中国科学院大学

University of Chinese Academy of Sciences

硕士学位论文

背包问题的精确分离算法研究

作者姓名：胡雨婷

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

2023 年 6 月

Exact Separation Algorithms for 0-1 Knapsack Problems

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Philosophy

in Computational Mathematics

By

HU Yuting

Supervisor: Professor DAI Yuhong

Academy of Mathematics and Systems Science, Chinese Academy of Sciences

June, 2023

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。承诺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享有著作权的研究成果，未在以往任何学位申请中全部或部分提交。对本论文所涉及的研究工作做出贡献的其他个人或集体，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收集、保存和使用学位论文的规定，即中国科学院大学有权按照学术研究公开原则和保护知识产权的原则，保留并向国家指定或中国科学院指定机构送交学位论文的电子版和印刷版文件，且电子版与印刷版内容应完全相同，允许该论文被检索、查阅和借阅，公布本学位论文的全部或部分内容，可以采用扫描、影印、缩印等复制手段以及其他法律许可的方式保存、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

混合整数规划问题是一类基本的最优化问题, 在工业生产、航空运输、经济与金融以及计算机通信等领域有广泛应用, 其理论和算法研究受到众多学者关注. 割平面方法是求解混合整数规划问题的关键方法之一. 其中, 背包问题是混合整数规划研究的一个经典问题, 精确分离算法是将分离问题转化为最优化问题进行求解的一种割平面算法. 本文将从精确分离的角度对两类特殊背包问题的分离问题进行研究.

对于多维背包多面体, 从精确分离的角度研究了聚合背包约束的分离算法. 在现有的精确分离算法框架下, 将聚合方法运用到行生成过程中, 设计了特殊的分离算法. 对带有多条约束的背包问题算例进行数值实验, 说明聚合方法可以有效提升问题的求解效果.

对于含 GUB 约束的背包多面体, 我们研究了其精确分离算法. 将线性松弛解的分离问题建模成最优化问题, 在行生成过程中考虑 GUB 约束的特性, 我们给出求解含 GUB 约束的背包子问题的动态规划算法, 并进一步改进以占优表储存的稀疏动态规划算法来求解序列升维系数. 对带 GUB 约束的装箱问题测试集进行数值测试, 验证了该精确分离算法的有效性.

关键词：混合整数规划, 背包问题, GUB 约束, 精确分离算法, 聚合方法

Abstract

Mixed integer programming problem is a kind of basic optimization problem, which is widely applied in the fields of industrial production, air transport, economy, finance and computer communication. The theoretical and computational study has attracted great attention. The cutting plane method is one of the most important methods for solving mixed integer programming problems. Knapsack problem is a classic problem in mixed integer programming, and the exact separation algorithm is a cutting plane algorithm that transforms the separation problem into an optimization problem. This thesis focuses on the separation problems of two special knapsack problems from the perspective of exact separation.

For the multi-dimensional knapsack polyhedron, the separation algorithm of aggregated knapsack constraints is studied on the basis of exact separation. Under the framework of the existing exact separation algorithm, the aggregation method is applied to the row generation and a special separation algorithm is designed. Numerical experiments on the knapsack problem with multiple constraints show that aggregation can effectively improve the performance of the solution.

For the knapsack polyhedron with GUB constraints, we discuss exact separation algorithm of these problems. The separation problem of linearly relaxation is modeled as an optimization problem, and the property of GUB constraints are considered in row generation. We propose dynamic programming algorithm of the knapsack subproblem with GUB constraints, and apply the dominant table to computing the coefficients of the sequence lifting inequalities. Numerical tests on testset, including the bin packing problem with GUB constraints, verify the effectiveness of the exact separation algorithm.

Key Words: Mixed integer programming, Knapsack problem, GUB constraint, Exact separation, Aggregation

目 录

图目录

表目录

符号列表

字符

$\mathbb{R}$	实数集
$\mathbb{R}^n$	n 维实向量集
$\mathbb{R}^{m\times n}$	$m \times n$ 维实矩阵集
$\mathbb{R}_+$	非负实数集
$\mathbb{Z}$	整数集
$\mathbb{Z}^n$	n 维整数向量集
$\mathbb{Z}^{m\times n}$	$m \times n$ 维整数矩阵集
$\mathbb{Z}_+$	非负整数集
$[n]$	指标集 $\{1, ..., n\}$
a^T	向量 a 的转置
$\|S\|$	集合 S 的元素个数
conv(S)	集合 S 的凸包

缩写

$\boldsymbol{K}$	一维背包集合
$\boldsymbol{MK}$	多维背包集合
$\boldsymbol{GUB}$	广义上限约束
$\boldsymbol{K_{GUB}}$	带有 GUB 约束的背包集合

第 1 章　绪论

混合整数规划是一类部分决策变量限制为整数的最优化问题, 在现实生活中有着广泛应用, 工业、经济、物流、计算机等领域中许多问题都可以归结为混合整数规划问题.

一般的混合整数规划模型如下:

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } & Ax \le b, \\ & l \le x \le u, \\ & x_j \in \mathbb{Z}, \forall j \in I, \end{aligned} \tag{1-1}$$

其中 $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $l \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$. $x \in \mathbb{R}^n$ 是决策变量, $c^T x$ 是目标函数, $Ax \le b$ 是线性约束, $l \le x \le u$ 是边界约束, 集合 $I \subseteq [n] = \{1, ..., n\}$ 是限制为整数的变量指标集.

对于公式 (1-1) 形式的整数规划问题, 定义其线性松弛问题如下:

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } & Ax \le b, \\ & l \le x \le u. \end{aligned} \tag{1-2}$$

线性松弛问题即在原问题的基础上不考虑整数条件, 其最优值小于或等于原整数规划问题的最优值, 这也为原问题提供了一个下界. 线性松弛问题的最优解被称为线性松弛解, 若松弛最优解为整数解, 则其也是原整数规划问题的最优解. 一般来说, 混合整数规划问题是 $\mathcal{NP}$-难的, 对于其解法的研究一直以来都是学术界关注的热点之一.

1.1　混合整数规划

混合整数规划的研究可以追溯到上世纪五十年代, 运筹学创始人 G.Dantzig[1] 对旅行商问题的整数规划模型进行研究, 提出了求解线性规划问题的单纯形法. 随着线性规划理论的进一步发展和应用, 人们发现现实中的很多生产问题都可以建模成混合整数规划问题. 因此, 对混合整数规划问题的研究与创新具有重要现

实意义, 学术界对混合整数规划问题的理论和求解算法也逐渐关注起来. 目前最主要的求解算法包括分支定界方法和割平面方法.

分支定界方法的主要思想是将原整数规划问题分解成若干个线性松弛子问题来找到最优解. 1960 年, Land 和 Doig [2] 设计求解整数规划问题的分支定界算法. 割平面方法是从求解原问题的线性松弛问题出发, 计算能分离当前松弛解和原可行集的割平面, 将其加入松弛问题, 重复这一过程最终求得整数最优解. 1958 年, Gomory [3] 提出割平面算法, 后由 Crowder [4] 等人将割平面方法用于求解一般整数规划问题. 在此基础上, 将分支定界法和割平面法结合起来形成分支割平面法. 分支割平面方法主要是在分支的过程中计算割平面, 将其加入松弛问题来得到更大的下界.

在后续对割平面方法进一步讨论之前, 我们需要先介绍集合的维数、有效不等式和刻面定义的不等式等概念.

定义 1.1. 如果若干向量 $x^1, x^2, ..., x^q \in \mathbb{R}^n$ 的线性方程

$$\sum_{i=1}^{q} \lambda_i x^i = 0$$

的唯一解是 $\lambda_1 = ... = \lambda_q = 0$, 则称这些向量是线性无关的.

定义 1.2. 如果若干向量 $x^0, x^1, x^2, ..., x^q \in \mathbb{R}^n$ 的线性方程组

$$\sum_{i=0}^{q} \lambda_i x^i = 0, \ \sum_{i=0}^{q} \lambda_i = 0$$

的唯一解是 $\lambda_0 = \lambda_1 = ... = \lambda_q = 0$, 则称这些向量是仿射无关的.

定义 1.3. 集合 $S \subseteq \mathbb{R}^n$ 的维数是 S 中最大的仿射无关点的个数减去 1, 记作 $\dim(S)$.

定义 1.4. 对于集合 $S \subseteq \mathbb{R}^n$, 若不等式 $\alpha^T x \leq \beta$ 满足 $\alpha^T y \leq \beta \ \forall y \in S$, 则称该不等式是集合 S 的有效不等式.

定义 1.5. 对于集合 $P \subseteq \mathbb{R}^n$, 若存在 $m \in \mathbb{Z}_+$, 矩阵 $A \in \mathbb{R}^{m \times n}$ 和向量 $b \in \mathbb{R}^m$ 使得 $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, 则称集合 P 是多面体.

定义 1.6. 对于多面体 $P \subseteq \mathbb{R}^n$, 如果集合 F 可以写作如下形式.

$$F := P \cap \{x \in \mathbb{R}^n : \alpha^T x = \beta\},$$

其中 $\alpha^T x \leq \beta$ 是多面体 P 的一个有效不等式, 则称该集合是多面体 P 的面. 进一步, 若 $\dim(F) = \dim(P) - 1$, 则称集合 F 是多面体 P 的刻面, 有效不等式 $\alpha^T x \leq \beta$ 是多面体 P 的刻面定义的不等式.

在混合整数规划问题中, 整数变量常见的来源主要有以下三种. 一是用于决策二值问题, 如背包问题中是否将某物品放入背包、设施选址问题中某个地方是否建立设施等. 二是描述物品个数, 如工件生产问题中各种工件生产的数量、网络流问题中的载体数量等. 三是非线性函数分段线性化, 如电力问题中的机组爬坡约束等. 可以看出, 在现实的生产生活中混合整数规划问题都有着广泛应用. 减少问题求解的时间、提高所得解的精度可以更大程度地降本增效, 混合整数规划求解器应运而生. 目前市面上常见的数学规划求解器有 GUROBI、CPLEX、LINDO、COPT、MOSEK、SCIP、CMIP 等.

1.2　背包问题

背包问题 (Knapsack Problems, KP) 是混合整数规划研究的一个经典问题, 它是指给定若干个有重量和收益的物品, 在放入的物品重量之和不超过一定承载量的前提下, 如何选择这些物品使得总收益最大. 一方面, 背包问题很容易描述, 但是根据物品的重量和收益属性也能扩展出多种问题. 另一方面, 背包问题不仅经常作为子问题出现在优化问题中, 还经常出现在组合优化的数学优化问题建模中. 在计算复杂度方面, 背包问题是一个弱 $\mathcal{NPC}$ 问题, 能用伪多项式时间复杂度的动态规划算法求解.

1.2.1　0-1 背包问题

近年来, 背包问题被应用在许多现实问题中. 因此, 背包问题及其各种变形与推广问题都是研究热点, 如 0-1 背包问题、完全背包问题、多维背包问题、二次背包问题等 [5]. 其中, 0-1 背包问题规定每个物品最多只能选一次, 是最基础的背包问题之一, 也得到学术界广泛的探讨和研究. 0-1 背包问题的数学模型如下.

$$\begin{aligned} \max \quad & \sum_{i\in[n]} c_i x_i \\ \text{s.t.} \quad & \sum_{i\in[n]} a_i x_i \leq b, \\ & x_i \in \{0,1\}, \forall i \in [n], \end{aligned} \tag{1-1}$$

其中 $[n]=\{1,...,n\}$, $c\in\mathbb{R}^n_+$, $a\in\mathbb{Z}^n_+$, $b\in\mathbb{Z}_+$. 不失一般性, 假设 $a_1\le a_2\le ...\le a_n$, $a_i\le b, \forall i\in[n]$ 并且 $\sum_{i\in[n]}a_i>b$.

目前对于 0-1 背包问题的求解算法主要分为以下两类. 一是精确算法, 如动态规划法、分支定界法等, 这类算法可以求得精确解, 但是随着问题规模变大会出现指数爆炸问题. 二是近似算法, 如遗传算法、粒子群算法等, 这些算法的计算复杂度相对精确算法较低, 但是难以得到精确解, 有时不能满足计算要求. 下面我们将详细介绍, 如何用动态规划算法求得对于 0-1 背包问题的最优值 [6], 并通过回溯找到对应的最优解.

定义问题 $P(k,z)$ 如下.

$$\begin{aligned} f(k,z)=\max\ & \sum_{i\in[k]}c_ix_i \\ \text{s.t.}\ & \sum_{i\in[k]}a_ix_i\le z, \\ & x_i\in\{0,1\},\forall i\in[k]. \end{aligned} \tag{1-2}$$

考虑一系列子问题 $P(k,z)$ 对应的最优值 $f(k,z)$, 其中 $k\in[n]$, $z\in[b]$, 则 $f(n,b)$ 即为原 0-1 背包问题的最优值. 为了表达方便, 不妨令 $f(0,z)=0,\forall z\in[b]$. 对于问题 (1-1) 有迭代公式:

$$f(k,z)=\begin{cases} f(k-1,z), & z<a_k, \\ \max\{f(k-1,z),c_k+f(k-1,z-a_k)\}, & otherwise. \end{cases}$$

对于第 k 个物品, 只有当 $a_k\le z$ 时才能比较是否将其放入背包. 如果不放入该物品, 则容量不变, 对应的最优值为 $f(k-1,z)$. 若放入该物品, 则容量要减少 a_k, 对应的最优值为 $f(k-1,z-a_k)$ 加上放入该物品得到的收益 c_k. 由于优化目标是最大化放入背包的物品总收益, 所以我们选择这两种情况下最大的总收益.

得到最优值 $f(n,b)$ 后, 我们对各变量依次进行回溯找最优解 x^*. 令 $\lambda:=b$, 对于某个 k, 当 $f(k,\lambda)=f(k-1,\lambda)$ 时, 则 x^*_k 取 0 , 否则 x_k 取 1, 并且更新 $\lambda:=\lambda-a_k$. 由算法过程, 我们可以得到 0-1 背包问题的动态规划算法的计算复杂度是 $O(nb)$.

1.2.2 0-1 背包多面体

对于背包约束, 定义 0-1 背包集合

$$K=\left\{x\in\{0,1\}^n:\sum_{i\in[n]}a_ix_i\le b\right\}.$$

0-1 背包多面体则为 0-1 背包集合的凸包,

$$P = \mathrm{conv}(K).$$

在求解一般的整数规划问题时, 由背包约束生成的割平面对问题的求解效率的提升有重要作用. 其中, 升维覆盖不等式是由原始背包约束得到的背包割 [4, 7]. 其它背包割平面还包括混合整数圆整割 (Mixed Integer Rounding Cuts) [8, 9]、升维投影割 (Lift-and-Project Cuts) [10]、分裂割 (Split cuts) [11]、群割 (Group Cuts) [12, 13] 等. Achterberg [14] 的研究说明当不考虑背包割平面时, CPLEX 12.5 的求解效率将会降低 14%. 对于 0-1 背包多面体, 已有的研究主要从背包集合的覆盖出发得到有效不等式.

