**项目进展报告-0409**

# 1. 模型回顾

## 1.1 定义

首先进行必要的定义，如表1-3所示。

表1 下标

|  |  |
| --- | --- |
| 符号 | 含义 |
| *i*, *j* | 某种货币，取值属于[1,*N*]之间，整数 |
| *k* | 某个渠道，取值属于[1,*K*]之间，整数 |
| *o* | 初始货币 |
| *d* | 最终货币 |

表2 变量

|  |  |
| --- | --- |
| 符号 | 含义 |
|  | 为经渠道*k*兑换成货币*j*，使用货币*i*的数量，非负连续变量 |
|  | 是否从渠道*k*将货币*i*兑换成货币*j*，0-1变量 |
|  | 是否将货币*i*兑换成货币*j*，0-1变量 |
|  | 货币*i*的访问序号，非负连续变量 |

表3 参数

|  |  |
| --- | --- |
| 符号 | 含义 |
|  | 渠道*k*中货币*i*的存量，非负数 |
|  | 待兑换的货币*o*的总量，非负数 |
|  | 从渠道*k*将货币*i*兑换成货币*j*的手续费，与渠道有关，非负数 |
|  | 从渠道*k*将货币*i*兑换成货币*j*的手续费，与交易量有关，非负数 |
| *GasBudget* | 手续费上限，正数 |
| *M* | 建模所需，一个足够大的正数 |

## 1.2 汇总

注意，相比之前版本，**增加部分决策变量上限约束**，以便于求解。

由此我们得到一个**带有分式项的非线性混合整数规划模型**，如下所示：



# 2. 测试案例

拟用前2个测试案例作为检测程序正确与否的工具，同时也做展示之用。需要说明的是，目前**并未考虑手续费约束，但保留子环路消除约束**。而测试案例3将专门讨论子环路消除约束的必要性。

