了上述结论, 因此面对这种情况, 需要设置终止条件来处理.

如果对某一条约束使用域传播算法, 约束中的变量界只会发生 $\frac{1}{1000}$ 或者 $\frac{1}{10000}$ 的变化, 域传播算法对这一条约束的执行效率就会非常低下, 需要及时制止这样的情况发生. 通过设置一个变量界变化的阈值, 变量界的改变超出了这个阈值, 才会被采用.

记 l_k', u_k' 为域传播后得到的变量 x_k 新的界, 如果得到的新的变量界满足如下条件:

$$u_k' < u_k - 0.01 \times \max\{\min\{u_k - l_k, |u_k|\}, 1\}$$

$$l_k' > l_k + 0.01 \times \max\{\min\{u_k - l_k, |l_k|\}, 1\}$$

即只有变量界变化的值大于某个阈值后, 该算法才会改变 x_k 的界.

对于小节开头给出的例子, 只需要做出 100 次域传播运算就会发现改变变量的界没有意义, 域传播终止. 那么是否能更快发现并终止域传播算法, 可以通过修改之前不等式条件前的系数 (论文中为 0.01) 来实现.

2.1.4 数值结果

测试算例为公开算例库 miplib2017, miplib3 等收集来的 27 个简单算例. 测试方法为自实现域传播算法, 把域传播算法计算后的模型生成一个预处理后的 mps

文件, 用 CPLEX 运行预处理后的 mps 文件与原 mps 比较最终值是否正确.

表 2.1 域传播算法整体效果合计 (共 27 个算例)

正确求解	固定变量	改变变量界	删除变量	改变左右端	删除约束
27	402	8742	402	572	485

表 2.2 域传播算法各个算例具体效果

	规模 (约束, 变量)	固定变量	改变变量界	改变左右端	删除约束
air03	(124,10757)	0	13	0	0
atm_5_10_1	(270,260)	0	68	0	0
bell3a	(124,10757)	16	104	31	30
blend2	(123,133)	2	119	0	0
dcmulti	(290,548)	0	533	18	18
dsbmip	(1182,1886)	0	2315	0	0
egout	(98,141)	31	115	102	46
enigma	(21,100)	0	0	0	0
fiber	(363,1298)	14	58	18	28
flugpl	(18,18)	0	12	1	1
gen	(780,870)	66	509	272	148
gesa3	(1368,1152)	0	434	0	0
gesa3_o	(1224,1152)	0	432	0	0
gt2	(29,188)	0	12	0	0
khb05250	(101,1350)	0	2594	0	0
l152lav	(97,1989)	0	0	0	0
lseu	(28,89)	0	0	0	0
misc03	(96,160)	0	2	0	0
misc06	(820,1808)	249	1398	107	201
mod008	(6,319)	0	0	0	0
mod010	(146,2655)	0	0	0	0
p0033	(16,33)	0	0	0	0

p0201	(133,201)	6	6	0	0
p0282	(241,282)	0	0	0	0
p0548	(176,548)	18	18	23	13
rgn	(24,180)	0	0	0	0
stein27	(118,27)	0	0	0	0

上表中固定的变量即被删除的变量, 因为域传播算法只能通过固定变量来删除.

值得一提的是, 在 misc06.mps 算例中发现了一个有趣的问题, 把上一节提到的控制域传播算法运行次数的参数从 0.01 增大到 0.05, 这个算例的最优值会从 1.2850860737×10^4 变成 1.2851076292×10^4, 而直接运行 CPLEX 计算原文件最优值为 1.2851076292×10^4. 原因是算例中一个变量的变量界因为上一节两个不等式表示的阈值变得更大而无法缩小界, 使得在 CPLEX 中出现了数值不稳定的情况. 故在设计算法的时候尽可能的用大一点的阈值使得最优解与 CPLEX 直接计算原 mps 文件结果保持一致.

2.1.5 误差分析与处理

上一节中提到因为域传播算法终止阈值设置导致在 CPLEX 运算中出现了不稳定的情况, 本节基于 mps[1]文件格式利用大整数处理浮点数数据的方法来消除误差或使得误差在可控范围内, 并与 SCIP 中处理误差的方式对比, 比较最终结果与 SCIP 计算的 27 个数据最优值的差值, 以及两个方法各自的优劣.

给定约束如下:

$$\underline{\beta} \leq a^T x \leq \bar{\beta}$$

$$l \leq x \leq u$$

l 表示变量 x 的下界, u 表示变量 x 的上界, l, u, x 都是向量. 记域传播算法后得到 x_k 的新的变量界为 $l_k^{'} \leq x_k \leq u_k^{'}$.

在 SCIP 中通常会采用在原数上加上或者减去一个很小的数来减小误差:

$$l_k^{''} = 10^{-5}\lfloor 10^5 l_k^{'} + 10^{-6} \rfloor$$

[1]Mathematical Programming System File, 分为 Data Files、GIS Files 和 Game Files 等类别. 在大多数情况下, 是 Data Files.

$$u_k'' = 10^{-5}\lceil 10^5 u_k' - 10^{-6}\rceil$$

l_k'', u_k'' 作为变量 x_k 的新的变量界代替原来变量界 l_k', u_k'.

大整数是基于修改数据存储方式的一种方法, 因此需要先介绍一种国际通用的混合整数线性规划模型数据的保存文件——mps. Mps 文件格式提供了一种定义数学规划问题的方法. 大型机 LP 系统上历史悠久的 mps 格式已成为公认的 LP 问题定义标准. 与 CPLEX LP 格式相反, mps 格式是面向列的格式：按列 (变量) 而不是按行 (约束) 来指定问题. Mps 文件在固定的列位置上定义了许多区域, 如下所示：

区域	1	2	3	4	5	6
列	2-3	5-12	15-22	25-36	40-47	50-61

一般区域 4, 6 来表示数据, 所以标准 mps 文件下用 12 列来表示数据.

从上表看 mps 文件最多用 12 列来保存数据, 那么 mps 文件中保存的小数最多保存到小数点后 10 位. 对于数据类型 long long, 最多精确保存 18 位整数, 如果 mps 文件中两个数据相乘, 最高可以达到小数点后 22 位, 因此 long long 型也无法满足精度需求, 需要用一种更长的整数类型记录数据, gmp[2]大整数库是一个不错的选择. 因为 mps 文件最多 12 列, 因此采用 gmp 库的新数据类型记录到小数点后 12 位, 12 位后的数据舍去, 使小数点后第十位的数据尽量准确. 因为小数在计算机中会以浮点数保存, 故需要把小数乘上 10^{12}, 用整数的方式表示小数点后十二位. 例如, $3.5 = 3.500000000000$, 数据中保存为 3500000000000. 保存的数据的真实值需要小数点向前移动 12 位.

根据之前对两种方法的叙述, 先简单地从算法消耗的空间, 算法消耗的时间, 数据的准确性, 算法可扩展性四个角度来比较两种方法：

- 空间：大整数方法需要保存的数据类型大小至少是 double 型的一倍, 故 SCIP 的方法更加节省空间.
- 时间：从数据结构分析, 对于任意一个需要在计算机中存储的小数, 大整数需要的存储空间比 double 类型更大, 故理论上四则运算的时间更长.

[2]GMP 是 The GNU MP Bignum Library, 是一个开源的数学运算库, 它可以用于任意精度的数学运算, 包括有符号整数、有理数和浮点数. 它本身并没有精度限制, 只取决于机器的硬件情况.

- 数据的准确性：double 型数据的截断误差是难以预测的, 而大整数方法的误差可以预测.
- 算法可扩展性：大整数方法采取的数据类型可以把数据精确到小数点后任意一位计算, 同样 double 数据类型并不具有这样的特性.

本文通过计算之前 27 个算例的最优值并与 SCIP 比较来说明大整数结构的准确性, 测试方法如下:

把上一章节中实现的域传播算法数据结构改成大整数, 测试上节中 27 个算例, 仅执行预处理过程, 把预处理后结果生成 mps 文件, 通过使用 SCIP 计算预处理后的 mps 文件与 SCIP 直接执行原始 mps 比较最后的最优解是否一致. 此次测试的目的主要是比较修改数据结构为大整数后是否会影响求得的全局最优解, 程序运行时间在这次测试中不进行比较.

参数如下:

表 2.3 大整数类型与 double 型比较的参数设计

V^{gmp}	大整数类型最优值
V^{double}	double 类型最优值
ξ	大整数类型与 double 类型最优值最优值差的绝对值
$\bar{\xi}$	27 个算例差的平均值

定义求误差平均值的公式公式如下:

$$\xi_i = |V_i^{gmp} - V_i^{double}|$$

$$\bar{\xi} = (\sum_{i=1}^{27} \xi_i)/27$$

最终 27 个算例差均值计算结果有 $\bar{\xi} < 10^{-8}$.

从求解结果上来看, 大整数的方法并不会影响最优值, 同时也有更高的精度, 但是因为所采取的实验方法的原因, 大整数的结构的效率在这无法比较. 进一步考虑这一个结果, 因为大整数精度可以为任意范围, 计算结果比计算机截断误差更容易控制且更加精确, 在 SCIP 中, 因为考虑域传播如果变量界的改变过小, 则域传播算例不再改变变量界, 终止. 因此对于一个更加高精度的方法来说, 域传播算法最后求得的解的范围必然小于或等于 SCIP 中域传播解的范围, 但 SCIP 域传播方法并未因为精度问题去掉可行解导致数值不稳定, 故求出来的最优解基本相

同. 对于大整数的数据结构来说, 误差的影响相比较计算机浮点型直接取阶段误差, 误差的量能在可控范围内.

2.2　隐含图算法

Savelsbergh[6] 于 1994 年提出了在预处理中加入探测算法, 并为实现探测算法, Savelsbergh 提出一种隐含图的结构. 隐含图从二元变量出发, 尝试把二元变量固定到某一个确定的取值后, 分析变量界.

考虑原模型中包含二元变量的问题, 对于任意二元变量 x_i, 假设把 x_i 固定到它的下界, 即 $x_i = 0$, 因为 x_i 的上界发生了变化 (从 1 变成 0), 故所有包含 x_i 的约束的积极界发生了改变, 因此这些包含 x_i 的约束中的其他变量的变量界也会因为域传播的关系发生改变, 把这种变化记录为隐含图.

给定线性约束:

$$\underline{\beta} \le a^T x \le \bar{\beta}$$

$$l \le x \le u$$

l 表示变量 x 的下界, u 表示变量 x 的上界, l, u, x 都是向量.

