

# 运小筹优化理论教程

OlittleR Tutorial Journal on Optimization



TBSI

清华-伯克利深圳学院  
Tsinghua-Berkeley Shenzhen Institute



Tsinghua University

Tsinghua-Berkeley Shenzhen Institute & Department of Industrial Engineering

卷: 总第 11 期 | 2022 年 3 月 - 2022 年 6 月合集 |

本期作者: 刘兴禄 蔡茂华 张瑞三 修宇璇 于乃康 勾笑盈 徐昶悦

本期题目: 运小筹运筹优化理论教程系列: 第 11 部分  
(OlittleR Tutorial Journal on Optimization: part 11)

本期主题: Benders Decomposition, Two-stage robust optimization, Column and constraint generation, Cutting stock problem, 非线性规划的 Gurobi 求解, C++调用 Gurobi 求解 CVRP, Gurobi 的矩阵建模方法 (MVar), Gurobi 获取非线性规划的对偶值, 二部图匹配, Gurobi 建模案例: 数论方程

本期发布: 刘兴禄

To cite these articles:

```
@misc{OlittleRTeam20220LRTJO,
  title={OlittleR Tutorial Journal on
Optimization: part 11},
  author={Xinglu Liu, Maohua Cai, Ruisan Zhang,
Naikang Yu, Yuxuan Xiu, Xiaoying Gou,
Changyue Xu and OlittleR Optimization
Editorial Team},
  year={2022},
  volume={11},
  pages={1--280},
  howpublished={\url{https://github.com/OlittleRer}}
}
```

推荐引用:

```
@misc{OlittleRTeam20220LRTJO,
  title={运小筹运筹优化理论教程系列: 第 11
部分},
  author={刘兴禄, 蔡茂华, 张瑞三, 于乃康,
修宇璇, 勾笑盈, 徐昶悦 & 运小筹优
化编辑团队},
  pages={1--280},
  year={2022},
  volume={11},
  howpublished={\url{https://github.com/Olit
tleRer}}
}
```



# 目录

0.1 运小筹团队成员 . . . . .	1
0.2 运小筹公众号简介 . . . . .	2
<b>第 1 章 2022-03-17: 【ORers' Bling Chat   【高光聊天记录集锦-02】: 运小筹读者群里那些热烈的讨论 . . . . .</b>	<b>3</b>
1.1 关于退化解的讨论 . . . . .	4
1.2 关于复杂逻辑约束建模 . . . . .	12
1.3 GUROBI 中取变量名的重要性 . . . . .	14
1.4 实现决策变量方差最小 . . . . .	16
<b>第 2 章 2022-03-25: 非线性优化   非线性问题线性化以及求解的详细案例及 Python+Gurobi 求解 . . . . .</b>	<b>18</b>
2.1 非线性数学规划的小案例 . . . . .	18
2.2 第 1 种方法: 完全直接调用 Gurobi 求解 . . . . .	19
2.3 第 2 种方法: 手动线性化 . . . . .	22
2.4 解决 $z =  x $ 的问题 . . . . .	25
<b>第 3 章 2022-03-30: 【ORers' Bling Chat   【高光聊天记录集锦-03】: 运小筹读者群里那些热烈的讨论 . . . . .</b>	<b>29</b>
3.1 关于 Dijkstra 算法和 A* 算法的讨论 . . . . .	30
3.2 关于神经网络的讨论 . . . . .	33
3.3 关于 tight formulation 的讨论 . . . . .	35
3.4 关于融合模型及其子模型的讨论 . . . . .	38
3.5 关于文章投稿的讨论 . . . . .	39
<b>第 4 章 2022-05-02: 优化   寻找新的建模视角——从直观解释对偶问题入手: 以 Cutting Stock Problem 的对偶问题为例 . . . . .</b>	<b>40</b>
4.1 Cutting Stock Problem 简介 . . . . .	41
4.2 Cutting Stock Problem 的 Column generation reformulation . . . . .	43
4.3 Cutting Stock Problem 的小例子 +Python 调用 Gurobi 求解 . . . . .	44

4.4	直观解释 CG 求解 Cutting Stock Problem 对偶问题的含义 . . . . .	49
4.4.1	CG 求解 Cutting Stock Problem 的原问题 . . . . .	49
4.5	其他问题的对偶视角解释 . . . . .	52
4.5.1	桌椅生产问题 . . . . .	52
4.5.2	最短路问题 . . . . .	53
4.6	参考文献 . . . . .	53
<b>第 5 章 2022-05-05: 优化   史上最【皮】数学题求解的新尝试: 一种求解器的视角 (Python 调用 Gurobi 实现) . . . . .</b>		<b>54</b>
5.1	从上海 Pizza 店 HomeSlice 说起 . . . . .	54
5.2	最【皮】数学题: 可以转化为一个运筹优化问题 . . . . .	58
5.3	方法 1: 引入辅助变量进行转化 . . . . .	59
5.4	方法 2: 将表达式展开 . . . . .	62
5.5	如果只要求得到整数解, 是否可以快速求得? . . . . .	66
5.6	总结 . . . . .	71
5.7	参考文献及资料 . . . . .	71
<b>第 6 章 2022-05-12: 优化算法   Benders Decomposition: 一份让你满意的【入门-编程实战-深入理解】的学习笔记 . . . . .</b>		<b>72</b>
6.1	大规模混合整数规划求解算法的综述 . . . . .	72
6.2	分解算法 . . . . .	74
6.3	Benders Decomposition 的算法原理 . . . . .	74
6.4	Benders Decomposition 的完整例子 + 代码实现 . . . . .	81
6.4.1	问题描述 . . . . .	81
6.4.2	MIP 模型 . . . . .	82
6.4.3	用 Benders Decomposition 分解求解小例子: 模型分解 . . . . .	83
6.4.4	代码实现: 逐步迭代版本 . . . . .	86
6.4.5	Benders Decomposition 过程的 Bounds 更新 . . . . .	105
6.4.6	Benders Decomposition 的深入理解 . . . . .	106
6.4.7	小结 . . . . .	116
6.4.8	完整集成版本代码 . . . . .	117
6.5	参考文献和资料 . . . . .	126
<b>第 7 章 2022-05-15: 【ORers' Bling Chat   【高光聊天记录集锦-05】: 运小筹读者群里那些热烈的讨论 . . . . .</b>		<b>127</b>
7.1	关于求解器/模型的问题 . . . . .	128
7.2	学习资料/学习路线 . . . . .	142
7.3	实际操作/论文投稿 . . . . .	143

7.4 启发式相关问题 . . . . .	147
<b>第 8 章 2022-06-02: 优化   二部图最大匹配问题的精确算法详解 (HK 算法和匈牙利算法): 一份让您满意的【理论介绍 + 代码实现】学习笔记 . . . . .</b>	<b>169</b>
8.1 前言 . . . . .	169
8.2 学习 HK 前的扩展 . . . . .	170
8.2.1 二分图 (二部图) . . . . .	170
8.3 增广路径 (Augmenting Path) . . . . .	173
8.4 BFS(Breadth First Search) . . . . .	174
8.4.1 概述 . . . . .	174
8.4.2 迭代过程 . . . . .	174
8.5 DFS(Depth First Search) . . . . .	175
8.5.1 概述 . . . . .	175
8.5.2 迭代过程 . . . . .	175
8.6 匈牙利算法 (KM Algorithm) . . . . .	176
8.7 HK 算法 . . . . .	179
8.7.1 HK 概述 . . . . .	179
8.7.2 HK 迭代过程 . . . . .	179
8.7.3 复杂度 . . . . .	182
8.7.4 代码实现 . . . . .	182
8.8 参考文献和资料 . . . . .	185
<b>第 9 章 2022-06-20: 优化求解器   Gurobi 的 MVar 类: 矩阵建模利器、求解对偶问题的备选方案 (附详细案例 + 代码) . . . . .</b>	<b>186</b>
9.1 前言 . . . . .	186
9.2 优化求解器的建模方式: 以 gurobi 为例 . . . . .	187
9.3 Gurobi 的 MVar (矩阵形式变量对象) . . . . .	188
9.3.1 一个线性规划的例子: Wyndor 玻璃厂 . . . . .	190
9.3.2 问题的建模 . . . . .	190
9.3.3 基于 Gurobi 进行按矩阵形式建模: 完整代码 . . . . .	192
9.3.4 将 Var 对象转成 MVar 对象 . . . . .	193
9.3.5 Model.getA() 函数的使用 . . . . .	194
9.3.6 基于矩阵形式建模的对偶问题的书写及建模 . . . . .	195
9.3.7 基于 Gurobi 进行按矩阵形式进行对偶问题的建模并求解 . . . . .	196
9.3.8 小结 . . . . .	197
9.3.9 参考资料 . . . . .	197

<b>第 10 章 2022-06-24: 优化   手把手教你用 C++ 调用 Gurobi 求解 CVRP: 环境配置、数学模型及完整代码 . . . . .</b>	<b>198</b>
10.1 前言 . . . . .	198
10.2 Visual studio2022 下配置 Gurobi . . . . .	199
10.3 CVRP: 问题描述及数学模型 . . . . .	207
10.4 C++ 调用 Gurobi 求解 CVRP 的完整代码 . . . . .	210
10.4.1 数据集生成 . . . . .	210
10.4.2 添加决策变量 . . . . .	211
10.4.3 添加目标函数 . . . . .	212
10.4.4 最终结果 . . . . .	215
10.5 小结 . . . . .	216
10.6 参考资料 . . . . .	216
<b>第 11 章 2022-06-27: 鲁棒优化   C&amp;CG 算法求解两阶段鲁棒优化: 全网最完整、最详细的【入门-完整推导-代码实现】笔记 . . . . .</b>	<b>217</b>
11.1 两阶段鲁棒优化问题 (Two-stage Robust Optimization) . . . . .	218
11.2 两阶段鲁棒优化问题的详细解读 . . . . .	220
11.3 Two-stage RO 和 Benders-dual cutting plane method . . . . .	222
11.4 The column-and-constraint generation algorithm . . . . .	228
11.5 补充: 为什么用 KKT . . . . .	239
11.5.1 Slater's 条件 . . . . .	240
11.5.2 KKT 条件求解非线性规划的小例子 . . . . .	240
11.6 推导 $SP_2$ 的 inner level 模型的 KKT 条件 . . . . .	242
11.7 C&CG 算法求解 Two-stage RO 的完整简洁步骤 . . . . .	244
11.8 Case study: robust location-transportation problem . . . . .	246
11.8.1 Two-stage robust location-transportation problem . . . . .	246
11.9 Experimental results and discussion . . . . .	248
11.10 Python 调用 Gurobi 求解确定性的模型 . . . . .	255
11.10.1 确定性的 location-transportation problem 的模型 . . . . .	255
11.10.2 Python 调用 Gurobi 完整代码 . . . . .	255
11.11 Python 调用 Gurobi 求解 Two-stage RO model: C&CG algorithm . . . . .	258
11.11.1 C&CG 算法的伪代码 . . . . .	258
11.11.2 C&CG 算法的主问题和子问题 . . . . .	259
11.12 Location-transportation problem: 确定性问题 . . . . .	261
11.12.1 模型 . . . . .	261
11.12.2 Python 调用 Gurobi 求解确定性问题 . . . . .	261

11.13 Two-stage robust location-transportation problem . . . . .	263
11.13.1 两阶段鲁棒优化模型 . . . . .	263
11.13.2 C&CG: 主问题 . . . . .	264
11.13.3 C&CG: 子问题 . . . . .	264
11.14 参考资料 . . . . .	273

## 0.1 运小筹团队成员

运小筹编委：运小筹团队编委（成员）的单位及联系方式如下：

刘兴禄	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:hsinglul@163.com">hsinglul@163.com</a>
何彦东	清华大学，深圳国际研究生院（博士后），邮箱: <a href="mailto:ydhe602@163.com">ydhe602@163.com</a>
段淇耀	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:duanqy71@163.com">duanqy71@163.com</a>
修宇璇	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:yuxuanxiu@gmail.com">yuxuanxiu@gmail.com</a>
曾文佳	清华大学，工业工程系硕士毕业邮箱: <a href="mailto:zwj19@mails.tsinghua.edu.cn">zwj19@mails.tsinghua.edu.cn</a>
夏 <u>国</u>	清华大学，工业工程系硕士毕业邮箱: <a href="mailto:xia970201@gmail.com">xia970201@gmail.com</a>
王梦彤	清华大学，工业工程系博士毕业，新加坡国立大学（博士后）邮箱: <a href="mailto:wmt15@mails.tsinghua.edu.cn">wmt15@mails.tsinghua.edu.cn</a>
段宏达	清华大学，工业工程系，邮箱: <a href="mailto:dhd18@mails.tsinghua.edu.cn">dhd18@mails.tsinghua.edu.cn</a>
王鑫	清华大学，工业工程系，邮箱: <a href="mailto:wangxin16@mails.tsinghua.edu.cn">wangxin16@mails.tsinghua.edu.cn</a>
王涵民	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:humminwang@163.com">humminwang@163.com</a>
张 <u>国</u>	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:zhangyihrm2015@163.com">zhangyihrm2015@163.com</a>
王基光	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:wangjg2020@163.com">wangjg2020@163.com</a>
陈琰钰	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:yanyu_chen_98@163.com">yanyu_chen_98@163.com</a>
周鹏翔	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:zpx20@mails.tsinghua.edu.cn">zpx20@mails.tsinghua.edu.cn</a>
徐璐	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:xu-l20@mails.tsinghua.edu.cn">xu-l20@mails.tsinghua.edu.cn</a>
臧永森	清华大学，工业工程系，邮箱: <a href="mailto:zangys15@tsinghua.org.cn">zangys15@tsinghua.org.cn</a>
左齐茹仪	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:zqry20@mails.tsinghua.edu.cn">zqry20@mails.tsinghua.edu.cn</a>
张婷	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:15927584840@163.com">15927584840@163.com</a>
杨影	清华大学，工业工程系，邮箱: <a href="mailto:1637352453@qq.com">1637352453@qq.com</a>
陈懋宁	清华大学，工业工程系，邮箱: <a href="mailto:catme1008@163.com">catme1008@163.com</a>
刘晨宇	清华大学，工业工程系，邮箱: <a href="mailto:liuchenyu0326@163.com">liuchenyu0326@163.com</a>
赖克凡	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:laikf21@mails.tsinghua.edu.cn">laikf21@mails.tsinghua.edu.cn</a>
曹可欣	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:caokx21@mails.tsinghua.edu.cn">caokx21@mails.tsinghua.edu.cn</a>
李心怡	清华大学，深圳国际研究生院，邮箱: <a href="mailto:yzlixinyi@163.com">yzlixinyi@163.com</a>
国昕怡	清华大学，深圳国际研究生院，邮箱: <a href="mailto:xinyiguoellen@163.com">xinyiguoellen@163.com</a>
勾笑盈	中国科学技术大学，管理学院，邮箱: <a href="mailto:xygou@mail.ustc.edu.cn">xygou@mail.ustc.edu.cn</a>
蔡茂华	莆田学院，数学与金融学院，邮箱: <a href="mailto:cmh001212@163.com">cmh001212@163.com</a>
张瑞三	四川大学，商学院，邮箱: <a href="mailto:sum3rebirth@gmail.com">sum3rebirth@gmail.com</a>
徐昶悦	同济大学，交通运输工程学院，邮箱: <a href="mailto:changyxu@163.com">changyxu@163.com</a>
张一白	香港科技大学（广州），邮箱: <a href="mailto:zhangybailiaoyang@outlook.com">zhangybailiaoyang@outlook.com</a>
樵溪子	清华大学，清华-伯克利深圳学院，邮箱: <a href="mailto:qiao_xizi@163.com">qiao_xizi@163.com</a>