## 2.1 测试案例1

仅考虑两个渠道，两种货币，其中的货币存量为，的货币存量为，兑换目标为：将1单位货币全部兑换为货币并最大化所得的货币数量，如图1所示：

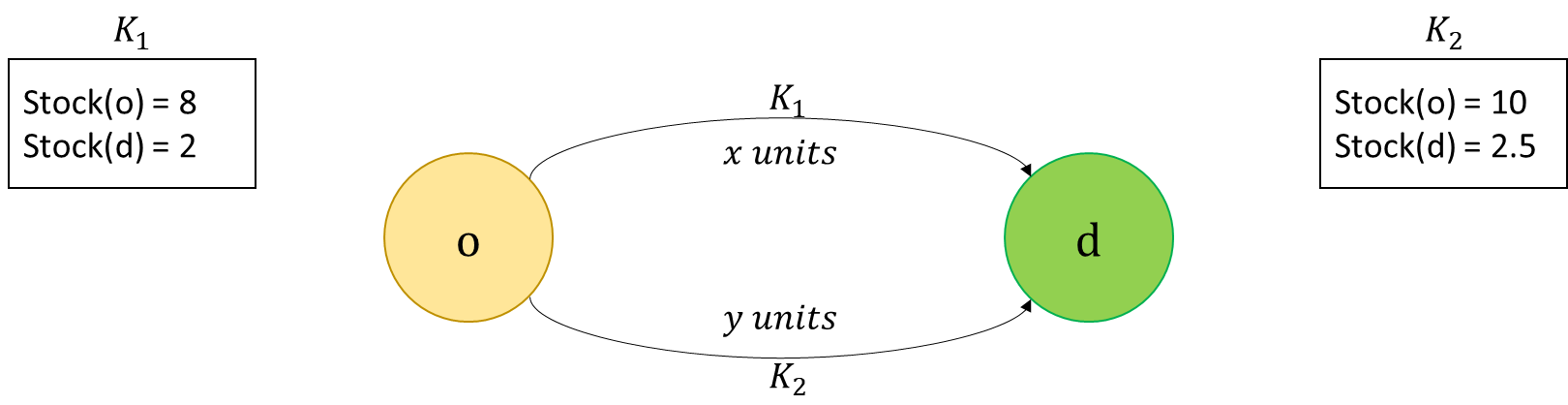
**

图1 测试案例1

经程序计算，最优兑换策略为：将0.4477单位和0.5522单位的货币分别经由渠道兑换为货币，总计获得0.2368单位的货币。

**数学推导：**

## 2.2 测试案例2

仅考虑两个渠道，三种货币，其中的货币存量为，的货币存量为，兑换目标为：将1单位货币全部兑换为货币并最大化所得的货币数量，如图2所示：

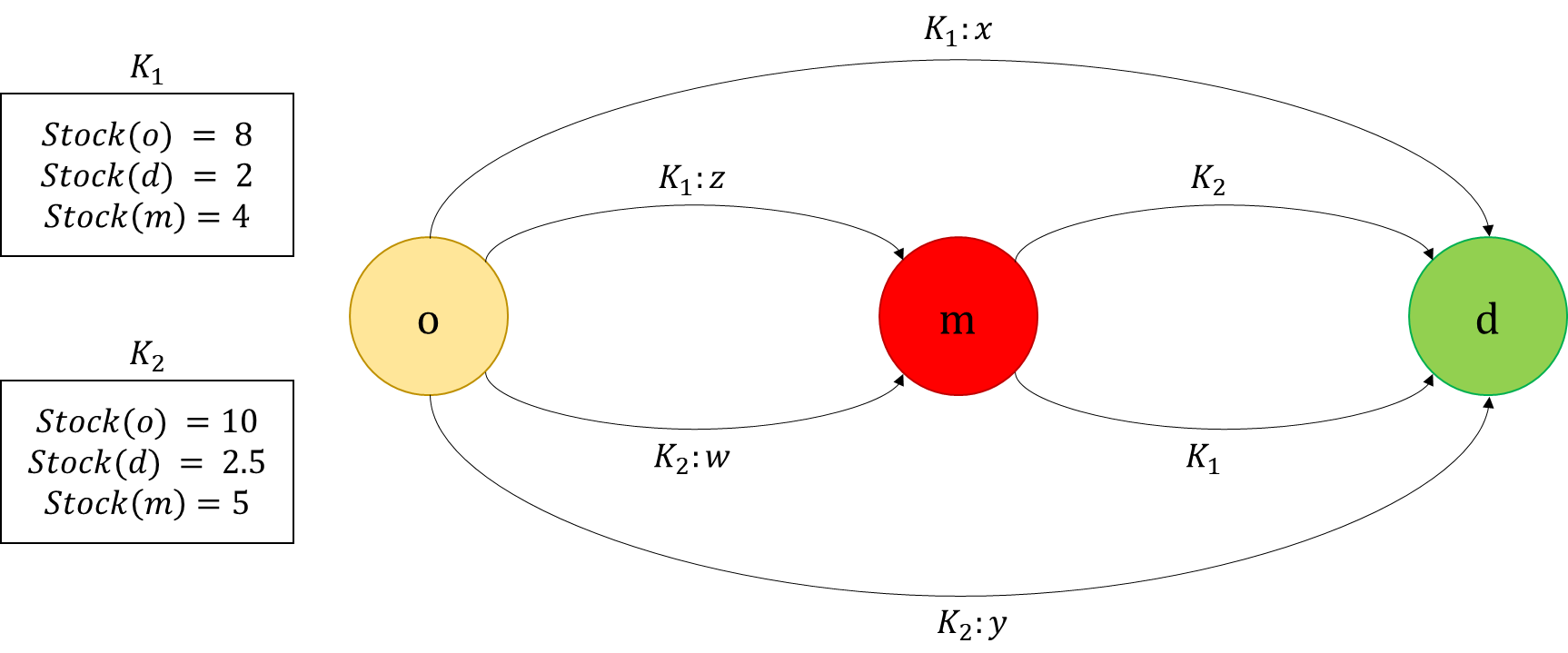
****

图2 测试案例2

经程序计算，使用最优兑换策略总计获得0.2410单位的货币。最优兑换策略如下：

**合理性检查：**本案例相比于案例1引入额外新币种，兑换路径数量增加，且两种货币量保持不变，因而最终收益的货币数量大于案例1

## 2.3 测试案例3

本案例讨论“子环路消除”的约束是否应该去除。

考虑两个渠道，四种货币，其中：的货币存量为，的货币存量为。

（1）保留“去除子环路”约束

兑换方案如图3所示，最佳目标值为1.6299单位：

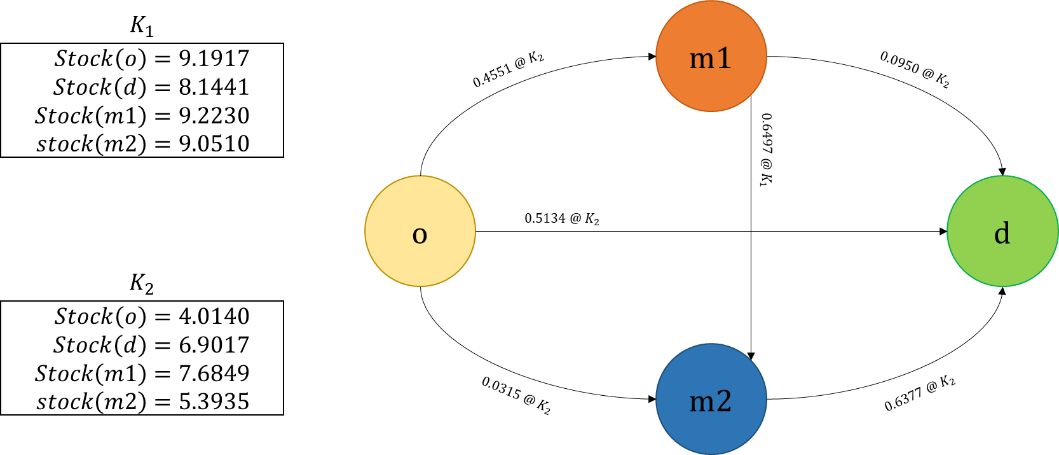


图3 测试案例3-有子环路消除

（2）删除“去除子环路”约束

兑换方案如图4所示，最佳目标值为1.6739单位：

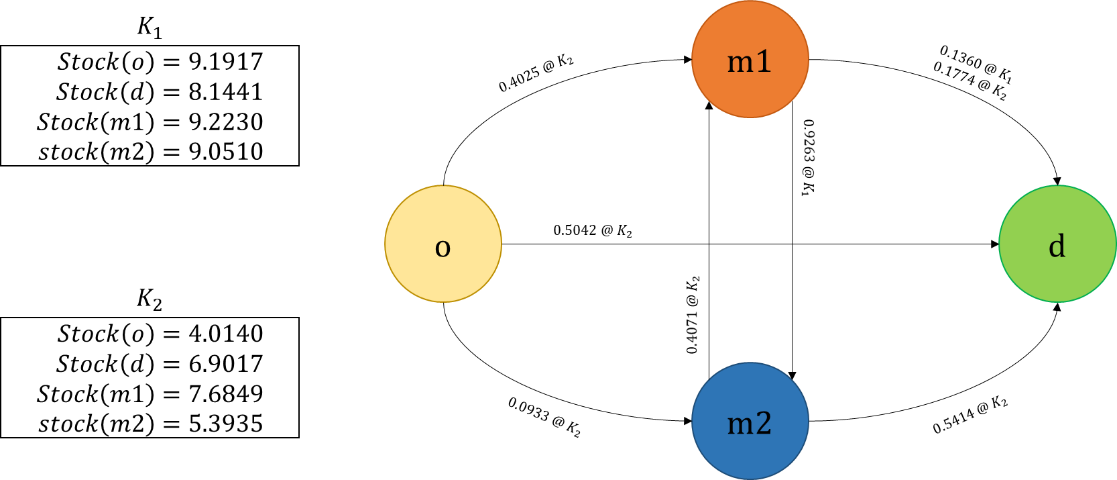


图4 测试案例3-无子环路消除

后者引入了**子环路**，虽然**提高了目标值**，但是在实际应用中此种行为并不合理，应予以避免。对本案例进行100次求解，保留“去除子环路”约束的平均求解时间为0.1316秒，删除“去除子环路”约束的平均求解时间为0.1981秒，时间增长约为50%。究其原因，删除“去除子环路”约束后由于**可行域扩大**，所以会**更加耗时**，因此**应保留“去除子环路”约束。**