定义 1.7. 如果 $C \subseteq [n]$ 满足 $\sum_{i\in C} a_i > b$, 则称 C 为集合 X 的覆盖. 如果 C 是集合 X 的覆盖并且其任意子集都不是集合 X 的覆盖, 则称 C 为集合 X 的极小覆盖.

基于覆盖这一概念, 有以下三类有效不等式.

定义 1.8. 覆盖不等式. 如果 C 为集合 X 的覆盖, 则覆盖不等式

$$\sum_{i\in C} x_i \le |C| - 1$$

是 0-1 背包多面体的有效不等式. 如果 C 为集合 X 的极小覆盖, 则称 $\sum_{i\in C} x_i \le |C| - 1$ 为极小覆盖不等式.

Wolsey [15] 证明极小覆盖不等式是低维空间的刻面定义不等式. 实际计算中, 覆盖不等式一般需要进一步升维来得到原始约束空间的有效不等式. 其中一种简单可行的方法是考虑扩展覆盖不等式 [16–18].

定义 1.9. 扩展覆盖不等式. 在覆盖 C 的基础上, 考虑扩展覆盖 $E(C) = C \cup \{j : a_i \le a_j \ \forall i \in C\}$, 有扩展覆盖不等式

$$\sum_{i\in E(C)} x_i \le |C| - 1.$$

或者, 根据 Padberg [19] 对背包约束上的有效不等式的升维有如下定理.

定理 1.1. *假设不等式 $\sum_{i=1}^{n-1} \alpha_i x_i \le \beta$ 为限制集 $K_n := K \cap \{x \in \mathbb{R}^n : x_n = 0\}$ (或 $K_n := K \cap \{x \in \mathbb{R}^n : x_n = 1\}$) 的有效不等式, 且 $K_n \ne \emptyset$, 令*

$$\alpha_n \le \beta - \max\left\{\sum_{i=1}^{n-1} a_i x_i \le b - a_n, x \in \{0,1\}^n\right\}$$

或

$$\alpha_n \geq \max\left\{\sum_{i=1}^{n-1} a_i x_i \leq b, x \in \{0,1\}^n\right\} - \beta$$

则 $\sum_{i=1}^{n} \alpha_i x_i \leq \beta$ (或 $\sum_{i=1}^{n-1} \alpha_i x_i + \alpha_n(1-x_n) \leq \beta$) 为原始背包约束 $\sum_{i\in[n]} a_i x_i \leq b$ 关于 $\sum_{i=1}^{n-1} \alpha_i x_i \leq \beta$ 的升维不等式.

当定理 1.1 中关于 α_n 的取值满足等号时, 称为极大升维不等式. 更进一步, 当 $\sum_{i=1}^{n-1} \alpha_i x_i \leq \beta$ 确定限制集 K_n 上的 d 维面时, 其极大升维不等式确定 K 上的至少 $d+1$ 维面. 其中, 对之前固定为下界 0 的变量进行升维称为向上升维, 对固定为上界 1 的变量进行升维称为向下升维. 一般地, 我们可以对覆盖不等式进行升维计算得到升维覆盖不等式:

定义 1.10. 升维覆盖不等式. 在覆盖不等式 $\sum_{i\in C} x_i \leq |C| - 1$ 的基础上, 考虑对 $x_i, i \in [n]\backslash C$ 逐个进行升维来改进覆盖不等式.

1.2.3 含 GUB 约束的背包问题

广义上限 (Generalized Upper Bound, GUB) 约束是指一组 0-1 变量之和小于或等于 1, 有如下形式:

$$\sum_{i\in S_j} x_i \leq 1, \forall j \in [m].$$

其中 S_j $\forall j \in [m]$ 表示 GUB 集合, $[m] = \{1, ..., m\}$ 表示 GUB 集合的下标集合. 不妨假设 $\cup_{j\in[m]} S_j = [n]$, 即每个变量 x_i $\forall i \in [n]$ 都至少属于一个 GUB 集合. 根据 GUB 约束的定义, 我们知道一个 GUB 集合里面的 0-1 变量至多只有一个变量能取到 1, 其余变量均为 0.

现实生活的资源分配中经常有物品与物品之间不相容或某项资源只能用在一个物品上, 这些情况均可以建模成含 GUB 约束的背包问题. 这类问题是 0-1 背包问题的一个扩展, 其目标是在满足背包约束的前提下, 还要考虑变量之间不相容的关系, 即 GUB 约束, 选择物品使得总收益最大. 当 $|S_j| = 1$ $\forall j \in [m]$ 时, 问题就是经典的 0-1 背包问题. 因此 0-1 背包问题可以看作是含 GUB 约束的背包问题的特例. Wolsey [20] 首次对含 GUB 约束的背包问题进行研究并引入 GUB 覆盖这一概念. 随后, 关于 GUB 约束的特殊性引起许多学者关注, 见 [21, 22]. 含 GUB 约束的背包问题在很多领域都具有重要作用, 比如食物分配问题 [23]、考试安排问题 [24]、有冲突的装箱问题 [25] 等. 关于含 GUB 约束的背包问题的求解方法主要有分支定界算法、遗传算法、局部搜索等.

当考虑问题中 GUB 约束这一特殊结构时, 可以计算得到更强的割平面. 目前已经有学者研究了含 GUB 约束的背包多面体. Nemhauser [26] 通过定义 0-1 变量的依赖集来给出升维覆盖割平面的系数. Sherali [27] 研究了含 GUB 约束的背包多面体的特性. Gu [7] 讨论了升维 GUB 覆盖不等式的理论和算法过程. Gokce [28] 考虑了“$ax \geq b$”的情况并定义了 α-强覆盖. Zeng [29] 研究含多个不相交 GUB 约束的背包问题的刻面定义不等式. Luiz [30] 对含冲突变量的背包问题提出不相交团割平面.

已有的研究工作都是从背包问题的覆盖不等式出发, 考虑 GUB 约束中 0-1 变量至多有一个取 1 的特点, 最后得到原问题的割平面. 我们考虑针对含 GUB 约束的背包多面体设计一般的割平面算法, 在精确分离算法的求解过程中结合 GUB 约束的特点, 计算得到原问题的刻面定义不等式.

1.3　精确分离算法

自割平面方法提出以来, 许多研究人员继承这一思想并提出各种不同的割平面, 如 Gomory 分数割 (Gomory’s Fractional Cuts)、Gomory 混合整数割 (Gomory Mixed Integer cuts)、混合整数圆整割 (Mixed Integer Rounding Cuts)、背包覆盖割 (Knapsack Cover Cuts)、升维投影割 (Lift-and-Project Cuts) 等.

一般地, 我们可以通过求解当前线性松弛解的分离问题来产生割平面. 对于松弛解 $\bar{x} \in \mathbb{R}^n$ 和多面体 $P = \text{conv}(X)$, 其中 X 为整数规划问题的可行点集合 X, 其分离问题是要么找到分离点 $\bar{x}$ 和多面体 $P = \text{conv}(X)$ 的超平面 (对应于不等式 $\sum_{i\in[n]} \alpha_i x_i \leq 1$), 要么证明 $\bar{x} \in P$. 上述的分离问题可以建模成如下的数学优化问题.

$$\begin{aligned} v = \max_{\alpha,\beta}\ & \alpha^T \bar{x} - \beta \\ \text{s.t. } & \alpha^T x \leq \beta, \forall x \in P, \\ & \alpha \in \Pi, \end{aligned} \tag{1-1}$$

其中 Π 是包含原点的凸紧集, $\beta, \alpha \in \mathbb{R}^n$ 为变量, 也是所求有效不等式的右端项和系数. 目标函数值 v 表示 $\bar{x}$ 的违背值, 约束条件 $\alpha^T x \leq \beta, \forall x \in P$ 表示多面体 P 中的每个可行点都满足不等式 $\alpha^T x \leq \beta$. 当最优值 $v \leq 0$ 时, 证明 $\bar{x} \in P$. 否则, 当最优值 $v > 0$ 时, 问题 (1-1) 的最优解 α^* 给出对于当前线性松弛解 $\bar{x}$ 最违背的有效不等式.

精确分离方法是将分离问题建模成数学优化问题, 从而对当前线性松弛解或证明其属于整数可行集, 则得到整数最优解, 或计算得到分离不等式, 将其加入线性松弛问题, 重复这一过程, 最后得到原整数规划问题的最优解. Boyd [31–33] 首次考虑通过求解背包多面体的分离问题来生成割平面. Yan [34] 等人进一步考虑混合整数背包集, Kaparis [35] 将其扩展到 0-1 变量的情况, Fukasawa [36–38] 则扩展到混合整数规划问题. Boccia [35] 描述了另一种使用极性概念的精确分离方法. 随后, 许多学者还针对其它类型的优化问题研究精确分离问题, 如背包集 [39, 40]、集合覆盖问题 [41]、广义分配问题 [42, 43]、旅行商问题的的局部割平面 [44] 等. Vasilyev [45] 针对背包约束设计了精确分离算法的一般流程, 主要分为预处理、行生成、系数规范化和序列升维等四步, 最后计算得到多面体的刻面定义不等式. Chen [46] 将不可分有容量限制的网络设计问题中的不可分流边集合作为问题的子结构, 研究了它的精确分离算法, 并提出了一种减少分离算法迭代次数的加速算法.

已有的工作已经建立了精确分离算法的基本框架, 但主要还是针对一维 0-1 背包多面体计算分离问题, 而为了进一步考虑多维 0-1 背包多面体, 我们将设计带有聚合方法的分离算法.

1.4 本论文的主要工作

本论文的工作主要围绕两类特殊背包多面体的分离问题展开, 一是研究多维 0-1 背包多面体的聚合分离算法, 二是针对含 GUB 约束的背包问题设计精确分离算法.

本论文安排如下. 在第 2 章, 我们先介绍一般 0-1 背包多面体的精确分离算法. 目前, 关于 0-1 背包多面体的精确分离算法主要是针对一维 0-1 背包多面体设计的. 考虑由一条背包约束定义的多面体与线性松弛解的分离问题, 将其建模为最优化问题. 在此基础上, 设计精确分离算法过程, 主要分为以下三步: 预处理、行生成和序列升维.

在第 3 章, 我们研究了多维 0-1 背包多面体的聚合分离算法. 对于含多条约束的背包多面体, 只考虑其中一条约束相关的一维背包多面体的分离问题可能不能找到割平面. 所以我们在现有的精确分离算法框架下, 将聚合方法运用到行生成过程中以此求得更强的割平面, 并给出多维 0-1 背包多面体的聚合分离算法过程. 数值实验结果表明, 聚合分离算可以有效提升问题的求解效果.

在第 4 章, 我们研究了含 GUB 约束的背包多面体的精确分离算法. 将线性

松弛解与含 GUB 约束的背包多面体的分离问题建模成优化问题, 求解该问题的精确分离算法主要分为以下三步: 预处理、行生成和序列升维. 为了进一步考虑 GUB 约束的特性, 我们改进预处理和序列升维的步骤. 并且在行生成过程中给出求解含 GUB 约束的背包子问题的动态规划算法. 在此基础上, 改进以占优表储存的稀疏动态规划算法来求解序列升维系数. 最后, 通过对带 GUB 约束的装箱问题测试集进行数值实验, 说明了该精确分离算法的有效性.

最后, 在第 5 章中, 我们对本论文的工作进行总结, 并在此基础上讨论未来的工作方向.

第 2 章　0-1 背包多面体的精确分离算法

2.1　引言

精确分离方法是将多面体 P 和线性松弛解 $\bar{x}$ 的分离问题建模成最优化问题, 要么计算得到对多面体 P 有效的割平面割去 $\bar{x}$, 要么证明 P 包含 $\bar{x}$. 精确分离算法首先由 Boyd 在 1992 年提出 [31]. Kaparis 等人 [35] 则研究了一维 0-1 背包多面体的的精确分离算法并且给出了 MIPLIB 实例的计算结果. Avella 等人 [43] 的研究表明对一维 0-1 背包多面体的精确分离方法是解决广义分配问题的有效方法. 一般 0-1 背包多面体的精确分离算法主要考虑对由一条背包约束定义的多面体计算分离问题, 所以接下来我们将以一维 0-1 背包多面体为例对精确分离算法进行说明.

在本章中, 我们考虑一维 0-1 背包多面体的精确分离算法. 回顾第 1 章, 有一维 0-1 背包集合和 0-1 背包多面体如下,

$$K = \left\{ x \in \{0,1\}^n : \sum_{i\in[n]} a_i x_i \leq b \right\},$$

$$P = \mathrm{conv}(K),$$

其中 $[n] = \{1, ..., n\}$, $c \in \mathbb{R}^n_+$, $a \in \mathbb{Z}^n_+$, $b \in \mathbb{Z}_+$. 不失一般性, 假设 $a_i \leq b, \forall i \in [n]$ 并且 $\sum_{i\in[n]} a_i > b$.

一般地, 我们先计算原问题的线性松弛问题得到松弛解, 定义集合 K 的线性松弛集合如下,

$$K_{LP} = \left\{ x \in [0,1]^n : \sum_{i\in[n]} a_i x_i \leq b \right\}.$$

在本章节中, 我们将在 2.2 节先介绍一维 0-1 背包多面体的分离问题, 将其建模为最优化问题, 并根据已有研究进行简化. 在此基础上, 2.3 节中我们给出 0-1 背包多面体的精确分离算法过程, 主要分为预处理、行生成和序列升维三步.

2.2 一维 0-1 背包多面体的分离问题

给定点 $\bar{x} \in [0,1]^n$, 一维 0-1 背包多面体 P 的分离问题是找到超平面严格分离多面体 P 和点 $\bar{x}$, 对应于找到有效不等式 $\alpha^T x \leq \beta$, 使得

$$\alpha^T x \leq \beta, \forall x \in P,$$

并且

$$\alpha^T \bar{x}_i > \beta,$$

或者证明 $\bar{x} \in P$. 对于当前线性松弛解 $\bar{x} \in K_{LP}$, 求解其分离问题等价于求解下述优化问题.

$$\begin{aligned} \upsilon = \max_{\alpha,\beta} \ & \alpha^T \bar{x} \\ \text{s. t. } & \alpha^T x \leq \beta, \forall x \in P, \\ & \alpha \in \Pi, \end{aligned} \tag{2-1}$$

其中 Π 是包含原点的凸紧集, $\beta \in \mathbb{R}$, $\alpha \in \mathbb{R}^n$ 为变量. 约束条件 $\alpha^T x \leq \beta, \forall x \in P$ 表示多面体 P 中的每个可行点都满足不等式 $\alpha^T x \leq \beta$. 目标函数值 υ 表示不等式代入 $\bar{x}$ 的左端值, 优化目标就是最大化左端值. 如果 $\upsilon \leq \beta$, 说明左端值最大时也不超过 β, 就证明了 $\bar{x} \in P$. 否则, 根据问题的最优解 α^*, 我们得到有效不等式 $\alpha^{*T} x_i \leq \beta$, 对于多面体 P, 其中的每个可行点都满足该不等式, 而 $\bar{x}$ 违背该不等式. 我们称该不等式严格分离当前松弛解 $\bar{x}$ 和多面体 P.