假设变量 x_j 在此约束中, 并且为二元变量; 变量 x_k 也在此约束中, 为整数变量或者连续变量. 以下不等式:

$$\underline{\beta} - \max\{\sum_{i=1,i\neq j,k}^{n} a_i * x_i | l_i \le x_i \le u_i\} \le a_j * x_j + a_k * x_k \le$$

$$\bar{\beta} - \min\{\sum_{i=1,i\neq j,k}^{n} a_i * x_i | l_i \le x_i \le u_i\}$$

对 x_j, x_k 恒成立. 因此当二元变量 x_j 的取值固定之后, 变量 x_k 的范围同样能确定下来. 分别固定 $x_j = 0, x_j = 1$, 有如下四种关系:

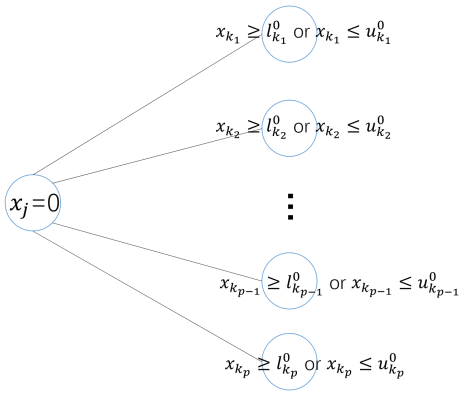
$$x_j = 0 \to x_k \ge l_k^0 \wedge x_j = 1 \to x_k \ge l_k^1 \Rightarrow x_k \ge \min\{l_k^0, l_k^1\} \quad \text{(a)}$$

$$x_j = 0 \to x_k \le u_k^0 \wedge x_j = 1 \to x_k \le u_k^1 \Rightarrow x_k \le \max\{u_k^0, u_k^1\} \quad \text{(b)}$$

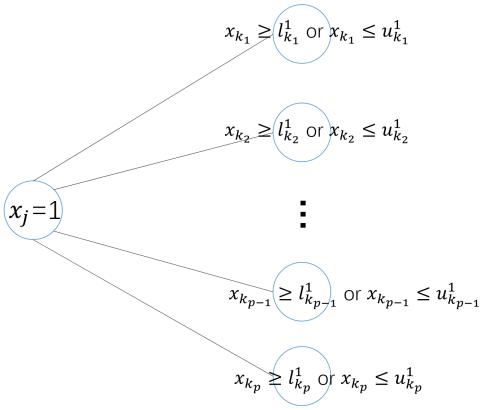
$$x_j = 0 \to x_k = l_k \wedge x_j = 1 \to x_k = u_k \Rightarrow x_k = l_k + (u_k - l_k)x_j \quad \text{(c)}$$

$$x_j = 0 \to x_k = u_k \wedge x_j = 1 \to x_k = l_k \Rightarrow x_k = u_k - (u_k - l_k)x_j \quad \text{(d)}$$

对与 x_j 处在同一约束中的所有变量重复上述过程就能找出隐藏在原模型中的这种关系, 并用图的形式清晰表示出来, 如下所示:



(a) $x_j = 0$



(b) $x_j = 1$

图 2.1 隐含图

图中 x_{k_i} 表示在域传播中受到 x_j 界变化影响的变量, $l^0_{k_i}$ 表示 $x_j = 0$ 时变量 x_{k_i} 的下界, $l^1_{k_i}$ 表示 $x_j = 1$ 时变量 x_{k_i} 的下界, $u^0_{k_i}$ 表示 $x_j = 0$ 时变量 x_{k_i} 的上界, $u^1_{k_i}$ 表示 $x_j = 1$ 时变量 x_{k_i} 的上界.

举例来看是如何把与二元变量在同一约束中的变量放入隐含图中的:

$$3 \leq x_1 + x_2 + x_3 \leq 4$$

$$x_1, x_2 \in \{0, 1\}$$

$$0 \leq x_3 \leq 5$$

当 $x_1 = 0$, 有:

$$3 \leq x_2 + x_3 \leq 4 \rightarrow 2 \leq x_3 \leq 4$$

当 $x_1 = 1$, 有:

$$2 \leq x_2 + x_3 \leq 3 \rightarrow 1 \leq x_3 \leq 3$$

隐含图中, 从 $x_1 = 0$ 这个端点出发, 可以推出 $2 \le x_3 \le 4$, 从 $x_1 = 1$ 出发, 可以推出 $1 \le x_3 \le 3$. 事实上, 在这个例子中, x_3 的界可以缩紧为 $1 \le x_3 \le 4$.

下面给出隐含图算法的流程图:

算法 5 隐含图算法

输入: 整数变量集合 I

输出: 隐含图 IG(implication graph)

1: **for** $x_i \in I$ **do**
2: $x_i = 0$;
3: 遍历包含 x_i 的所有约束, 调用域传播算法, 求出约束中其他变量的新界 $l_k^0 \le x_k \le u_k^0$;
4: $l_k^0 \le x_k \le u_k^0$ 保存到 IG 中;
5: $x_i = 1$;
6: 遍历包含 x_i 的所有约束, 调用域传播算法, 求出约束中其他变量的新界 $l_k^1 \le x_k \le u_k^1$;
7: $l_k^1 \le x_k \le u_k^1$ 保存到 IG 中;
8: 恢复 x_i 的界, $0 \le x_i \le 1$;
9: **end for**

隐含图算法时间复杂度计算: 第一步遍历模型中的所有约束, 因为 $|I| \le n$, 耗费时间为 O(n); 第二步遍历二元变量所在的所有约束, 必然少于总约束个数 m, 所以总的时间为 O(mn).

2.3　探测算法

探测算法作为域传播算法的推广, 使得域传播算法能更有效的应用在包含二元变量的模型中. 算法包含四个部分：隐含图, 冲突图, 域传播, 探测. 算法思想如下:

记 $x_k \in \{0, 1\}$, $k \in B$, B 为模型中所有二元变量构成的集合, 记 $x_j \in [l_j, u_j]$, $j \in N$ 表示其他变量.l_j^0, u_j^0 表示 $x_k = 0$ 时, 变量 x_j 的变量界：l_j^1, u_j^1 表示 $x_k = 1$ 时, 变量 x_j 的变量界. 通过观察得到如下四点结论:

- 如果 x_k 固定成 0 导致原问题不可行了, 那么 x_k 固定为 1. 同样, 如果 x_k 固定成 1 导致原问题不可行了, 那么 x_k 固定为 0.
- 如果 $l_j^0 = u_j^0$, 且 $l_j^1 = u_j^1$, x_j 可以用 x_k 来替代, $x_j = l_j^0 + (l_j^1 - l_j^0)x_k$.
- $l_j' = \min\{l_j^0, l_j^1\}$, $u_j' = \max\{u_j^0, u_j^1\}$ 为变量 x_j 新的变量界.

- $x_k = 0 \to l_j^0 \le x_j \le u_j^0$, $x_k = 1 \to l_j^1 \le x_j \le u_j^1$ 可以保存在隐含图中.

下面给出算法流程图:

算法 6 探测算法

输入: 二元变量集合 B

输出: 更新隐含图 IG(Implication Graph), 或者推测出原问题是否可行

1: **for** $x_i \in B$ **do**
2: 如果 x_i 已经被固定或者被其他变量表示, 继续;
3: **for** $v = 0, 1$ **do**
4: 尝试固定变量 $x_i = v$;
5: 在隐含图和冲突图中传播 $x_i = v$ 的信息, 获得其他与 x_i 同在隐含图以及冲突图中的变量的界信息 $[\hat{l}^v, \hat{u}^v] \subseteq [l, u]$;
6: 调用域传播算法去获得新的变量界 $[\tilde{l}^v, \tilde{u}^v] \subseteq [\hat{l}^v, \hat{u}^v]$;
7: 恢复 x_k 的界为 $x_k \in \{0, 1\}$, 以及在 4, 5, 6 步中改变的变量的界为之前的 $[l, u]$;
8: **end for**
9: 如果尝试固定变量 $x_i = 0$ 在第 7 步域传播算法导出冲突, 固定变量 $x_i = 1$, 同样, 如果尝试固定变量 $x_i = 1$ 在第 7 步域传播算法导出冲突, 固定变量 $x_i = 0$;
10: **for** $x_j = 1, ..., n, j \ne i$ **do**
11: $l_j = \min\{\tilde{l}_j^0, \tilde{l}_j^1\}$, $u_j = \max\{\tilde{u}_j^0, \tilde{u}_j^1\}$;
12: 如果 $\tilde{l}_j^0 = \tilde{u}_j^0$, 且 $\tilde{l}_j^1 = \tilde{u}_j^1$, 用 x_i 代替 x_j, $x_j = \tilde{l}_j^0 + (\tilde{l}_j^1 - \tilde{l}_j^0)x_i$;
13: 如果 $\tilde{l}_j^0 > \hat{l}_j$, 添加 $x_i = 0 \to x_j \ge \tilde{l}_j^0$ 到隐含图;
14: 如果 $\tilde{u}_j^0 < \hat{u}_j$, 添加 $x_i = 0 \to x_j \le \tilde{u}_j^0$ 到隐含图;
15: 如果 $\tilde{l}_j^1 > \hat{l}_j$, 添加 $x_i = 1 \to x_j \ge \tilde{l}_j^1$ 到隐含图;
16: 如果 $\tilde{u}_j^1 < \hat{l}_j$, 添加 $x_i = 1 \to x_j \le \tilde{u}_j^1$ 到隐含图;
17: **end for**
18: **end for**

算法时间复杂度考虑: 假设在第 3 步 for 循环中平均调用域传播的次数为 k. 因为 $|B| \le n$, 故探测算法的时间复杂度为 $O(m^k n^2)$.

2.3.1 终止条件

从上一节探测算法的时间复杂度来看, 当二元变量数目足够大的时候, 算法是比较耗费时间的, 这与预处理算法设计初衷相违背 (预处理算法通常都是时间复杂度为 1 次的算法, 来保证预处理不会耗费过长的时间). 需要在算法中设计一

些参数来控制算法运行时间, 同时还要保证算法有效性.

在设置参数前先介绍一个预处理中常用的概念：锁数. 锁数是和变量相关的一个参数, 分为上锁数和下锁数. 上锁数表示限制变量增大的约束个数, 下锁数表示限制变量减小的约束个数. 例如有如下约束:

$$x_1 - x_2 \leq 1$$

x_1 的上锁数为 1, 下锁数为 0;x_2 的上锁数为 0, 下锁数为 1.