# 3. 求解时间的影响因素分析

模型的规模主要由货币种类数量和交易所数量决定，本节研究以上两种因素对于求解时间的影响。所使用的算力信息如下：

* CPU：Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
* 内存：16 GB
* 操作系统：Windows 10专业版
* 解释器版本：Python 3.8.1
* 求解软件：Gurobi 9.1.1

程序首先生成相应数量的货币种类和交易所，每个交易所中的各类货币库存为1到10间的随机数（均匀分布），手续费初始化为0，初始币种持有量为1。

而后，求解每组参数下的决策结果，并记录求解时间（单位：秒），结果如表4所示：

表4 货币种类和交易所数量灵敏度分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **中间货币种类数量**  **交易所数量** | **2** | **3** | **4** | **5** |
| **1** | 0.006 | 0.027 | 0.061 | 1.123 |
| **2** | 0.022 | 0.032 | 0.149 | 0.331 |
| **3** | 0.018 | 0.068 | 0.180 | 0.356 |
| **4** | 0.023 | 0.054 | 0.793 | 121.570 |
| **5** | 0.007 | 0.111 | 0.782 | 200.570 |
| **6** | 0.024 | 0.075 | 0.617 | 10.616 |
| **7** | 0.023 | 0.520 | 9.716 | 2980.800 |
| **8** | 0.023 | 0.307 | 11.027 | 4923.100 |