为了问题 (2-1) 是可行且有界的, 我们不妨假设 β 是一个固定值. 由文献 [45] 可知, 不同的范数会影响割的效果, 其中线性范数能最大化违背值 υ 和右端项 β 之比, 与 L_1 和 L_∞ 范数相比可以得到更好的结果. 此外, Balas [47] 证明了非平凡的刻面定义不等式系数是非负的且 $\beta > 0$, 所以我们对问题 (2-1) 规范化, 令 $\beta = 1$. 因此, 接下来我们只考虑线性范数定义下的分离问题, 简化问题 (2-1) 为问题 (2-2).

$$\begin{aligned} \upsilon = \max_{\alpha} \ & \sum_{i \in [n]} \alpha_i \bar{x}_i \\ \text{s. t. } & \sum_{i \in [n]} \alpha_i x_i \leq 1, \forall x \in P. \end{aligned} \tag{2-2}$$

因为 $\forall i \in [n]$ 有假设 $a_i \leq b$, 所以 $e_i \in P$, 则有 $\alpha_i \leq 1$. 因此, 问题 (2-2) 中 $0 \leq \alpha_i \leq 1, \forall i \in [n]$.

2.3　精确分离算法

由于多面体 P 中有许多点, 所以分离问题 (2-2) 也包含大量约束. 一方面, 要列举多面体 P 每一个点是非常困难的, 另一方面, 随着分离问题 (2-2) 中约束个数的增加, 求解时间将呈指数级增长. 文献 [45] 中将精确分离算法主要分为以下三步.

(1) **预处理**. 对原多面体 P 进行预处理, 固定松弛解为 0 或 1 的变量, 得到一个低维的多面体.

(2) **行生成**. 通过行生成的方式, 计算低维多面体上的分离问题找到一个违背当前松弛解的有效不等式, 或者证明这样的不等式不存在.

(3) **序列升维**. 在预处理过程中被固定为 0 或 1 的变量需要相应被序列升维, 从而得到一个对原多面体 P 更强的有效不等式.

2.3.1　预处理

首先, 为了减小问题规模和减少计算时间, 在求解分离问题前我们对原多面体 P 进行预处理来得到低维多面体. 给定点 $\bar{x} \in P$ 和多面体 P, 固定松弛解中取值为 0 或 1 的变量, 得到多面体 P 的低维多面体 $P(\bar{x})$ 如下.

$$P(\bar{x}) = \mathrm{conv}(K(\bar{x})),$$

其中

$$K(\bar{x}) = \left\{ x \in \{0,1\}^{\bar{N}} : \sum_{i \in \bar{N}} a_i x_i \le \bar{b} \right\},$$

$\bar{N} = \{i \in [n] : 0 < \bar{x}_i < 1\}$, 且 $\bar{b} = b - \sum_{i \in [n], \bar{x}_i = 1} a_i$. 对降维后的多面体 $P(\bar{x})$ 建立分离问题.

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i \in \bar{N}} \alpha_i \bar{x}_i \\ \text{s. t.} \quad & \sum_{i \in \bar{N}} \alpha_i x_i \le 1, \forall x \in P(\bar{x}) \\ & 0 \le \alpha_i \le 1, i \in \bar{N}. \end{aligned} \tag{2-1}$$

在计算分离问题前, 我们先固定一部分变量得到低维多面体是因为相较于原多面体 P 上的分离问题 (2-2), 问题 (2-1) 的变量和约束规模都更小, 可以减少问题的求解时间. 并且, 低维多面体 $P(\bar{x})$ 的任意有效不等式可以通过序列升维过程转化为原多面体 P 的有效不等式. 进一步, 如果低维多面体 $P(\bar{x})$ 的有效不等式是刻面定义不等式, 则得到的原多面体 P 的有效不等式也是刻面定义不等式.

2.3.2 行生成

即使是降维后的分离问题也可能有指数多条约束, 所以我们采用行生成方法来求解分离问题 (2-1). 行生成方法是先计算带有部分约束的分离问题, 要么证明点 $\bar{x} \in P(\bar{x})$, 要么得到一条不等式再带入降维后的原背包约束验证该不等式是否有效, 若存在背包可行点违背该不等式, 则将该点作为新的约束添加到分离问题再求解, 重复以上过程直到求得最终解. 我们称带有部分约束的分离问题为局部分离问题, 形式如下.

$$\begin{aligned} v(U) = \max_{\alpha} \ & \sum_{i \in \bar{N}} \alpha_i \bar{x}_i \\ \text{s. t.} \ & \sum_{i \in \bar{N}} \alpha_i x_i \leq 1, \forall x \in U, \\ & 0 \leq \alpha_i \leq 1, \forall i \in \bar{N}, \end{aligned} \tag{2-2}$$

其中 $U \subseteq P(\bar{x})$. 一般来说, 我们初始化 $U = \varnothing$, 并且为了保证局部分离问题 2-2 的有界性, 如 2.2 节所证, 令 $0 \leq \alpha_i \leq 1, \forall i \in \bar{N}$.

如果最优值 $v(U) > 1$, 则找到分离当前松弛解 $\bar{x}$ 和低维多面体 $P(\bar{x})$ 的有效不等式

$$\sum_{i \in \bar{N}} \alpha_i^* x_i \leq 1, \tag{2-3}$$

其中 α^* 是局部分离问题的最优解. 如果 $v(U) \leq 1$, 则证明 $\bar{x} \in P(\bar{x})$. 在得到不等式后继续验证其对多面体 P 是否有效. 求解背包子问题

$$z = \max_{x} \left\{ \sum_{i \in \bar{N}} \alpha_i^* x_i \ : \ \sum_{i \in \bar{N}} a_i x_i \leq b, x_i \in \{0,1\} \ \forall i \in \bar{N} \right\}, \tag{2-4}$$

如果 $z \leq 1$, 则不等式 (2-3) 对多面体 P 有效. 否则, 最优解对应的背包可行点违背了不等式 (2-3), 要将其加入子集合 U 并继续求解局部分离问题. 按这样的流程不断进行迭代, 最终找到一个分离当前松弛解的有效不等式, 或者证明对于多面体 P 来说这样的不等式不存在. 我们将用例 2.1 演示上述计算过程, 并在算法 1 中给出该分离问题的行生成过程.

例 2.1. 考虑 0-1 背包多面体:

$$P = \text{conv}(K),$$

其中

$$K = \{x \in \{0,1\}^3 \ : \ x_1 + x_2 + 2x_3 \leq 2\}.$$

假设当前线性松弛解为 $\bar{x}=(\frac{1}{2},\frac{1}{2},\frac{1}{2})$, 从 $U=\varnothing$ 开始, 有局部分离问题

$$v(U)=\max\ \frac{1}{2}\alpha_1+\frac{1}{2}\alpha_2+\frac{1}{2}\alpha_3 \tag{2-5a}$$

$$\text{s.t. } 0\le\alpha_i\le 1, i\in[3]. \tag{2-5b}$$

其中目标函数 (2-5a) 的系数对应松弛解 $\bar{x}$, 优化目标是最大化违背值. 约束 (2-5b) 为初始化界约束. 求解该优化问题得到最优值为 $v(U)=\frac{3}{2}$, 最优解为 $\alpha^*=(1,1,1)$. 此时 $v(U)>1$, 需要求解如下背包子问题来验证其对应的不等式是否有效.

$$z=\max_x\left\{x_1+x_2+x_3:\ x_1+x_2+2x_3\le 2, x_i\in\{0,1\}\ \forall i\in[3]\right\}, \tag{2-6}$$

可以得到最优值 $z=2$, 最优解为 $x^*=(1,1,0)$. 此时最优值 $z>1$, 所以将最优解 x^* 作为约束加入到局部分离问题中, 于是局部分离问题更新如下.

$$v(U)=\max\ \frac{1}{2}\alpha_1+\frac{1}{2}\alpha_2+\frac{1}{2}\alpha_3 \tag{2-7a}$$

$$\text{s.t. } \alpha_1+\alpha_2\le 1, \tag{2-7b}$$

$$0\le\alpha_i\le 1, i\in[3]. \tag{2-7c}$$

其中约束 (2-7b) 对应于不等式违背的背包可行点 $x^*=(1,1,0)$. 再求解该优化问题, 得到最优值为 $v(U)=1$, 此时无法通过这条约束来生成分离松弛解的有效不等式.

算法 1 一维 0-1 背包多面体的分离问题的行生成过程

输入: 松弛解 $\bar{x}$, 集合 K 和子集合 $U=\varnothing$;

输出: 一个被 $\bar{x}$ 违背的有效不等式 $\sum_{i\in\bar{N}}\alpha_i^*x_i\le 1$, 或证明这样的不等式不存在;

步骤 1: 求解局部分离问题 (2-2) 得到最优解 α^* 和最优值 $v(U)$;

if $v(U)>1$ **then**

　步骤 2: 求解背包子问题 (2-3) 得到最优解 x^* 和最优值 z^*;

　if $z^*>1$ **then**

　　将违背点 x^* 加入子集合 U 并转入步骤 1;

　else

　　算法终止, 并且返回有效不等式 $\sum_{i\in\bar{N}}\alpha_i^*x_i\le 1$;

　end if

else

　算法终止, 并且证明 $\bar{x}\in P(\bar{x})$;

end if

在算法 1 中, 步骤 1 求解局部分离问题 (2-2) 得到最优解 α^* 和最优值 $\upsilon(U)$. 如果 $\upsilon(U) \leq 1$, 则证明 $\bar{x} \in P(\bar{x})$. 否则将进入步骤 2, 用改良后的 MINKNAP 算法 [48] 求解目标函数值为实数的背包子问题得到最优解 x^* 和最优值 z^*. 如果 $z^* \leq 1$, 则我们得到违背 $\bar{x}$ 但对集合 $P(\bar{x})$ 有效的不等式 $\sum_{i\in\bar{N}} \alpha_i^* x_i \leq 1$. 不然, 将违背的背包可行点 x^* 对应的约束加入局部分离问题 (2-2) 中并再次转入步骤 1. 如上述一样, 交替求解局部分离问题和背包子问题找到一个违背的有效不等式, 或者证明这样的有效不等式不存在.

2.3.3 序列升维

在得到对低维多面体 $P(\bar{x})$ 有效的分离不等式后, 为了避免数值不稳定, 先将不等式的系数化为整数, 并且重新计算右端项 β 以保证有效性.

在这之后, 考虑预处理阶段固定的变量按一定顺序相应进行序列升维, 从而得到原多面体 P 的有效不等式. 文献 [43] 分析精确分离算法过程中最耗时的部分正是序列升维. 假设计算得到的有效不等式 (2-3) 中已经有前 $k-1$ 个变量, 即有不等式

$$\sum_{i\in[k-1]} \alpha_i x_i \leq \beta.$$

我们要升维第 k 个变量 x_k, 根据升维定理 1.1, 有如下两种情形. 记

$$W(z) = \max_x \left\{ \sum_{i\in[k-1]} \alpha_i x_i : \sum_{i\in[k-1]} a_i x_i \leq z, x_i \in \{0,1\}, \forall i \in [k-1] \right\},$$

其中 $z \in \mathbb{Z}^+$, 且 $z \leq \bar{b}$.

1) 如果 $\bar{x}_k = 0$, 那么升维后的不等式是

$$\sum_{i\in[k-1]} \alpha_i x_i + \alpha_k x_k \leq \beta,$$

其中 $\alpha_k = \beta - W(\bar{b} - a_k)$.

2) 如果 $\bar{x}_k = 1$, 那么升维后的不等式是

$$\sum_{i\in[k-1]} \alpha_i x_i + \alpha_k (x_k - 1) \leq \beta,$$

其中 $\alpha_k = W(\bar{b} + a_k) - \beta$. 并且, 因为原来 $\bar{x}_k = 1$, 在升维后需要更新 $\bar{b} \to \bar{b} + a_k$.

可以看到在上述升维过程中, 需要求解若干个 0-1 背包子问题 $W(z)$, 但与行生成过程相比, 这里只需要求得子问题的最优值, 并不需要回溯得到对应的最优解. 此外, 这些背包子问题只有总容量不一样, 背包约束中的其它系数均一样. 因

此, 我们可以直接用 0-1 背包问题的动态规划算法来求解这些升维过程中的子问题. 该算法的时间复杂度为 $O(nb)$, 空间复杂度为 $O(b)$. 对 $\bar{N}$ 中各个变量计算升维系数后, 我们得到了原多面体 $\boldsymbol{P}$ 的刻面定义不等式.

第 3 章　多维背包多面体的聚合分离算法

3.1　引言

在前文所述的精确分离过程中, 我们都是基于一维背包多面体计算分离问题和进行序列升维. 但是一般问题往往含有多条约束, 而当对于一条背包约束无法分离当前松弛解的时候, 可以考虑聚合若干条背包约束生成新的一条背包约束来计算割平面.

割平面方法是现在求解混合整数规划最主要的算法之一, 而其中一些重要的割平面是聚合割平面, 如 Chvátal–Gomory (CG) 割、升维背包覆盖割、权重不等式等. 一般来说, 聚合割平面是通过聚合问题的原始约束获得一个隐含的不等式约束, 再基于这条聚合约束生成一个对其定义的整数凸包有效的割平面. Marchand [9] 通过简单聚合约束来生成 MIR 割平面. Fukasawa [36] 从数值实验的角度分析了各类聚合割平面. Bodur [49] 研究了装填问题的 Chvátal–Gomory (CG) 割平面和一般聚合割平面, 并通过数值实验说明由聚合约束产生的割平面比由单个约束产生的割平面更强.

在本章节中, 我们主要考虑多维 0-1 背包多面体.

$$P_{MK} = \mathrm{conv}(MK)$$

其中 $MK = \left\{x \in \{0,1\}^n : \sum_{i\in[n]} a_{ij}x_i \le b_j, j \in [m]\right\}$, $[n] = \{1,...,n\}$, $c \in \mathbb{R}^n_+$, $a \in \mathbb{Z}^{n\times m}_+$, $b \in \mathbb{Z}^m_+$. 不失一般性, 假设对于每条约束 $a_{ij} \le b_j$, $\forall i \in [n]$ 并且 $\sum_{i\in[n]} a_{ij} > b_j$.