## 0.2 运小筹公众号简介

本公众号致力于分享运筹优化 (LP、MIP、NLP、随机规划、鲁棒优化)、凸优化、强化学习等研究领域的内容以及涉及到的算法的代码实现。编程语言和工具包括 Java、Python、Matlab、CPLEX、Gurobi、SCIP 等。

欢迎读者朋友们加入我们的读者群，大家一起探讨学术问题。具体加群方式，见微信公众号最新一期的推文末尾。

## Chapter 1

# 2022-03-17: 【ORers' Bling Chat | 【高光聊天记录集锦-02】: 运小筹读者群 里那些热烈的讨论

作者: 运小筹读者群 1~6 的各位读者朋友们

图文整理: 蔡茂华, 莆田学院, 数学与金融学院, 本科在读

审校: 刘兴禄, 清华大学, 博士在读

这里特此声明, 群聊内容不保证 100% 正确, 不保证 100% 可信, 如有错误, 纯属正常, 另外, 希望发现错误的小伙伴们不吝赐教, 在留言区留言哦

注: 为方便理解问题, 整理过程中有筛选合并等, 敬请谅解。

BTW, 本部分聊天记录是运小筹读者 2 群在 2022 年春节大年三十到大年初七之间的精彩讨论。

## Chapter 2

# 2022-03-25: 非线性优化 | 非线性问题 线性化以及求解的详细案例及 Python+Gurobi 求解

刘兴禄, 清华大学, 清华-伯克利深圳学院, 博士在读

在数学规划问题中, 很多时候我们会遇到非线性的目标函数或者约束。此时一些小伙伴也许会不知所措。之前, 我们写过一篇推文来讲解如何处理这样的问题。链接如下:

这里, 我们再提供一个小案例, 以丰富这方面的参考资料。这个例子来源于一个师弟, 他最近跟我讨论了这个问题, 我觉得正好可以作为一个例子来介绍如何使用 Gurobi 求解非线性问题。

### 2.1 非线性数学规划的小案例

考虑下面的最小化问题。

$$\min f(x, y) \tag{2.1.1}$$

$$s.t. (x - 1)^2 + (y - 1)^2 - 1 \leq 0 \tag{2.1.2}$$

$$|x| + y - 2 \leq 0 \tag{2.1.3}$$

其中,  $f(x, y) = \max\{|x|, y + 4\}$ .

可以看到, 目标函数是一个带  $\max$  的函数, 是非线性的; 第一个约束是 2 次方, 第二个约束带绝对值。

## Chapter 3

# 2022-03-30: 【ORers' Bling Chat | 【高光聊天记录集锦-03】：运小筹读者群 里那些热烈的讨论

作者：运小筹读者群 1~6 的各位读者朋友们

图文整理：徐昶悦，同济大学，博士在读

审校：刘兴禄，清华大学，博士在读

这里特此声明，群聊内容不保证 100% 正确，不保证 100% 可信，如有错误，纯属正常，另外，希望发现错误的小伙伴们不吝赐教，在留言区留言哦

注：为方便理解问题，整理过程中有筛选合并等，敬请谅解。

## Chapter 4

# 2022-05-02: 优化 | 寻找新的建模视角 ——从直观解释对偶问题入手：以 Cutting Stock Problem 的对偶问题 为例

刘兴禄，清华大学，清华-伯克利深圳学院，博士在读

本文致谢：感谢【运小筹读者 2 群】的小伙伴们与我就该话题展开的热烈而有用讨论。以及幸新杰老师提供的有用的信息。

Cutting Stock Problem 是运筹优化中一个非常重要的问题。该问题与一个非常强大的算法框架【Column Generation】直接相关。本节我们从对偶的角度，来理解 Cutting Stock Problem。

另外，很多人懂的 Column Generation 的过程，但是也许有些读者并不了解 Column Generation 迭代过程中，问题的 Lower bound 如何计算。因为一些读者在 Column Generation 的过程中，仅仅关注子问题的 reduced cost 是否为负，不需要去关心每一步 Column Generation 的迭代中，全局的 Upper bound 和 Lower bound。因为在 CG 结束迭代的时候，CG reformulation 得到的全局 Lower bound 恰好等于当前 RMP 的线性松弛 RLMP 的 LP relaxation 解。也就是此时

$$\text{global LB} = z_{\text{RLMP}}$$

所以大家不关心也没关系。不过，知道 Lower bound 如何计算，会让你对整个算法理解更上一层楼，这篇推文，我们就与大家共同推导一下 Column Generation 求解 Cutting Stock Problem 的过程中，Lower bound 如何计算。

## Chapter 5

# 2022-05-05：优化 | 史上最【皮】数学题求解的新尝试：一种求解器的视角（Python 调用 Gurobi 实现）

作者：蔡茂华，莆田学院，数学专业本科在读

作者：刘兴禄，清华大学，清华-伯克利深圳学院，博士在读

### 5.1 从上海 Pizza 店 HomeSlice 说起

最近，由于新冠疫情，上海已经封城数日。随着疫情逐渐好转，一些单位开始了复工复产。这几天，微博上一则关于上海一家 Pizza 店复工复产的消息引起了大家的注意。

在此，希望上海疫情早日结束，希望全世界疫情早日结束，大家平平安安!!! 也希望大家能多多运用运筹优化的理论，来帮助提高防疫的效率，发挥运筹优化的力量!!!

事情是这样的，\*\* 上海有一家名为 HomeSlice (斯莱仕披萨) 的 Pizza 店复工复产了，但是由于 Pizza 店生产能力有限，导致 Pizza 供应不足，Pizza 店不得不限制配送资格的发放。但是独特的是，该 Pizza 店提高平日送资格获取门槛的方式是：做数学题!!! 这个操作让网友们眼前一亮。\*\*

声明：该事件是否真实存在，小编不得而知。只是觉得有趣，拿来跟大家一起学习优化而已。

先不看数学题，小编想先看看这家店具体在哪里，哈哈。小编上百度地图查了一下这家店，发现 HomeSlice 在上海有多家分店，比如：

- HOMESLICE 斯莱仕披萨 (浦东首店)
- HOMESLICE 斯莱仕披萨 (158 坊店)

## 5.4. 方法 2: 将表达式展开

运筹学理论教程 (OlittleR Tutorial Journal on Optimization): 2022 年 3 月- 2022 年 6 月合集 (总第 11 期)

245726741	47000	infeasible	17740		1.00000	-	0.4 6470s
244022350	49700	1.00000	21225	1	-	1.00000	- 0.4 6475s
244118852	49725	1.00000	18652	1	-	1.00000	- 0.4 6480s
244213871	49746	infeasible	23472		-	1.00000	- 0.4 6485s
244311170	49734	1.00000	19056	1	-	1.00000	- 0.4 6490s
244406488	49769	1.00000	28980	1	-	1.00000	- 0.4 6495s
244503284	49783	1.00000	24842	1	-	1.00000	- 0.4 6500s
244594978	49801	infeasible	16379		-	1.00000	- 0.4 6505s
244694304	49789	1.00000	32280	1	-	1.00000	- 0.4 6510s
244788779	49854	1.00000	16701	1	-	1.00000	- 0.4 6515s
244880189	49875	1.00000	18931	1	-	1.00000	- 0.4 6520s
244974158	49842	1.00000	18164	1	-	1.00000	- 0.4 6525s
245070205	49876	1.00000	20965	1	-	1.00000	- 0.4 6530s
245163156	49904	infeasible	29051		-	1.00000	- 0.4 6535s
245258236	49908	infeasible	20671		-	1.00000	- 0.4 6540s
245353026	49925	infeasible	25658		-	1.00000	- 0.4 6545s
245427494	49953	1.00000	18750	1	-	1.00000	- 0.4 6550s
245500282	49928	infeasible	21721		-	1.00000	- 0.4 6555s
245594520	49945	1.00000	20240	1	-	1.00000	- 0.4 6560s
245688831	50001	infeasible	16315		-	1.00000	- 0.4 6565s
245774819	49995	1.00000	19378	1	-	1.00000	- 0.4 6570s
245856604	49983	infeasible	20282		-	1.00000	- 0.4 6575s
245950883	49969	infeasible	16123		-	1.00000	- 0.4 6580s
246040305	49960	infeasible	22134		-	1.00000	- 0.4 6585DN @若尘1212

Figure 5.4.1: 求解日志

可以看到, Gurobi 的 Branch and cut tree 都已经探寻了 2 亿个点了, 但是还是找不到解。由此小编意识到了问题的严重性, 所以开始网上冲浪寻求这道题的真正答案。

果然问题很严重, 正确答案这是一个结果长达 80 位的整数。如下所示

$$a = 154476802108746166441951315019919837485664325669565431700026634898253202035277999, \quad (5.4.18)$$

$$b = 36875131794129999827197811565225474825492979968971970996283137471637224634055579, \quad (5.4.19)$$

$$c = 4373612677928697257861252602371390152816537558161613618621437993378423467772036 \quad (5.4.20)$$

好家伙, 怪不得 Gurobi 找不到正整数可行解呢, 这也有点太出乎意料了。

小编们不禁感叹: 这题也太【皮】了。80 位的巨无霸整数, 逗呢? 哈哈哈哈

如果大家对原问题的解析解法有兴趣的可以以下链接看看: <https://zhuanlan.zhihu.com/p/59072397>.

## 5.5 如果只要求得到整数解，是否可以快速求得？

既然正整数解这么皮，那我们只要求找到一组整数解呢？

于是我转移了目标：既然很难算出方程等于 4 的正整数解，那我可以算算满足条件的整数解。因此，我们可以将原模型更改为

$$\min 1 \quad (5.5.1)$$

$$s.t. \quad x_1^3 + x_1^2 x_2 + x_1^2 x_3 + x_1 x_2 x_3 + x_1 x_2^2 + x_2^3 + x_1 x_2 x_3 + x_2^2 x_3 + x_1 x_2 x_3 + x_2 x_3^2 + x_1 x_3^2 + x_3^3 \\ (5.5.2)$$

$$= 4(x_1^2 x_2 + x_1^2 x_3 + x_2^2 x_1 + x_2^2 x_3 + x_2 x_3^2 + x_1 x_3^2 + 2x_1 x_2 x_3) \quad (5.5.3)$$

$$x_1 + x_2 \neq 0 \quad (5.5.4)$$

$$x_1 + x_3 \neq 0 \quad (5.5.5)$$

$$x_2 + x_3 \neq 0 \quad (5.5.6)$$

$$x_i \text{ integer}, \quad \forall i \in 1, 2, 3 \quad (5.5.7)$$

但是， $x_1 + x_2 \neq 0$  这种约束无法直接在求解器中添加，因此，我们需要引入辅助变量和辅助约束。

$$x_i \neq 0 \quad (5.5.8)$$

就等价于

$$x_1 + x_2 > 0 \text{ or } x_1 + x_2 < 0 \quad (5.5.9)$$

而求解器的约束，只能是  $\geq, \leq, =$  的三者之一，不能是  $>, <$ ，因此我们必须再次进行转化。

注：求解器的约束，只能是  $\geq, \leq, =$  的三者之一，不能是  $>, <$ 。

为此，我们引入一个较小的正数  $\epsilon = 0.00001$ ，来完成这部分约束：

$$|x_1 + x_2| \geq \epsilon \quad (5.5.10)$$

$$|x_1 + x_3| \geq \epsilon \quad (5.5.11)$$

$$|x_2 + x_3| \geq \epsilon \quad (5.5.12)$$

上面三个绝对值约束仍然无法直接用 Gurobi 实现，还需要进一步转化。

为此，在调用求解器的时候，我们需要引入额外的辅助变量来表示左边绝对值的值，比如  $u_1, u_2, u_3$ ，并令

$$u_1 = |x_1 + x_2| \quad (5.5.13)$$

$$u_2 = |x_1 + x_3| \quad (5.5.14)$$

$$u_3 = |x_2 + x_3| \quad (5.5.15)$$

但是，上述模型，还是不能直接求解，因为 Gurobi 的添加绝对值约束的函数为

- `Model.addGenConstrAbs(resvar, argvar)`: 表示添加约束 `resvar==abs(argvar)`，只能是单个变量的操作，不能写多个的和。