因为进入算法的二元变量数目会影响算法的速度, 通常根据某种排序规则会选择部分二元变量进入算法, 故在执行算法前, 先给每个二元变量一个合适的分数十分重要. 给出的打分规则如下:

$$x_i^{score} = \zeta^- + \zeta^+ + |IG^-| + |IG^+| + 5|CG^-| + 5|CG^+|$$

其中 ζ^- 表示 x_i 的下锁数, ζ^+ 表示 x_i 的上锁数;$|IG^-|$ 表示 $x_i = 0$ 的隐含图包含的变量个数, $|IG^+|$ 表示 $x_i = 1$ 的隐含图包含的变量个数; $|CG^-|$ 表示 $x_i = 0$ 的冲突图包含的变量个数, $|CG^+|$ 表示 $x_i = 1$ 的冲突图包含的变量个数.

某个变量分数越低, 表示算法对这个变量效果越有限, 所以当固定或者替换 50 个变量时候, 考虑终止算法. 仅仅限制算法对其有效果的变量是不够的, 对没效果的变量也需要做出一个统计, 给出参数 nuseless 表示如果二元变量 x_k 通过探测算法不会使得某个变量固定或者被其他二元变量替代, 则 nuseless 值加 1; 参数 ntotaluseless 表示如果二元变量 x_k 通过探测算法不会使得某个变量的界缩紧, 则 ntotaluseless 值加 1. 通过给参数 nuseless 和 ntotaluseless 设置一个上限来终止算法. 当然, 如果这两个参数监测的改变发生了, 那么这两个参数会被重置为 0.

2.3.2　数值结果

测试算例为公开算例库 miplib2017, miplib3 等收集来的 27 个简单算例. 测试方法为 CMIP 下自实现探测算法, 把执行算法后的模型生成一个预处理后的 mps 文件, 用 CPLEX 运行预处理后的 mps 文件与原 mps 比较最终值是否正确. 同时从时间, 固定变量, 聚合变量, 改变变量界发生次数统计算法运行效果.

表 2.4 探测算法效果 (仅列出 27 个算例中有效果的算例)

问题名字	是否正确求解	时间	固定变量	聚合变量	改变变量界

atm_5_10_1	是	0.0	2	0	9
dcmulti	是	0.0	1	0	30
dsbmip	是	0.0	24	0	108
gesa3	是	0.0	0	0	8
gesa3_o	是	0.0	0	0	2
misc03	是	0.01	15	0	0
p0548	是	0.0	3	0	0

列表中所给出的 27 个算例算法效果并不明显, 因为探测算法其实是域传播算法的一个扩展, 算例在执行探测算法前先执行了域传播算法, 所以固定变量, 聚合变量, 改变变量界的效果有限.

因探测算法误差主要来源于域传播算法产生的截断误差, 故这里不再对算法数值稳定性做分析.

第 3 章　机位分配问题

3.1　问题背景

机位分配问题, 是指把有限的机位资源提供给不同的飞机, 使得对每一架的安排都能让其所属的航空公司满意. 把机位看做资源, 选择哪架飞机到哪个机位看做任务, 则这是一个资源分配问题, 而任务和资源的分配问题可以分为如下四类:

- 多任务对多资源: 任务-资源分配问题较复杂的情况下, 是一个任务需要多个资源支撑, 一个资源可以同时支撑多个资源.
- 单任务对多资源: 每个任务需要多个资源共同完成, 并且独占这些资源. 这些资源不能够为其他任务服务.
- 多任务对单资源: 每个资源可以供给多个任务, 每个任务只利用一个资源.
- 单任务对单资源: 每个任务只占用一个资源, 每个资源只能供给一个任务.

机位分配问题属于其中的“多任务对单资源”. 机位的停放就是任务, 机位就是资源. 在占用时间不重叠的情况下, 一个机位, 一天可以供给多个停放任务.

对于同一个机位的使用, 两个飞机的占用时间一般是不能重叠的. 比较特殊的情况是, 机位的使用是有冲突的, 比如说 A、B 两个机位, 不能同时停放飞机. 当然这种冲突还有更复杂的情况, 但是对于建模来说并不影响, 此处不一一列举.

不同的飞机, 由于航空公司之类的各种属性的限制, 有些机位不能放, 有些机位放的收益不同, 对于某航班不能放在某些机位的这类约束必须满足. 不同飞机放在不同机位带来的收益, 最终在目标函数的权重中体现 (比如, 客机就是偏好放在廊桥上).

3.2　问题模型

根据上一节问题背景的简述, 总结机位分配问题如下:

- 目标函数: $\max c_{ij}x_{ij}$, 其中 c_{ij} 表示飞机 i 放机位 j 能获得的分数, x_{ij} 表示飞机 i 是否放机位 j, 为变量.
- 约束 (1): $\sum_{j\in S} x_j \le 1$, 其中集合 S 是所有机位构成的集合, 此约束表示一架飞机最多放在一个机位上.

- 约束 (2)：$x_{i_1 j_1} + x_{i_2 j_2} \leq 1$ 表示飞机 i_1 放在机位 j_1 上, 同时飞机 i_2 放在机位 j_2 上的时候, 冲突.
- 约束 (3)：$x_{ij} \in \{0, 1\}$.

所有航班构成集合 F, 所有机位构成集合 S, 综上四点, 机位分配问题的整数规划模型如下:

$$\max \quad \sum_{i,j} c_{i,j} x_{i,j}$$

$$s.t. \quad \sum_{j \in S} x_{i,j} \leq 1, i \in F \tag{1}$$

$$x_{i_1,j} + x_{i_2,j} \leq 1, i_1, i_2 \in F, j \in S, i_1 \neq i_2 \tag{2}$$

$$x_{i,j} \in \{0, 1\}, i \in F, j \in S \tag{3}$$

完成建模后, 从数据集中读取数据得到的模型统计如下:

表 3.1 原机位分配问题的 mps 文件规模

文件大小	0.97g
约束个数	7711515
变量个数	64896

用开源求解器 SCIP 运算, 在测试的数据集上无法求解此模型.

3.3 对偶固定算法

3.3.1 锁数

在叙述对偶固定算法前需要重提锁数的概念. 给定约束如下:

$$\underline{\beta} \leq a^T x \leq \bar{\beta}$$

$$l \leq x \leq u$$

对于在上述约束中的任意变量 x_i, 变量前的系数为 a_i. 假设约束的左端项为负无穷, 即 $\underline{\beta} = -\infty$. 当 $a_i > 0$ 时, 这条约束会阻碍 x_i 向上变化, 即阻碍 x_i 增大; 当 $a_i < 0$ 时, 这条约束会阻碍 x_i 向下变化. 所以, 当 $a_i > 0$, 这个变量的上锁数增加 1; 当 $a_i < 0$, 这个变量下锁数增加 1. 当约束右端为正无穷的情况类似. 若约束左右端都为一个有限的值, 显然这条约束中所有变量的上下锁数都加 1.

3.3.2　算法思想

论文之前的一些预处理方法的求解技术都是基于模型原始形式的求解算法. 它们纯粹基于模型的可行性论证, 因此独立于目标函数. 相反, 可以基于目标函数的最优性来考虑一些求解方法, 例如, 如果能证明对于每一个最优解 x^*, 存在另外一个解 $\hat{x}$ 满足 $c^* = c^T x^* = c^T \hat{x}$, 且 $\hat{x}$ 存在某一特定的下标 j, 有 $\hat{x}_j = v$, 那么就可以把 x_j 固定为 v, 从而减少变量个数.

通过这样的想法, 尝试利用目标函数的信息固定一些变量, 同时必须确保模型的可行性不会改变, 这样的预处理方法同样也不会改变原模型的最优性.

算法流程图如下:

算法 7 对偶固定

输入: 所有变量集合 X

输出: 一些变量被固定到某一确定值

1: **for** $x_i \in X$ **do**
2:　　**if** $c_i \geq 0$ 且 x_i 下锁数为 0 **then**
3:　　$x_j = l_j$;
4:　　**end if**
5:　　**if** $c_i \leq 0$ 且 x_i 上锁数为 0 **then**
6:　　$x_j = u_j$;
7:　　**end if**
8:　　**if** $c_i \neq 0$ 且 x_i 被固定为无穷 **then**
9:　　原问题无解或者不可行;
10:　　**end if**
11: **end for**

可行性分析:

对任意模型中的变量 x_i, 如果 $c_i \geq 0$, 且 x_i 下锁数为 0, x_i 会被固定到它的下界. 因为 x_i 的下锁数为 0, 没有约束阻碍变量向下变化, 故把变量固定到它的下界后原问题依然可行. 同理, 当满足第二个 if 语句的条件后, 变量被固定到它的上界依然可行.

最优性分析:

因为原问题为最小化目标, 当 $c_i \geq 0$ 时, x_i 越小越好, 故推测得出当 $x_i = l_j$ 的时候, 目标达最小; 同理, 当 $c_i \leq 0$, $x_i = u_i$, 目标取得最小值.

通过上述分析, 对偶固定算法不仅不改变原模型可行性, 且能减少变量个数.

3.3.3 数值结果

在机位分配问题中, 如果某些航班放在特定机位不会产生任何收益, 目标函数的系数会存在为 0 的情况, 使得对偶固定算法能很好地运用在这一类模型上.

本章节一开始的模型规模如下:

表 3.2 原机位分配问题的 mps 文件规模

文件大小	0.97G
约束个数	7711515
变量个数	64896

测试环境如下所示:

表 3.3 对偶固定算法测试环境

CPU	Intel(R) Core(TM) i7-8700K CPU
CPU 核数	12
使用核数	1
CPU 主频	3696MHz
内存大小	16g
系统	64 位 windows 下子 ubuntu 系统

运行了对偶固定算法后模型规模如下:

表 3.4 对偶固定算法测试结果

约束个数	7711515
变量个数	42009
变量减小个数	22887
算法时间	<0.01s

与 CPLEX 运行结果对比:

表 3.5 CPLEX 对机位分配问题的测试结果

约束个数	14431
约束减少个数	7697085
变量个数	42008
变量减小个数	22888
算法时间	8.5s

实验结果可以看出, CPLEX 也是通过对偶固定的方法来减少变量总数的, 至于减少约束个数, 则是利用了下面所说的团聚合方法.

3.4　团聚合算法

3.4.1　团聚合

团在图论中表示完全子图, 在冲突图中表示对于团中的任意二元变量都不能同时取 1 的值. 为了成功解决机位分配问题, 团聚合算法是关键.

