中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

学　　校	国防科技大学
参赛队号	22910020061
队员姓名	1.毕德顺
	2.梁观平
	3.夏乐

中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

题 目 基于遗传算法的基本块排布模型

摘 要：

在当今国际社会日益严峻的形势下，芯片作为信息产业的核心，成为了大国之间的兵家必争之处。但我国在高性能芯片领域基础十分薄弱，芯片的算力、体积、稳定性以及精细化程度等方面，都远远落后于世界先进水平。同时随着中美贸易摩擦持续升级以及工业逆全球化的趋势日益增强，提高国产芯片性能，加强我国高精尖芯片的自主研发能力势在必行。本文主要聚焦于可编程交换芯片，讨论在 PISA 架构下如 何对其资源进行更优的排布，以实现最高的资源利用率。

针对问题一，本文建立了基于遗传算法的基本块排布模型。首先将编号为 0-606 的基本块和编号为 607-606+N 的流水线等级标志位组成原始的基本块排布染色体模型。然后通过 PFIH 方式插入将整条染色体打乱，打乱后的染色体通过 N 条相互邻近的流水线等级标志位将基本块分成 N+1 个流水线等级，其中位于两个邻近标志位之间的基本块便被确定为排布到同一流水线等级。为了衡量每次进化的子代的优劣性，本文根据数据依赖约束、控制依赖约束以及资源约束得到的违反三种数据依赖类型的基本块对数VB_{sj}、违反控制依赖的基本块对数VB_{kj}、四种资源的超限量分别记为VR_1、VR_2、VR_3、VR_4，违反约束 5 的流水线等级对数记为VPC_5以及偶数级资源约束的超限等级数VPC_6确定了成本损失函数的惩罚项，然后联合流水线等级数 NPC 确定了损失函数，进而确定了适应度值评分函数。通过 matlab 程序对遗传算法进行 5000 次的迭代，最终本文确定了在满足问题一中各类约束条件下的最优流水线等级数 NPC=78。

针对问题二，同问题一为了衡量每次进化的子代的优劣性，仍需确定要满足的资源约束，得到四种资源的超限量分别记为$VR_1^{'}$、$VR_2^{'}$、$VR_3^{'}$、$VR_4^{'}$，违反约束 5 的流水线等级对数记为$VPC_5^{'}$以及偶数级资源约束的超限等级数$VPC_6^{'}$，进而确定了成本损失函数的惩罚项，然后联合流水线等级数 NPC 确定了损失函数，最终确定了适应度值评分函数。通过 matlab 程序对遗传算法进行 5000 次的迭代，最终本文确定了在满足问题二中各类约束条件下的最优流水线等级数 NPC=64。

关键词：遗传算法，基本块排布，依赖约束，深度优先搜索，邻接矩阵

一、引 言

1.1 问题重述

1.1.1 问题背景

随着电子技术的发展和人类社会科技的进步，芯片的运算速度、体积以及性能已经得到了不小的提升。但传统的芯片功能面狭小，难以匹配多种多样的网络协议，可编程功能的交换芯片(programmable switching ASIC)应运而生。该类芯片可以灵活添加新的网络协议与网络功能，不必因网络协议变换重新设计，大大加快了研发效率。随着网络的不断开放，该芯片的重要性与日俱增，具有广阔的应用前景[1]。

可编程交换芯片与传统芯片相比，最显著的区别在于：传统芯片的报文处理与转发逻辑固化，而在可编程交换芯片中这些可以按需调整。因此设计者可以依据上层系统需求来设置芯片的报文处理与转发逻辑，芯片即会根据预定流程来处理报文。

PISA 架构是当前可编程交换芯片的主流架构，其由识别报文的报文解析、修改报文数据的报文处理流水线以及报文重组三个部分组成[2]。本文主要聚焦于芯片报文处理流水线的资源排布，在满足数据依赖、控制依赖以及资源约束等条件的同时，合理对各级流水线资源进行分配，以实现更高的资源利用率。

1.1.2 问题描述

由于各流水线等级上的基本块之间存在数据依赖、控制依赖以及资源约束，因此如何在满足以上三种约束条件的情况下，将基本块合理分配到各流水线等级上是可编程交换芯片面临的一个难题。题目分别给出了各基本块所需资源数据、各基本块读写变量情况以及各基本块邻接情况，本文需要根据以上三个数据解决以下问题：