因此，我们还要再次更改，变成

$$u_1 = x_1 + x_2 \quad (5.5.16)$$

$$u_2 = x_1 + x_3 \quad (5.5.17)$$

$$u_3 = x_2 + x_3 \quad (5.5.18)$$

$$u_1^{abs} = |u_1| \quad (5.5.19)$$

$$u_2^{abs} = |u_2| \quad (5.5.20)$$

$$u_3^{abs} = |u_3| \quad (5.5.21)$$

这样做才能直接调用 Gurobi 实现。

自此，改模型可以直接使用 Gurobi 求解了，也就是  $x_i \neq 0$  的约束就被很好的处理了。模型的完整形式如下：

$$\min 1 \quad (5.5.22)$$

$$\begin{aligned} s.t. \quad & x_1^3 + x_1^2 x_2 + x_1^2 x_3 + x_1 x_2 x_3 + x_1 x_2^2 + x_2^3 + x_1 x_2 x_3 + x_2^2 x_3 + x_1 x_2 x_3 + x_2 x_3^2 + x_1 x_3^2 + x_3^3 \\ & = 4(x_1^2 x_2 + x_1^2 x_3 + x_2^2 x_1 + x_2^2 x_3 + x_2 x_3^2 + x_1 x_3^2 + 2x_1 x_2 x_3) \end{aligned} \quad (5.5.23)$$

$$u_1 = x_1 + x_2 \quad (5.5.25)$$

$$u_2 = x_1 + x_3 \quad (5.5.26)$$

$$u_3 = x_2 + x_3 \quad (5.5.27)$$

$$u_1^{abs} = |u_1| \quad (5.5.28)$$

$$u_2^{abs} = |u_2| \quad (5.5.29)$$

$$u_3^{abs} = |u_3| \quad (5.5.30)$$

$$u_i^{abs} \geq \epsilon, \quad \forall i = 1, 2, 3 \quad (5.5.31)$$

$$x_i \text{ integer}, \quad \forall i \in 1, 2, 3 \quad (5.5.32)$$

$$u_i \text{ free}, \quad \forall i \in 1, 2, 3 \quad (5.5.33)$$

我们现在用 Python 调用 Gurobi 求解该问题, 代码如下

Code

```

1  from gurobipy import *
2  m = Model("test")
3  m.setParam('NonConvex', 2) # 非凸模型求解参数
4  a = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="a")
5  b = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="b")
6  c = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="c")
7  m1 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m1")
8  m2 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m2")
9  m3 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m3")
10 m4 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m4")
11 m5 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m5")
12 m6 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m6")
13 m7 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="m7")
14 u1 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="u1")
15 u2 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="u2")
16 u3 = m.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="u3")
17 abs_u1 = m.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="abs_u1")
18 abs_u2 = m.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="abs_u2")
19 abs_u3 = m.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.INTEGER, name="abs_u3")
20 lhs = m.addVar(vtype=GRB.INTEGER, name="lhs")
21 rhs = m.addVar(vtype=GRB.INTEGER, name="rhs")
22
23 m.addConstr(m1 == a*a)      #a*a
24 m.addConstr(m2 == b*b)      #b*b
25 m.addConstr(m3 == c*c)      #c*c
26 m.addConstr(m4 == a*b)      #a*b
27 m.addConstr(m5 == a*c)      #a*c
28 m.addConstr(m6==b*c)       #b*c
29 m.addConstr(lhs==m1*(a+b+c)+3*m4*c+(a+b+c)*m2+m3*(a+b+c))    # 左项式
30 m.addConstr(rhs==4*(m1*(b+c)+m2*(a+c)+(a+b)*m3+2*c*m4))        # 右项式
31 m.addConstr(lhs==rhs)      # 左右项式相等
32
33 m.addConstr(u1 == a + b)
34 m.addConstr(u2 == a + c)
35 m.addConstr(u3 == b + c)

```

```

36
37 m.addGenConstrAbs(abs_u1, u1)
38 m.addGenConstrAbs(abs_u2, u2)
39 m.addGenConstrAbs(abs_u3, u3)
40
41 m.addConstr(abs_u1 >= 0.00001)
42 m.addConstr(abs_u2 >= 0.00001)
43 m.addConstr(abs_u3 >= 0.00001)
44
45 m.setObjective(1, GRB.MINIMIZE)
46 m.optimize()
47 a1=a.x
48 b1=b.x
49 c1=c.x
50 print('a: ',a1)
51 print('b: ',b1)
52 print('c: ',c1)

```

最后算出的求解结果和日志如下:

Code

```

1 resolve added 2 rows and 7 columns
2 Presolve time: 0.00s
3 Presolved: 72 rows, 36 columns, 171 nonzeros
4 Presolved model has 6 SOS constraint(s)
5 Presolved model has 16 bilinear constraint(s)
6 Variable types: 1 continuous, 35 integer (6 binary)
7
8 Root relaxation: objective 1.000000e+00, 19 iterations, 0.00 seconds
9
10      Nodes    |    Current Node    |    Objective Bounds        |     Work
11      Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
12
13          0      0  1.000000      0    10      -    1.000000      -      -    0s
14          0      0  1.000000      0     2      -    1.000000      -      -    0s
15          0      0  1.000000      0     2      -    1.000000      -      -    0s
16          0      2  1.000000      0     2      -    1.000000      -      -    0s
17      555679  9368  1.000000    329     1      -    1.000000      -    0.9    5s
18 *1091923  932                  34      1.0000000    1.000000  0.00%   1.2    9s
19
20 Cutting planes:
21   Gomory: 1
22
23 Explored 1093808 nodes (1307260 simplex iterations) in 9.91 seconds
24 Thread count was 16 (of 16 available processors)
25
26 Solution count 1: 1
27
28 Optimal solution found (tolerance 1.00e-04)
29 Best objective 1.000000000000e+00, best bound 1.000000000000e+00, gap 0.0000%
```

```

30 a: -1.0
31 b: 11.0
32 c: 4.0

```

可以看到，模型得到了一个可行解

$$a = -1 \quad (5.5.34)$$

$$b = 11 \quad (5.5.35)$$

$$c = 4 \quad (5.5.36)$$

$$\frac{a}{b+c} + \frac{b}{a+c} + \frac{c}{a+b} = \frac{-1}{11+4} + \frac{11}{-1+4} + \frac{4}{-1+11} = 4 \quad (5.5.37)$$

完美。

另外，小编也调整了一下参数，获得了其他的可行解。

比如

$$a = -330 \quad (5.5.38)$$

$$b = -120 \quad (5.5.39)$$

$$c = 30 \quad (5.5.40)$$

$$\frac{a}{b+c} + \frac{b}{a+c} + \frac{c}{a+b} = 4 \quad (5.5.41)$$

```

Optimal solution found (tolerance 1.00e-04)
Best objective 1.00000000000e+00, best bound 1.00000000000e+00, gap 0.0000%
a: -330.0
b: -120.0
c: 30.0

```

CSDN @若尘1212

Figure 5.5.1: 求解结果

可见，得到整数解还是非常容易的（不止一组答案，有很多组）。

那我们现在再来看看，方程

$$\frac{a}{b+c} + \frac{b}{a+c} + \frac{c}{a+b} = k \quad (5.5.42)$$

$k$  等于 3、2、1 时的正整数解求解情况。

- 当  $k = 3$  时：求解起来也很费劲，跑了很久也没有答案，所以小编放弃了。
- 当  $k = 2$  时：求解起来很快，得到了  $a = 1 b = 1 c = 3$  的一组结果
- 当  $k = 1$  时：求解显示无解。

从上面的求解情况来看，我们发现在这个模型中，当  $k$  逐渐变大后，求解难度是呈爆炸级别增长的。

小编经常会看到有人会在问：\*\*“我试了一个决策变量为  $x$  个的模型，求解起来很快啊！但为什么决策变量增加到  $x + 2$  个，求解器就跑得很慢。”\*\*

对于这个问题，仅仅是将  $k$  从 2 改为 4，结果就从 1 位数爆炸增长成 80 位数，其他问题虽然改变一些条件获取解的难度不会增长得这么恐怖。但是一个模型的参数、规模改变，或多或少都会对求解进程造成影响。至于影响情况是怎样的？还得聪明的读者们去针对于自己的模型深入分析。

## 5.6 总结

本篇推文以最近微博发布的一则引起网友热议的 Pizza 店事件为背景，将其采用的数学题，从优化的角度进行了探讨。旨在以此为例，讲解对于一些有趣的问题，如何从优化的角度来进行求解。

本文中涉及到了非线性规划模型的多个转化技巧，以及求解器使用的注意事项。可以帮助广大运筹优化初学者迅速掌握如何处理模型中的非线性项。

当然，本文对于不是搞运筹优化研究的其他专业的小伙伴也会有一些启发。除了使用解析的办法，运筹优化提供了一些新的视角，感兴趣的其他小伙伴也可以多多关注运筹优化领域，也许，你的问题的解决方案，就藏在运筹优化中哦。

## 5.7 参考文献及资料

- [1]. Bremner A, Macleod A. An unusual cubic representation problem[C]//Annales Mathematicae et Informaticae. 2014: 29-41.
- [2]. 微博 ID 为【兔撕鸡大老爷】的微博正文, 2022.05.01
- [3]. 知乎: <https://zhuanlan.zhihu.com/p/59072397>

## Chapter 6

# 2022-05-12: 优化算法 | Benders Decomposition: 一份让你满意的【入门-编程实战-深入理解】的学习笔记

作者: 刘兴禄, 清华大学, 清华-伯克利深圳学院, 博士在读

初次完稿日期: 2020.10.08

### 6.1 大规模混合整数规划求解算法的综述

大规模混合整数规划 (mixed integer programming, MIP) 的求解在运筹优化中具有至关重要的地位。这里我们强调一下, 不做特殊说明的话, 我们考虑的都是混合整数线性规划 (mixed integer linear programming, MILP)。求解 MIP 的常用精确算法主要包括:

- branch and bound
- cutting plane
- branch and cut
- column generation(不保证得到最优解, 需要结合其他技巧获得最优解)
- branch and price
- Lagrangian relaxation(不保证得到最优解, 需要结合其他技巧获得最优解)
- Dantzig-Wolfe decomposition
- Benders decomposition
- ...

其他衍生的拓展方法, 本文不做介绍。

其中, branch and bound、branch and cut 是非常通用的求解框架, 适用于所有的 MIP。

Lagrangian relaxation 并不能保证得到最优解，但是运气好的时候，也可以得到最优解。不过 Lagrangian relaxation 可以得到比线性松弛 (LP relaxation) 更紧的界限 (Bound)，因此可以用来加速 branch and bound 或者 branch and cut。

而 column generation 和 branch and price 这两个框架的通用性就弱很多。这两个框架都是基于模型重构的。只有可以重构为相应问题的模型，才可以使用这两种方法。此外，column generation 不能保证获得全局最优解（具体原因可以查看相关文献，也可以阅读笔者即将出版的教材，见参考文献 1）。而将 branch and bound 的框架嵌入 column generation 中，构成 branch and price 框架，则可以保证获得全局最优解。

以上精确算法中：

- branch and bound 是一种分而治之的思想，本质上是一种隐枚举 (branching)。但是由于加入了 prune(剪枝) 和 bounding (定界) 的步骤，使得该框架在实际求解中要比真正的纯枚举要高效得多。

- branch and cut 是目前最流行的求解 MIP 的求解器的通用算法框架。由于其在 branch and bound 的基础上，加入了 cutting plane 的步骤，用于割去当前节点的最优小数解，从而逼近该节点的凸包，从而显著地加速了求解过程。

- column generation 和 branch and price 是基于模型重构而来的算法。通常是将原来的 MIP 分解成为一个主问题 (Master Problem) 和若干个子问题 (Subproblem)，然后迭代求解。当然，子问题又叫定价子问题 (Pricing Problem)。由于分解之后，各个部分的求解相对容易，以及主问题一般是更为紧凑的模型，因此会提供更好的界限，从而加速收敛。这些框架都是融合了模型重构和分解的思想。

- Lagrangian relaxation 是一种松弛的思想。通过结合对偶理论，得到比单纯的线性松弛更好的界限。

- Dantzig-Wolfe decomposition 和 Benders decomposition 是根据模型的特殊结构，将模型分解为更容易求解的小问题，通过小问题之间的迭代求解和交互，最终达到精确求解模型的目的的精确算法。但是二者的分解思路并不相同。Dantzig-Wolfe decomposition 是基于一个表示定理得来的分解方法，该方法需要 MIP 的约束矩阵符合块角状的特征，通用性有限，使用之前需要考察模型是否符合该结构。而 Benders decomposition 实际上是较为通用的分解框架，CPLEX 中也包含了 Benders decomposition 的算法组件，用户可以自己定制分解策略。Benders decomposition 的主要思想是，将较复杂的模型分解成为 2 部分，分别求解 2 部分都较为容易，通过两部分之间的交互迭代，最终算法得到收敛，得到最优解。

每一种精确算法都不存在绝对的优劣关系，它们各有特点，都蕴含着精妙的思想。在实际情况中，我们需要灵活应用，巧妙结合，从而达到显著加速求解的目的。

本文就来着重介绍分解算法中的重要成员：Benders decomposition。这里强调一下，本文涉及的 Benders decomposition，是最基本的 Benders decomposition，更高级别的 Benders decomposition 的拓展算法，请读者自行阅读相关文献。

## 6.2 分解算法

上面讲到，分解算法，是基于模型的特点，将大问题拆解称为若干小问题。单独求解这些小问题的难度，远低于直接求解原来的大模型。这些小问题之间需要进行一定的交互，从而保证整分解后的模型能够收敛到原来模型的最优解，从而保证分解方法的全局最优性。

这里需要明确，分解  $\neq$  拆解。分解的操作，实际上是有艺术性和理论性的。

分解包括拆解 + 耦合。其中拆解就是将问题拆成若干小问题，耦合就是将各个小问题联系起来，形成交互，从而保证全局最优性。

也许目前的直观解释还不够具体，不过没关系，我们先说清楚整体的思想，然后再以具体的理论加以解释。

## 6.3 Benders Decomposition 的算法原理

考虑如下的 MIP 模型：

$$\min_{x,y} \quad cx + fy \quad (6.3.1)$$

$$\text{s.t.} \quad Ax + By = b \quad (6.3.2)$$

$$x \geq 0 \quad (6.3.3)$$

$$y \in Y \subseteq \mathbb{R}^n \quad (6.3.4)$$

其中  $c \in \mathbb{R}^{1 \times m}$ ,  $f \in \mathbb{R}^{1 \times n}$ , 是行向量,  $x, y$  是列向量, 且  $x, y$  是决策变量, 其维度分别为  $m \times 1$  和  $n \times 1$ 。 $A, B$  是约束矩阵, 其维度分别为  $A \in \mathbb{R}^{r \times m}$ ,  $B \in \mathbb{R}^{r \times n}$ 。 $b$  为右端常数, 其维度为  $b \in \mathbb{R}^{r \times 1}$ 。注意这里的符号,  $\mathbb{R}^{r \times m}$  其实就是表示是一个  $r \times m$  的一个实数矩阵。

该 MIP 模型具有下面的特点：

-  $x$  是连续型 (continuous) 决策变量, 较为简单; -  $y$  是复杂决策变量, 可以认为是 0-1 型 (binary) 决策变量或者整数 (integer) 型决策变量, 相较  $x$  而言, 略复杂, 因此在模型中我们用了数学语言  $y \in Y \subseteq \mathbb{R}^n$  来表达。前一个  $y \in Y$ , 这个  $Y$  可以是一系列的  $y \in \{0, 1\}$  的约束, 也可以是  $y$  integer 的约束。当然, 还可以是  $y \geq 0$  的连续约束, 都可以。为了方便, 我们统一用  $y \in Y$  来表示。

由于  $x$  为简单变量,  $y$  为复杂变量, 当规模较大时, 直接求解上述 MIP 比较困难。这个困难的原因是：- 复杂变量  $y$  搅合进来了, 这个罪魁祸首给问题求解带来了挑战。

因此我们设想：如果我们将这个捣蛋者  $y$  先搞定, 剩下的部分中, 只有  $x$  是决策变量, 问题就变成了线性规划 (Linear programming, LP) 了, 这不就简单太多了嘛?