给定约束如下:

$$x_1 + x_2 \leq 1 \tag{1}$$

$$x_2 + x_3 \leq 1 \tag{2}$$

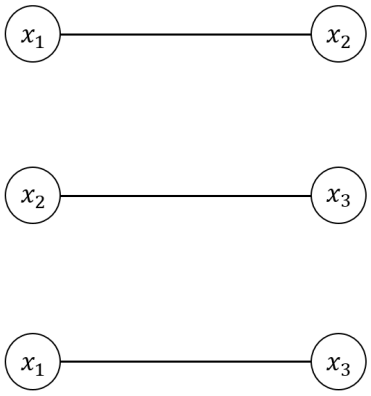
$$x_1 + x_3 \leq 1 \tag{3}$$

其中 x_1, x_2, x_3 都是二元变量. 假设第四条约束如下:

$$x_1 + x_2 + x_3 \leq 1 \tag{4}$$

用约束 (4) 表示上述三条约束, 约束 (4) 使解空间更紧. 例如, 对于松弛解 (0.5, 0.5, 0.5), 不在约束 (4) 表示的解空间内, 但在约束 (1) ~ (3) 表示的解空间内.

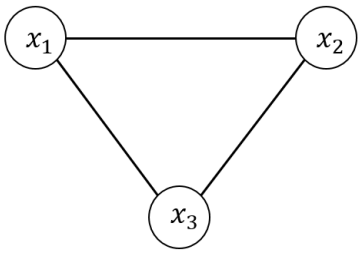
从图的角度来看, 公式 (1), (2), (3) 表示如下图所示的三条线段:



(a) $x_1+x_2 \le 1, x_2+x_3 \le 1, x_1+x_3 \le 1$

图 3.1 团聚合示例 (聚合前)

而公式 (4) 表示把三条线段对应顶点两两拼接构成的三角形, 是一个完全图:



(a) $x_1 + x_2 + x_3 \le 1$

图 3.2 团聚合示例 (聚合后)

从这个例子来看, 三条约束通过团聚合缩减到一条, 六个非零元通过团聚合缩减到三个, 故模型中大量的第 (2) 类约束可以通过团聚合的方法缩小约束和非零元的规模, 且解空间更优秀.

3.4.2 算法设计

3.4.2.1 问题与图的等价关系

因为团等价于图中的完全图, 而问题约束规模过大, 所以考虑用图的邻接矩阵来保存第 (2) 类约束等价的图.

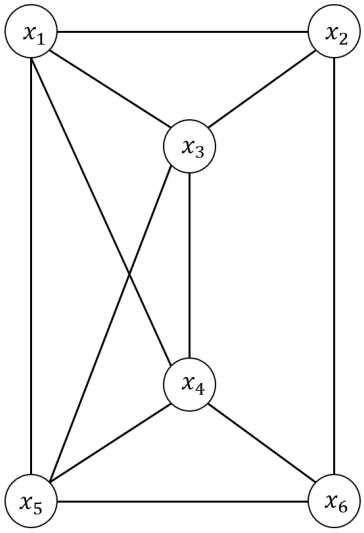
定义 3.1. 邻接矩阵: $G =< V, E >$ 是简单有向图, $|V| = n$, 定义 $n \times n$ 的矩阵 $P = (p_{ij})$, 其中 p_{ij} 为顶点 v_i 与 v_j 相邻次数.

对于之前给定的机位分配问题, 所需要的图是一个无向图, 故只用邻接矩阵的上三角来表示第 (2) 类约束对应的图.

考虑完如何保存约束的信息后, 再考虑团聚合算法和在图中寻找团的等价性. 做团聚合算法的目的是不去掉原模型可行解的情况下缩小约束规模, 那么在图中搜索团的时候必须把每一条边都划分到一个团中, 即对图中所有的边进行一个划分, 每一条边都需要在一个团中, 且最后划分的团的数量尽可能小.

3.4.2.2　算法思想

在本小节中通过用图和示例举出如何在邻接矩阵中搜索团. 给出图和其邻接矩阵如下:



(a) 算法示例 1

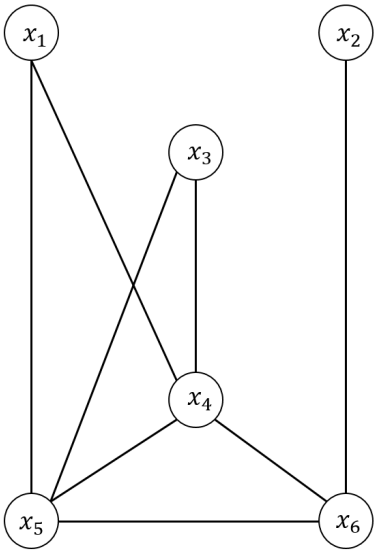
图 3.3 团聚合算法讲解图 1

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

矩阵 1

从矩阵右上的每一行开始搜索矩阵的非零元, 第一个是在第一行第二列的位置, 表示变量 x_1, x_2 不能同时为 1, 而对于两个变量来说, 这就是一个团. 接下来需要去用团聚合的办法把这个团扩大. 通过观察矩阵第一行第三列, 以及矩阵第二行第三列, 这两个位置的值都为 1, 从列来看, 表示变量 x_3 即与 x_1 冲突, 又与 x_2 冲突, 那么变量 x_3 也能添加到团中. 接着从列的方向看, 并未再发现存在一列, 这一列在 1, 2, 3 行位置的值为 1, 那么这个团无法再扩大了.

此时, 图中的三条边已经被分配到一个团中, 为了避免边的重复分配, 需要在矩阵中把这些边删除, 即第一行二、三列, 第二行三列的位置设为 0, 所得图和矩阵如下:



(a) 算法示例 2

图 3.4 团聚合算法讲解图 2

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

矩阵 2

但是这样的做法会存在一个缺陷, 从图——算法示例 1 来看, 变量 x_1, x_3,

x_4, x_5 在一个团中, 但删除关联 x_1, x_3 的边后, x_1, x_3, x_4, x_5 至少需要两个团来表示这四个变量之间的关系, 这样最后得到的团的个数增加了, 与算法设计初衷相违背, 因此关联 x_1, x_3 的边无法删除.

这只是一个简单的图的观察得到的结论, 如果图的点和边数量够多, 图的结构过于复杂, 是无法准备判断出是否某条边需要被删除, 某一条边又无法被删除 (因为删除后会使得最后得到的团过多, 算法效果并不好). 一个很简单的处理办法是: 只需要假意删除已经找到的团中的这些边, 这时候矩阵的左下部分可以利用起来了.

把第一行二、三列, 第二行三列的关于对角线对称的位置设置为 1, 所得矩阵如下:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

矩阵 3

因为在搜索团的时候只会考虑矩阵的右上部分, 不会重复搜索, 剩下需要解决的是从列以及行比较看是否能在已发现的团中加入新的顶点.

接着之前的步骤进行, 发现第一行第四列的值为 1, 把 x_1, x_4 加入一个新团中, 观察第二列和第二行发现, x_2 并未与 x_4 关联, 故 x_2 无法加入团; 观察第三列和第三行, 发现变量 x_3 可以加入团, 同样的方法可以发现变量 x_5 可以加入团. 继续之前矩阵右上角置 0 的规则和矩阵左下角置 1 的规则, 得到矩阵如下:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

矩阵 4

当利用起矩阵左下部分的时候, x_1, x_3, x_4, x_5 构成的团能被探测到, 不会出现变量 x_1, x_3, x_4, x_5 需要至少两个团来表示的情况, 但是此时决定变量是否能加入团的规则发生了一点改变, 之前只需要考虑右上部分矩阵变量 x_k 与团中变量 $\{x_{i_1}, ..., x_{i_j}\}$ 对应邻接矩阵位置 $P_{i_1,k}, ..., P_{i_j,k}$(这里假设 $i_1, ..., i_j < k$) 是否为 1, 现在需要考虑 $P_{i_1,k} \vee P_{k,i_1}, ..., P_{i_j,k} \vee P_{k,i_j}$ 的值是否为 1, 至此, 问题得到完美解决.

接着把上述方法重复执行得到的矩阵如下:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

矩阵 5

这一步是因为在邻接矩阵中先发现的边为连接变量 x_2, x_6 的边, 从图来看团 $\{x_2, x_6\}$ 并不能加入新的变量, 因此通过这一步矩阵右上方只有一个 1 转移到关于对角线对称的矩阵左下方位置.

再重复执行一遍此方法过程的矩阵如下:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

矩阵 6

矩阵 6 的右上部分已经不存在为 1 的值了, 至此, 所有的边都被分配到了团中, 所得的四个团为 $\{x_1, x_2, x_3\}, \{x_1, x_3, x_4, x_5\}, \{x_4, x_5, x_6\}, \{x_2, x_6\}$.

3.4.2.3 算法流程图

算法 8 团聚合

输入: 图顶点集合 V, $|V| = n$, 邻接矩阵 P

输出: 聚合后的团表 $Cliques$

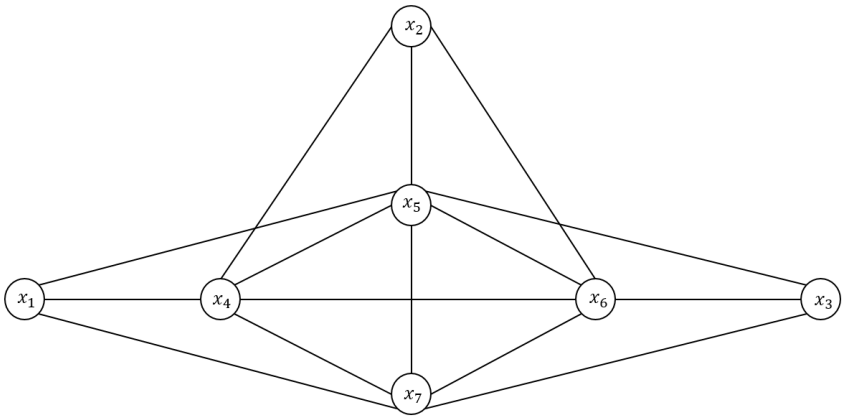
for $i = 1, ..., n$ **do**
  for $j = i + 1, ..., n$ **do**
    if $P_{ij} = 1$ **then**
      把顶点 x_i, x_j 加入团 S 中;
      for $k = 1, ..., n$ **do**
        if $k \neq i, j$ **then**
          for $l \in S$ **do**
            比较邻接矩阵 P 对于位置, $(P_{lk}||P_{kl})$ 的取值是否为 1;
          end for
          if 第 8 步对任意 $l \in S$ 都成立 **then**
            把 x_k 加入团 S;
            for $l \in S$ **do**
              if $l \neq k$ **then**
                if $k < l$ **then**
                  $P_{kl} = 0$;
                  $P_{lk} = 1$;
                else
                  $P_{lk} = 0$;
                  $P_{kl} = 1$;
                end if
              end if
            end for
          end if
        end if
      end for
      把团 S 加入团表 $Cliques$;
      $S \leftarrow \varnothing$;
    end if
  end for
end for

算法时间复杂度分析：设极大团最多包含 k_{max} 个顶点. 前两个 for 循环为变量邻接矩阵右上部分非零元, 即约束个数, 复杂度为 O(m); 第三个 for 循环为遍历

顶点个数, 实际上只需遍历与选取边的两个顶点相连的点, 复杂度为 O(k_{max}); 第四个 for 循环遍历团中变量个数, 复杂度为 O(k_{max}); 故算法复杂度为 O(mk_{max}^2).