问题一：通过给定数据，分析各基本块之间的依赖关系，建立各基本块之间的关系程序流程图。同时依据流图确定各基本块之间的数据依赖和控制依赖关系，再结合各类资源使用情况，建立起基本块排布的模型。在满足资源利用率最高的条件下，给出基本块的排布表。

问题二：在上题建立的基本块排布模型上，对三种约束条件进行修改，将各流水线等级上同一执行流程所需资源的最大值作为本流水线等级所需资源，即放宽资源约束条件，求解流水线等级数更少的基本块排布方案。

1.2 模型假设

1、假设参加交叉过程的个体数占种群的比例为 0.9；

2、假设子代中发生变异的个体占种群的比例为 0.05。

1.3 符号说明

符号	符号含义
N	基本块序号后插入的分段标志数量
M_NEIGH_{k*k}	邻接矩阵

$block_i$	第 i 号基本块
M_ACCESS_{k*k}	可达矩阵
$M\_CONFILICT_{c*c}$	基本块间的变量冲突矩阵
Xi	变量 i
M_SJTYPE_{k*k}	数据依赖矩阵
ind_i	基本块 i 在整条染色体的位置索引
$\grave{ind}_i$	基本块 i 在某一流水线的位置索引
VB_{sj}	违反三种数据依赖的基本块对数
α	违反数据依赖的惩罚系数
chrom	一条染色体：基本块序号排列
Index i	基本块 i 在 chrom 中的位置索引
violate_control	排列中违反控制依赖的对数
VB_{kz}	违反控制依赖的基本块对数
β	违反控制依赖的惩罚系数
PCB{i}	流水线中的第 i 级
ri	第 i 种资源
violate_ri	第 i 种资源的超限量
VR_i	第 i 种资源的超限量（简化表示）
flag	流水线的资源标志位
NPC	流水线总的级数
pc	级数遍历变量
violated_pc5	违反约束 5 的级数对的量

VPC_i	违反约束 i 的流水线等级对数(i 取 5,6)
violated_pcnum	TCAM 资源的偶数级超限量
θ	违反约束 1-4 的惩罚系数
γ	违反约束 5,6 的惩罚系数
$F_{\cos t}$	成本损失函数
F_{fit}	适应度函数
$M\_Adjacency_{k*k}$	k 阶标志矩阵
$VR_i^{'}$	第 i 种资源的超限量（问题二）
$VPC_i^{'}$	违反约束 i 的流水线等级对数（问题二）
$\theta^{'}$	违反约束 1-4 的惩罚系数（问题二）
$\gamma^{'}$	违反约束 5,6 的惩罚系数（问题二）

二、问题一的分析与求解

2.1 问题一的分析

首先从求解各基本块之间的邻接关系入手，得出各基本块之间控制依赖关系。通过题中给出的邻接基本块信息（即附件三），使用 Python 中的 pyvis 库绘制出来各个基本块间的程序流图，如图 2.1 所示。