Benders Decomposition 就是基于这样的思想。

我们观察到, 如果给定  $y$  的值, 假定为  $\bar{y}$ , 那么剩余部分的模型就可以写成

$$\text{SP: } \min_x \quad cx \quad (6.3.5)$$

$$\text{s.t. } Ax = b - B\bar{y} \quad (6.3.6)$$

$$x \geq 0 \quad (6.3.7)$$

该模型是 LP, 求解难度比之前的 MIP 降低了好几档次。因为 MIP 是 NP-hard 问题, 而 LP 是多项式时间可精确求解的, 不是 NP-hard。该问题一般被称之为子问题 (Subproblem problem, SP)。

如何给出  $y$  的值  $\bar{y}$  呢? 我们可以将关于  $x$  的部分全部忽略掉, 变成下面的模型

$$\text{MP}_0 : \quad \min_y \quad fy \quad (6.3.8)$$

$$\text{s.t. } y \in Y \subseteq \mathbb{R}^n \quad (6.3.9)$$

求解  $\text{MP}_0$  非常容易, 求解得到的解, 即可作为  $\bar{y}$  传给 SP。该问题一般被称之为为主问题 (Masterproblem, MP)。 $\text{MP}_0$  加了下标 0 表示初始的 MP(因为后续迭代过程, MP 需要加入 cut)。

但是, 分别求解模型 MP 和 SP 并不能等同于求解了原 MIP。要想达到等价地求解原 MIP 的目的, 还需要 MP 和 SP 之间的交互。

为什么要交互呢?

原因是:

- MP 删除了所有关于  $x$  的约束, 相当于抛弃了  $x$  的影响。但是实际上  $x$  一定会影响 MP。我们通过一种先试错, 再补救的过程, 先删除所有的  $x$  的相关部分, 构成初始的 MP, 通过求解 SP, 获得一些信息, 这些信息反映了  $x$  对  $y$  的影响, 我们通过某种方式, 将  $x$  对  $y$  的影响加回到 MP 中, 进行补救。通过整个过程的迭代, 我们最终就可以达到求解原 MIP 的目的。

其中, 补救的步骤, 实际上专业术语叫做 recourse, 反应在具体形式上, 就是以 cutting plane 的形式, 将补救措施加回到 MP 中。

至于如何补救补救, 那就需要用到对偶理论的知识。首先说明, Benders Decomposition 分解的补救措施, 是以两种 cutting plane 的形式加回到 MP 中的, 分别为 - Benders optimality cut 和

- Benders feasibility cut。

这两种 Cut 来源于 SP 的对偶, 由于 SP 是 LP, 因此我们可以将 SP 对偶, 变成 Dual SP, 其形式如下:

## Chapter 8

# 2022-06-02: 优化 | 二部图最大匹配问题的精确算法详解 (HK 算法和匈牙利算法): 一份让您满意的【理论介绍 + 代码实现】学习笔记

作者: 张瑞三, 四川大学, 硕士在读

审校: 刘兴禄, 清华大学, 清华-伯克利深圳学院

### 8.1 前言

二分图最大匹配问题是一个经典问题, 顾名思义就是寻找二分图两个独立集合中的最大匹配 (最大匹配中匹配指的是一对一匹配), 例如相亲活动中, 男生集合和女生集合, 相互有意的即存在配对的可能 (不排除花心大萝卜的存在, 同时对多个异性有好感), 相亲成功的最多的配对情况就是我们要找的最大匹配。在很多领域有重要应用, 例如求最小点覆盖或者最大独立集, 但是它却可以在多项式时间内精确求解, 因此该问题不是 NP hard。目前已经有很多种在多项式时间内精确求解二部图最大匹配问题的算法, 如 Kuhn - Munkres algorithm (匈牙利算法) 和 Hopcroft-Karp algorithm(以下简称 HK 算法) 等。今天我们介绍的主角就是 HK 算法, 它是一种非常高效的求解二部图最大匹配的一种精确算法 (复杂度为  $O(|E| * \sqrt{V})$ , 这里的  $V$  是图中点数,  $E$  是边数) 后面我们会进行证明), 也就是说——该算法可以得到最优解。HK 算法通过不断寻找图中的增广路, 从而找到可以提升当前解的方向, 进而通过算法迭代, 不断改进当前解, 直到找不到任何增广路为止, 此时, 说明当前解已经达到最优。在最后我们也提供了 HK 算法中寻找增广路径的 python 代码供以参考。霍普克洛夫特 - 卡

普算法 (Hopcroft–Karp algorithm) 该算法由 John Hopcroft 和 Richard Karp (1973) 发现。正如之前的匹配方法, 如匈牙利算法和 Edmonds (1965) 的工作一样, Hopcroft-Karp 算法通过寻找增广路径反复增加部分匹配的大小: 在进入和离开之间交替的边缘序列。

## 8.2 学习 HK 前的扩展

在学习 HK 算法之前, 需要补充一些知识点, 以便更好地理解算法结构。首先我们解决的是二分图的最大匹配问题, 所以我们需要先对二分图进行了解; 其次我们去找最大匹配的时候是通过迭代寻找增广路径, 所以增广路径的定义需要明确。看到目录的同学可能会问, 是讲 HK 算法的, 为什么学前还要学习匈牙利算法 (KM) 呀? 回答: 实际上 HK 算法是在 KM 算法上的优化, KM 算法在每次迭代中, 只是找到一条增广路, 然后基于该增广路, 对当前解进行改进。但是每次迭代中, 却往往存在多条增广路, 如果能找到多条增广路, 算法效果会得到进一步提升。而 HK 算法正式利用了这一点, 在每一次算法迭代中, 同时使用了 DFS 和 BFS, 因此可以在单次迭代中找到多条增广路, 从而进一步提升了效率, 缩小了算法复杂度。

### 8.2.1 二分图 (二部图)

定义: 二分图 (又名二部图), 是图论中的一种特殊模型。设  $G = (V, E)$  是一个无向图, 如果顶点  $V$  可分割为两个互不相交的子集  $(A, B)$ , 并且图中的每条边  $(i, j)$  所关联的两个顶点  $i$  和  $j$  分别属于这两个不同的顶点集 ( $i \in A, j \in B$ ), 则称图  $G$  为一个二分图。简而言之, 就是顶点集  $V$  可分割为两个互不相交的子集, 并且图中每条边依附的两个顶点都分属于这两个互不相交的子集, 两个子集内的顶点不相邻。

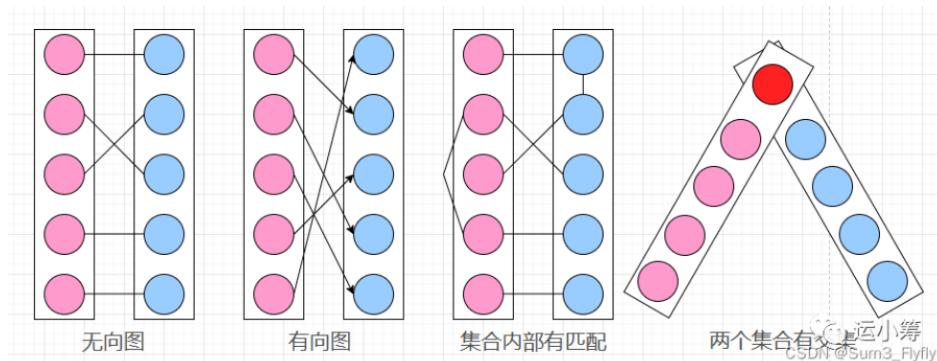


Figure 8.2.1: 二分图与其他

我们把上图中四个图编个号: (1)-(4)

- (2) 图很明显其中的边是有向的;

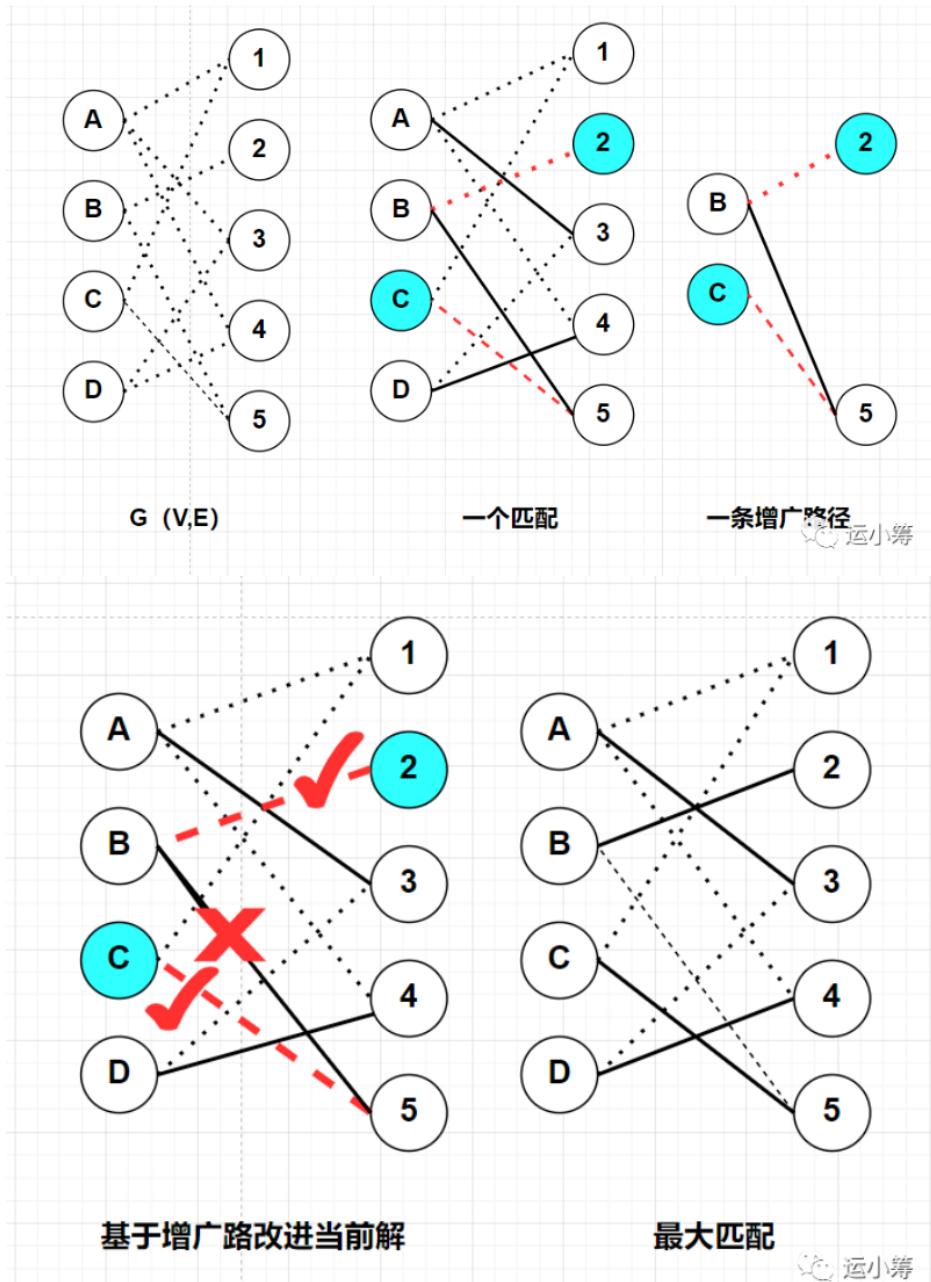


Figure 8.2.2: 最大匹配

当然，可以看出完美匹配一定是最小覆盖，所有的点都已经被覆盖，我们也找不到新的增广路径了。如上图中的“一个匹配”中就仅是一个普通的匹配，而且我们很容易发现一条增广路径：2-B-5-C，然后就得到了最后的“最大匹配”的结果。

### 8.3 增广路径 (Augmenting Path)

增广路径需要满足以下三个条件:

- 开始于一个未配对的向量
- 终止于一个未配对的向量
- 路径中的节点在匹配和不匹配之间交替

在上面讲解最大匹配的配图中，我们就通过增广路径找到了最大匹配，我们来将这个过程分解一下：

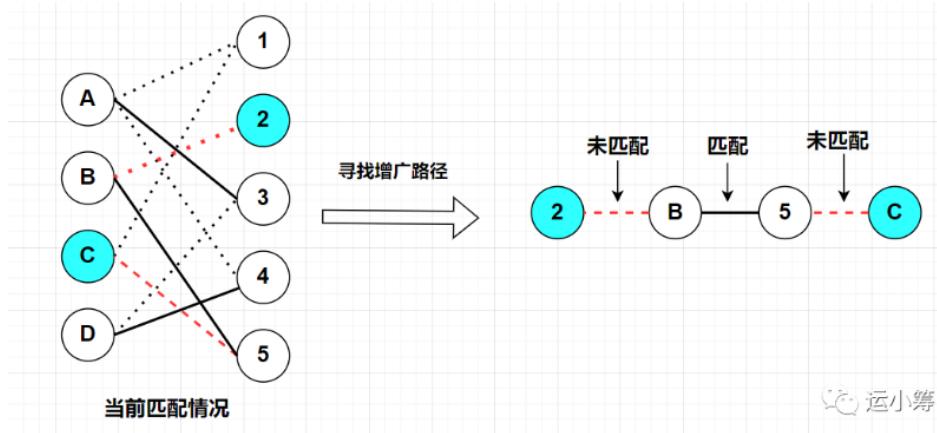


Figure 8.3.1: 增广路径-1

然后我们将配对的边断开，再把未配对的边连上：

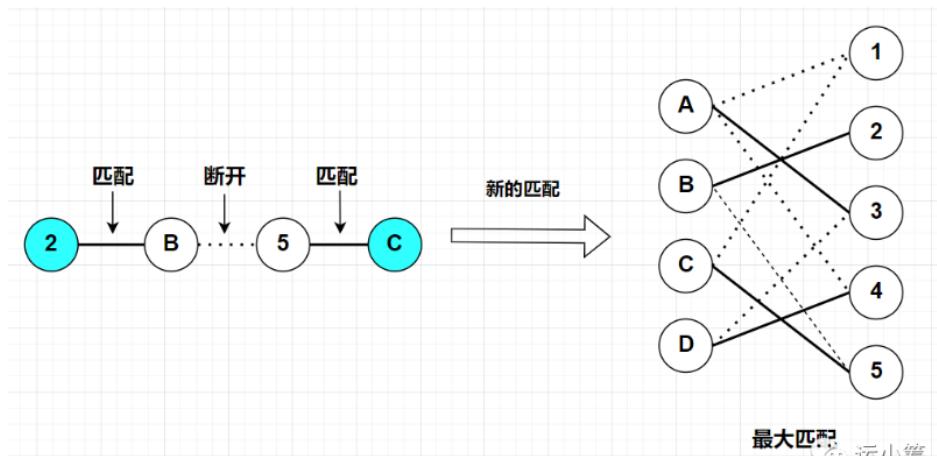


Figure 8.3.2: 增广路径-2

可以看到，经过这一次操作，图中的匹配由原来的 3 对匹配，变成了 4 对匹配。不难发

现，当我们找到一条增广路径的时候，未匹配边的数量比已匹配边的数量恰好多 1。所以如果将一条增广路径中匹配的边断开，并将所有的未匹配边变成匹配边，就得到了一个更优的解（匹配 +1）。