而后，分别在固定货币种类数量（以C代表）和交易所数量（以E代表）的条件下，对计算时间进行灵敏度分析，如图5所示（为表现时间变化趋势，每条曲线用其最大值归一化）。

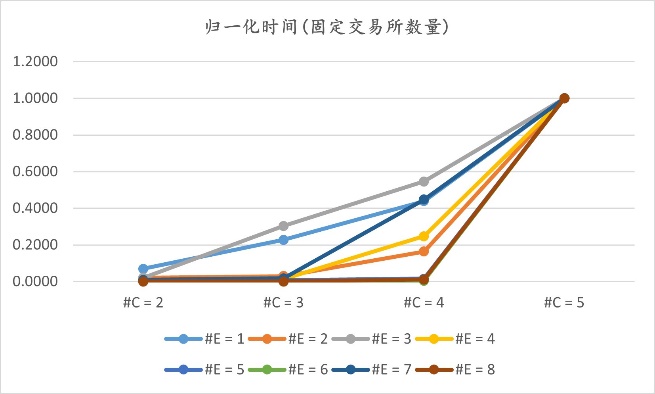
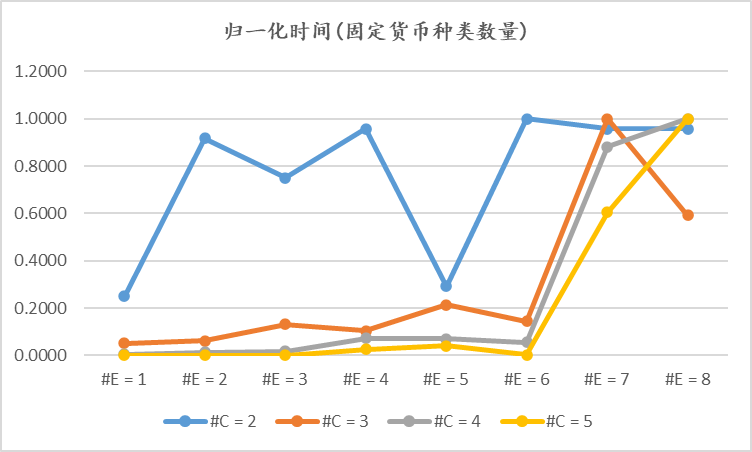


图5 求解时间与货币种类数量和交易所数量的关系

由此可知：

（1）交易所数量对计算耗时的影响关系较为复杂，并非单调递增。

（2）随着货币数量增加，计算时间单调递增。对于大规模问题，计算成本将无法承受，必须进行简化。

# 4. 精度与效率的权衡

以上模型的求解均取得全局最优解，对应的精度要求最高。但在实际应用中，为保障快速计算兑换路径，可考虑适当降低精度要求。精度要求可以用相对间隙（Gap）指标表示，定义为：(可行解目标值-松弛解目标值)/松弛解目标值。实验中观察到，将Gap从1.0%下降到默认值0.01%将消耗绝大部分时间，却不会明显提高解的质量。因此，在Gurobi中，MIPGap参数可调节上述Gap指标的取值。

现对MIPGap做灵敏度分析，首先生成一份含有4种货币7个交易所的案例，以不同的MIPGap参数求解模型，进行10次试验求平均值，并将所得结果记录在表5中。

表5 MIPGap灵敏度分析

|  |  |  |  |
| --- | --- | --- | --- |
| **MIPGap参数** | **当前最优目标值** | **当前最优目标值**  **/全局最优目标值** | **求解时间** |
| 0.01% | 9.790 | 100.000% | 59.599 |
| 0.10% | 9.790 | 100.000% | 26.276 |
| 0.50% | 9.790 | 99.999% | 3.291 |
| 1.00% | 9.790 | 99.997% | 1.555 |
| 2.00% | 9.786 | 99.960% | 0.944 |
| 3.00% | 9.786 | 99.958% | 0.862 |
| 4.00% | 9.758 | 99.674% | 0.770 |
| 5.00% | 9.753 | 99.620% | 0.613 |
| 10.0% | 9.729 | 99.374% | 0.230 |