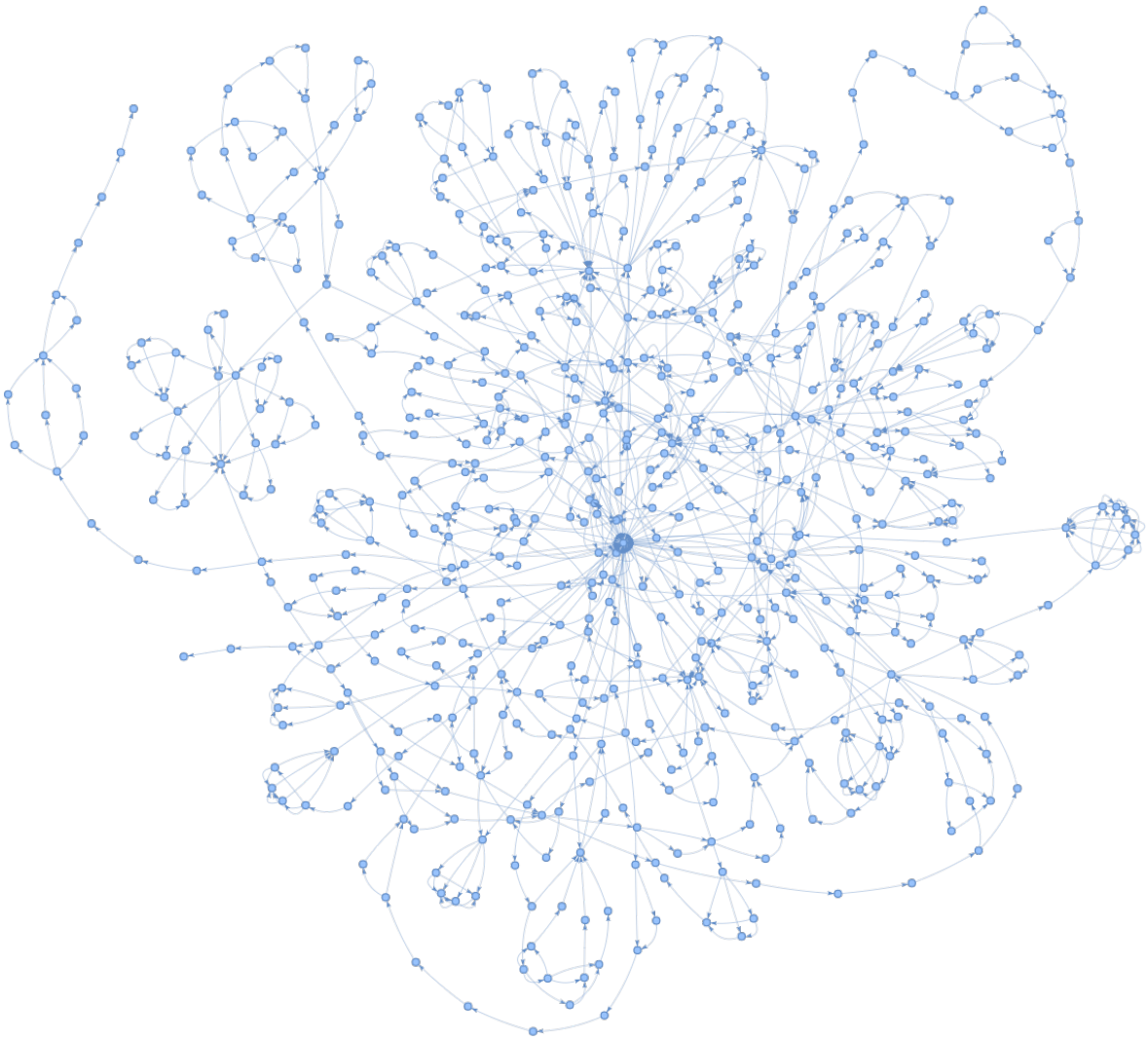


图 2.1 基本块单向流图

通过对图分析得出，出度为零的叶子节点为：2、88、360、381、454；入度为零的起始节点只有一个：365，如图 2.2 所示。

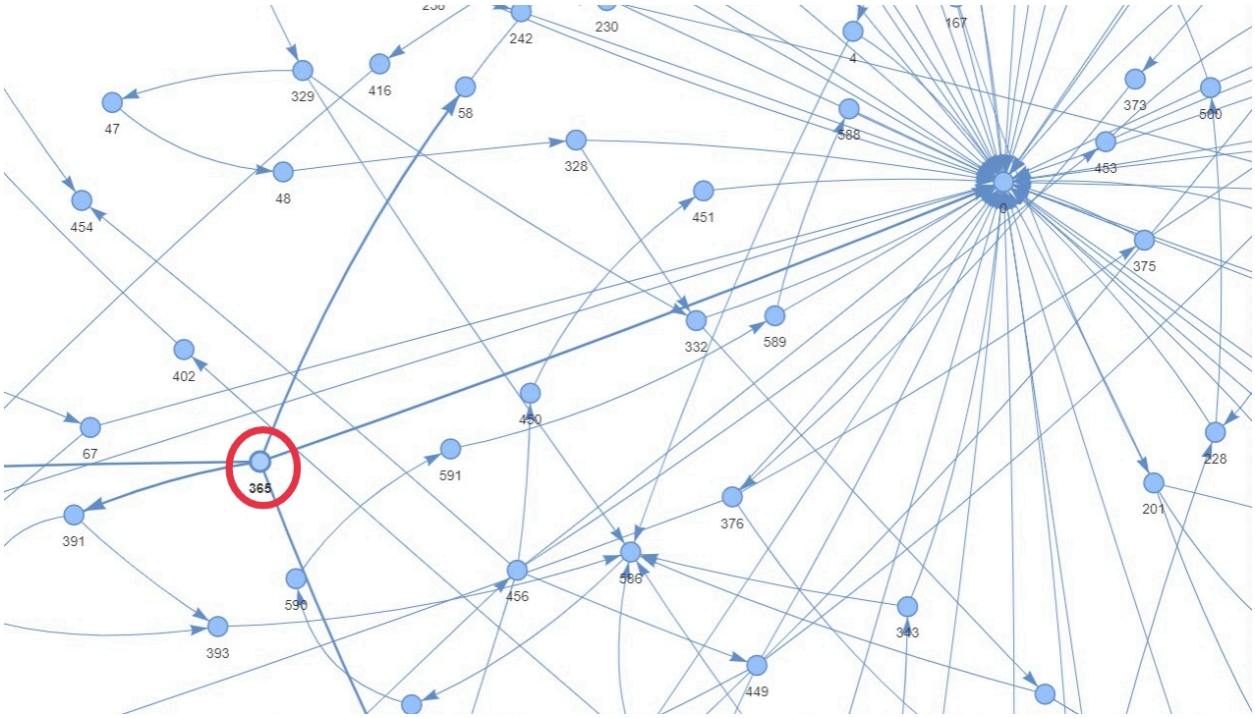


图 2.2 起始节点附近的单向流图（放大）

由于基本块个数较多，需要分析的排布组合数量极多，本文选用了遗传算法来进行求解。

2.2 基于遗传算法的基本块排布模型

所谓遗传算法，主要是借助自然选择理论与遗传学说，将需解决的问题转变为一个类似生物进化的过程[3]。在进化过程中，通过对染色体进行复制、杂交、变异等操作产生下一代。接着通过适应度函数对个体进行评分，淘汰掉适应度值低的部分。经多次迭代，产生适应度值高的个体，以求得到最优化的个体。

如图 2.3 所示，一条染色体由不同的 01 组合构成，遗传算法可以让染色体在迭代过程中通过交叉、变异等操作（如图 2.4 所示）获得自己想要的 01 组合。由于基本块排布问题也是将基本块分派到不同的流水线等级上，也即是不同的数字在满足约束条件下的组合问题，因此本文建立了基于遗传算法的基本块排布模型。