目前, 精确分离方法基本是针对问题中某一条原始背包约束进行设计的, 但是对于多维 0-1 背包问题仅考虑一条背包约束有可能不能生成割平面. 由前文的相关研究可以看出, 通过聚合约束生成的割平面有良好的数值表现, 所以我们考虑将原问题的若干条背包约束聚合产生一条新的背包约束, 设计聚合分离算法, 从而尝试找到能分离当前松弛解的割平面. 一方面, 聚合多条约束可以有效利用这些约束的信息, 在计算分离问题的行生成过程中一定程度上避免加入原问题描述的多面体中并不包含的点. 另一方面, 对于单条背包约束目前已有较为完善的精确分离算法框架, 所以我们可以直接运用其中的部分方法, 比如序列升维的计算过程, 也可以根据聚合约束优化行生成的过程来计算割平面.

在本章节中, 我们将说明某些情况下仅考虑一条背包约束不能生成割平面, 因此需要考虑新的方法来应用分离算法. 根据之前关于聚合割平面研究的启发, 将原约束与其它背包约束聚合, 再计算对应的分离问题来生成割平面. 接下来, 我们设计带有聚合方法的分离算法, 给出聚合约束的具体过程. 最后, 为了验证聚合分离算法的有效性, 我们进行了相关数值实验.

本节的安排如下. 3.2 节基于 0-1 背包多面体的分离问题, 举例说明用以往的精确分离算法计算多约束优化问题的割平面时遇到的问题, 再在此基础上讨论聚合约束的可行性. 3.3 节给分离算法中聚合技术的实现细节. 最后, 3.4 节展示聚合分离算法的计算效果并进行小结.

3.2 多维 0-1 背包多面体的分离问题

目前, 关于 0-1 背包多面体的精确分离算法主要是对于其中某一条背包约束进行的. 在第 2 章中, 我们已经详细介绍了一维 0-1 背包多面体的分离问题和精确分离算法过程. 但是, 在计算多维 0-1 背包多面体的分离问题的行生成过程中, 只考虑一条约束可能会将一些对原问题多面体不可行但是对该约束可行的点加入局部分离问题中进行求解, 最终无法根据该条约束得到分离当前松弛解的割平面. 下面我们将用一个例子说明这种可能性.

例 3.1. 考虑多约束的 0-1 背包多面体:

$$P_{MK} = \mathrm{conv}(MK),$$

其中

$$MK = \{x \in \{0,1\}^3 : x_1 + x_2 + 2x_3 \le 2,\ x_1 + 2x_2 + x_3 \le 2\}.$$

假设当前线性松弛解为 $\bar{x} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, 先考虑基于约束 $x_1 + x_2 + 2x_3 \le 2$ 的分离问题, 根据例 2.1, 可以得到局部分离问题

$$\max\ \frac{1}{2}\alpha_1 + \frac{1}{2}\alpha_2 + \frac{1}{2}\alpha_3 \tag{3-1a}$$

$$\text{s.t. } \alpha_1 + \alpha_2 \le 1, \tag{3-1b}$$

$$0 \le \alpha_i \le 1, i \in [3], \tag{3-1c}$$

其中目标函数 (3-1a) 的系数对应松弛解, 优化目标是最大化违背值. 约束 (3-1b) 对应原背包问题中约束 $x_1 + x_2 + 2x_3 \le 2$ 的一个可行点. 求解该优化问题得到最优值为 1, 此时无法通过这条约束来生成分离松弛解的有效不等式.

对以上例子进行分析, 可以发现问题 (3-1) 并不完全对应原背包多面体 P_{MK} 的分离问题. 约束 (3-1b) 代表的点 $(1,1,0)$ 违背约束 $x_1+2x_2+x_3\leq 2$, 并不是原背包问题的可行点. 如果在行生成的过程中也考虑约束 $x_1+2x_2+x_3\leq 2$, 则可以避免加入一些非可行点, 这样可以让分离问题引入更少的约束, 得到更大的违背值, 从而更有可能找到分离当前松弛解的有效不等式.

首先, 考虑有两条背包约束

$$a^Tx\leq b \tag{3-2}$$

和

$$c^Tx\leq d, \tag{3-3}$$

聚合这两条约束得到

$$(a+kc)^Tx\leq b+kd, \tag{3-4}$$

其中 $k\in\mathbb{Z}^+$ 是聚合系数. 对于聚合系数我们有如下结论.

命题 3.1. *考虑两条背包约束 $a^Tx\leq b$ 和 $c^Tx\leq d$, 若分离问题中某个点 x^* 满足 $a^Tx^*+s=b$ 且 $c^Tx^*=d+t$, 其中 $s\geq 0$, $t>0$, 令*

$$k>\frac{s}{t},$$

则聚合约束 (3-4) 对应的分离问题中可以删去点 x^*.

下面我们将继续用例子 3.1 演示分离问题行生成过程中的聚合方法.

例 3.2. 考虑约束 (3-1b) 代表的点 $(1,1,0)$, 计算得到 $s=0$, $t=1$, 所以 $k>0$. 取 $k=1$, 聚合约束 $x_1+x_2+2x_3\leq 2$ 和 $x_1+2x_2+x_3\leq 2$ 得到聚合约束

$$2x_1+3x_2+3x_3\leq 4.$$

则局部分离问题 (3-1) 可以去除约束 (3-1b), 得到新的优化问题如下.

$$\begin{aligned}\max\ &\frac{1}{2}\alpha_1+\frac{1}{2}\alpha_2+\frac{1}{2}\alpha_3\\ \text{s.t. }&0\leq\alpha_i\leq 1, i\in[3].\end{aligned} \tag{3-5}$$

求解以上问题可以得到最优解 $(1,1,1)$, 此时 $v(U)>1$, 需要求解新约束定义的背包子问题来验证其对应的不等式是否有效.

$$z=\max_x\left\{x_1+x_2+x_3:2x_1+3x_2+3x_3\leq 4, x_i\in\{0,1\}\ \forall i\in[3]\right\}, \tag{3-6}$$

可以得到最优值 $z=1$. 于是, 我们得到有效不等式 $x_1+x_2+x_3\leq 1$.

3.3 聚合分离算法

在本节中, 我们将介绍聚合分离算法, 主要分为预处理、行生成、序列升维等三步. 基于 3.2 节中对聚合背包约束的讨论, 我们将在行生成的过程中引入聚合方法.

(1) **预处理**. 在求解分离问题前对原问题进行预处理, 固定松弛解为 0 或 1 的变量, 得到一个低维的多面体.

(2) **行生成**. 通过行生成的方式, 对于问题中的某一条背包约束计算低维多面体上的分离问题找到一个违背当前松弛解的有效不等式, 或者证明这样的不等式不存在, 再选取一条背包约束来产生新的聚合约束, 计算新约束的分离问题来试图找到分离不等式.

(3) **序列升维**. 在预处理过程中被固定为 0 或 1 的变量需要相应被序列升维, 从而得到一个对原多面体更强的有效不等式.

3.3.1 预处理

给定线性松弛解 $\bar{x}$, 我们先固定松弛解中取值为 0 或 1 的变量. 实际计算中, 先记录松弛解中取值为 0 或 1 的变量下标, 每条背包约束在后续行生成、聚合以及序列升维过程中将根据记录的变量下标进行相应的计算.

3.3.2 带聚合方法的行生成

下面回顾一维 0-1 背包多面体的分离问题的行生成过程, 并在此基础上说明行生成过程中的聚合方法. 首先, 我们先考虑原问题中的某一条背包约束 $ax \leq b$ 描述的多面体 P. 如第 2 章中所述, 从计算局部分离问题开始, 形式如下.

$$\begin{aligned} v(U) = \max_{\alpha} \ & \sum_{i \in \bar{N}} \alpha_i \bar{x}_i \\ \text{s. t.} \ & \sum_{i \in \bar{N}} \alpha_i x_i \leq 1, \forall x \in U, \\ & 0 \leq \alpha_i \leq 1, \forall i \in \bar{N}, \end{aligned} \tag{3-1}$$

其中 $\bar{N} = \{i \in [n] : 0 < \bar{x}_i < 1\}$ 为预处理过程中没有固定的部分变量, $P(\bar{x}) = \text{conv}(\left\{ x \in \{0,1\}^{\bar{N}} : \sum_{i \in \bar{N}} a_i x_i \leq \bar{b} \right\})$ 为多面体 P 对应的低维多面体, $U \subseteq P(\bar{x})$. 如果最优值 $v(U) > 1$, 则找到分离松弛解 $\bar{x}$ 和低维多面体 $P(\bar{x})$ 的有效不等式

$$\sum_{i \in \bar{N}} \alpha_i^* x_i \leq 1, \tag{3-2}$$

其中 α^* 是问题 3-1 的最优解. 如果 $\upsilon(U) \leq 1$, 则证明 $\bar{x} \in P(\bar{x})$, 说明当只考虑约束 $ax \leq b$ 时, 无法得到分离当前松弛解的有效不等式. 在得到不等式后继续验证其对当前背包约束描述的多面体是否有效. 求解背包子问题

$$z = \max_{x} \left\{ \sum_{i \in \bar{N}} \alpha_i^* x_i : \sum_{i \in \bar{N}} a_i x_i \leq b, x_i \in \{0,1\}\ \forall i \in \bar{N} \right\}, \tag{3-3}$$

如果 $z \leq 1$, 则不等式 (3-2) 对当前背包约束 $ax \leq b$ 描述的多面体 P 有效. 否则, 最优解对应的背包可行点违背了不等式 (3-2), 要将其加入子集合 U 并继续求解局部分离问题. 按这样的流程不断进行迭代, 最终找到一个分离当前松弛解的有效不等式, 或者证明对于当前背包约束描述的多面体来说这样的不等式不存在.

正如前文所述, 对于多维 0-1 背包多面体, 若只对其中一条背包约束计算分离问题很有可能不能找到分离不等式, 因此我们需要将这条约束与其它约束聚合, 得到新的约束再进一步更新分离问题. 假设目前考虑的背包约束是 $a^T x \leq b$, 如果对于当前约束不能找到分离松弛解的有效不等式, 那么我们将结合聚合方法来进一步计算分离问题.

首先, 选取一条背包约束 $c^T x \leq d$, 一般来说选择最“违背”的约束, 比如可以选择违背当前局部分离问题中背包可行点最多的约束, 也可以选择违背值最大的约束. 接下来, 根据 3.2 节的理论分析, 将两条约束进行聚合得到新约束 $(a + kc)^T x \leq b + kd$, 其中

$$k > \max \left\{ \frac{s_i}{t_i} : a^T x^i + s_i = b, c^T x^i = d + t_i, i \in V \right\},$$

其中 $V = \{x^i : c^T x^i > d, x^i \in U\}$. 为了让新约束也保持背包约束的结构, 取 k 为整数, 这样可以得到一条聚合背包约束. 由 k 的定义, 有 V 中各点违背聚合约束, 所以在继续行生成过程前, 我们可以先更新局部分离问题中的 U, 即删去 V 包含的点. 得到聚合约束后, 将行生成过程中的背包子问题更新如下,

$$z = \max_{x} \left\{ \sum_{i \in \bar{N}} \alpha_i^* x_i : \sum_{i \in \bar{N}} (a_i + kc_i) x_i \leq b_i + kd_i, x_i \in \{0,1\}\ \forall i \in \bar{N} \right\}. \tag{3-4}$$

这样则称完成一轮聚合, 若对于新约束仍不能找到分离当前松弛解的有效不等式, 则可以考虑选取另一条约束再进行一轮聚合. 但是由聚合系数 k 的选取可知, 每次聚合得到的新背包约束的系数都会变大, 使得行生成过程中局部分离问题和背包子问题的计算时间增加, 因此需要控制聚合的次数. 我们在算法 2 中给出带聚合方法的行生成过程.

算法 2 带聚合方法的行生成过程

输入: 松弛解 $\bar{x}$, 集合 MK 以及其中一条背包约束 $a^T x \leq b$ 和子集合 $U = \emptyset$;

输出: 一个被 $\bar{x}$ 违背的有效不等式 $\sum_{i \in \bar{N}} \alpha_i^* x_i \leq 1$, 或证明这样的不等式不存在;

步骤 1: 求解局部分离问题 (3-1) 得到最优解 α^* 和最优值 $\upsilon(U)$;

if $\upsilon(U) > 1$ **then**

步骤 2: 求解背包子问题 (3-3) 或 (3-4) 得到最优解 x^* 和最优值 z^*;

if $z^* > 1$ **then**

将违背点 x^* 加入子集合 U 并转入步骤 1;

else

算法终止, 并且返回有效不等式 $\sum_{i \in \bar{N}} \alpha_i^* x_i \leq 1$;

end if

else

选取 MK 中另一条背包约束 $c^T x \leq d$, 并删去 U 中违背该约束的点;

将其与当前的约束聚合得到新的约束, 并更新背包子问题的约束条件转入步骤 2;

end if

在算法 2 中, 先考虑某一条约束 $a^T x \leq b$ 的分离问题. 步骤 1 调用求解器 CPLEX 计算其局部分离问题得到最优解 α^* 和最优值 $\upsilon(U)$. 如果 $\upsilon(U) > 1$, 则进入步骤 2, 用 bouknap 算法 [50] 求解背包子问题得到最优解 x^* 和最优值 z^*. 如果 $z^* \leq 1$, 则可以根据当前约束得到违背 $\bar{x}$ 但对低维多面体有效的不等式 $\sum_{i \in \bar{N}} \alpha_i^* x_i \leq 1$. 否则, 将违背点 x^* 加入子集合 U 后再转入步骤 1. 如果 $\upsilon(U) \leq 1$, 则说明对于当前约束不能找到分离松弛解 $\bar{x}$ 的有效不等式, 我们选取另一条约束 $c^T x \leq d$, 并且删去子集合 U 中违背该约束的点, 即删去局部分离问题中对应的约束. 下一步, 将选取的约束与当前约束进行聚合得到新的约束 $(a + kc)^T x \leq b + kd$, 并且更新背包子问题的约束为新约束, 再继续对局部分离问题用行生成的方式进行计算. 最后, 在一定的聚合次数内重复这一过程直到找到割平面, 否则就终止算法.

3.3.3 序列升维

在得到对低维多面体有效的分离不等式后, 参考文献 [51], 我们先将不等式的系数化为整数, 并重新计算右端项以保证有效性. 在这之后, 需要结合最终的聚

合约束按照预处理阶段固定的变量相应进行序列升维, 从而得到原问题的有效不等式. 根据聚合过程, 最终的聚合约束也是一条背包约束, 不妨假设对背包约束 $a^T x \le b$ 进行序列升维. 并且假设计算得到的有效不等式 (3-2) 中已经有前 $k-1$ 个变量, 即有不等式

$$\sum_{i\in[k-1]} \alpha_i x_i \le \beta,$$

我们要升维第 k 个变量 x_k, 有如下两种情形. 记

$$W(z) = \max_x \left\{ \sum_{i\in[k-1]} \alpha_i x_i : \sum_{i\in[k-1]} a_i x_i \le z, x_i \in \{0,1\}\ \forall i \in [k-1] \right\}.$$

1) 如果 $\bar{x}_k = 0$, 那么升维后的不等式是

$$\sum_{i\in[k-1]} \alpha_i x_i + \alpha_k x_k \le \beta,$$

其中 $\alpha_k = \beta - W(\bar{b} - a_k)$.