如果图中找不到任何增广路，说明该解不能被再次改进，当前解就是最优解了。这也是为什么找不找得到增广路是判断是否达到最优的充要条件。

## 8.4 BFS(Breadth First Search)

上面提到了 HK 算法通过 BFS 寻找多条增广路径的，所以在这里需要先介绍一下 BFS。

### 8.4.1 概述

广度优先搜索（也称宽度优先搜索，缩写 BFS）是连通图的一种遍历算法这一算法也是很多重要的图的算法的原型。Dijkstra 单源最短路径算法和 Prim 最小生成树算法都采用了和宽度优先搜索类似的思想。其别名又叫 BFS，属于一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位置，彻底地搜索整张图，直到找到结果为止。基本过程，BFS 是从根节点开始，沿着树（图）的宽度遍历树（图）的节点。如果所有节点均被访问，则算法中止。

### 8.4.2 迭代过程

已知图  $G = (V, E)$ ，设  $node_x$  是其中的一个源顶点，按和  $node_x$  由近到远的遍历所有能到达的点，即存在边  $edge \in E$  连接两个点，并计算  $node_x$  到所有这些顶点的距离（最少边数），该算法同时能生成一棵根为  $node_x$  且包括所有可达顶点的宽度优先树。这样每一层全部都遍历完再遍历下一层的思想就是“广度优先”。

- step1：将图中所有结点储存在队列中，代码是使用 python 实现的，可以使用有序的数据结构来实现，按照所属层数来创建相应 index 的列表（HK 算法交替路的始末）step1：将图中所有结点储存在队列中，代码是使用 python 实现的，可以使用有序的数据结构来实现，按照所属层数来创建相应 index 的列表（HK 算法交替路的始末）
- step2：从最上层中取出第一个端点 start node，从该起始点出发，访其未访问的所有匹配点 neighbor nodes，并将已经访问的点从队列中删除；
- step3：对所有点重复 step2:，直至队列为空。

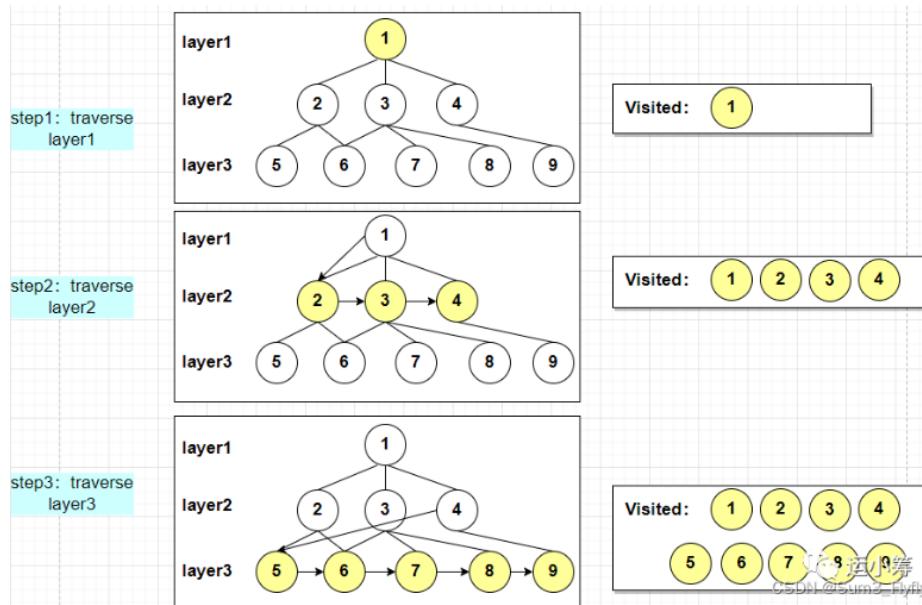


Figure 8.4.1: BFS

## 8.5 DFS(Depth First Search)

KM 算法和 HK 算法都是通过 DFS 来寻找增广路径的，所以也需要介绍一下 DFS。

### 8.5.1 概述

深度优先搜索属于图算法的一种，英文缩写为 DFS 即 Depth First Search. 其过程简要来说是对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次。

### 8.5.2 迭代过程

已知图  $G = (V, E)$ ，设  $node_x$  是其中的一个源顶点，从最近的可到达的点中选择一个，然后按照这样的步骤，一直找到一个根节点，这便是一条路径，这样找到所有的路径。这种“一条路走到底”的思想便是”深度优先“。

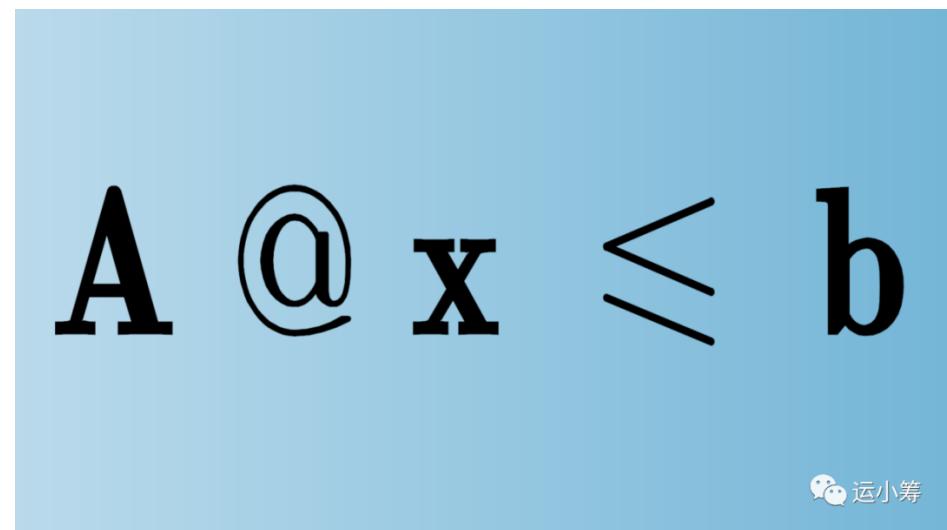
- step1: 从选择的源顶点 start node 出发，选择一个邻接点 neighbor1，再选择 neighbor1 的邻接点，直到访问到根节点；
- step2: 从这条路径的根节点返回上一层，寻找是否存在同层的点，有则遍历同层点的路径，按照层数逐渐向上层寻找，按照上述步骤遍历所有的路径；
- step3: 对所有匹配邻接点，按照 step1 和 step2 遍历所有路径；

## Chapter 9

# 2022-06-20: 优化求解器 | Gurobi 的 MVar 类: 矩阵建模利器、求解对偶问 题的备选方案 (附详细案例 + 代码)

作者: 刘兴禄, 清华大学博士在读

作者: 修宇璇, 清华大学博士在读



### 9.1 前言

本文涉及到的模型 (LP, MIP) 均是为了说明问题, 即使是 MIP, 我们也将其当做 LP 看待。> - LP: linear programming, 线性规划; > - MIP: mixed integer programming, 混合整

数(线性)规划。

## 9.2 优化求解器的建模方式：以 gurobi 为例

我们考虑如下的线性规划问题

$$\begin{aligned} \text{Maximize} \quad & Z = 3x_1 + 5x_2 \\ \text{subject to} \quad & \end{aligned}$$

$$x_1 \leq 4$$

$$2x_2 \leq 12$$

$$3x_1 + 2x_2 \leq 18$$

$$\text{and } x_1 \geq 0, \quad x_2 \geq 0$$

该模型有 2 个决策变量  $x_1$  和  $x_2$ , 3 个约束。

如果我们使用 Gurobi 对其进行建模，一般有下面 3 (或者说 4) 种方法：

1. **按行建模**: 逐行进行建模。通过使用 `quicksum` 或者创建表达式 `LinExpr` 对象结合 `addTerms`, 拼凑表达式, 最后使用 `Model.addConstr` 完成添加约束的操作, 从而完成建模;
2. **按列建模**: 逐列进行建模。通过创建 `Column` 对象, 使用 `addTerms` 函数拼凑好 `Column` 对象, 最后使用 `Model.addVar` 函数直接完成建模。
3. **按非零系数建模**: 类似于逐列进行建模, 可以使用 `addTerms` 拼凑表达式。具体实现跟按行建模类似。
4. **按矩阵方式建模**: 通过拼凑约束系数矩阵  $A$ , 然后将决策变量转化为 `MVar` 的对象, 最后直接使用重载运算符 `@` 完成约束的添加, 即如 `Model.addConstr(A @ x == b)`, 就可以完成建模。

前 3 中建模方法我们已经在之前的推文中有所涉及, 见推文:

本文主要来介绍第四种: 按矩阵方式建模。主要内容包括:

1. 按矩阵方式建模的详细案例和代码;
2. 按矩阵方式建模, 并通过求解器轻松得到对偶问题的例子和代码。

在介绍具体内容之前, 笔者首先指出按照矩阵建模的好处和不足:

**优点:**

1. 添加约束代码量少, 只需要拼凑好系数矩阵, 即可一行代码搞定约束的添加;
2. 便于直接得到对偶问题的系数矩阵 ( $A^T$ ), 从而快速得到对偶问题的具体形式。

**缺点:**

1. 对于约束系数稀疏的模型, 会占用不必要的内存来存储那些系数为 0 的部分, 模型较大时, 有可能导致内存溢出;

$$\text{Minimize } w = [y_1, y_2, y_3] \begin{bmatrix} 4 \\ 12 \\ 18 \end{bmatrix}$$

subject to

$$[y_1, y_2, y_3] \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 3 & 2 \end{bmatrix} \geq [3, 5]$$

$$\text{and } \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

### 9.3.7 基于 Gurobi 进行按矩阵形式进行对偶问题的建模并求解

在矩阵建模的方式下，我们实现对偶问题的建模非常简单，只需要按照上一节的介绍，将原问题的系数矩阵  $\mathbf{A}$  转置，得到  $\mathbf{A}^T$ ，并且创建对偶问题的决策变量  $\mathbf{y}$ ，然后就可以轻松完成对偶问题的建模。具体代码如下。

Code

```

1 if __name__ == "__main__":
2     m = grb.Model("LP")
3     m.setParam('OutputFlag', 1)
4     y = m.addMVar(3, lb=0, ub=grb.GRB.INFINITY)
5     c = np.array([3, 5])
6     A = np.array([[1, 0],
7                   [0, 2],
8                   [3, 2]])
9     b = np.array([4, 12, 18])
10
11    m.addConstr(A.T @ y >= c)
12
13    m.Params.timelimit = 99999999999999999999999999999999
14
15    m.setObjective(b @ y, grb.GRB.MINIMIZE)
16
17    m.update()
18    m.optimize()
19    print('y_1={}'.format(y[0].x))
20    print('y_2={}'.format(y[1].x))
21    print('y_3={}'.format(y[2].x))

```

求解结果如下：

Code

```
1 Presolve time: 0.00s
```

```

2 Presolved: 2 rows, 3 columns, 4 nonzeros
3
4 Iteration      Objective       Primal Inf.       Dual Inf.       Time
5       0    0.0000000e+00    3.250000e+00    0.000000e+00    0s
6       2    3.6000000e+01    0.000000e+00    0.000000e+00    0s
7
8 Solved in 2 iterations and 0.00 seconds (0.00 work units)
9 Optimal objective 3.600000000e+01
10 y_1=0.0
11 y_2=1.5
12 y_3=1.0

```

我们将对偶问题的模型导出查看：

Code

```
1 m.write('dual.lp')
```

Code

```

1 \ Model LP
2 \ LP format - for model browsing. Use MPS format to capture full model detail.
3 Minimize
4   4 C0 + 12 C1 + 18 C2
5 Subject To
6   R0: C0 + 3 C2 >= 3
7   R1: 2 C1 + 2 C2 >= 5
8 Bounds
9 End

```

可见是完全吻合的。

### 9.3.8 小结

本文介绍了 Gurobi 的各种建模方式，包括按行建模、按列建模、按非零元素建模以及按矩阵形式建模。

我们详细介绍了按矩阵形式建模的案例及其在快速完成对偶问题建模中的使用。

但是笔者还是需要说明一点：按矩阵形式建模，虽然可以很方便地用求解器完成对偶问题的建模，但是如果科技论文中，需要大家写出对偶问题的具体公式形式，则这种方法不再奏效。从利用求解器实现对偶问题的求解角度来讲，本文介绍的办法完全没有问题。但是还是鼓励大家直接将对偶问题的具体公式形式写出来，这样对今后的研究更有好处。

### 9.3.9 参考资料

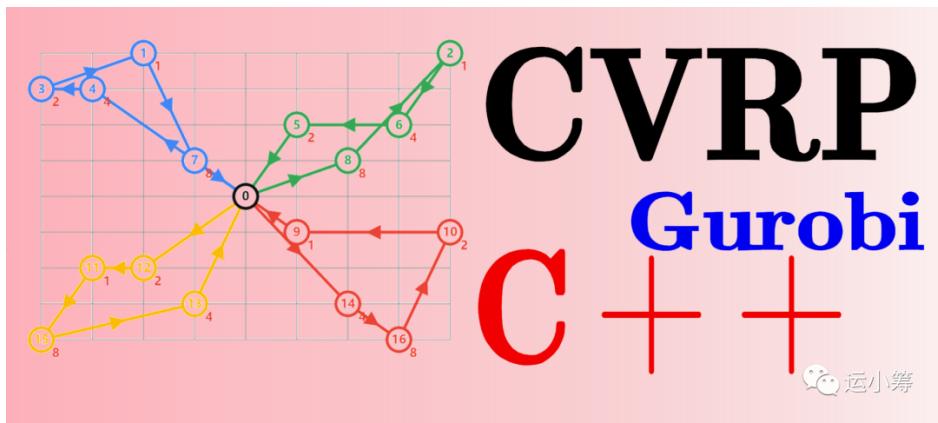
1. Gurobi Optimizer Reference Manual-9.5.1
2. Hillier F S. Introduction to operations research[J]. 1967.