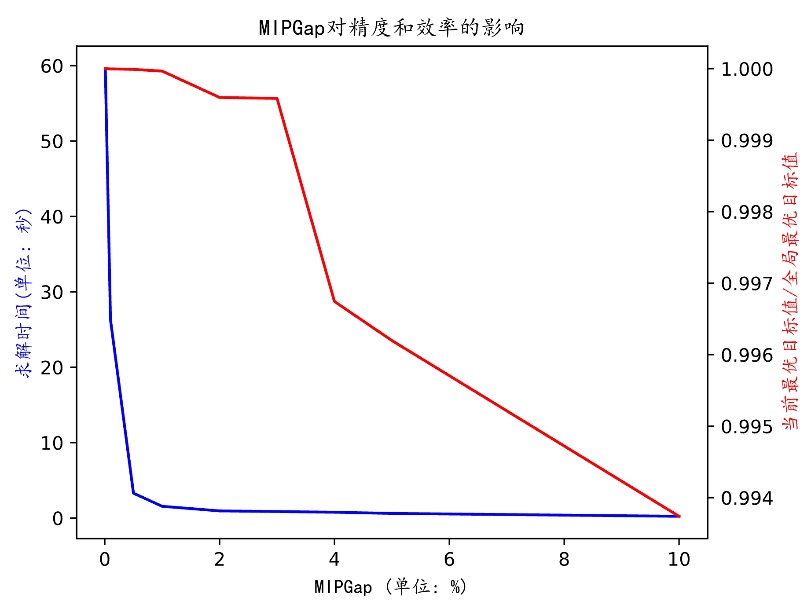
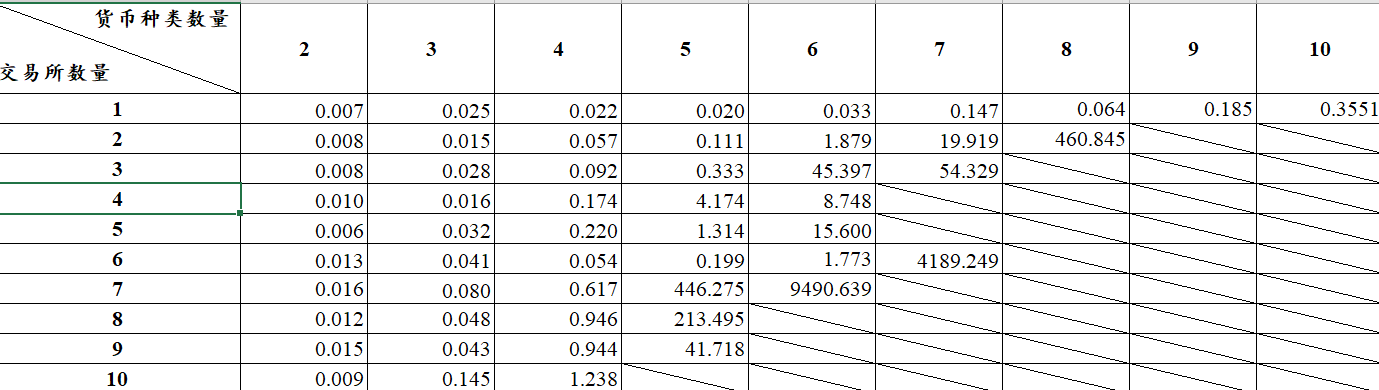


图 6 精度要求对结果质量和计算时间的影响

如图6所示，求解时间和目标值都随MIPGap递减。当MIPGap从0开始略有增长时，精度损失不大，但效率显著提升。在实际应用中，应注意**适当牺牲精度换取效率**。

特别地，由于实际问题中，精度要求不能低于5%，故以MIPGap=0.05来进行试验，结果如下。**其中不显示的结果均为计算时间严重超标**。由此可知，在实际过程中，单纯追求精确解的计算成本过高，必须对模型近似处理。



# 5. 后续工作

**（1）尝试增加一些基于应用场景的限制来简化问题**

* 限制交易阶段或次数
* 限制中转货币种类数（不超过4个），注意不包括初始和目标货币
* 限制交易所数量（不超过8个）

**（2）对数学模型进行近似处理（凸转化）**

* 分段线性去近似分式
* 分式线性化传统方法，保证不引入二次项得到松弛问题

**（3）使用数值优化方法（这个优先级较低）**

* 放松子环路约束，使用SQP等方法做局部最优解。同时采用multi-start方法，并行多个process，对多个局部最优解优中选优。并尝试对所得解进行邻域搜索以保证可行。

（**4）实盘测试**

使用一家交易所（Uniswap），采用网站上的实际数据对模型算法进行测试