2) 如果 $\bar{x}_k = 1$, 那么升维后的不等式是

$$\sum_{i\in[k-1]} \alpha_i x_i + \alpha_k (x_k - 1) \le \beta,$$

其中 $\alpha_k = W(\bar{b} + a_k) - \beta$, 并 $\bar{b} \to \bar{b} + a_k$.

可以看到在升维过程中, 需要求解若干个 0-1 背包子问题. 文献 [45] 中建议用 0-1 背包问题的动态规划算法来计算. 但是一般的动态规划表有大部分状态是重复的, 只有在部分点会发生变化. 因此, 参考文献 [46], 我们用以占优表储存的稀疏动态规划表来求解这些升维过程中的子问题.

回顾 0-1 背包问题动态规划的迭代公式,

$$f(k+1,z) = \begin{cases} f(k,z), & z < a_{k+1}, \\ \max\{f(k,z), c_{k+1} + f(k, z-a_{k+1})\}, & otherwise. \end{cases}$$

对于第 $k+1$ 个变量, 只有当 $a_{k+1} \le z$ 时才能考虑是否可以取 $x_{k+1} = 1$. 如果 $x_{k+1} = 0$, 则容量 z 不变, 对应的最优值为 $f(k,z)$. 若 $x_{k+1} = 1$, 则容量要减少 a_{k+1}, 对应的最优值为 $f(k, z-a_{k+1})$ 加上该变量的目标函数值 c_{k+1}. 由于优化目标是最大化放入背包的物品总收益, 所以我们选择这两种情况下最大的总收益 $\max\{f(k,z), c_{k+1} + f(k, z-a_{k+1})\}$, 其算法过程如下.

算法 3 加入 0-1 变量的动态规划表更新过程

输入: 变量 x_{k+1} 对应的系数 c_{k+1} 和 a_{k+1}, 上一步迭代的动态规划表 $f(k,z)$, b;

输出: 动态规划表 $f(k+1,z)$;

$f(k+1,z) := f(k,z)$;

for $z = a_{k+1}, a_{k+1}+1, ..., b$ **do**

  if $c_{k+1} + f(k, z-a_{k+1}) > f(k,z)$ **then**

    $f(k+1,z) := c_{k+1} + f(k, z-a_{k+1})$;

  end if

end for

在一般的动态规划算法中, 随着 z 的变化, 最优值 $f(k,z)$ 的变化是跳跃的. 实际上, 如果只记录这些跳跃的节点, 我们也可以逐步更新动态规划表, 最终得到问题的最优值. 接下来, 将通过例 3.3 说明基于占优表的动态规划算法过程.

例 3.3. 考虑如下 0-1 背包问题,

$$opt = \max_x \{8x_1 + 6x_2 + 2x_3 + x_4 : 15x_1 + 5x_2 + 2x_3 + x_4 \le 17, \\ x \in \{0,1\}^4\}.$$

当 $k = 2$ 时, 关于该问题的动态规划表如下.

表 3-1 $f(2,z)$ **的动态规划表**

z	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$f(2,z)$	0	0	0	0	0	6	6	6	6	6	6	6	6	6
z	14	15	16	17										
$f(2,z)$	6	8	8	8										

如果只记录 $f(2,z)$ 发生变化的点, 我们用占优表的形式呈现如下.

表 3-2 $f(2,z)$ **的占优表**

z	0	5	15
$f(2,z)$	0	6	8

比较表 2-1 和表 2-2, 可以看到用占优表形式储存的动态规划表能更少地占用储存空间. 现在假设已有 $f(k,z)$ 的占优表, 考虑下一个变量 x_{k+1}.

记 $\{(w_i^k, p_i^k)\}_{i=1}^{l_k}$ 表示 $f(k,z)$ 的占优表, 其中 l_k 表示该占优表的长度, (w_i^k, p_i^k) 表示该占优表第 i 对数, 分别对应背包问题的已用容量和对应的收益. 对于某两对 $(w_{i_1}^k, p_{i_1}^k)$ 和 $(w_{i_2}^k, p_{i_2}^k)$, 如果有 $w_{i_1}^k \le w_{i_2}^k$ 且 $p_{i_1}^k \ge p_{i_2}^k$, 则称 $(w_{i_1}^k, p_{i_1}^k)$ 占优 $(w_{i_2}^k, p_{i_2}^k)$, 即用更少的容量得到更大的收益. 下面, 我们将用例 3.3 中的算例演示如何生成加入下一个变量的占优表.

例 3.4. 现在我们有 $f(2,z)$ 的占优表 $\{(w_i^2, p_i^2)\}_{i=1}^{l_2}$ 和放入变量 x_3 后的占优表 $\{(w_i^2 + c_3, p_i^2 + a_3)\}_{i=1}^{l_2}$ 如下.

表 3-3 占优表 $\{(w_i^2, p_i^2)\}_{i=1}^{l_2}$ 和 $\{(w_i^2 + c_3, p_i^2 + a_3)\}_{i=1}^{l_2}$

w	0	5	15
p	0	6	8

w	2	7	17
p	2	8	10

依次比较两张表中每一对数字, 按照 $w_1^{k+1} < w_2^{k+1} < ... < w_{l_{k+1}}^{k+1} \le b$ 的顺序并且去除被占优的部分, 得到 $f(3,z)$ 的占优表 $\{(w_i^3, p_i^3)\}_{i=1}^{l_3}$ 如下.

表 3-4 $f(3,z)$ 的占优表

w	0	2	5	7	17
p	0	2	6	8	10

对于下一个考虑的变量 x_{k+1}, $\{(w_i^k + c_{k+1}, p_i^k + a_{k+1})\}_{i=1}^{l_k}$ 表示在放入第 $k+1$ 个物品后各个已有容量下的最大收益. 我们将其与已有的占优表 $\{(w_i^k, p_i^k)\}_{i=1}^{l_k}$ 归并为一个新的占优表, 并且去掉被占优的 (w,p) 对, 即可得到考虑第 $k+1$ 个变量后的动态规划表. 如上述过程, 我们给出加入下一个变量的占优表更新算法如下.

算法 4 加入 0-1 变量的占优表更新过程

输入: 变量 x_{k+1} 对应的系数 c_{k+1} 和 $ak+1$, $f(k,z)$ 的占优表 $\{(w_i^k, p_i^k)\}_{i=1}^{l_k}$, b;

输出: $f(k+1,z)$ 的占优表 $\{(w_i^{k+1}, p_i^{k+1})\}_{i=1}^{l_{k+1}}$;

$i=1, j=1, l=1$;

while $i \le l_k, j \le l_k$ 且 $w_l^k \le b$ **do**

  if $w_i^k \le w_j^k + a_{k+1}$ 且 $p_i^k \ge p_j^k + c_{k+1}$ **then**

    $j := j+1$;

  else if $w_i^k \ge w_j^k + a_{k+1}$ 且 $p_i^k \le p_j^k + c_{k+1}$ **then**

    $i := i+1$;

  else if $w_i^k < w_j^k + a_{k+1}$ **then**

    $w_l^k := w_i^k, p_l^k := p_i^k$;

    $i := i+1, l := l+1$;

  else if $w_i^k > w_j^k + a_{k+1}$ **then**

    $w_l^k := w_j^k + a_{k+1}, p_l^k := p_j^k + c_{k+1}$;

    $j := j+1, l := l+1$;

  end if

end while

while $i \le l_k$ 且 $w_l^k \le b$ **do**

  $w_l^k := w_i^k, p_l^k := p_i^k$;

  $i := i+1, l := l+1$;

end while

while $j \le l_k$ 且 $w_l^k \le b$ **do**

  $w_l^k := w_j^k + a_{k+1}, p_l^k := p_j^k + c_{k+1}$;

  $j := j+1, l := l+1$;

end while

在算法 4 中, 我们依次比较表 $\{(w_i^k, p_i^k)\}_{i=1}^{l_k}$ 和表 $\{(w_i^k + c_{k+1}, p_i^k + a_{k+1})\}_{i=1}^{l_k}$ 中每个 (w,p) 对. 当其中某个 (w,p) 对被占优时, 则会被跳过, 即移除被占优的部分. 当比较的两个 (w,p) 对互相直接没有占优关系时, 则将对应的已用容量 w 较小的 (w,p) 对先放入新的占优表. 如果其中一张表中的 (w,p) 对先被比较完, 则将另外一张表的剩余部分直接赋给新的占优表. 此外, 在比较时如果已用容量 w 大于背包约束右端项 b 的话, 算法将终止.

3.4　数值实验

在本节中, 我们将通过数值实验分析聚合分离算法的效果. 实验是用 C++ 接入 IBM ILOG CPLEX optimizer 12.7.1 编写实现的. 在数值实验中, 我们关闭预处理方法以保持算例的原结构. 此外, 采用单线程求解, 求解时间上限设置为 3600s, 其它参数设置为 CPLEX 默认参数.

测试集中算例为随机生成的多维 0-1 背包问题, 记为 MKP. 测试集有 80 个算例. 问题的变量个数和约束个数有不同设置, 每条背包约束有一定的稀疏度, 每种参数设置下有 20 个算例. 目标函数系数服从 $[90, 100]$ 的均匀分布, 背包约束的系数取值服从 $[10, 50]$ 的均匀分布.

测试集 MKP 的具体参数信息如表 3-5 所示.

表 3-5 测试集 MKP 的参数信息

参数名	参数设置	参数名	参数设置
物品个数	50	约束个数	50
目标函数系数取值	$[90, 100]$	约束系数稀疏度	0.4、0.5、0.6、0.7
右端项取值	$b_j = \sum_{i\in[n]} a_{ij} / 2$	约束系数取值	$[10, 50]$

表 3-6 展示了聚合分离算法与一般精确分离算法 (EXACT) 在测试集 MKP 上的数值结果. 其中, 我们考虑最大聚合次数分别为 1 次 (AGG1) 或 2 次 (AGG2). 并且, 将问题按背包约束系数稀疏度 (Density) 分为 4 个部分, 每种稀疏度下统计至少有一种情况下能在规定时间内求解的算例数 (“#”), 比较了求解问题数 (Solved)、平均求解时间 (Time)、平均搜索的节点数 (Nodes).

表 3-6 聚合分离算法与一般精确分离算法在测试集 MKP 上的表现

Density	#	EXACT			AGG1			AGG2		
		Solved	Nodes	Time	Solved	Nodes	Time	Solved	Nodes	Time
0.4	20	20	16361	6.51	20	14593	6.02	20	13094	5.52
0.5	20	20	125478	33.88	20	104492	28.33	20	101228	27.60
0.6	20	20	259877	64.33	20	238878	57.30	20	235736	53.87
0.7	19	19	1291851	266.14	19	1177005	239.44	19	1138961	228.26

一般来说, 聚合分离算法求解问题的平均搜索节点数和平均求解时间都比一般精确分离算法少. 进一步, 我们考虑在被聚合分离算法影响的算例上算法的表现.

表 3-7 展示了聚合分离算法和一般精确分离算法 (EXACT) 在测试集 MKP 中被影响的算例上的表现. 其中, 我们考虑最大聚合次数分别为 1 次 (AGG1) 或 2 次 (AGG2). 并且, 将问题按背包约束系数稀疏度 (Density) 分为 4 个部分, 每种稀疏度下统计至少有一种情况下聚合分离成功的算例数 (“#”), 比较了平均求解时间 (Time)、平均搜索的节点数 (Nodes). 两种最大聚合次数的设置下还比较了平均聚合分离成功的次数 (Effect).

表 3-7 聚合分离算法与一般精确分离算法在被影响的算例上的表现

Density	#	EXACT		AGG1			AGG2		
		Nodes	Time	Effect	Nodes	Time	Effect	Nodes	Time
0.4	3	25807	8.10	2	20533	6.36	3	19094	5.02
0.5	9	125478	58.88	2	104492	32.23	9	101228	31.34
0.6	14	378944	90.18	3	351602	83.63	13	350421	81.09
0.7	14	1585966	311.89	5	1445354	266.72	15	1367620	258.96

稀疏度越大, 就有越多算例至少在一种最大聚合次数设置下聚合分离成功, 说明加入聚合方法后更可能计算得到割平面, 并且稀疏度越大时, 各约束间重合的变量更多, 更容易聚合分离成功. 当允许的聚合次数较大时, 聚合分离成功的次数会更多, 因为这样可以行生成过程中删去更多的背包可行点, 使得违背值尽可能大, 从而找到分离松弛解的有效不等式. 与表 3-6 相比较可以发现, 当只考虑聚合方法有影响的例子时, 聚合分离算法的计算效果更好.

第 4 章　含 GUB 约束的背包多面体的精确分离算法

4.1　引言

含 GUB 约束的背包问题是要求在满足给定的背包约束和 GUB 约束的前提下, 最大化放入背包的物品总收益. 问题的数学模型如下:

$$\max \sum_{j\in[m]}\sum_{i\in S_j} c_i x_i \tag{4-1a}$$

$$\text{s.t.} \sum_{j\in[m]}\sum_{i\in S_j} a_i x_i \le b, \tag{4-1b}$$

$$\sum_{i\in S_j} x_i \le 1, \forall j \in [m], \tag{4-1c}$$

$$x_i \in \{0,1\}, \forall i \in [n], \tag{4-1d}$$

其中 $[n] = \{1, ..., n\}$ 记作物品的下标集合, 0-1 变量 $x_i \in \{0,1\}$ 表示物品 i 是否放入背包. 模型中 $c_i \in \mathbb{R}_+$ 表示物品 i 的收益, $a_i \in \mathbb{Z}_+$ 表示物品 i 所占的容量, $b \in \mathbb{Z}_+$ 表示背包的总容量. $S_j\ \forall j \in [m]$ 表示若干个 GUB 集合, $[m] = \{1, ..., m\}$ 记作 GUB 集合的下标集合. 不妨假设 $\cup_{j\in[m]} S_j = [n]$. 我们考虑不重叠的 GUB 集合, 即若 $j_1 \ne j_2$, $S_{j_1} \cap S_{j_2} = \varnothing$. 目标函数 (4-1a) 是最大化放入背包的物品总收益, 约束 (4-1b) 是容量约束. 约束 (4-1c) 是 GUB 约束, 表示同属于一个 GUB 集合的变量至多只有一个能取到 1. 这里提出的含 GUB 约束的 0-1 背包多面体的精确分离算法可以被视为对传统的背包多面体精确分离算法的推广, 尤其当 $|S_j| = 1\ \forall j \in [m]$ 时, 含 GUB 约束的 0-1 背包多面体就是经典的 0-1 背包多面体.