## Chapter 10

# 2022-06-24: 优化 | 手把手教你用 C++ 调用 Gurobi 求解 CVRP: 环境 配置、数学模型及完整代码

作者: 于乃康, 昆明理工大学, 机电学院工业工程博士在读

修改: 刘兴禄, 清华大学博士在读



### 10.1 前言

本文基于一个经典的组合优化问题——Capacitated Vehicle Routing Problem(CVRP, 带容量约束的车辆路径规划问题) 来介绍在“C++”环境下如何调用 Gurobi 求解器进行最优化模型的建模和求解。本文用到的平台是 visual studio2022 和 Gurobi9.5.1。

本文内容分为两个部分

• 3.4

双击刚才新建属性页，在 C / C ++ / 常规/附加包含目录下，添加: ..... \win64\include

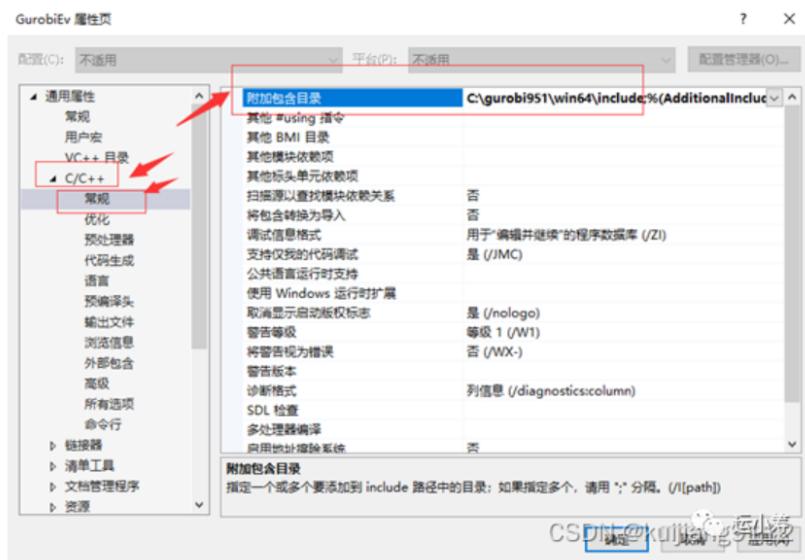


Figure 10.2.7: 第三步：为项目配置 gurobi-4

• 3.5

在链接器/常规/附加库目录下, 添加: .....\\win64 \\ lib

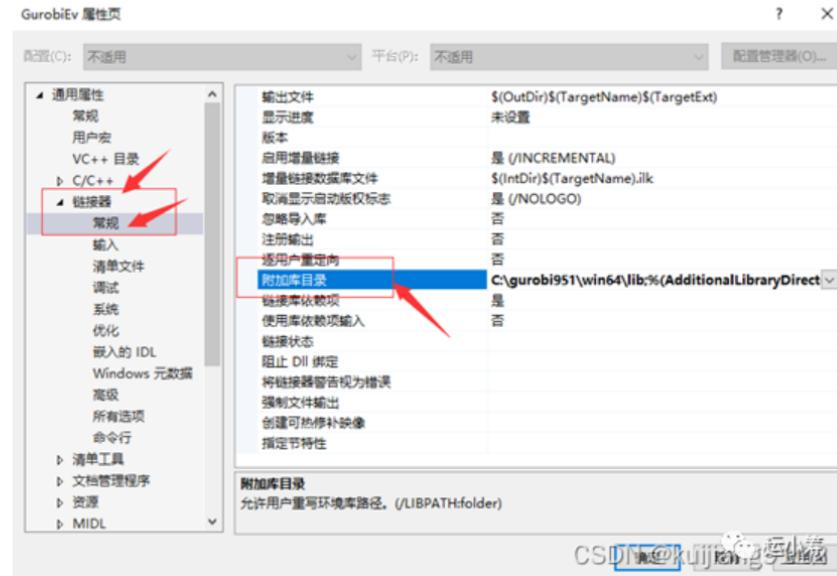


Figure 10.2.8: 第三步: 为项目配置 gurobi-5

• 3.6

在链接器/输入/附加依赖项下，添加gurobi95.lib和gurobi\_c++mdd2019.lib(这两个lib文件名与你的gurobi版本有关，在你的gurobi安装文件夹...win64\lib下找到对应版本的lib)

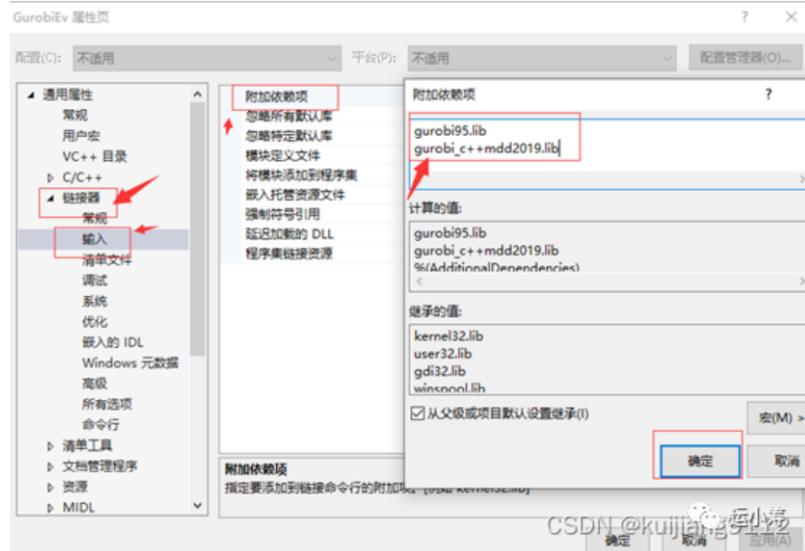


Figure 10.2.9: 第三步：为项目配置 gurobi-6

- 3.7 测试

打开gurobi安装目录hellip;\win64\examples\c++中，找到mip1\_c++.cpp文件，复制到刚才建的.cpp文件中，点击“本地Windows调试”运行，看到运行结果如示，说明环境配置没问题。

```

Microsoft Visual Studio 调试控制台
Gurobi Optimizer version 9.5.1 build v9.5.lrc2 (win64)
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
Optimize a model with 2 rows, 3 columns and 5 nonzeros
Model fingerprint: 0x98886187
Variable types: 0 continuous, 3 integer (3 binary)
Coefficient statistics:
    Matrix range [1e+00, 3e+00]
    Objective range [1e+00, 2e+00]
    Bounds range [1e+00, 1e+00]
    RHS range [1e+00, 4e+00]
Found heuristic solution: objective 2.0000000
Presolve removed 2 rows and 3 columns
Presolve time: 0.00s
Presolve: All rows and columns removed
Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 16 available processors)

Solution count 2: 3 2

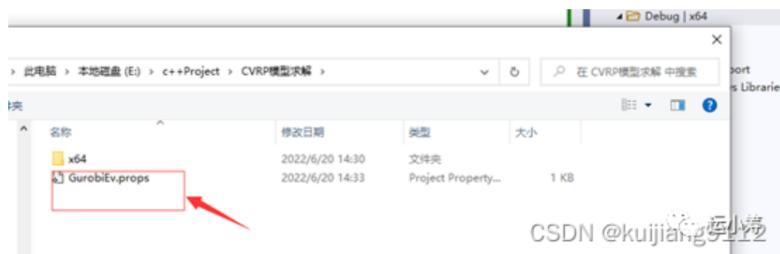
Optimal solution found (tolerance 1.00e-04)
Best objective 3.00000000000e+00, best bound 3.00000000000e+00, gap 0.0000%
x 1
y 0
z 1
Obj: 3
  
```

CSDN @kuijiang9运小差

Figure 10.2.10: 第三步：为项目配置 gurobi-7

- 3.8

在其他工程中如果需要gurobi环境的话，直接在项目的“属性管理器”->右击“debug|x64”->“添加现有属性表”，把我们刚才配置好的GurobiEv.props添加进去就可以直接使用啦！



CSDN @kuijiang9运小差

Figure 10.2.11: 第三步：为项目配置 gurobi-8

## 10.3 CVRP: 问题描述及数学模型

我们回顾一下 Paolo Toth 等编著的经典《Vehicle Routing: Problems, Methods, and Applications》一书和 [2] 的关于 CVRP (Capacitated Vehicle Routing Problem) 问题的描

述:

我们回顾一下Paolo Toth等编著的经典《Vehicle Routing: Problems, Methods, and Applications》一书和[2]的关于CVRP (Capacitated Vehicle Routing Problem) 问题的描述:

In the CVRP, the transportation requests consist of the distribution of goods from a single *depot*, denoted as point 0, to a given set of  $n$  other points, typically referred to as *customers*,  $N = \{1, 2, \dots, n\}$ . The amount that has to be delivered to customer  $i \in N$  is the customer's *demand*, which is given by a scalar  $q_i \geq 0$ , e.g., the weight of the goods to deliver. The *fleet*  $K = \{1, 2, \dots, |K|\}$  is assumed to be *homogeneous*, meaning that  $|K|$  vehicles are available at the depot, all have the same capacity  $Q > 0$ , and are operating at identical costs. A vehicle that services a customer subset  $S \subseteq N$  starts at the depot, moves once to each of the customers in  $S$ , and finally returns to the depot. A vehicle moving from  $i$  to  $j$  incurs the *travel cost*  $c_{ij}$ .

CVRP问题就是为  $K$  辆可用的车设计行驶路径, 使得所有的顾客需求都能被满足, 每辆车的载重不能超过容量, 且每一个顾客访问且只被访问一次, 同时使得所有车辆的总行驶距离最小。

©运筹学

CSDN @刘兴林

CVRP 问题就是为  $K$  辆可用的车设计行驶路径, 使得所有的顾客需求都能被满足, 并且每辆车的载重不能超过容量, 且每一个顾客访问且只被访问一次, 同时使得所有车辆的总行驶距离最小。

在给出完整的数学模型之前, 我们首先引入下面的参数和决策变量。

### 参数

- $c_{ij}$ : 表示客户点  $i$  到客户点  $j$  的距离;
- $d_i$ : 表示客户点  $i$  的配送需求量;
- $K$ : 表示车辆数;
- $N$ : 表示包含中心仓库的客户点数;
- $Q$ : 表示车辆的载重量;

### 决策变量

$$x_{ijk} = \begin{cases} 1, & \text{车辆 } k \text{ 从 } i \text{ 到 } j, \\ 0, & \text{其他.} \end{cases} \quad (10.3.1)$$

$$y_{ik} = \begin{cases} 1, & \text{客户 } i \text{ 由车辆 } k \text{ 服务,} \\ 0, & \text{其他.} \end{cases} \quad (10.3.2)$$

$$\mu_{ik}, \text{ 非负数, 表示第 } k \text{ 辆车在点 } i \text{ 的载重量.} \quad (10.3.3)$$

注意, 上述决策变量中:

$x_{ijk}$  可以被认为是 routing decision(路径决策), 表示车辆的路径决策;

$y_{ik}$  为 assignment decision (分配决策), 表示顾客和车辆之间的匹配关系。

$\mu_{ik}$  为 vehicle load decision(车辆载重决策), 表示车辆的载重量, 该约束可以起到消除子环路的作用。

至于为什么可以用来消除子环路, 见往期推文。

基于上述参数和决策变量, CVRP 可以被建模为一个整数规划模型, 其完整形式如下:

$$\min \sum_{i=0}^N \sum_{j=0}^N \sum_{k=1}^K c_{ij} x_{ijk} \quad (1)$$

$$\text{s.t. } \sum_{k=1}^K y_{ik} = 1, \quad \forall i \in \{1, \dots, N\}, \quad (2)$$

$$\sum_{i=0}^N x_{ijk} = y_{jk}, \quad \forall j \in \{1, \dots, N\}, \forall k \in \{1, \dots, K\} \quad (3)$$

$$\sum_{i=0}^N x_{ijk} = y_{ik}, \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, \dots, K\} \quad (4)$$

$$\sum_{i=0}^N d_i y_{ik} \leq Q, \quad \forall k \in \{1, \dots, K\}, \quad (5)$$

$$\mu_{jk} + d_j - \mu_{ik} + Q(1 - x_{ijk}) \leq 0, \quad \forall i, j \in \{1, \dots, N\}, i \neq j, k \in \{1, \dots, K\}. \quad (6)$$

$$x_{ijk}, y_{ik} \in \{0, 1\}, \quad \forall i, j \in \{1, \dots, N\}, i \neq j, k \in \{1, \dots, K\}. \quad (7)$$

$$\mu_{ik} \text{ integer}, \quad \forall i \in \{1, \dots, N\}, k \in \{1, \dots, K\}. \quad (8)$$

(1) 为目标函数, 表示最小化总路径;

约束 (2) 表示每个顾客都一定被服务;

约束 (2) - (3) 为流平衡约束 (flow conservation), 表示经过一个客户点, 就离开那个客户点;

约束 (5) 保证每辆车的载重不超过其最大限度;

约束 (6) 是子回路消除约束, 可参见 [1] 和 Toth 关于子回路消除的介绍。

关于子环路消除, 也可参考本公众号的往期推文。

CVRP 是经典的组合优化问题, 是 NP-hard。本文作为基础教学文章, 仅展示调用最优化求解器求解 CVRP 的内容。下面就是 C++ 调用 Gurobi 求解 CVRP 的详细代码。