3.4.2.4 冲突信息重复

上述给出的算法对变量搜索极大团的顺序不同, 最后得出的团表不一定相同, 例如:



(a) 算法示例 3

图 3.5 团聚合算法讲解图 3

这个图的邻接矩阵为:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

这个图的特殊之处在于, 图中存在四个极大团 $\{x_1, x_4, x_5, x_7\}$, $\{x_2, x_4, x_5, x_6\}$, $\{x_3, x_5, x_6, x_7\}$, $\{x_4, x_5, x_6, x_7\}$, 但是极大团 $\{x_4, x_5, x_6, x_7\}$ 被其他三个极大团给包住了, 且另外三个极大团的信息足够表示极大团 $\{x_4, x_5, x_6, x_7\}$, 当使用团聚合算

法后, 所得矩阵如下:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

通过团聚合算法, 极大团 $\{x_4, x_5, x_6, x_7\}$ 并不会保存在团表中, 因为 $\{x_4, x_5, x_6, x_7\}$ 中的任意一条边都存在于其他三个极大团中. 因此, 在分支定界过程中, 对于松弛解 $\bar{x} = (0.1, 0.1, 0.1, 0.3, 0.3, 0.3, 0.3)$ 这样的解可能是某个子问题的最优解, 所以约束

$$x_4 + x_5 + x_6 + x_7 \leq 1$$

需要加入团表来去掉类似上述 $\bar{x}$ 的最优解, 这也是一种割平面方程. 但本文中给出的团聚合算法, 基于变量下标顺序搜索极大团, 可能会出现例子中 $\{x_4, x_5, x_6, x_7\}$ 无法加入团表的情况, 这是算法的不足之处.

3.4.3　数值结果

3.4.3.1　原模型实验

本章节一开始的模型规模如下:

表 3.6 原机位分配问题的 mps 文件规模

文件大小	0.97G
约束个数	7711515
变量个数	64896

测试环境如下所示:

表 3.7 团聚合算法的测试环境

CPU	Intel(R) Core(TM) i7-8700K CPU
CPU 核数	12
使用核数	1
CPU 主频	3696MHz
内存大小	16g
系统	64 位 windows 下子 ubuntu 系统

运行了团聚合算法后模型规模如下:

表 3.8 团聚合算法的测试结果

文件大小	50.6M
约束个数	21340
变量个数	64896
算法时间	68s

Table3 对应的数学模型 SCIP 和 CMIP 都无法求解, 给出 Table4 对应的模型求解如下:

表 3.9 SCIP, CMIP 求解团聚合后的模型的对比

	时间	最优解
SCIP	76.86s	-3.28970000000000e+05
CMIP	40s	-3.28970000000000e+05

所得的最优解与 CPLEX 计算结果一致, 故认为算法正确.

3.4.3.2 修改目标函数后的模型实验

从对 CPLEX 的分析看出, CPLEX 运用了对偶固定和团聚合这两种预处理方法, 为了让团聚合算法有一个更好的对比, 稍加对目标函数的修改使得对偶固定方法无法运行来进行团聚合算法的一个对比.

根据对实验数据的分析, 把目标函数中没有的变量添加一个绝对值很小的负

系数加入目标函数, 测试环境与上一节相同, CPLEX 采用单线程运行, 仅执行预处理, 测试结果如下:

表 3.10 CPLEX, CMIP 求解修改模型后的机位分配问题的对比

	时间	预处理后变量个数	预处理后约束个数
CMIP	70.12s	64896	21340
CPLEX	25.08s	64896	21340

因为没有考虑在时间上的优化处理, 算法的耗时是 CPLEX 三倍以上. 从最后变量个数来观察, CPLEX 也没有预处理方法来固定变量, 故执行团聚合算法的时候顶点个数与 CMIP 一致的, 为 64896 个. 最后预处理过后得到极大团的个数为约束的个数, CMIP 是 21340 个, 与 CPLEX 一致.

第 4 章　冲突图在辛烷算法中的应用

辛烷算法是一种通过枚举八面体的扩展面找到可行的整数点的启发式方法. 算法以 octane[13, 27] 命名, 主要通过对八面体结构特征的研究, 建立纯 0,1 整数规划问题所有整数解与超立方体面之间的一一映射, 从几何的角度寻找问题可行解.

4.1　几何背景

考虑混合整数规划中的一种特殊情况, 所有的变量都是整数变量, 且变量下界为 0, 上界为 1, 形式如下:

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \in \{0,1\}^n \end{aligned}$$

假设 $n = 2$, 问题的解空间在如下正方形区域内:

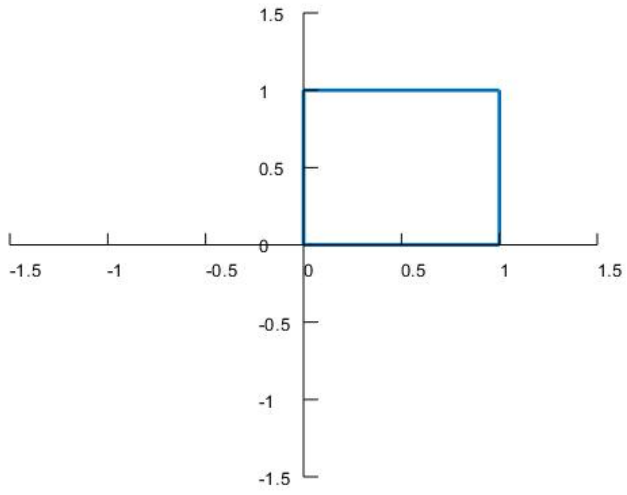


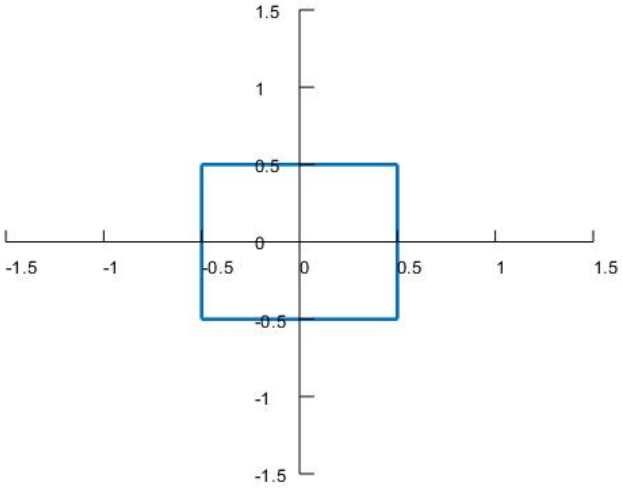
图 4.1 二维空间纯 0, 1 规划松弛解空间

定义如下两个集合 K, K^*:

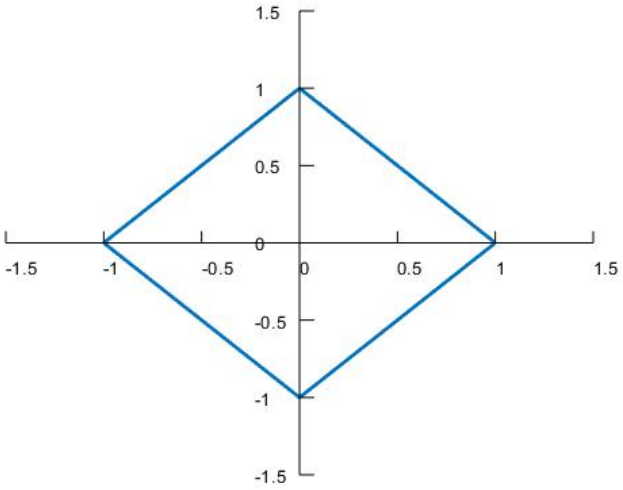
$$K := \{x \in R^2 | -\frac{e}{2} \leq x \leq \frac{e}{2}\}, e := \{1\}^2$$

$$K^* := \{x \in R^2 | \, \|x\|_1 \leq 1\}$$

空间 K 以及 K^* 绘图如下:

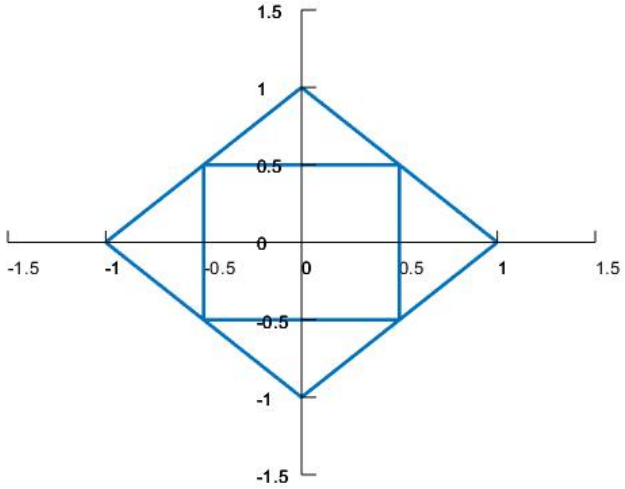


(a) K



(a) K^*

空间 K 为二维空间纯 0, 1 规划解空间的一个平移, 即把正方形的中心从 $(\frac{1}{2}, \frac{1}{2})$ 移到 $(0, 0)$ 的位置. 把空间 K 和 K^* 在一张图中看:



(a) K **和** K^*

K 的四个顶点和 K^* 的四条边可以建立一个一一对应的关系.