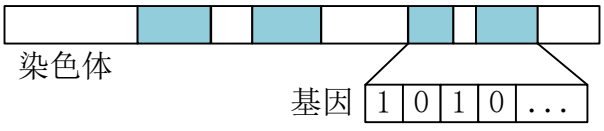


图 2.3 染色体示意图

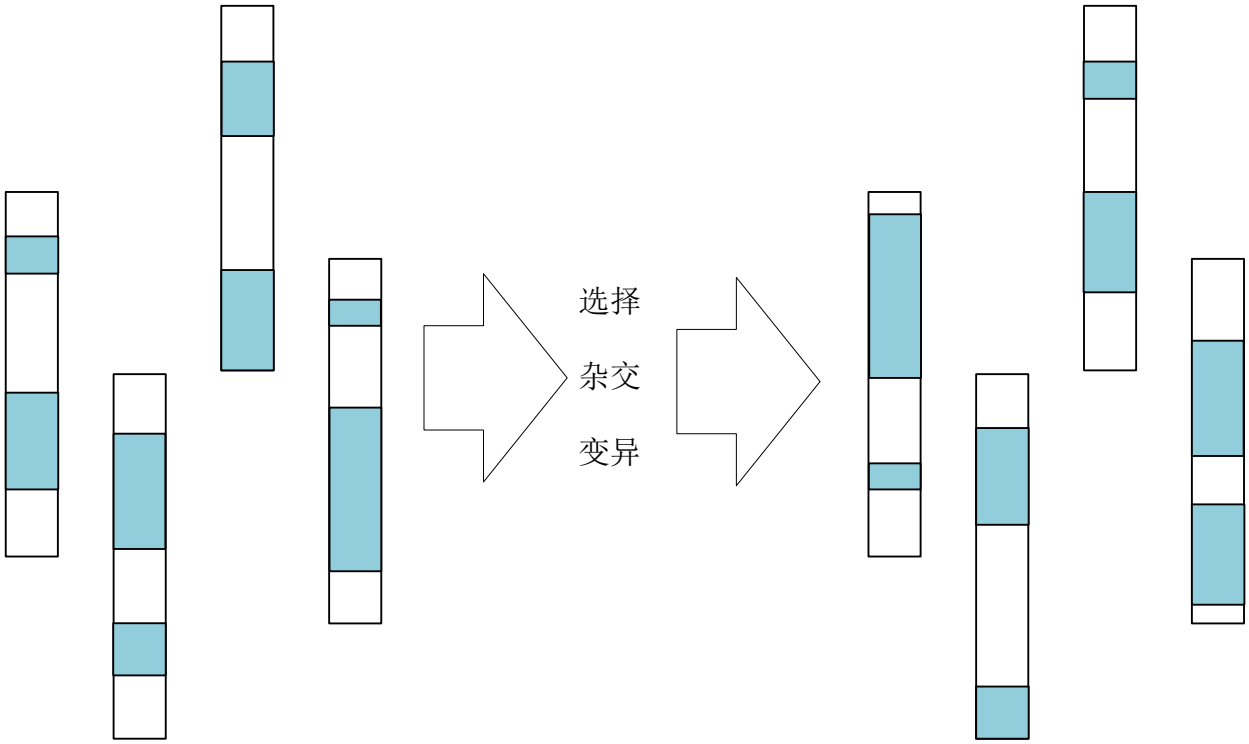


图 2.4 迭代过程

2.2.1 初始数据表示

如图 2.5 所示，本文将基本块编号 0-606 依次排列组成染色体白色基因部分，并假设流水线等级数为 N+1。为了将基本块排布到 N+1 个流水线等级上，文中设置了 N 个标志值依次排列在基本块编号后面，即染色体蓝色基因部分。白色基因部分和蓝色基因部分共同组成了本文的基本块排布染色体模型。

0	1	2	…	606	607	608	…	606+N

图 2.5 基本块排布染色体模型

其中标志值挑选的均为大于基本块最大序号 606 的 N 个连续值，因此只需求得邻近的两个大于最大基本块序号的标志值位置，然后将它们之间的基本块定义为在这一流水线等级即可。如图 2.6 所示为随机生成的染色体的一部分片段，假设标志值 624 与 634 之间没有其他大于 606 的标志值，便可以将以 2 起始，469 终止的基本块分布到同一流水线等级。

…	624	2	16	…	354	469	634	…

图 2.6 随机生成的染色体片段

2.2.2 母代种群

通过 PFIH 式插值将整条染色体打乱[4]，便可以得到各种排列不同的染色体。本文通过这种方式获得了 100 个候选染色体组，又称母代种群。

2.2.3 适应度值评分函数

为了评估随机得到的母代种群质量，本文通过数据依赖、控制依赖以及资源约束建立了适应度评分函数。适应度值相近的染色体组，会倾向于结合产生更优的子代个体。

下面是数据依赖约束、控制依赖约束以及资源约束的定义以及实现情况，通过这三类约束与其对应的惩罚因子便可以联合流水线等级个数共同组成成本损失函数，从而进一步得到适应度值评分函数。

（1）数据依赖约束

数据依赖是语句或代码块间数据流造成的一种约束。这种约束具体来说就是规范语句或者代码块间的正确执行顺序，以防止它们对同一变量进行读写操作时产生三类冲突：写后读冲突，写后写冲突以及读后写冲突。因此为了对数据依赖进行建模，必须寻找合适表征执行顺序的方式。首先已知 attachment3.csv 文件中储存了各基本块在流程图中的邻接基本块信息。因此可以利用程序轻松地得到各基本块的邻接矩阵，命名为 M_NEIGH_{k*k}，具体定义如公式(2-1)所示：

$$M\_NEIGH_{k*k}=\begin{cases} m_{i,j}=0, block_i 与 block_j 不邻接 \\ m_{i,j}=1, block_i 与 block_j 邻接 \end{cases} \tag{2-1}$$