## 10.4 C++ 调用 Gurobi 求解 CVRP 的完整代码

### 10.4.1 数据集生成

Code

```

1 #include<iostream>
2 #include<vector>
3 #include"gurobi_c++.h"
4 #include <ctime>
5 #include <cstdlib>
6 #include<string>
7 #include<algorithm>
8 using namespace std;
9
10
11 #define VehicleNum 5 //车辆数
12 #define CustomNum 30//客户数量
13 #define VehicleCap 100 //车辆最大载重量
14 #define sCusN 7 //模型实际大小
15
16 vector<vector<double>>DistanceMatrix(CustomNum, vector<double>(CustomNum)); //距离矩阵
17 vector<int>Demand(CustomNum); //需求矩阵
18 const double CustomCoordinate[][2] = {
19 {23,95}, {820,915},{470,302},{950,905},{868,744},
20 {891,987},{76,625}, {8,541},{219,988},{662,636},
21 {318,402},{307,984},{689,486},{675,946},{665,331},
22 {392,143},{957,725},{442,911},{576,762},{985,369},
23 {170,460},{861,246},{192,61},{121,1000},{858,541},
24 {261,204},{572,74},{788,22},{303,694},{97,241}
25 }; //客户点坐标
26
27 class RandomNumber
28 {
29 public:
30     RandomNumber()
31     {
32         srand(time(0));
33     }
34
35     int get(int begin = 0, int end = 1)
36     {
37         return rand() % (end - begin + 1) + begin;
38     }
39 };
40
41 double discal(const double* city1, const double* city2)
42 {
43     double x1 = city1[0]; //城市 1 横坐标
44     double y1 = city1[1];

```

## Chapter 11

# 2022-06-27: 鲁棒优化 | C&CG 算法 求解两阶段鲁棒优化：全网最完整、最 详细的【入门-完整推导-代码实现】笔记

作者：刘兴禄，清华大学，清华-伯克利深圳学院博士在读

初次完稿日期：2021.09.28

补充完善 + 审校：张瑞三，四川大学，硕士在读



特别感谢 Zeng Bo 老师非常耐心地解答我的疑惑，您的解答对我来讲非常重要。

本文非常详细、完整地介绍了求解 Two-stage Robust Optimization Model 的一种精确算法：Column and Constraint Generation Algorithm。该算法由 Zeng Bo, Zhao Long 于 2013 年首次提出，相关论文发表在期刊《Operations Research Letters》上，论文题目为《Solving

two-stage robust optimization problems using a column-and-constraint generation method》。本文对该论文进行了非常细致的解读，包括：

- 对鲁棒优化问题进行了分类；
- 对两阶段鲁棒优化模型进行了详细解析；
- 对 Bender-dual 算法进行了讲解；
- 对 Column-and-constraint generation method 进行了非常详细的解读、完整的推导和完整的解释；
- 对论文中的案例进行了相当详细的推导；
- 对 Column-and-constraint generation method 进行了完整的代码复现 (Python 调用 Gurobi 实现)。



**Figure 11.0.1:** Solving two-stage robust optimization problems using a column-and-constraint generation method

## 11.1 两阶段鲁棒优化问题 (Two-stage Robust Optimization)

鲁棒优化是一类考虑参数不确定的数学规划问题，是运筹学中比较高阶的方法。近几年，关于鲁棒优化的研究越来越火热。常见的鲁棒优化问题包括基本的鲁棒优化、多阶段鲁棒优化、分布式鲁棒优化等。

鲁棒优化旨在处理优化问题中参数不确定的挑战，致力于优化最坏情况 (worst case) 下的解，或者最坏参数分布下的解 (对应分布式鲁棒优化)，从而使最终得到的解具有非常强的抵抗不确定性或者风险的能力。

处理参数不确定的优化问题，还有一类方法，叫作随机规划，关于二者的区别，见往期推文。

两阶段鲁棒优化问题，是多阶段鲁棒优化的一个特例 (multi-stage robust optimization)。

在两阶段鲁棒优化问题中，一般包含两个不同层级的决策变量，又称为第一阶段的决策 (first-stage decision) 和第二阶段的决策 (second-stage decision)。比如：一个联合优化问题，同时考虑

- 战略层决策和战术层决策；
- 战术层决策和操作层决策
- ...

一个典型例子，就是选址-配送规划问题。选址决策属于战略层决策，配送问题，可以视为战术层或者作业层决策。

两阶段鲁棒优化问题描述如下：

第一阶段的决策相关的参数是确定的，并且第一阶段的决策需要首先做出；

第二阶段的决策的相关参数有一些是不确定的。第二阶段的决策需要在第一阶段的决策确定之后，等到第二阶段参数的揭示后，再做出相应的第二阶段决策。

两阶段鲁棒优化的目标为：对两阶段决策进行联合优化，同时考虑第二阶段参数的不确定性。也就是优化在第二阶段参数取到最坏情况下的两个阶段决策对应的总的目标值。

下面我们用标准的数学语言来描述两阶段鲁棒优化问题。两阶段鲁棒优化问题的数学模型如下所示 (参考自文献 [1])

$$\min_{\mathbf{y}} \mathbf{c}^T \mathbf{y} + \max_{u \in \mathcal{U}} \min_{\mathbf{x} \in \mathbf{F}(\mathbf{y}, u)} \mathbf{b}^T \mathbf{x} \quad (1) \quad (11.1.1)$$

$$s.t. \quad \mathbf{A}\mathbf{y} \geq \mathbf{d}, \quad (2) \quad (11.1.2)$$

$$\mathbf{G}\mathbf{x} \geq \mathbf{h} - \mathbf{E}\mathbf{y} - \mathbf{M}u, \quad u \in \mathcal{U} \quad (3) \quad (11.1.3)$$

$$\mathbf{S}_{\mathbf{y}} \subseteq \mathbb{R}_+^n \quad (4) \quad (11.1.4)$$

$$\mathbf{S}_{\mathbf{x}} \subseteq \mathbb{R}_+^m \quad (5) \quad (11.1.5)$$

其中：

$\mathbf{y}$  为第一阶段的决策变量；

$\mathbf{x}$  为第二阶段的决策变量；

$\mathbf{c}$  为第一阶段的决策变量相关的参数，是确定的；

$u$  为第二阶段的决策变量相关的参数，是不确定的，且其不确定参数的取值范围由不确定集  $\mathcal{U}$  刻画；具体的例子见往期推文；

目标函数中，第一项  $\min_{\mathbf{y}} \mathbf{c}^T \mathbf{y}$  为第一阶段决策对应的目标函数。该部分决策需要首先确定；

目标函数中，第二项  $\max_{u \in \mathcal{U}} \min_{\mathbf{x} \in \mathbf{F}(\mathbf{y}, u)} \mathbf{b}^T \mathbf{x}$  为第二阶段决策对应的目标函数。该部分决策需要在第一阶段决策确定之后再做出；该部分目标函数旨在找到 worst case；也就是在最

坏情况下，第二阶段的目标值；

整个目标函数，就是在优化在最坏情况下的总目标函数，使得最坏的情况最好，使得解具有非常好的鲁棒性。

约束 (2) 只跟第一阶段的决策有关；

约束 (3) 中跟第一阶段，第二阶段的决策都有关，且包含不确定参数  $u$ 。

以上就是两阶段鲁棒优化的一个简要介绍。目前还比较晦涩的数学符号，我们将在下文进行更通俗的解读。

注意，如果  $u$  的取值可能性非常少，我们就可以通过穷举  $u$  的取值，显式地将每个可能的  $u$  对应的约束都加进模型，然后将两阶段鲁棒优化模型等价为一个可直接求解的一阶段数学规划模型，从而可以达到直接求解上述 Two-stage robust optimization 模型的目的。因为最坏情况，一定是对应着某一种  $u$  的取值，而我们已经穷举了所有  $u$  的取值可能，且将其显式地写进了模型，因此原两阶段鲁棒优化模型可以直接被解决。但是通常， $u$  的取值非常多（组合数级别或者无穷多），穷举  $u$  往往是低效甚至不可行的，因此，需要利用一些其他方法来求解 Two-stage robust optimization model。本文将要介绍的 Column and Constraint Generation 算法就是其中一种（此外，文献 [1] 中提到的 Benders-dual algorithm 也是一种）。

## 11.2 两阶段鲁棒优化问题的详细解读

参考文献 [1] 中提到：Two-stage RO 的计算非常困难，即使是一个简单的 Two-stage RO 问题也可能是 NP-Hard 的。为了克服计算负担，目前主要流行两种求解策略。第一种是使用 \*\*approximation algorithms\*\*，它假设第二阶段的决策是不确定性的简单函数，如仿射函数。第二类算法是利用 Benders 分解法求出精确解，即利用第二阶段决策问题的对偶解逐步构造第一阶段决策的价值函数。因此，我们称它们为 \*\*Benders-dual cutting plane algorithms\*\* 算法。

文献 [1] 中关于两阶段鲁棒优化问题模型的描述如下：

second-stage decision variables, respectively. Unless mentioned explicitly, they can take either discrete or continuous values. The uncertainty set  $\mathcal{U}$  could be a discrete set or a polyhedron. The general form of two-stage RO formulation is

$$\begin{aligned} & \min_{\mathbf{y}} \mathbf{c}^T \mathbf{y} + \max_{u \in \mathcal{U}} \min_{\mathbf{x} \in F(\mathbf{y}, u)} \mathbf{b}^T \mathbf{x} \\ & \text{s.t. } \mathbf{A}\mathbf{y} \geq \mathbf{d}, \quad \mathbf{y} \in \mathbf{S}_y \end{aligned} \quad (1)$$

where  $F(\mathbf{y}, u) = \{\mathbf{x} \in \mathbf{S}_x : \mathbf{G}\mathbf{x} \geq \mathbf{h} - \mathbf{E}\mathbf{y} - \mathbf{M}u\}$  with  $\mathbf{S}_y \subseteq \mathbb{R}_+^n$  and  $\mathbf{S}_x \subseteq \mathbb{R}_+^m$ . A few cutting plane based methods have been developed and implemented to derive the exact solution when  $\mathbf{S}_x = \mathbb{R}_+^m$  [19,21,8,15]. Because they are designed in the line of Benders' decomposition [2,14] and make use of the dual information of the second-stage decision problem, we call them *Benders-dual cutting plane methods* or *Benders-dual methods* for short. We briefly describe them as follows.

CSDN @刘兴禄

Figure 11.2.1: 两阶段鲁棒优化问题模型的描述

该论文探讨的问题，是基于如下的假设：

- 决策变量类型假设：first-stage and second-stage are linear optimization
- 参数不确定性 (uncertainty): is either a finite discrete set or a polyhedron. 换句话说，改论文考虑的不确定集为是一个有限的离散集合或者一个多面体集合，比如
  - a finite discrete set: 不确定参数的可能取值为一个有限的可选集，例如： $u \in \mathcal{U} = \{\text{scenario}_1, \text{scenario}_2, \text{scenario}_3\}$ .
  - a polyhedron: 不确定参数的可能取值的范围为一个多面体，例如  $u \in \mathcal{U} = \{1 \leq u_1 \leq 2, 3 \leq u_2 \leq 5\}$ ，该不确定集为一个矩形，学术上一般称其为盒子不确定集 (box uncertainty set)。

根据上图，两阶段鲁棒优化中，第一阶段的决策需要首先做出，其优化方向为 min。

但是第二阶段的决策，却是一个双层问题 (bi-level)。具体来讲，目标函数第二项的符号

为

$$\max_{u \in \mathcal{U}} \min_{\mathbf{x} \in F(\mathbf{y}, u)} \mathbf{b}^T \mathbf{x}$$

这个符号需要详细解释一下：

$\mathbf{x} \in F(\mathbf{y}, u)$  的意思：表示 given 不确定参数  $u$  的一个取值，以及 given first-stage 决策变量  $\mathbf{y}$  的取值，我们决策  $\mathbf{x}$ ，使得  $\mathbf{b}^T \mathbf{x}$  最小。并且满足如下约束

$$F(\mathbf{y}, u) = \{\mathbf{x} \in \mathbf{S}_x : \mathbf{G}\mathbf{x} \geq \mathbf{h} - \mathbf{E}\mathbf{y} - \mathbf{M}u\} \quad (11.2.1)$$

with  $\mathbf{S}_y \subseteq \mathbb{R}_+^n$  and  $\mathbf{S}_x \subseteq \mathbb{R}_+^m$ 。

其实  $\mathbf{S}_y \subseteq \mathbb{R}_+^n$  and  $\mathbf{S}_x \subseteq \mathbb{R}_+^m$  就是说， $\mathbf{x}$  和  $\mathbf{y}$  分别是  $n \times 1$  维和  $m \times 1$  维的非负连续决策变量而已。

我们将上图中的模型展开，写成一个比较容易看明白的形式：

$$\min_{\mathbf{y}} \quad \mathbf{c}^T \mathbf{y} + \max_{u \in \mathcal{U}} \min_{\mathbf{x} \in \mathbf{F}(\mathbf{y}, u)} \mathbf{b}^T \mathbf{x} \quad (1) \quad (11.2.2)$$

$$s.t. \quad \mathbf{A}\mathbf{y} \geq \mathbf{d}, \quad (2) \quad (11.2.3)$$

$$\mathbf{G}\mathbf{x} \geq \mathbf{h} - \mathbf{E}\mathbf{y} - \mathbf{M}u, \quad u \in \mathcal{U} \quad (3) \quad (11.2.4)$$

$$\mathbf{S}_y \subseteq \mathbb{R}_+^n \quad (4) \quad (11.2.5)$$

$$\mathbf{S}_x \subseteq \mathbb{R}_+^m \quad (5) \quad (11.2.6)$$

只不过，论文截图中  $\mathbf{y} \in \mathbf{S}_y$ ，就等价于  $\mathbf{y}$  是下面模型的可行解

$$\min_{\mathbf{y}} \quad \mathbf{c}^T \mathbf{y} + \max_{u \in \mathcal{U}} \min_{\mathbf{x} \in \mathbf{F}(\mathbf{y}, u)} \mathbf{b}^T \mathbf{x} \quad (11.2.7)$$

$$s.t. \quad \mathbf{A}\mathbf{y} \geq \mathbf{d}, \quad (11.2.8)$$

$$\mathbf{S}_y \subseteq \mathbb{R}_+^n \quad (11.2.9)$$

其中  $u, \mathbf{x}$  是一个给定的值。

上述 Two-stage Robust Optimization 模型，一般是无法直接调用优化求解器进行求解的，需要设计对应的算法进行求解。下面就来介绍求解该模型的算法。

### 11.3 Two-stage RO 和 Benders-dual cutting plane method

类似于使用标准的 Benders Decomposition 算法求解确定性问题，Two-stage RO 也是可以使用 Benders Decomposition 算法来求解的。文献中将该算法称之为 Benders-dual cutting plane method。

本文仅对 Benders-dual cutting plane method 做简要介绍。小编其实早就已经写好了 Benders-dual cutting plane method 的完整笔记和代码，只不过想留着后面再分享，尽请期待！

Benders-dual cutting plane method 求解 Two-stage RO 问题可以类比 Benders Decomposition 求解确定性问题的情形。我们可以将 Two-stage RO 分解为：

一个主问题 (Master Problem)，主要包含第一阶段的决策  $\mathbf{y}$  及只跟  $\mathbf{y}$  有关的约束、子问

We note some significant differences between Benders-dual method and the above algorithm:

- (i) *Decision variables in the master problem.* The C&CG algorithm increases the dimensionality of the solution space by introducing a set of new variables in each iteration, while the Benders-dual algorithm keeps working with the same set of variables.
- (ii) *Feasibility cut.* The C&CG algorithm provides a general approach to deal with the feasibility issue, while current approaches for the Benders-dual algorithm are problem-specific.
- (iii) *Computational complexities.* Compared with the Benders-dual algorithm, the C&CG algorithm solves the master program with a larger number of variables and constraints. However, under the relatively complete recourse assumption, according to [Propositions 1 and 2](#), the number of iterations in the C&CG algorithm is reduced by the order of  $O(q)$  if the second-stage decision problem is an LP. Actually, as the number of extreme points is exponential with respect to numbers of variables and constraints (in the second stage), such a reduction is very significant. The computational study presented in [21] and in [Section 4](#) confirms this point.
- (iv) *Solution capability.* Different from the Benders-dual algorithm, which requires the second-stage problem to be an LP problem, the C&CG algorithm is indifferent to the variable types in the second stage. We recently extended this algorithm in a nested fashion to deal with two-stage RO with a mixed integer recourse problem [20].
- (v) *Strength of the cut.* Under the relatively complete recourse assumption, the following proposition (see A3 in the Electronic Companion [11] for the proof) shows that the optimal value of  $\mathbf{MP}_1$  is an underestimation of that of  $\mathbf{MP}_2$ .