把 K 和 K^* 扩展到 n 维空间中:

$$K := \{x \in R^n | -\frac{e}{2} \leq x \leq \frac{e}{2}\}, e := \{1\}^n$$

$$K^* := \{x \in R^n | \|x\|_1 \leq \frac{n}{2}\} = \{x \in R^n | \delta^T x \leq \frac{n}{2}, \forall \delta \in \{\pm 1\}\}$$

K 的顶点和 K^* 的面 $\delta^T x = \frac{n}{2}$ 可以建立一个一一对应的关系如下:

$$\delta_i = \begin{cases} -1, & x_i = 0 \\ 1, & x_i = 1 \end{cases}$$

通常整数规划中的启发式方法是在求解原问题的松弛问题后得到一个松弛解 $\bar{x}$, 再给定一个搜索方向和距离, 在搜索方向上找一个距离 $\bar{x}$ 最近的整数解. 在辛烷算法中, 把松弛解 $\bar{x}$ 的坐标减去 $\frac{e}{2}$, 给定一个搜索方向 r, 定义距离如下:

定义 4.1.　如果存在 $\lambda > 0$ 使得 $\delta^T(\overline{x} + \lambda r) = \frac{n}{2}$ 成立, 则 δ 称为一个可达面, λ 为 $\bar{x}$ 到可达面 δ 的距离.

在之前的分析中, 任意一个面 δ 都与一个整数解一一对应, 所以 $\bar{x}$ 到一个可达面的距离可以看做到一个整数解的距离, 因此寻找离松弛解 $\bar{x}$ 最近的一个整数解就是寻找离松弛解 $\bar{x}$ 最近的一个可达面.

4.2 最先可达面

在上一节中给出的可达面的定义, 因为参数 λ 是一个大于 0 的参数, 把 λ 取值最小的一个可达面称为最先可达面, 所对应的顶点也是距离当前松弛解最近的顶点. 通过可达面寻找最先可达面的算法最早被 Balas[28] 提出.

在说明得到最先可达面的算法前先叙述一个结果. 假设搜索方向 r 在 n 维空间中的任意分量皆大于 0, 证明 $\delta = (1, ..., 1)$ 是一个可达面. 因为 $\frac{n}{2} - \delta^T \overline{x} \geq 0$, $\delta^T r \geq 0$, 则 λ 取值为一个大于 0 的数, 所以面 $\delta = (1, ..., 1)$ 是一个可达面. 同样, 如果搜索方向存在小于 0 的分量, 用坐标变换的方法使得搜索方向分量全部大于 0. 这样选定一个搜索方向后, 通过坐标变化使得 $\delta = (1, ..., 1)$ 恒为一个可达面.

下一步则从可达面 $\delta = (1, ..., 1)$ 出发得到最先可达面. 先定义几个符号如下:

对 $\delta = \{\pm 1\}^n$ 以及 $I \subseteq N$, $N = \{1, ..., n\}$:

$$p(\delta, I) := -\sum_{i \in I} \delta_i \overline{x}_i \ , \ P(\delta) := \frac{n}{2} + p(\delta, N)$$
$$q(\delta, I) := \sum_{i \in I} \delta_i r_i \ , \ Q(\delta) := q(\delta, N)$$
$$f(\delta, I) := p(\delta, I)/q(\delta, I)$$

从可达面的定义来看 $P(\delta) = \frac{n}{2} - \delta^T \overline{x}$, $Q(\delta) = \delta^T r$, $P(\delta)$ 与 $Q(\delta)$ 的比值为 λ. 再定义可达面的一种操作—翻转如下:

定义 4.2. 假设 δ 为 K^* 的一个面.

1. 翻转 δ 的第 i 个分量即用 $-\delta_i$ 来代替 δ_i. 对于集合 $I \subseteq N$, 用 $\delta \diamond I$ 来表示翻转集合 I 中所有元素为下标的 δ 的分量.

2. 对于 δ 为 K^* 的一个可达面, $I \subseteq N$:

(a.) 如果 $\lambda(\delta \diamond I) < \lambda(\delta)$, 称集合 I 是一个减小翻转.

(b.) 如果 $\lambda(\delta \diamond I) \leq \lambda(\delta)$, 称集合 I 是一个非增翻转.

下面不加证明的给出如下定理:

定理 4.1. *δ 为 K^* 的一个可达面, 并且集合 $I \subseteq N$. 则集合 I 是一个减小翻转当且仅当如下成立:*

(1.) $q(\delta, I) > 0$ 且 $\lambda(\delta) < f(\delta, I)$;

(2.) $q(\delta, I) < 0$ 且 $\lambda(\delta) > f(\delta, I)$;

(3.) $q(\delta, I) = 0$ 且 $0 < p(\delta, I)$.

对于可达面 $\delta = (1, ..., 1)$ 来说, 对于任意 I 来说 $q(\delta, I) > 0$, $\lambda(\delta)$ 是一个可以计算出的常数, 因此找到一个集合 I 使得计算出的 $f(\delta, I)$ 的值大于 $\lambda(\delta)$ 就可以通过翻转得到一个距离当前松弛解更近的可达面.

给出算法框图如下所示:

算法 9 最先可达面算法

输入: 线性规划可行解 $\bar{x}$, 搜索方向 r

输出: 最先可达面 δ^*

1: 坐标变化 $\bar{x} - \frac{e}{2}$;
2: 对于搜索方向小于 0 的 r_i, 用 $-x_i$ 替代 x_i 使得搜索方向大于 0;
3: 以 $\frac{\bar{x}_i}{r_i}$ 非减的顺序重新排列 x 的下标;
4: $\delta^* \leftarrow e$;
5: **for** $i = 1, ..., n$ **do**
6:　　**if** $\delta^* \diamond i$ 是一个减小翻转 **then**
7:　　$\delta^* \leftarrow \delta^* \diamond i$;
8:　　**end if**
9: **end for**

算法第三步中 $-\frac{\bar{x}_i}{r_i}$ 是定理 4.1 中的 $f(\delta, I)$, 因此通过算法第三步的排序可以迅速判断出是否 $\delta = (1, ..., 1)$ 为最先可达面. 假设变量个数为 n, 算法第三步排序时间复杂度为 O($nlogn$), 循环时间复杂度为 O(n), 因此整个算法时间复杂度为 O($nlogn$).

4.3　整数点集合搜索

在二维空间中, 假设线性规划松弛可行解为 $x = (\frac{1}{2}, \frac{1}{2})$, 搜索方向为 $r = (-\frac{3}{2}, -\frac{1}{2})$, 最先可达面并不唯一, 因此把所有的最先可达面构成一个集合 F.

在启发式算法中, 无法判断距离松弛解最近的哪个整数点会使得目标函数值达到一个更优秀的值, 所以需要把距离松弛解某一段距离下的所有整数点拿出来作比较, 这个距离可以考虑不为最小距离, 稍微大于最小距离的整数点依然可以纳为考虑范围, 因为现在还不存在证明整数规划的最优值一定距离线性规划最优值最近. 因此, 在之前叙述的算法中仅仅得到一个最先可达面是不够的, 需要取得一些距离松弛解较近的整数解的集合, 这些整数解最好是可行的. 因此 Avis 和

Fukuda[29] 提出关于这些整数点集的反向搜索算法.

考虑一个辅助图 $G = (V, E, \omega)$ 来帮助完成上述的工作:

$$V := \{\delta \in \{\pm 1\}^n : Q(\delta) > 0\}$$

$$E := \{(\delta, \delta \diamond i) : \lambda(\delta) \leq \lambda(\delta \diamond i)\}$$

$$\omega(\delta, \delta \diamond i) := \lambda(\delta \diamond i) - \lambda(\delta), (\delta, \delta \diamond i) \in E$$

顶点集合 V 是所有可达面的构成的, 图中的边表示任意两个可达面仅仅通过一次翻转得到, 边上的权重为两个顶点距离的差, 为正数. 通过这样的一个转换, 寻找一些距离松弛解最近的顶点等价于在图 G 中从某个最先可达面出发寻找一条最短路径树, 在权重存在相同值的情况下用 Dijkstra 算法时间复杂度为 $O(kn^2 logk)$, 其中 k 为最短路径树顶点个数. 这个时间复杂度过高难以接受, 因此需要一个时间复杂度更低的算法实现.

Balas, Ceria, Dawande, Margot 和 Pataki[27] 找到了一个反向搜索函数 f 来构造上述辅助图 G 的一个子图, 使得 G 存在一个根节点无父节点, 对根节点外的任意顶点都存在唯一一个的父节点. 构造方法如下:

定义 4.3. 对 $\forall i \in N$, 有:

1. 对于可达面 δ 的一个 $+$ 到 $-$ 的减小翻转, 如果 $\delta_i = 1$, 那么 $\delta \diamond i$ 必然比 δ 先到达;

2. 对于可达面 δ 的一个 $-$ 到 $+$ 的非增翻转, 如果 $\delta_i = -1$, 那么 $\delta \diamond i$ 不会 δ 晚到达;

通过上述定义构造反向搜索函数 f, 即定义除根节点外的任意顶点的父节点. 如果对可达面 δ 存在至少一个 $+$ 到 $-$ 的减小翻转, 令 i 为所有减少翻转中下标最小的值, 定义 δ 父节点 $pred(\delta) := \delta \diamond i$; 如果可达面 δ 不存在 $+$ 到 $-$ 的减小翻转, 但至少存在一个 $-$ 到 $+$ 的非增翻转, 令 i 为所有非增翻转中下标最大的值, 定义 δ 父节点 $pred(\delta) := \delta \diamond i$.

Balas 等人证明了通过上述定义的反向搜索函数必然存在一个根节点 δ^*, 为最先可达面, 且通过一个复杂度为 $O(knlogk)$ 的算法可以得到 k 个最先在搜索方向 r 上找到的 K^* 中的面.

算法流程图如下:

算法 10 整数点集合搜索

输入: 线性规划可行解 $\bar{x}$, 搜索方向 r, 最先可达面 δ^*, 可达面集合 L

输出: 整数点集合 F

1: **for** $i = n, ..., 1$ **do**
2:　　**if** $\delta^* \diamond i$ 是 $-$ 到 $+$ 的非增翻转 **then**
3:　　$\delta^* \leftarrow \delta^* \diamond i$;
4:　　**end if**
5: **end for**
6: 初始化 $L \leftarrow \delta^*$;
7: **for** $i = 1, ..., k$ **do**
8:　　依次翻转 L 中第 i 个面的所有位置, 如果翻转后所得面的到 $\bar{x}$ 的距离比翻转前的面更大或者相等, 且翻转后所得面不在 L 中, 则插入翻转后得到的面到 L;
9: **end for**
10: 把 L 中的面转化成对应的整数点插入 F;
11: 检查 F 中的整数点是否可行;

算法第 1 步 for 循环实际上是寻找根节点, 第 7 步 for 循环搜索 k 个最先在搜索方向 r 上找到的 K^* 中的面. 搜索算法复杂度为 O($knlogk$), 这里不再给出证明, 具体证明见参考文献 [27].