其中，k 表示基本块编号，$i\le k$，$j\le k$。

虽然邻接矩阵可以表征相邻基本块的执行顺序，但是在整个代码里面，需要知道的是各基本块的执行顺序，因此便需要求解有向通路，在有向通路上依此经过的编号就是代码块的执行先后顺序。有向通路在程序中并不好直接表示，因此本文使用有向可达矩阵间接表征有向通路，矩阵名为 M_ACCESS_{k*k}，定义如公式(2-2)所示：

$$M\_ACCESS_{k*k}=\begin{cases} m_{i,j}=0, block_i 与 block_j 不可达 \\ m_{i,j}=1, block_i 与 block_j 可达 \end{cases} \tag{2-2}$$

其中，k 表示基本块编号，$i\le k$，$j\le k$。

可达矩阵 M_ACCESS_{k*k} 邻接矩阵的计算表达式见公式(2-3)：

$$M\_ACCESS_{k*k}=M\_NEIGH^1+M\_NEIGH^2+\cdots+M\_NEIGH^k \tag{2-3}$$

因此很容易通过 matlab 将可达矩阵计算出来。

然后并不是所有基本块都需要严格按照有向通路顺序去执行，只有当它们之间存在变量冲突时才需要考虑执行顺序，因此求出对于变量 X_i，基本块的冲突集合 $M\_CONFILICT_{c*c}$ 是非常重要的。本文已知 attachment2.csv 文件中储存了各基本块对于变量的读写信息，显然，这不能很直观的看出各级本块是否对某一变量存在读写冲突。因此，本文首先对存储格式进行变换得到两个表格：变量读冲突表格 R.xlsx 以及变量写冲突 W.xlsx。通过观察发现编号为 0 的基本块只对变量进行了写操作，因此本文利用这一特性，求解出 $M\_CONFILICT_{c*c}$，其定义如公式(2-4)所示：

$$M\_CONFILICT_{c*c}=\begin{cases} m_{i,j}=+n, 基本块 block_j 对变量 X_i 存在写操作 \\ m_{i,j}=-n, 基本块 block_j 对变量 X_i 存在读操作 \end{cases} \tag{2-4}$$

其中 c 表示变量 X_i 的个数，n 表示基本块 $block_j$ 的编号。

本文利用有向可达矩阵 M_ACCESS_{k*k} 以及变量冲突矩阵 $M\_CONFILICT_{c*c}$ 求解数据依赖约束算法，如下所示：

数据依赖约束算法实现

输入：有向可达矩阵 M_ACCESS_{k*k} 及 $M\_CONFILICT_{c*c}$

输出：数据依赖类型矩阵 M_SJTYPE_{k*k}

```
1  For row = 1 To c:
2   For col1 = 1 To c:
3    If M_CONFILICT[row,col1]不为空，从头开始遍历本行
4     For col2 = 1 To c
5      if M_CONFILICT[row,col2]也不为空，则分别将abs(M_CONFILICT[row,col1])+1
         以及abs(M_CONFILICT[row,col2])+1作为M_ACCESS_{k*k}的行列索引
6      If M_ACCESS⌈abs(M_CONFILICT[row,col1])+1,abs(M_CONFILICT[row,col2])+1⌉
         等于 1，即存在通路
7        根据M_CONFILICT[row,col1]以及M_CONFILICT[row,col2]的正负性对
         M_ACCESS 进行幅值，以此进行数据依赖类型判定
8     End For
9   End For
10 End For
```

为了对数据依赖类型矩阵 M_SJTYPE_{k*k} 有更加直观的认识，对其进行如下定义，见公式(2-5)：

$$M\_SJTYPE_{k*k}=\begin{cases} m_{i,j}=0, \text{基本块} block_i \text{与基本块} block_j \text{不存在通路} \\ m_{i,j}=1, \text{基本块} block_i \text{与基本块} block_j \text{存在通路} \\ m_{i,j}=t, \text{基本块} block_i \text{与基本块} block_j \text{存在数据依赖} \end{cases} \tag{2-5}$$