 运筹学  
CSDN @刘兴禄

本文的参考论文假设不确定变量  $u_l$  属于一个不确定的场景集合，即是一个离散的集合，我们可以比较容易地通过枚举比较每种情景下的解的质量；接下来就是考虑很复杂的情况，即多面体不确定集 polyhedral uncertainty set。处理多面体不确定集的求解方法主要有 outer approximation algorithm 和 mixed integer linear reformulations，前者基于启发式，后者利用不确定集的特殊结构将双线性规划转化为等价的 MIP。然而，准确地求解具有一般多面体不确定性集的两阶段反问题仍然是一个具有挑战性的问题。

这里我们把前面提到的求解  $\mathbf{SP}_2$  方法再次回顾一遍：

$$\mathbf{SP}_2 : \mathcal{Q}(\mathbf{y}) = \left\{ \max_{u \in \mathcal{U}} \min_x \mathbf{b}^T \mathbf{x} : \mathbf{Gx} \geq \mathbf{h} - \mathbf{Ey} - \mathbf{Mu}, \mathbf{x} \in \mathbf{S}_x \right\} \quad (11.4.11)$$

求解  $\mathbf{SP}_2$  的可选方法：

1. 启发式算法
2. 通过将  $\mathbf{SP}_2$  的 inner level 的部分取对偶，将  $\mathbf{SP}_2$  变成一阶段的 bilinear 的二次规划模型，然后使用 Gurobi, COPT 等求解器来求解；
3. 使用 Karush-Kuhn-Tucker(KKT) 条件将 inner level 的模型做等价转化，然后直接变为 single leve 的 MIP 进行求解。

本文着重介绍基于 KKT 条件的解法。这里一定注意：KKT 条件应用的前提是强对偶成立。

1. 问题是凸的，且强对偶成立  $\Leftrightarrow$  KKT 条件成立。(KKT 条件成立的充要条件)
2. 问题是非凸的，且强对偶成立  $\Rightarrow$  KKT 条件成立。(KKT 条件成立的必要条件)

以  $\mathbf{SP}_2$  为例子，我们将 inner level 的模型 (inner level 是凸的，且强对偶显然成立，因此 KKT 条件成立是充要条件)：

$$\min_{\mathbf{x}} \quad \mathbf{b}^T \mathbf{x} \quad (11.4.12)$$

$$s.t. \quad \mathbf{Gx} \geq \mathbf{h} - \mathbf{Ey} - \mathbf{Mu}, \quad (11.4.13)$$

$$\mathbf{S}_{\mathbf{x}} \subseteq \mathbb{R}_+^m \quad (11.4.14)$$

通过拉格朗日对偶结合 KKT 条件，等价转化成下面的 KKT 条件方程组：

$$\mathbf{Gx} \geq \mathbf{h} - \mathbf{Ey} - \mathbf{Mu} \quad (11.4.15)$$

$$\mathbf{G}^T \boldsymbol{\pi} \leq \mathbf{b} \quad (11.4.16)$$

$$(\mathbf{Gx} - \mathbf{h} + \mathbf{Ey} + \mathbf{Mu})_i \boldsymbol{\pi}_i = 0, \quad \forall i \quad (11.4.17)$$

$$(\mathbf{b} - \mathbf{G}^T \boldsymbol{\pi})_j x_j = 0, \quad \forall j \quad (11.4.18)$$

$$\mathbf{u} \in \mathcal{U}, \quad \mathbf{x} \in \mathbf{S}_{\mathbf{x}}, \quad \boldsymbol{\pi} \geq 0. \quad (11.4.19)$$

求解这个方程组，就等价于求解了 inner level 的模型。

这里需要强调一句：

KKT 条件，实际上就是把一个有目标的约束规划问题，转化成了一个无目标的方程组。求解无目标的方程组，得到的解就是有目标的约束规划问题的最优解。将解带入目标函数，得到目标函数值，就是有目标的约束规划问题的最优值。

相信很多小伙伴一定很疑惑，上面的 KKT 条件方程组是怎么来的呢？别着急，我们在后文中有完整的推导过程。

然后我们将 outer level 的部分考虑进来，则整个 bi-level 的 **SP<sub>2</sub>** 就等价为下面 single level 的模型：

**SP<sub>2</sub>** 的等价形式：

$$\max_{\mathbf{x}, u, \pi} \mathbf{b}^T \mathbf{x} \quad (7) \quad (11.4.20)$$

$$s.t. \quad \mathbf{Gx} \geq \mathbf{h} - \mathbf{Ey} - \mathbf{Mu} \quad (8) \quad (11.4.21)$$

$$\mathbf{G}^T \boldsymbol{\pi} \leq \mathbf{b} \quad (9) \quad (11.4.22)$$

$$(\mathbf{Gx} - \mathbf{h} + \mathbf{Ey} + \mathbf{Mu})_i \boldsymbol{\pi}_i = 0, \quad \forall i \quad (10) \quad (11.4.23)$$

$$(\mathbf{b} - \mathbf{G}^T \boldsymbol{\pi})_j x_j = 0, \quad \forall j \quad (11) \quad (11.4.24)$$

$$\mathbf{u} \in \mathcal{U}, \quad \mathbf{x} \in \mathbf{S}_{\mathbf{x}}, \quad \boldsymbol{\pi} \geq 0. \quad (12) \quad (11.4.25)$$

在构造的新模型中含有两个等式约束，即 (10) 和 (11)，有没有熟悉的感觉，没错，(10) 和 (11) 就是互补松弛性条件。

上述模型中，约束 (10) 和 (11) 是非线性的，但是由于是乘积等于 0 的形式，因此可以进行等价线性化，从而将其进一步等价转化成 MIP，从而直接可以求解。

具体转化方法，见下面截自论文中的图片。

Next, we present a method to deal with general polyhedra uncertainty sets. Several solution methods are developed for both relatively simple cardinality uncertainty sets and structured polyhedral uncertainty sets, including an outer approximation algorithm [8] and mixed integer linear reformulations [19,21,15,13]. The first is a heuristic procedure to solve  $\mathbf{SP}_1$  with a general polyhedral uncertainty set. The latter group uses the specific structure of the uncertainty set to convert the bilinear program  $\mathbf{SP}_1$  into an equivalent mixed integer linear program. Nevertheless, it remains a challenging problem to exactly solve two-stage RC

运筹学  
CSDN @刘兴

with a general polyhedral uncertainty set. To address this issue, we make use of the classical Karush–Kuhn–Tucker (KKT) conditions to handle a general polyhedral uncertainty set, provided that the relatively complete recourse assumption holds.

Consider  $\mathbf{SP}_2$ . Let  $\pi$  be the vector of dual variables to the second-stage decision problem. Using KKT conditions,  $\mathbf{SP}_2$  is equivalent to the following:

$$\max \mathbf{b}^T \mathbf{x} \quad (15)$$

$$\text{s.t. } \mathbf{Gx} \geq \mathbf{h} - \mathbf{Ey} - \mathbf{Mu} \quad (16)$$

$$\mathbf{G}^T \pi \leq \mathbf{b} \quad (17)$$

$$(\mathbf{Gx} - \mathbf{h} + \mathbf{Ey} + \mathbf{Mu})_i \pi_i = 0, \quad \forall i \quad (18)$$

$$(\mathbf{b} - \mathbf{G}^T \pi)_j x_j = 0, \quad \forall j \quad (19)$$

$$u \in \mathcal{U}, \quad \mathbf{x} \in \mathbf{S}_x, \quad \pi \geq \mathbf{0}. \quad (20)$$

Constraints in (18) and (19) are complementary slackness conditions, where  $i$  and  $j$  are appropriate indices for variables or constraints. By making use of the big-M method, they can be linearized by introducing binary variables. For example, we introduce a binary variable  $v_j$  for a constraint in (19). Then, it can be reformulated as

$$x_j \leq M v_j, \quad (\mathbf{b} - \mathbf{G}^T \pi)_j \leq M(1 - v_j), \quad v_j \in \{0, 1\}. \quad (21)$$

So,  $\mathbf{SP}_2$  can be converted into a 0-1 mixed integer program and computed by an existing solver. We recognize that if a tight bound on big-M can be analytically obtained, e.g., the study in [13] on the robust location-transportation problem, a better performance can be achieved.

运筹学  
CSDN @刘兴

## 11.5 补充：为什么用 KKT

由于后面需要用到 KKT 条件求解  $\mathbf{SP}_2$ , 但是一些小伙伴可能对 KKT 条件并不熟悉, 因此我们先在这里补充讲解一下 KKT 条件。

**KKT(Karush-Kuhn-Tucker) 条件**是非线性规划领域里最重要的理论成果之一, 是确定某点为极值点的必要条件。对于凸优化, KKT 就是优化极值点的充分必要条件。但实际上这话是不严谨的, 还缺少一个 regularity 条件, 在使用 KKT 条件之前需要验证 regularity 条件, 否则就无法保证 KKT 条件给出的结论一定成立。(证明请看王源师兄的推文——[【学界】关于 KKT 条件的深入探讨](<https://zhuanlan.zhihu.com/p/33229011>))

首先, 我们考虑下面的约束优化问题

$$\min z = f(x_1, x_2, \dots, x_n) \quad (11.5.1)$$

$$s.t. \quad g_1(x_1, x_2, \dots, x_n) \leq b_1 \quad (11.5.2)$$

$$g_2(x_1, x_2, \dots, x_n) \leq b_2 \quad (11.5.3)$$

$$\cdot \quad (11.5.4)$$

$$\cdot \quad (11.5.5)$$

$$\cdot \quad (11.5.6)$$

$$g_m(x_1, x_2, \dots, x_n) \leq b_m \quad (11.5.7)$$

$$(11.5.8)$$

其中  $f, g_1, \dots, g_m$  可微且一阶导数连续。

如果有  $\bar{x} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$  是一个最优解, 那么  $\bar{x} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$  肯定是满足这  $m$  个约束条件的, 其次还一定存在一组非负实数  $(\bar{\lambda}_1, \bar{\lambda}_2, \dots, \bar{\lambda}_m)$  使得下式成立

$$\frac{\partial f(\bar{x})}{\partial x_j} - \sum_{i=1}^m \bar{\lambda}_i \frac{\partial g_i(\bar{x})}{\partial x_j} = 0, \quad j = 1, 2, \dots, n \quad (11.5.9)$$

$$\bar{\lambda}_i [b_i - g_i(\bar{x})] = 0, \quad i = 1, 2, \dots, m; \quad (11.5.10)$$

$$\bar{\lambda}_i \geq 0 \quad i = 1, 2, \dots, m \quad (11.5.11)$$

其中  $\lambda_i$  是引入的广义拉格朗日乘子, 就是对应约束的影子价格, 上式也就是这个优化问题的 KKT 条件。

下面我们用一个具体的例子来加深对 KKT 条件的理解。

除此之外, 前文介绍到, KKT 条件成立的一个必要条件是, 强对偶成立。那么什么情况

下强对偶成立呢？这就不得不提到 Slater's condition。

### 11.5.1 Slater's 条件

**Slater's condition:** 如果满足原问题是凸优化问题，并且至少存在一个相对内点（什么叫相对内点？答：就是一个可以让所有不等式约束都取严格  $>$  或者  $<$  的可行点），那么该问题就具有强对偶性。

例如，考虑以下凸优化问题

$$\min f_0(x) \quad (11.5.12)$$

$$s.t. \quad f_i(x) \leq 0, \quad \forall i = 1, 2, \dots, m \quad (11.5.13)$$

$$Ax = b \quad (11.5.14)$$

$$(11.5.15)$$

该问题中如果至少存在一个可行解  $x'$  满足：使不等式约束  $f_i(x')$  取严格  $<$ ，即  $f_i(x') < 0, \forall i = 1, 2, \dots, m$ ，那么该问题的强对偶性成立。

综上，KKT 条件成立的前提：

- 目标函数和约束函数均可微且一阶导数连续；
- 问题强对偶成立 (满足 Slater's condition)；

### 11.5.2 KKT 条件求解非线性规划的小例子

考虑下面的二次规划问题：

$$\max z = x_1^2 - 2x_2^2 + 27x_1 + 45x_2 - 10x_3 \quad (11.5.16)$$

$$s.t. \quad x_1 + x_2 - x_3 \leq 0 \quad (11.5.17)$$

$$x_3 - 17.25 \leq 0 \quad (11.5.18)$$

$$(11.5.19)$$

首先，我们来验证该问题是否为 convex 或者 concave。我们得到目标函数  $\max z = x_1^2 - 2x_2^2 + 27x_1 + 45x_2 - 10x_3$  的海塞矩阵为（我们忽略  $x_3$  这个变量，因为关于  $x_3$  的部分是线性的，我们只考虑关于  $x_1$  和  $x_2$  的部分）

$$H = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix} \quad (11.5.20)$$