定义含 GUB 约束的 0-1 背包集合和对应的多面体如下,

$$K_{GUB} = \left\{ x \in \{0,1\}^n : \sum_{j\in[m]}\sum_{i\in S_j} a_i x_i \le b, \sum_{i\in S_j} x_i \le 1, \forall j \in [m] \right\},$$

$$P_{GUB} = \text{conv}(K_{GUB}).$$

集合 K_{GUB} 的线性松弛集合 LP_{GUB} 为

$$LP_{GUB} = \left\{ x \in [0,1]^n : \sum_{j\in[m]}\sum_{i\in S_j} a_i x_i \le b, \sum_{i\in S_j} x_i \le 1, \forall j \in [m] \right\},$$

即将 0-1 变量松弛为取值为 [0, 1] 的连续变量, 因此 $K_{GUB} \subseteq LP_{GUB}$.

目前, 对于含 GUB 约束的 0-1 背包多面体的割平面方法主要是以覆盖不等式为基础扩展得到的特殊割平面. 而精确分离算法能由当前松弛解出发, 求解分离问题, 得到一般化的割平面.

在本章节中, 我们将针对含 GUB 约束的 0-1 背包多面体设计精确分离算法. 考虑含 GUB 约束的 0-1 背包多面体的分离问题, 即给定点 $\bar{x} \in [0,1]^n$, 要么找到一个超平面分离点 $\bar{x}$ 和多面体 P_{GUB}, 要么证明 $\bar{x} \in P_{GUB}$. 我们首先将含 GUB 约束的 0-1 背包多面体的分离问题建模成一个优化问题. 求解该分离问题的精确分离算法主要分为以下三步: 预处理、行生成、序列升维. 最后, 在一定的测试环境下, 我们对该精确分离算法进行数值实验.

本章的安排如下. 4.2 节分析了含 GUB 约束的 0-1 背包多面体的性质并将其分离问题建模成优化问题. 4.3 节给出求解该分离问题的精确分离算法, 包括 4.3.1 节中计算分离问题前根据当前松弛解对优化问题进行预处理, 4.3.2 节中用行生成方法求解预处理后低维多面体的分离问题, 给出求解含 GUB 约束的背包子问题的动态规划算法, 4.3.3 节中改进以占优表储存的稀疏动态规划算法来求解序列升维系数, 得到原多面体的割平面. 最后, 4.4 节展示了相关数值实验结果, 并在此基础上进行分析和小结.

4.2 含 GUB 约束的 0-1 背包多面体的分离问题

给定点 $\bar{x} \in [0,1]^n$, 含 GUB 约束的 0-1 背包多面体 P_{GUB} 的分离问题是找到超平面严格分离多面体 P_{GUB} 和点 $\bar{x}$, 对应于找到有效不等式 $\sum_{j\in[m]} \sum_{i\in S_j} \alpha_i x_i \leq \beta$, 使得

$$\sum_{j\in[m]} \sum_{i\in S_j} \alpha_i x_i \leq \beta, \forall x \in P_{GUB},$$

并且

$$\sum_{j\in[m]} \sum_{i\in S_j} \alpha_i \bar{x}_i > \beta,$$

或者证明 $\bar{x} \in P_{GUB}$. 对于当前松弛解 $\bar{x} \in LP_{GUB}$, 求解其分离问题等价于求解下述优化问题.

$$\begin{aligned} v = \max_{\alpha,\beta} \quad & \sum_{j\in[m]} \sum_{i\in S_j} \alpha_i \bar{x}_i \\ \text{s. t.} \quad & \sum_{j\in[m]} \sum_{i\in S_j} \alpha_i x_i \le \beta, \forall x \in P_{GUB}, \\ & \alpha \in \Pi, \end{aligned} \tag{4-1}$$

其中 Π 是包含原点的凸紧集, $\beta \in \mathbb{R}$, $\alpha \in \mathbb{R}^n$ 为变量.

如果 $v \le \beta$, 就证明了 $\bar{x} \in P_{GUB}$. 否则, 根据问题的最优解 α^*, 我们得到有效不等式 $\sum_{j\in[m]} \sum_{i\in S_j} \alpha_i^* x_i \le \beta$ 严格分离当前松弛解 $\bar{x}$ 和多面体 P_{GUB}. 类似于 2.2 节, 求解其分离问题等价于求解下述优化问题, 其中 $0 \le \alpha_i \le 1, \forall i \in [n]$.

$$\begin{aligned} v = \max_{\alpha} \quad & \sum_{j\in[m]} \sum_{i\in S_j} \alpha_i \bar{x}_i \\ \text{s. t.} \quad & \sum_{j\in[m]} \sum_{i\in S_j} \alpha_i x_i \le 1, \forall x \in P_{GUB}. \end{aligned} \tag{4-2}$$

4.3　精确分离算法

在本节中, 我们将针对含 GUB 约束的背包多面体设计精确分离算法. 正如前文所述, 精确分离算法主要分为预处理、行生成、序列升维等三步, 其中每一步都将根据 GUB 约束的特性进行改进.

(1) **预处理**. 为了减少计算时间, 在求解分离问题前我们对原问题进行预处理. 首先对松弛解取值全为 0 或 1 的 GUB 集合中的变量进行固定, 得到一个低维的多面体.

(2) **行生成**. 通过行生成方法, 求解一系列局部分离问题来计算低维多面体上的分离问题找到一个违背当前松弛解的有效不等式, 或者证明这样的不等式不存在.

(3) **序列升维**. 在预处理过程中被固定的变量按所在的 GUB 集合被序列升维, 从而得到一个对原多面体更强的有效不等式.

4.3.1　预处理

给定点 $\bar{x} \in LP_{GUB}$, 我们首先考虑固定部分变量得到多面体 P_{GUB} 的低维多面体 $P_{GUB}(\bar{x})$ 如下:

$$P_{GUB}(\bar{x}) = \text{conv}(K_{GUB}(\bar{x})),$$

其中

$$K_{GUB}(\bar{x}) = \left\{ x \in \{0,1\}^{n-\sum_{t\in T}|S_t|} : \sum_{t\in T}\sum_{i\in S_t} a_i x_i \le \bar{b}, \sum_{i\in S_t} x_i \le 1, \forall t \in T \right\},$$

$T = \{t \in [m] : \bar{x}_i \in \{0,1\}, \forall i \in S_t\}$, $\bar{b} = b - \sum_{t\in T, i\in S_t, \bar{x}_i=1} a_i$. 这说明, 当同属于一个 GUB 集合的变量松弛解只有 0, 1 两种取值时, 我们将固定这些 GUB 集合中的变量, 从而得到低维多面体 $P_{GUB}(\bar{x})$. 类似于问题 (4-2), 建立 $P_{GUB}(\bar{x})$ 上的分离问题如下.

$$\begin{aligned} \max_{\alpha} \quad & \sum_{j\in[m]\backslash T}\sum_{i\in S_j} \alpha_i \bar{x}_i \\ \text{s. t.} \quad & \sum_{j\in[m]\backslash T}\sum_{i\in S_j} \alpha_i x_i \le 1, \forall x \in P_{GUB}(\bar{x}). \end{aligned} \tag{4-1}$$

4.3.2 行生成

不同于直接求解可能有指数条约束的问题 (4-1), 行生成将从部分约束出发, 用迭代方法求解局部分离问题.

$$\begin{aligned} v(U) = \max_{\alpha} \quad & \sum_{j\in[m]\backslash T}\sum_{i\in S_j} \alpha_i \bar{x}_i \\ \text{s. t.} \quad & \sum_{j\in[m]\backslash T}\sum_{i\in S_j} \alpha_i x_i \le 1, \forall x \in U, \\ & 0 \le \alpha_i \le 1, \forall i \in S_j, \forall j \in [m]\backslash T, \end{aligned} \tag{4-2}$$

其中 $U \subseteq P_{GUB}(\bar{x})$. 我们初始化 $U = \varnothing$, 并令 $0 \le \alpha_i \le 1, \forall i \in [n]$. 对于局部分离问题, 如果违背值 $v(U) \le 1$, 则可以证明 $\bar{x} \in P_{GUB}(\bar{x})$. 否则, 我们得到不等式

$$\sum_{j\in[m]\backslash T}\sum_{i\in S_j} \alpha_i^* x_i \le 1, \tag{4-3}$$

其中 α^* 是当前局部分离问题的最优解, 点 $\bar{x}$ 违背该不等式. 接下来, 还需要验证该不等式是否对集合 $K_{GUB}(\bar{x})$ 有效. 求解如下带 GUB 约束的背包子问题.

$$z = \max_{x} \left\{ \sum_{j\in[m]\backslash T}\sum_{i\in S_j} \alpha_i^* x_i : x \in K_{GUB}(\bar{x}) \right\}. \tag{4-4}$$

如果 $z \le 1$, 则不等式 (4-3) 对带 GUB 约束的背包集合 $K_{GUB}(\bar{x})$ 有效. 否则, 求解问题 (4-4) 得到的解违背了不等式 (4-3), 需要将这个解对应的背包可行点加入子

集合 U 并且再次求解局部分离问题, 直至得到一个被当前松弛解违背的有效不等式, 或者证明这样的不等式不存在. 接下来, 我们在算法 5 中给出该分离问题的行生成过程.

算法 5 含 GUB 约束的背包多面体的分离问题的行生成过程

输入: 松弛解 $\bar{x}$, 集合 $K_{GUB}(\bar{x})$ 和子集合 $U = \varnothing$;

输出: 一个被 $\bar{x}$ 违背的有效不等式 $\sum_{j\in[m]\setminus T}\sum_{i\in S_j}\alpha_i^* x_i \le 1$, 或证明这样的不等式不存在;

步骤 1: 求解局部分离问题 (4-2) 得到最优解 α^* 和最优值 $v(U)$;

if $v(U) > 1$ **then**

　步骤 2: 求解带 GUB 约束的背包子问题 (4-4) 得到最优解 x^* 和最优值 z^*;

　if $z^* > 1$ **then**

　　将违背点 x^* 加入子集合 U 并转入步骤 1;

　else

　　终止算法过程并且返回有效不等式 $\sum_{j\in[m]\setminus T}\sum_{i\in S_j}\alpha_i^* x_i \le 1$;

　end if

else

　终止算法过程并且证明 $\bar{x} \in P_{GUB}(\bar{x})$;

end if

在算法 5 中, 步骤 1 调用求解器 CPLEX 求解局部分离问题 (4-2) 得到最优解 α^* 和最优值 $v(U)$. 如果 $v(U) \le 1$, 则证明 $\bar{x} \in P_{GUB}(\bar{x})$. 否则将进入步骤 2, 用改进后的动态规划算法求解带 GUB 约束的背包子问题得到最优解 x^* 和最优值 z^*. 如果 $z^* \le 1$, 则我们得到违背 $\bar{x}$ 但对集合 $P_{GUB}(\bar{x})$ 有效的不等式 $\sum_{j\in[m]\setminus T}\sum_{i\in S_j}\alpha_i^* x_i \le 1$. 不然, 将违背的背包可行点 x^* 对应的约束加入局部分离问题 (4-2) 中并再次转入步骤 1.

在上述的行生成过程中, 需要求解一系列带 GUB 约束的背包子问题. 为了尽可能找到最违背的背包可行点, 我们考虑使用精确算法求解这些子问题. 基于一般 0-1 背包问题的动态规划算法, 对于带 GUB 约束的背包问题定义问题 $P_{GUB}(k, z)$ 如下.

$$
\begin{aligned}
f_{GUB}(k,z) = \max \quad & \sum_{j\in[k]}\sum_{i\in S_j} c_i x_i \\
\text{s.t.} \quad & \sum_{j\in[k]}\sum_{i\in S_j} a_i x_i \le z, \\
& \sum_{i\in S_j} x_i \le 1, j \in [k], \\
& x_i \in \{0,1\}, \forall i \in S_j, \forall j \in [k],
\end{aligned}
\tag{4-5}
$$

其中 $f_{GUB}(m,b)$ 对应于原来的带 GUB 约束的背包问题的最优值. 不妨令 $f(0,z) = 0, \forall z \in [b]$. 因为 GUB 集合中至多只有一个变量可以取到 1, 所以对于 $f_{GUB}(k,z)$ 有递推公式如下.

$$
f_{GUB}(k,z) = \max\left\{f_{GUB}(k-1,z), c_i + f_{GUB}(k-1,z-a_i)\ \forall i \in I_k(z)\right\},
$$

其中 $I_k(z) = \{i \in S_k : a_i \le z\}$. 对于某个 GUB 集合 S_k 和其中某个物品 i, 只有当 $a_i \le z$ 时才能比较是否放入第 i 个物品. 若一个物品都不放入, 则容量不变, 对应的最优值为 $f_{GUB}(k-1,z)$. 若放入第 i 个物品, 则容量要减少 a_i, 对应的最优值为 $f_{GUB}(k-1,z-a_i)$ 加上放入该物品得到的收益 c_i. 由于优化目标是最大化放入背包的物品总收益, 而且对于这个 GUB 集合来说, 其中至多只有一个物品能放入背包, 所以我们选择各种情况下最大的总收益, 其计算过程见算法 6.

算法 6 加入 GUB 变量组的动态规划表更新过程

输入: GUB 变量组对应的系数 c_i 和 a_i, $\forall i \in S_{k+1}$, 上一步迭代的动态规划表 $f_{GUB}(k,z)$, b;

输出: 动态规划表 $f_{GUB}(k+1,z)$;

$f_{GUB}(k+1,z) := f_{GUB}(k,z)$;

for $z = 1, ..., b$ **do**

  for $i \in S_{k+1}$ **do**

    if $z \ge a_i$ 且 $c_i + f_{GUB}(k,z-a_i) > f_{GUB}(k,z)$ **then**

      $f_{GUB}(k+1,z) := c_i + f_{GUB}(k,z-a_i)$;

    end if

  end for

end for

得到最优值 $f_{GUB}(m,b)$ 后, 我们对各 GUB 集合依次回溯找最优解 x^*. 令 $\lambda := b$, 对于某个 k, 如果 $f_{GUB}(k,\lambda) = f_{GUB}(k-1,\lambda)$, 则 $x_i^* = 0\ \forall i \in S_k$. 否则,

对于某个 $i \in S_k$, 有 $f_{GUB}(k,\lambda) = c_i + f_{GUB}(k-1, \lambda - a_i)$, 则 $x_i^* = 1$, 并且更新 $\lambda := \lambda - a_i$. 对于带 GUB 约束的背包问题, 动态规划算法的计算复杂度是 $O(nb)$.

4.3.3　序列升维

在行生成过程中我们已经计算得到对多面体 $P_{GUB}(\bar{x})$ 有效的不等式 (4-3), 但是它对原多面体 P_{GUB} 不一定有效, 还需要对预处理阶段中固定的 GUB 变量组按一定的顺序进行序列升维.