4.4　搜索方向选择

4.4.1　目标方向

给定纯 0, 1 规划问题形式如下:

$$\min \quad cx$$

$$Ax \leq b$$

$$x \in \{0,1\}^n$$

选择 $-c$ 作为搜索方向 r, 因为原问题为极小化问题, 当 $c_i > 0$ 时, 令 $x_i = 0$, 对应 $\delta = -1$; 而当 $c_i < 0$ 时, 令 $x_i = 1$, 对应 $\delta = 1$.

4.4.2　结合冲突图的新搜索方向

给定原问题的线性松弛解 $\bar{x}$, 从第一个分量开始, 在冲突图中搜索是否发现这个分量, 如果发现某个分量 x_{i_1} 在冲突图 $x_{i_1} + ... + x_{i_k} \leq 1$ 中, 令冲突图中松弛

值最大的 $x_{i_{max}}$ 对应的搜索方向这个分量的取值为 1, 冲突图中其他变量对应搜索方向分量取值为 -1; 不在冲突图中的变量对应搜索方向分量为目标函数系数的负值. 在设计结合冲突图的新搜索方向中, 对应搜索方向 r 的分量有如下四种情况:

1. r_i 对应的 x_i 不在冲突图中, $r_i = -c_i$, c_i 为目标函数 x_i 前的系数;

2. r_i 对应的 x_i 在冲突图中某个团 S 中, 如果 S 中存在一个变量 x_j 对应的 $r_j = 1$, 则 $r_i = -1$;

3. r_i 对应的 x_i 在冲突图中某个团 S 中, 如果 S 中不存在变量对应的搜索方向分量取值为 1, 且 $\bar{x}_i$ 为团 S 中所有变量松弛解值的最大值, 则 $r_i = 1$;

4. r_i 对应的 x_i 在冲突图中某个团 S 中, 如果 S 中不存在变量对应的搜索方向分量取值为 1, 且 $\bar{x}_i$ 不为团 S 中所有变量松弛解值的最大值, 则 $r_j = -1$.

举例如下, 假设原问题包含 x_1, x_2, x_3, x_4 四个变量, 且 x_1, x_2, x_3 在冲突图中, 有 $x_1 + x_2 + x_3 \leq 1$. 此时 $\bar{x} = (0.3, 0.3, 0.4, 0.5)$, 因为 x_3 此时的松弛值最大, 所以对应的 $r_3 = 1$, 当 x_3 取值为 1 时, x_1, x_2 必须取值为 0, 所以 $r_1 = r_2 = -1$ 表示向 x_1, x_2 为 0 的方向搜索, r_4 的值由目标函数的系数决定, 尽可能在不违背约束的情况下使得目标函数取得更小的值.

4.5 辛烷算法

4.5.1 算法流程图

算法 11 辛烷算法

输入: 线性规划可行解 $\bar{x}$, 搜索方向 r, 可达面集合 L

输出: 整数点集合 F

1: 坐标变化 $\bar{x} - \frac{e}{2}$;
2: 对于搜索方向小于 0 的 r_i, 用 $-x_i$ 替代 x_i 使得搜索方向大于 0;
3: 以 $\frac{\bar{x}_i}{r_i}$ 非减的顺序重新排列 x 的下标;
4: $\delta^* \leftarrow e$;
5: **for** $i = 1, ..., n$ **do**
6: **if** $\delta^* \diamond i$ 是一个减小翻转 **then**
7: $\delta^* \leftarrow \delta^* \diamond i$;
8: **end if**
9: **end for**
10: **for** $i = n, ..., 1$ **do**

11: **if** $\delta^* \diamond i$ 是 $-$ 到 $+$ 的非增翻转 **then**

12: $\delta^* \leftarrow \delta^* \diamond i$;

13: **end if**

14: **end for**

15: 初始化 $L \leftarrow \delta^*$;

16: **for** $i = 1, ..., k$ **do**

17: 依次翻转 L 中第 i 个面的所有位置, 如果翻转后所得面的到 $\bar{x}$ 的距离比翻转前的面更大或者相等, 且翻转后所得面不在 L 中, 则插入翻转后得到的面到 L;

18: **end for**

19: 把 L 中的面转化成对应的整数点插入 F;

20: 检查 F 中的整数点是否可行;

4.5.2　数值结果

实验方法：一般化第四章中的机位分配模型, 用邻接矩阵的方式表达出两个变量的冲突关系, 用随机撒点的方法模拟机位分配模型中的变量冲突约束 $x_{i_1 j_1} + x_{i_2 j_2} \leq 1$, 目标函数系数为绝对值 10000 内的随机数, 正负皆可. 通过 Highs[1]求解器求解出松弛解的值, 调用辛烷算法观察是否能找到可行解, 再调用 CPLEX 求随机撒点机位分配模型的最优值, 比较目标值最小的可行解 V_{sol} 与最优值 V_{opt} 之间的 gap$:= (V_{sol} - V_{opt})/V_{opt}$, 对比目标搜索方向和新设计搜索方向区别.

实验规模: 采用 100, 500, 1000 三种变量个数, 每种实验 100 次, 通过平均矩阵稠密度 (邻接矩阵 1 的比例), 平均约束个数 (团聚合后团的个数), 找到可行解次数, 平均可行解个数, 平均 gap, 平均算法运行时间六个角度比较.

[1]HiGHS 是根据 MIT 许可免费提供的开源串行和并行求解器，适用于大规模稀疏线性规划和混合整数规划模型.

测试环境如下:

表 4.1 辛烷算法的测试环境

CPU	Intel(R) Core(TM) i7-8700K CPU
CPU 核数	12
使用核数	1
CPU 主频	3696MHz
内存大小	16g
系统	64 位 windows 下子 ubuntu 系统

因两种搜索方向在同一个随机模型下比较, 故下表 4.2 ~ 4.4 中平均矩阵稠密度, 平均约束个数是一致的. 数值结果如下:

表 4.2 包含 100 个变量约束随机的机位分配模型的数值结果

	平均矩阵稠密度	平均约束个数	找到可行解次数
目标搜索方向	50.95%	811.54	0
结合冲突图的搜索方向			100
	平均可行解个数	平均 gap	平均时间消耗
目标搜索方向	0	∞	0.03
结合冲突图的搜索方向	40.04	40.15%	0.05

表格 4.2 中, 在 100 个变量的随机约束模型中, 基于冲突图的搜索方向每次都能找到可行解, 而基于目标函数的搜索方向无法找到可行解, 因此基于目标函数的搜索方向无需保存找到的可行解, 故时间更少.

表 4.3 包含 500 个变量约束随机的机位分配模型的数值结果

	平均矩阵稠密度	平均约束个数	找到可行解次数
目标搜索方向	50.23%	21615.70	0
结合冲突图的搜索方向			100
	平均可行解个数	平均 gap	平均时间消耗
目标搜索方向	0	∞	0.11
结合冲突图的搜索方向	46.83	85.26%	0.84

表格 4.3 中, 在 500 个变量的随机约束模型中, 基于冲突图的搜索方向依然每次能找到可行解, 但是相比于 100 个变量, 搜索到的可行解与问题最优解 gap 更大.

表 4.4 包含 1000 个变量约束随机的机位分配模型的数值结果

	平均矩阵稠密度	平均约束个数	找到可行解次数
目标搜索方向	50.14%	87926.10	0
结合冲突图的搜索方向			100
	平均可行解个数	平均 gap	平均时间消耗
目标搜索方向	0	∞	0.22
结合冲突图的搜索方向	61.98	92.00%	4.35

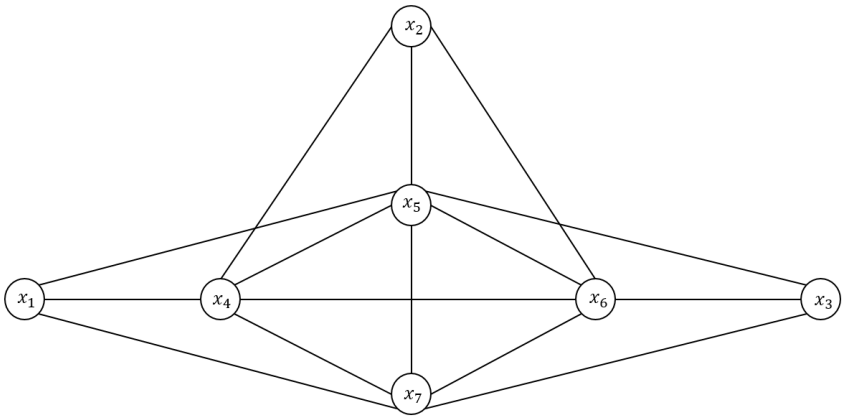
表格 4.4 中, 在 1000 个变量的随机约束模型中, 基于冲突图的搜索方向每次仍能找到可行解, 但是相比于 500 个变量, 搜索到的可行解与问题最优解 gap 继续变大. 因为在基于冲突图的搜索方向中, 主要是从问题是否可行出发来寻找解, 较少考虑问题的最优性, 故随着变量规模的增大, 找到一个启发解的质量越来越差. 因此一个更合理的搜索方向应该同时考虑可行性和最优性.

第 5 章　总结与展望

在混合整数规划问题中一些特殊的问题会存在一些特殊的结构, 在包含二元变量的问题中, 冲突图就是问题中的特殊结构, 利用冲突图的信息可以使得一些算法更加完善.

在第二章中, 探测算法实际上是利用冲突图的信息, 使得域传播能得到进一步的扩展, 变量的界更紧. 因为该算法耗费时间比大多数预处理算法更长, 因此在执行前都会先执行域传播算法, 故探测算法的效果并不会这么突出, 且需要加入一系列限制参数及时终止算法.