不妨假设, 有效不等式 (4-3) 中已经有前 $k-1$ 组变量, 即有不等式

$$\sum_{j\in[k-1]} \sum_{i\in S_j} \alpha_i x_i \le \beta,$$

我们要升维第 k 组变量 $x_i\ \forall i \in S_k$, 有如下两种情形. 记

$$W_{GUB}(z) = \max_{x\in\{0,1\}^{\sum_{j\in[k-1]}|S_j|}} \left\{ \sum_{j\in[k-1]} \sum_{i\in S_j} \alpha_i x_i : \sum_{j\in[k-1]} \sum_{i\in S_j} a_i x_i \le z, \sum_{i\in S_j} x_i \le 1\ \forall j \in [k-1] \right\}.$$

1) 如果 $\bar{x}_i = 0\ \forall i \in S_k$, 那么升维后的不等式是

$$\sum_{j\in[k-1]} \sum_{i\in S_j} \alpha_i x_i + \sum_{i\in S_k} \alpha_i x_i \le \beta,$$

其中 $\alpha_i = \beta - W_{GUB}(\bar{b} - a_i)\ \forall i \in S_k$.

2) 如果 $\exists i_k \in S_k$, 有 $\bar{x}_{i_k} = 1$. 由于 GUB 集合的定义, 则对 $\forall i \in S_k$ 且 $i \ne i_k$, 有 $\bar{x}_i = 0$. 那么升维后的不等式是

$$\sum_{j\in[k-1]} \sum_{i\in S_j} \alpha_i x_i + \alpha_{i_k}(x_{i_k} - 1) + \sum_{i\ne i_k, i\in S_k} \alpha_i x_i \le \beta,$$

其中 $\alpha_{i_k} = W_{GUB}(\bar{b} + a_{i_k}) - \beta$, $\alpha_i = \beta - W_{GUB}(\bar{b} + a_{i_k} - a_i) + \alpha_{i_k}\ \forall i \ne i_k, i \in S_k$. 并且, 因为原来 $\bar{x}_{i_k} = 1$, 在升维后需要更新 $\bar{b} \to \bar{b} + a_{i_k}$.

与 3.3.3 节的序列升维过程相比, 带 GUB 约束的背包多面体每次对一整个 GUB 集合的变量进行升维. 因此, 我们需要改进原有的以占优表储存的稀疏动态规划算法来求解这些升维过程中的子问题. 接下来, 在例 3.3 的基础上, 我们通过例 4.1 说明改进后的算法过程.

例 4.1. 考虑带 GUB 约束的背包问题:

$$\begin{aligned} opt = \{8x_1 + 6x_2 + 2x_3 + x_4 : & 15x_1 + 5x_2 + 2x_3 + x_4 \le 17, \\ & x_3 + x_4 \le 1, x \in \{0,1\}^4\}. \end{aligned}$$

当 $k=2$ 时, 关于该问题以占优表形式储存的动态规划表如下.

表 4-1 $f_{GUB}(2,z)$ 的占优表

z	0	5	15
$f_{GUB}(2,z)$	0	6	8

现在假设已有 $f_{GUB}(k,z)$ 的占优表, 考虑下一个 GUB 变量组 $x_i\ \forall i \in S_{k+1}$. 下面, 我们将用例 4.1 中的算例演示如何逐步生成占优表.

表 4-2 占优表 $\{(w_i^2,p_i^2)\}_{i=1}^{l_2}$ 和 $\{(w_i^2+c_3,p_i^2+a_3)\}_{i=1}^{l_2}$

w	0	5	15
p	0	6	8

w	2	7	17
p	2	8	10

表 4-3 中间阶段的占优表和 $\{(w_i^2+c_4,p_i^2+a_4)\}_{i=1}^{l_2}$

w	0	2	5	7	17
p	0	2	6	8	10

w	1	6	16
p	1	7	9

表 4-4 $f_{GUB}(3,z)$ 的占优表

z	0	2	5	6	7	16	17
$f_{GUB}(3,z)$	0	2	6	7	8	9	10

不同于 3.3.3 节的动态规划表更新过程, 因为同属于一个 GUB 集合的变量至多只有一个能取到 1, 所以对于第 $k+1$ 个 GUB 变量组我们需要逐步合并已有占优表 $\{(w_i^k,p_i^k)\}_{i=1}^{l_k}$ 和 $\{(w_i^k+c_j,p_i^k+a_j)\}_{i=1}^{l_k}\ \forall j \in S_{k+1}$ 得到 $f_{GUB}(k+1,z)$ 的占优表. 如上述过程, 我们给出加入 GUB 变量组的占优表更新算法如下.

算法 7 加入 GUB 变量组的占优表更新过程

输入: GUB 变量组对应的系数 $c_i\ \forall i \in S_{k+1}$ 和 $a_i\ \forall i \in S_{k+1}$, $f_{GUB}(k,z)$ 的占优表 $\{(w_i^k, p_i^k)\}_{i=1}^{l_k}$, b;

输出: $f_{GUB}(k+1,z)$ 的占优表 $\{(w_i^{k+1}, p_i^{k+1})\}_{i=1}^{l_{k+1}}$;

$\{(w_i^{mid}, p_i^{mid})\}_{i=1}^{l_{mid}}$ 记为中间阶段的占优表;

$\{(w_i^{mid}, p_i^{mid})\}_{i=1}^{l_{mid}} = \{(w_i^k, p_i^k)\}_{i=1}^{l_k}$;

for $t \in S_{k+1}$ **do**

　参考算法 4, 合并 $\{(w_i^{mid}, p_i^{mid})\}_{i=1}^{l_{mid}}$ 和 $\{(w_i^k + c_t, p_i^k + a_t)\}_{i=1}^{l_k}$;

end for

$\{(w_i^{k+1}, p_i^{k+1})\}_{i=1}^{l_{k+1}} = \{(w_i^{mid}, p_i^{mid})\}_{i=1}^{l_{mid}}$;

4.4　数值实验

对于含 GUB 约束的背包问题, 我们用 C++ 接入 IBM ILOG CPLEX optimizer 12.7.1 编写代码测试精确分离算法的有效性. 为了不改变算例的原结构, 在数值实验中我们关闭了预处理方法. 并且, 我们设置只在根节点计算用户割平面, 采用单线程求解, 求解时间上限设置为 3600s. 除此之外, 其它参数设置沿用 CPLEX 默认参数.

首先, 我们考虑带冲突关系的装箱问题 (Bin Packing Problem with Conflicts, BPPC) [52, 53]. 装箱问题是指给定若干有容量的物品要装到若干个箱子, 优化目标是使用的箱子尽可能少 [54]. 问题的数学模型如下.

$$
\begin{aligned}
\min\ & \sum_{j\in[m]} y_j \\
\text{s.t.}\ & \sum_{i\in[n]} a_i x_{ij} \le v_j y_j, \forall j \in [m], \\
& \sum_{j\in[m]} x_{ij} = 1, \forall i \in [n], \\
& x_{ij} \in \{0,1\}, \forall i \in [n], j \in [m], \\
& y_j \in \{0,1\}, \forall j \in [m],
\end{aligned}
\tag{4-1}
$$

其中 $[n] = \{1, ..., n\}$, $a \in \mathbb{Z}_+^{n\times m}$ 表示各个物品在各个箱子中所占容量, $v \in \mathbb{Z}_+^m$ 表示各个箱子的总容量. 带冲突关系的装箱问题则是在一般的装箱问题的基础上, 令某些物品存在冲突关系, 不能放进同一个箱子. BPPC 问题是 $\mathcal{NP}$-难的, 它可以看作是装箱问题和顶点着色问题的一般形式 [6, 55, 56]. 问题 (4-1) 中最关键的是第

一条约束, 我们称其为装箱约束, 不失一般性, 考虑如下形式的装箱约束,

$$\sum_{i\in[n]} a_i x_i \le v y. \tag{4-2}$$

约束 (4-2) 和背包约束的结构类似, 因此其分离问题的求解可以转化为对背包约束分离问题的求解. 下面, 我们将说明两者之间的关系.

命题 4.1. *若不等式 $\sum_{i\in[n]} \alpha_i x_i \le 1$ 为多面体 $P_1 = \{x \in \mathbb{R}^n : \sum_{i\in[n]} a_i x_i \le v\}$ 的刻面定义不等式, 则对应的不等式 $\sum_{i\in[n]} \alpha_i x_i \le y$ 为多面体 $P_2 = \{x \in \mathbb{R}^n : \sum_{i\in[n]} a_i x_i \le v y\}$ 的刻面定义不等式.*

证明. 假设不等式 $\sum_{i\in[n]} \alpha_i x_i \le 1$ 为多面体 $P_1 = \{x \in \mathbb{R}^n : \sum_{i\in[n]} a_i x_i \le v\}$ 的刻面定义不等式, 则存在 n 个仿射无关的点 $x^1, ..., x^n \in P_1$ 使得 $\sum_{i\in[n]} \alpha_i x_i^j = 1, \forall j \in [n]$. 则点 $(x^1, 1), ..., (x^n, 1) \in P_2$ 也是 n 个仿射无关的点, 并且有 $\sum_{i\in[n]} \alpha_i x_i^j - 1 = 0, \forall j \in [n]$, 所以不等式 $\sum_{i\in[n]} \alpha_i x_i - y \le 0$ 为多面体 $P_2 = \{x \in \mathbb{R}^n : \sum_{i\in[n]} a_i x_i \le v y\}$ 的刻面定义不等式. □

由命题 (4.1), 我们可以对装箱约束描述的多面体运用一维 0-1 背包多面体的精确分离算法. 我们将在两个测试集上测试含 GUB 约束的背包多面体的精确分离算法.

第一个测试集来自于 Muritiba 等的工作中采用的带冲突关系的装箱问题测试集 [25], 记作 Muritiba2010. 测试集包含 50 个算例. 原测试集的问题中存在若干对变量存在冲突关系, 我们先将问题松弛为带 GUB 约束的装箱问题, 即当一组变量中两两之间都冲突时, 令这组变量同属于一个 GUB 集合, 在这样的构造规则下可以满足划分的若干个 GUB 集合是互不相交的. 此外, 测试集将根据原问题中冲突变量对的密度进行进一步划分, 冲突变量对的密度越大对应的松弛问题中 GUB 集合就越大, 所以我们也可以比较不同密度下精确分离算法的性能差异. 第二个测试集是参考文献 [22, 25] 中采用的测试集生成的, 记作 BPPCasym. 测试集包含 120 个算例. 之前的测试集是对称的装箱问题, 即物品在每个箱子中所占容量和冲突关系是一样的, 进一步考虑非对称的装箱问题. 我们将用这两个测试集来比较精确分离算法得到的割平面和 CPLEX 已有的割平面的效果.

测试集 Muritiba2010 的各参数设置如表 4-5 所示.

表 4-5 测试集 Muritiba2010 的参数信息

参数名	参数设置	参数名	参数设置
物品个数	120	箱子最大个数	120
物品所占容量	[20, 100]	箱子容量	150
冲突约束稀疏度	0.1、0.2、0.3、0.4、0.5		

表 4-6 展示了测试集 Muritiba2010 上 CPLEX 默认设置 (CPX) 以及使用针对含 GUB 约束的背包多面体的精确分离算法 (GUBEXACT) 的数值结果. 其中, 我们将问题按冲突约束稀疏度 (Density) 分为 5 个部分, 每种稀疏度下统计至少有一种情况下能在规定时间内求解的算例数 (“#”), 比较了求解问题数 (Solved)、平均求解时间 (Time)、平均搜索的节点数 (Nodes).

表 4-6 精确分离算法与默认设置在测试集 Muritiba2010 上的表现

Density(%)	#	CPX			GUBEXACT		
		Solved	Nodes	Time	Solved	Nodes	Time
10	8	7	129620	882.07	8	69823	408.03
20	8	7	83604	570.68	8	23516	180.19
30	8	7	29017	453.85	8	8063	149.77
40	8	7	29560	390.55	8	17101	205.56
50	9	9	13760	277.10	9	8503	209.58

随着稀疏度升高, 问题更易求解, 这是因为稀疏度大时, 互相冲突的变量越多, GUB 集合越大, 问题规模相对变小. 每种稀疏度下, 精确分离算法求解问题所用的平均时间都比默认设置下少. 当稀疏度为 0.1 至 0.4 时, 使用精确分离算法可以多求解 1 个问题. 当稀疏度较小时, 针对 GUB 约束设计的精确分离算法的效果更好, 可能是因为稀疏度越小 GUB 集合更多.

测试集 BPPCasym 的各参数设置如表 4-7 所示.

表 4-7 测试集 BPPCasym 的参数信息

参数名	参数设置	参数名	参数设置
物品个数	50、70、90、100、110、120	箱子最大个数	与物品个数对应
物品所占容量	[20, 100]	箱子容量	$v_j = \sum_{i\in[n]} a_{ij} / 2$
GUB 集合个数	[30, 50]	箱子种类	5

表 4-8 展示了测试集 BPPCasym 上 CPLEX 默认设置 (CPX) 以及使用针对含 GUB 约束的背包多面体的精确分离算法 (GUBEXACT) 的数值结果. 其中, 我们按箱子个数 (Size) 将问题分为 6 个部分, 每种问题规模下统计至少有一种情况下能在规定时间内求解的算例数 (“#”), 比较了求解问题数 (Solved)、平均求解时间 (Time)、平均搜索的节点数 (Nodes) 和 *gap closed* (Gap). 其中 *gap closed* [57] 定义如下.

$$\frac{z_{root} - z_{LP}}{z_{ub} - z_{LP}} \times 100\%$$

其中 z_{root} 表示根节点在加入割平面后线性松弛的最优值, z_{LP} 表示加入割平面前线性松弛的最优值, z_{ub} 表示已知的最优值.

gap closed 越大, 说明根节点处割平面效果越好. 如果算例在根节点求解到最优值, 则 *gap closed* 为 100%. 测试集 Muritiba2010 的算例为对称的装箱问题, *gap closed* 要么为 100% 要么为 0%, 比较这一数值意义不大. 我们考虑测试集 BPPCasym 中非对称的装箱问题, 进行数值实验测试精确分离算法是否可以提升 *gap closed*.

表 4-8 精确分离算法与默认设置在测试集 BPPCasym 上的表现

Size	#	CPX				GUBEXACT			
		Solved	Nodes	Time	Gap(%)	Solved	Nodes	Time	Gap(%)
50	20	20	2492	5.20	12.98	20	210	3.08	20.03
70	20	20	1032	35.32	11.40	20	439	8.26	17.23
90	17	16	20968	219.80	7.12	15	3813	127.33	13.79
100	15	15	99468	1102.35	5.84	15	38595	594.66	7.03
110	14	13	115953	1203.35	3.16	13	47843	764.71	4.46
120	12	11	101395	1272.76	2.83	12	40438	932.15	4.33

当物品个数越大时, 问题规模越大, 求解难度越大. 一般来说当问题规模较小时, 在根节点的求解效果更好. 含 GUB 约束的背包多面体的精确分离算法比 CPLEX 默认设置有一定效果提升, 求解搜索的平均节点数减少, *gap closed* 提升. 尤其对于难问题, 平均的求解节点数明显减少.

第 5 章　总结与展望

本文的工作主要围绕背包多面体的分离问题展开, 一是对多维背包多面体设计聚合分离算法, 二是针对含 GUB 约束的背包多面体设计精确分离算法.

本文首先实现了一维 0-1 背包多面体的精确分离算法, 再考虑多维背包多面体的分离问题, 在行生成过程中添加了背包约束的聚合方法, 提出了多维背包多面体的聚合分离算法. 数值实验结果表明聚合分离算法可以提升问题求解效果.

其次, 我们提出了含 GUB 约束的背包多面体的精确分离算法. 考虑 GUB 约束的特性分别在精确分离算法框架的三步——预处理、行生成和序列升维中改进原有的精确分离算法. 我们先在预处理阶段将变量按 GUB 集合分别进行固定. 再在行生成过程中给出求解含 GUB 约束的背包子问题的动态规划算法, 并以此求解验证不等式有效性的背包子问题. 在此基础上, 在序列升维过程中改进以占优表储存的稀疏动态规划算法来求解序列升维系数. 最后, 数值实验表明, 含 GUB 约束的背包多面体的精确分离算法可以提升求解含 GUB 约束的装箱问题的效率. 而对于非对称含 GUB 约束的装箱问题, 该算法还可以进一步提升问题在根节点的 *gap closed*.

当然本文研究内容问题还有可以提升的方面. 目前我们根据已加入的背包可行点对各条约束的违背值或违背次数来选取最违背的约束进行聚合, 而如何选取要聚合的约束来得到生成更好的割平面需要从理论上进一步进行探索. 当约束规模变大时, 对含 GUB 约束的背包多面体的精确分离算法的计算时间将大幅度增加, 尤其是行生成过程部分, 因此还需要研究加速行生成过程的方法.

参考文献

[1] Dantzig G F R. Solution of a large-scale traveling-salesman problem [J]. Journal of the Operations Research Society of America, 1954, 2(4): 393-410.

[2] Land A D A. An automatic method for solving discrete programming problems [J]. Econometrica, 1960, 28(3): 497.

[3] Gomory R E. Outline of an algorithm for integer solutions to linear programs [J]. Bulletin of the American Mathematical Society, 1958, 64(5): 275-278.

[4] Crowder H P M, Johnson E L. Solving large-scale zero-one linear programming problems [J]. Operations Research, 1983, 31(5): 803-834.

[5] Kellerer H, Pferschy U, Pisinger D, et al. Multidimensional knapsack problems [M]. Springer, 2004.

[6] Martello S, Toth P. Knapsack problems: algorithms and computer implementations [M]. John Wiley & Sons, Inc., 1990.

[7] Gu Z, Nemhauser G L, Savelsbergh M W. Lifted flow cover inequalities for mixed 0-1 integer programs [J]. Mathematical Programming, 1999, 85: 439-467.

[8] Cornuéjols G, Li Y, Vandenbussche D. K-cuts: A variation of gomory mixed integer cuts from the lp tableau [J]. INFORMS Journal on Computing, 2003, 15(4): 385-396.

[9] Marchand H, Wolsey L A. Aggregation and mixed integer rounding to solve mips [J]. Operations Research, 2001, 49(3): 363-371.

[10] Balas E, Perregaard M. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer gomory cuts for 0-1 programming [J]. Mathematical Programming, 2003, 94(2): 221-245.

[11] Cook W, Kannan R, Schrijver A. Chvátal closures for mixed integer programming problems [J]. Mathematical Programming, 1990, 47(1-3): 155-174.

[12] Dash S, Günlük O. On the strength of gomory mixed-integer cuts as group cuts [J]. Mathematical Programming, 2008, 115: 387-407.

[13] Gomory R E, Johnson E L. Some continuous functions related to corner polyhedra [J]. Mathematical Programming, 1972, 3: 23-85.

[14] Achterberg T, Wunderling R. Mixed integer programming: Analyzing 12 years of progress [J]. Facets of Combinatorial Optimization: Festschrift for Martin Grötschel, 2013: 449-481.

[15] Wolsey L A, Nemhauser G L. Integer and combinatorial optimization: volume 55 [M]. John Wiley & Sons, 1999.

[16] Glover F. Unit coefficient inequalities for zero-one programming [J]. Management Science Report Series, 1973(73-7).

[17] Wolsey L A. Faces for a linear inequality in 0–1 variables [J]. Mathematical Programming, 1975, 8(1): 165-178.

[18] Balas E, Zemel E. Facets of the knapsack polytope from minimal covers [J]. SIAM Journal on Applied Mathematics, 1978, 34(1): 119-148.

[19] Padberg M W. A note on zero-one programming [J]. Operations Research, 1975, 23(4): 833-837.

[20] Wolsey L A. Valid inequalities for 0–1 knapsacks and mips with generalised upper bound constraints [J]. Discrete Applied Mathematics, 1990, 29(2-3): 251-261.

[21] Johnson E L, Padberg M W. A note of the knapsack problem with special ordered sets [J]. Operations Research Letters, 1981, 1(1): 18-22.

[22] Yamada T, Kataoka S, Watanabe K. Heuristic and exact algorithms for the disjunctively constrained knapsack problem [J]. Information Processing Society of Japan Journal, 2002, 43 (9).

[23] Christofides N. The vehicle routing problem [J]. Combinatorial Optimization, 1979.

[24] Laporte G, Desroches S. Examination timetabling by computer [J]. Computers & Operations Research, 1984, 11(4): 351-360.

[25] Muritiba A E F, Iori M, Malaguti E, et al. Algorithms for the bin packing problem with conflicts [J]. Informs Journal on Computing, 2010, 22(3): 401-415.

[26] Nemhauser G L, Vance P H. Lifted cover facets of the 0–1 knapsack polytope with gub constraints [J]. Operations Research Letters, 1994, 16(5): 255-263.

[27] Sherali H D, Lee Y. Sequential and simultaneous liftings of minimal cover inequalities for generalized upper bound constrained knapsack polytopes [J]. SIAM Journal on Discrete Mathematics, 1995, 8(1): 133-153.

[28] Gokce E I, Wilhelm W E. Valid inequalities for the multi-dimensional multiple-choice 0–1 knapsack problem [J]. Discrete Optimization, 2015, 17: 25-54.

[29] Zeng B, Richard J P P. A polyhedral study on 0–1 knapsack problems with disjoint cardinality constraints: facet-defining inequalities by sequential lifting [J]. Discrete Optimization, 2011, 8(2): 277-301.

[30] Luiz T A, Santos H G, Uchoa E. Cover by disjoint cliques cuts for the knapsack problem with conflicting items [J]. Operations Research Letters, 2021, 49(6): 844-850.

[31] Andrew Boyd E. A pseudopolynomial network flow formulation for exact knapsack separation [J]. Networks, 1992, 22(5): 503-514.

[32] Boyd E A. Generating fenchel cutting planes for knapsack polyhedra [J]. SIAM Journal on Optimization, 1993, 3(4): 734-750.

[33] Boyd E A. Fenchel cutting planes for integer programs [J]. Operations Research, 1994, 42(1): 53-64.

[34] Yan X Q, Boyd E A. Cutting planes for mixed-integer knapsack polyhedra [J]. Mathematical Programming, 1998, 81: 257-262.

[35] Kaparis K, Letchford A N. Separation algorithms for 0-1 knapsack polytopes [J]. Mathematical Programming, 2010, 124: 69-91.

[36] Fukasawa R, Goycoolea M. On the exact separation of mixed integer knapsack cuts [J]. Mathematical Programming, 2011, 128: 19-41.

[37] Goycoolea M G. Cutting planes for large mixed integer programming models [D]. Georgia Institute of Technology, 2006.

[38] Fukasawa R. Single-row mixed-integer programs: Theory and computations [D]. Georgia Institute of Technology, 2008.

[39] Avella P, Boccia M, Vasilyev I. Computational testing of a separation procedure for the knapsack set with a single continuous variable [J]. INFORMS Journal on Computing, 2012, 24(1): 165-171.

[40] Gabrel V, Minoux M. A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems [J]. Operations Research Letters, 2002, 30(4): 252-264.

[41] Avella P, Boccia M, Vasilyev I. Computational experience with general cutting planes for the set covering problem [J]. Operations Research Letters, 2009, 37(1): 16-20.

[42] Vasilyev I L. A cutting plane method for knapsack polytope [J]. Journal of Computer and Systems Sciences International, 2009, 48: 70-77.

[43] Avella P, Boccia M, Vasilyev I. A computational study of exact knapsack separation for the generalized assignment problem [J]. Computational Optimization and Applications, 2010, 45: 543-555.

[44] Applegate D, Bixby R, Chvátal V, et al. Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems [J]. Mathematical Programming, 2003, 97: 91-153.

[45] Vasilyev I, Boccia M, Hanafi S. An implementation of exact knapsack separation [J]. Journal of Global Optimization, 2016, 66(1): 127-150.

[46] Chen L, Chen W K, Yang M M, et al. An exact separation algorithm for unsplittable flow capacitated network design arc-set polyhedron [J]. Journal of Global Optimization, 2021, 81: 659-689.

[47] Balas E. Facets of the knapsack polytope [J]. Mathematical Programming, 1975, 8: 146-164.

[48] Ceselli A, Righini G. A branch-and-price algorithm for the capacitated p-median problem [J]. Networks: An International Journal, 2005, 45(3): 125-142.

[49] Bodur M, Del Pia A, Dey S S, et al. Aggregation-based cutting-planes for packing and covering integer programs [J]. Mathematical Programming, 2018, 171: 331-359.

[50] Pisinger D. A minimal algorithm for the bounded knapsack problem [J]. INFORMS Journal on Computing, 2000, 12(1): 75-82.

[51] Achterberg T. Scip: solving constraint integer programs [J]. Mathematical Programming Computation, 2009, 1: 1-41.

[52] Jansen K, Öhring S. Approximation algorithms for time constrained scheduling [J]. Information and Computation, 1997, 132(2): 85-108.

[53] Jansen K. An approximation scheme for bin packing with conflicts [J]. Journal of Combinatorial Optimization, 1999, 3(4): 363-377.

[54] Gendreau M, Laporte G, Semet F. Heuristics and lower bounds for the bin packing problem with conflicts [J]. Computers & Operations Research, 2004, 31(3): 347-358.

[55] Jensen T R, Toft B. Graph coloring problems [M]. John Wiley & Sons, 2011.

[56] Malaguti E, Toth P. A survey on vertex coloring problems [J]. International Transactions in Operational Research, 2010, 17(1): 1-34.

[57] Wolter K. Implementation of cutting plane separators for mixed integer programs [J]. Dipolma thesis, Technische Universität Berlin, 2006.

[58] Hojny C, Gally T, Habeck O, et al. Knapsack polytopes: a survey [J]. Annals of Operations Research, 2020, 292: 469-517.

[59] Dey S S, Iroume A, Wang G. The strength of multi-row aggregation cuts for sign-pattern integer programs [J]. Operations Research Letters, 2018, 46(6): 611-615.

[60] Nemhauser G L, Wolsey L A. A recursive procedure to generate all cuts for 0–1 mixed integer programs [J]. Mathematical Programming, 1990, 46(1-3): 379-390.

致 谢

我的三年硕士生涯即将结束, 这三年间我得到了身边很多人的帮助. 值此论文完成时, 向关心和帮助我的老师、同门师兄弟姐妹、同学、朋友和家人表示由衷感谢!

首先, 要感谢我的本科兼硕士导师戴彧虹研究员! 还记得本科期间第一次与戴老师见面, 戴老师带着我们去吃了怡宾烤鸭店的烤鸭, 老师平易近人的态度和渊博的知识给我留下深刻的印象. 在这之后的本科生活中, 戴老师每学期都会了解我们的学习生活, 并在关键时刻给予我建议和帮助. 进入研究生阶段后, 在戴老师的指引下, 我接触到了整数规划, 感受到了数学在实际生活中的魅力. 在课题组讨论中, 戴老师会认真指导我们的研究并进一步引导我们思考前沿问题, 使我收获颇丰. 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!

感谢课题组的袁亚湘院士, 袁老师教导我们要将学术问题具象化, 在专注于自己的研究时也要多思考国际上各个相关领域的发展方向. 感谢课题组的刘歆老师, 刘老师在研一时为我们带来了通俗易懂并收获满满的凸分析课程. 感谢课题组的刘亚锋老师, 刘老师在讨论班上以及面试时对我的学习提供了帮助. 感谢课题组的史斌老师、马俊杰老师、孙聪老师在我讨论班上对我所做的工作指出问题提供建议.

感谢优化课题组的所有师兄弟姐妹们. 感谢陈伟坤师兄和陈亮师兄对我科研的指导和帮助. 感谢赵浩天师兄、黄磊师兄、姜博鸥师姐、吉振远师兄、张瑞进师兄、陈圣杰师兄、王磊师兄、陈硕师兄、谢鹏程师兄、王圣超师兄、裴骞师兄、张跃师兄、赵成师兄、胡雨宽师兄、武哲宇师姐、张后山师兄、洛江耀师兄、吕维师兄、章煜海、吴鹏举、王博睿、王薪潼、苏昭纲、张亦、刘上琳、王子岳、汤宇杨师弟、周长宇师弟、李冠达师弟、姜林硕师妹、郭钰琦师妹、王彦儒师妹、黄诗语师妹对我的帮助.

感谢雁栖湖校区 371 宿舍的室友们, 我们在学习上互相帮忙, 共同进步, 在课余一起打牌玩游戏, 放松心情. 感谢我的秋招小伙伴殷亚鸾, 感谢我的自行车老师潘婕妤和刘昊宸, 感谢我的室友廖扬菲, 感谢我的七年同学邓芷芸、金白西和尹杰! 感谢雁栖湖校区丰富的水果, 中关村校区食堂三楼的吊锅鸡, 宿舍门口的煎饼果子和知春路路口的麦当劳, 吃好喝好才能生活好!

最后尤其要感谢我的妈妈, 感谢妈妈对我的包容和照顾, 在我最需要陪伴的时候耐心倾听! 感谢我的姐妹陈理雅, 偶尔联系但是感情依旧!

2023 年 6 月

作者简历及攻读学位期间发表的学术论文与其他相关学术成果

作者简历：

2016 年 9 月——2020 年 6 月, 在中国科学院大学数学与应用数学系获得学士学位.

2020 年 9 月——2023 年 6 月, 在中国科学院数学与系统科学研究院计算数学所攻读硕士学位.