第三章中的团聚合算法, 算法在设计方向上暂时没有考虑内存消耗问题, 把一个稀疏矩阵的 0 元存储下来是一个巨大的内存消耗, 这样的内存消耗是没有意义的. 同时在选择顶点扩充团的时候, 只需要找原来团中度最小的顶点周围的顶点判断是否加入团, 可以减少对顶点的判断次数. 在搜索内部极大团算法的例子中:



(a) 算法示例 5

尽管从预处理的角度来看, 确实去掉极大团 $\{x_4, x_5, x_6, x_7\}$ 后原问题的约束个数会减少 (每个极大团都是原问题的一条约束), 因此约束矩阵非零元的个数也会减少, 但是从求解松弛问题来考虑, 因为去掉了约束:

$$x_4 + x_5 + x_6 + x_7 \leq 1$$

在求解松弛问题的过程中可能得到一个 $\hat{x}$, 有 $\hat{x}_4 = \hat{x}_5 = \hat{x}_6 = \hat{x}_7 = 0.3$, 这样的

松弛解不是一个整数解, 从可行解的凸包看, 去掉上述约束, 使得可行解的凸包变大, 不利于之后分支定界过程, 因此是否再次搜索加入一些内部极大团, 需要从求解问题的整个过程来考虑. 运行搜索内部极大团算法需要耗费更多的时间, 感性上来看, 找到的极大团越多, 需要的时间越多, 但会对后面分支定界过程有帮助, 去掉了一些不需要的松弛解. 一个有效的算法, 既能在预处理阶段尽可能消去非零元个数, 又能使得预处理后的模型尽可能接近原模型可行解的凸包.

在第四章中对辛烷算法实验中, 只针对从机位分配问题中抽象出的一类特殊的模型有不错的效果, 对于其他更加一般的模型效果不佳, 且这个算法比较耗费时间, 所以在现在开源界第一的求解器 SCIP 中一般是关闭的. 需要再寻找一些更加合适的搜索方向使得该算法寻找到的解更加优秀.

混合整数规划是一个非常难解决的问题, 随着模型规模的不断增大, 得到问题全局最优解的时间往往可能是几个小时或者更甚至几十个小时, 完全达不到工业应用需求. 在设计混合整数规划算法的时候, 不仅需要算法的时间复杂度在理论上有保证, 同时也要考虑实际的一些情况去加速算法. 在数据结构的选择上也要在算法运行速度和内存消耗方面做一个取舍, 即尽可能缩小内存消耗的情况下又不至于算法运行速度大幅下降.

附录 A　中国科学院大学学位论文撰写要求

学位论文是研究生科研工作成果的集中体现，是评判学位申请者学术水平、授予其学位的主要依据，是科研领域重要的文献资料。根据《科学技术报告、学位论文和学术论文的编写格式》（GB/T 7713-1987）、《学位论文编写规则》（GB/T 7713.1-2006）和《文后参考文献著录规则》（GB7714—87）等国家有关标准，结合中国科学院大学（以下简称“国科大”）的实际情况，特制订本规定。

参考文献

[1] WOLSEY L A. Integer Programming: volume 42[M]. Wiley Online Library, 1998.

[2] LAWLER E L, WOOD D E. Branch-and-bound methods: A survey[J]. Oper. Res., 1966, 14 (4):699-719.

[3] MAROS I. Computational Techniques of the Simplex Method: volume 61[M]. Springer Science & Business Media, 2012.

[4] ACHTERBERG T. SCIP: solving constraint integer programs[J]. Mathematical Programming Computation, 2009, 1(1):1-41.

[5] BREARLEY A L, MITRA G, WILLIAMS H P. Analysis of mathematical programming problems prior to applying the simplex algorithm[J]. Math. Program., 1975, 8(1):54-83.

[6] SAVELSBERGH M W. Preprocessing and probing techniques for mixed integer programming problems[J]. ORSA Journal on Computing, 1994, 6(4):445-454.

[7] GAMRATH G, KOCH T, MARTIN A, et al. Progress in presolving for mixed integer programming[J]. Math. Program. Comput., 2015, 7(4):367-398.

[8] ACHTERBERG T, BIXBY R E, GU Z, et al. Presolve reductions in mixed integer programming[J]. INFORMS J. Comput., 2020, 32(2):473-506.

[9] CONFORTI M, CORNUÉJOLS G, ZAMBELLI G, et al. Integer Programming: volume 271 [M]. Springer, 2014.

[10] GOMORY R E. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem[M]//JÜNGER M, LIEBLING T M, NADDEF D, et al. 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art. Springer, 2010: 77-103.

[11] GOMORY R. Outline of an algorithm for integer solutions to linear programs[J]. Bulletin of the American Mathematical Society, 1958, 64:275-278.

[12] GOMORY R E. Solving linear programming problems in integers[J]. Combinatorial Analysis, 1960, 10:211-215.

[13] BERTHOLD T. Primal heuristics for mixed integer programs[J]. PhD thesis, Berlin: Technische Universität, 2006.

[14] BRITO S S. Conflict graphs in mixed-integer linear programming: preprocessing, heuristics and cutting planes.[J]. PhD thesis, Universidade Federal de Ouro Preto, 2020.

[15] ATAMTÜRK A, NEMHAUSER G L, SAVELSBERGH M W. Conflict graphs in solving integer programming problems[J]. European Journal of Operational Research, 2000, 121(1): 40-55.

[16] MARTIN A. Integer programs with block structure[J]. Habilitations-Schriftm, Technische Universität Berlin, 1998.

[17] SILVA J P M, SAKALLAH K A. GRASP: A search algorithm for propositional satisfiability [J]. IEEE Trans. Computers, 1999, 48(5):506-521.

[18] DAVIS M, LOGEMANN G, LOVELAND D. A machine program for theorem-proving[J]. Communications of the ACM, 1962, 5(7):394-397.

[19] DAVIS M, PUTNAM H. A computing procedure for quantification theory[J]. Journal of the ACM (JACM), 1960, 7(3):201-215.

[20] PADBERG M W. On the facial structure of set packing polyhedra[J]. Math. Program., 1973, 5(1):199-215.

[21] JOHNSON E L, NEMHAUSER G L. Recent developments and future directions in mathematical programming[J]. IBM Systems Journal, 1992, 31(1):79-93.

[22] HOFFMAN K L, PADBERG M. Solving airline crew scheduling problems by branch-and-cut [J]. Management science, 1993, 39(6):657-682.

[23] BIXBY R E, LEE E K. Solving a truck dispatching scheduling problem using branch-and-cut [J]. Oper. Res., 1998, 46(3):355-367.

[24] BORNDÖRFER R. Aspects of set packing, partitioning, and covering[J]. PhD thesis, Technische Universitat Berlin, Berlin, Germany, 1998.

[25] SANTOS H G, TOFFOLO T A, GOMES R A, et al. Integer programming techniques for the nurse rostering problem[J]. Annals of Operations Research, 2016, 239(1):225-251.

[26] BRITO S S, SANTOS H G. Preprocessing and cutting planes with conflict graphs[J]. Computers & Operations Research, 2021, 128:105176.

[27] BALAS E, CERIA S, DAWANDE M, et al. Octane: A new heuristic for pure 0–1 programs [J]. Operations Research, 2001, 49(2):207-225.

[28] BALAS E. Ranking the facets of the octahedron[J]. Discret. Math., 1972, 2(1):1-15.

[29] AVIS D, FUKUDA K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra[C]//DRYSDALE R L S. Proceedings of the Seventh Annual Symposium on Computational Geometry, North Conway, NH, USA, , June 10-12, 1991. ACM, 1991: 98-104.

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

2014 年 9 月——2018 年 6 月在中国海洋大学数学科学学院数学与应用数学系获得学士学位.

2018 年 9 月——2021 年 6 月在中国科学院大学数学与系统科学研究院计算数学系攻读硕士学位.

致　谢

在毕业论文即将定稿的时候, 我的三年硕士生涯也接近尾声了, 三年时光不经意间逝去, 依然还记得 2018 年刚入学时候对新的生活, 新的校园, 新的朋友们充满了好奇. 依然还记得 2018 年 9 月份国科大开学典礼上白春礼校长的发言:“国有疑难可问谁, 强国一代有我在!”在这硕士三年的最后时刻, 谨此向给与我关心和帮助的老师们, 同学们, 朋友们和家人表示由衷感谢.

首先要感谢我的导师戴彧虹研究员, 感谢戴老师三年以来对我的关心和指导, 论文的顺利完成离不开戴老师一直以来对我的帮助. 还记得当初研究生入学考试复试完之后第一次与戴老师见面, 戴老师渊博的知识给我带来很深刻的印象, 记得当时因为学习上的疑惑咨询戴老师, 戴老师以他独到的见解和渊博的知识给了我一个极其具有启发的回答. 在三年的学习期间, 在戴老师的指导下使我能接触最前沿的科学知识, 去了解, 探索, 并具有一些发现, 使得我能顺利完成一些研究项目课题, 给我带来了很大的收获. 硕士三年的学习对我以后的成长发展至关重要, 在此向戴老师表示由衷的感谢.

感谢优化课题组的袁亚湘老师、刘歆老师、刘亚锋老师、孙聪老师在讨论班上对我的指导和提出的建议, 使得我在科研的道路上走的更轻松和, 更远.

感谢在国科大雁栖湖校区给我上过课的王晓老师、郭田德老师、许志强老师等, 你们给我敞开了知识的大门, 让我学到了很多专业知识. 感谢院里的办公老师们, 吴继萍老师、丁如娟老师、钱莹老师、张纪平老师、刘颖老师、胡洁老师、邵欣老师、尹永华老师、刘霞老师、关华老师等.

感谢优化课题做的所有师兄弟姐妹们. 感谢陈伟坤师兄、陈亮师兄、傅凯师兄、杨沐明师兄、张瑞进师兄、吉振远师兄、赵浩天师兄、陈雅丹师姐、肖纳川师兄、吴宇宸师兄等, 同届的陈圣杰、黄磊、姜博鸥、王磊、汪思维, 还有裴骞师弟、张跃师弟、赵成师弟、王圣超师弟、谢鹏程师弟、陈硕师弟、武哲宇师妹、胡雨宽师弟、胡雨婷师妹、章煜海师弟等, 很高兴我能和你们一起在科学院学习. 感谢陈伟坤师兄在平时整数规划的学习中给我的帮助, 在学习过程中给了我很多方法和建议, 使得我毕业论文能顺利完成.

感谢在雁栖湖宿舍的小伙伴们, 许锐航、夏梓耕、朱子恒、叶子诚、徐戍、唐

斌、饶一鹏、王溪、符松韧. 在一起共同学习的时光很开心. 感谢符松韧愿意当我的答辩秘书, 帮助我处理了很多事情.

最后感谢在我失望难过的时候一直默默支持我的家人们, 你们的支持、鼓励、对我无私的爱是我不断前进的最大动力. 感谢一直以来你们的教导、付出, 家永远是旅人最温暖的港湾!

三年的硕士经历, 三年的学习生涯, 让我成长的收获的是用言语无法描述的, 再次感谢所有给我关心帮助过的人!