其中，k 表示基本块编号，$i \le k$，$j \le k$。t 表示数据依赖类型，$t=2$ 表示存在写后读依赖，$t=3$ 表示存在写后写依赖，$t=4$ 表示存在读后写依赖。

通过阅读可知在 PISA 架构中，当基本块 $block_i$ 和 $block_j$ 存在写后读数据依赖或写后写数据依赖时，基本块 $block_i$ 排布的流水线级数需要小于基本块 $block_j$ 排布的级数；当基本块 $block_i$ 和 $block_j$ 存在读后写数据依赖时，基本块 $block_i$ 排布的流水线级数需要小于等于基本块 $block_j$ 排布的级数。为了将数据依赖作为约束惩罚条件，本文对违反三种数据依赖类型的基本块对数进行统计。因此当基本块违反写后读数据依赖或写后写数据依赖时基本块 $block_i$ 排布的流水线级数需要大于等于基本块 $block_j$ 排布的级数。由前面流水线级数和各基本块建立的染色体模型可得基本块 $block_i$ 排布的流水线级数需要大于等于基本块 $block_j$ 排布的级数等价于 $block_i$ 在整条染色体的位置索引 ind_i 大于 $block_j$ 的位置索引 ind_j 的情况与当 $block_i$ 和 $block_j$ 在同一流水线级数时 $block_i$ 在这段流水线等级染色体的位置索引 $\grave{ind}_i$ 小于 $block_j$ 的位置索引取 $\grave{ind}_j$ 并集，即

$$Violated_{2,3}=C\left(ind_i>ind_j\right)\cup C\left(\grave{ind}_i<\grave{ind}_j\right) \tag{2-6}$$

其中 ind_i 表示 $block_i$ 整条染色体的位置索引，$\grave{ind}_i$ 表示 $block_i$ 在某段流水线等级染色体时的位置索引。以上三种情况如图 2.7，图 2.8，图 2.9 所示：

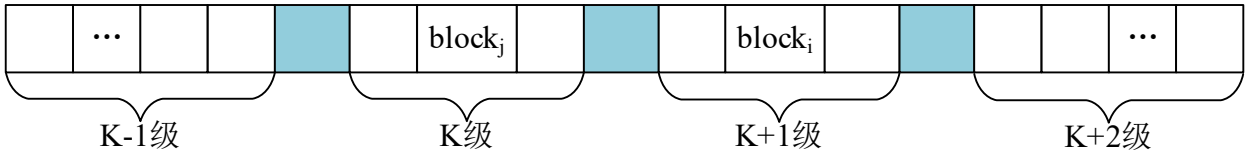


图 2.7 基本块分处不同级数时

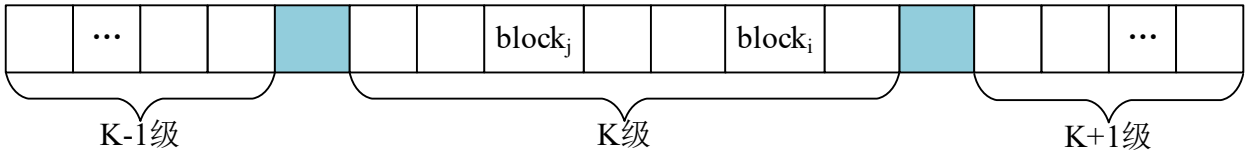


图 2.8 基本块处于同一级数时

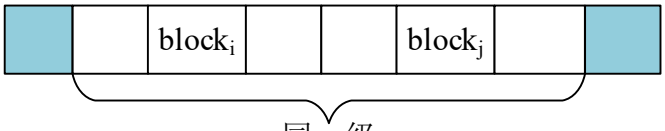


图 2.9 基本在整条染色体上

然而当基本块违反读后写数据依赖时，并不是基本块 $block_i$ 排布的流水线级数需要小于或等于基本块 $block_j$ 排布的级数情况的取反，即并不等同于基本块 $block_i$ 排布的流水线级数需要大于基本块 $block_j$ 排布的级数。这是因为读后写数据依赖本身就存在执行的先后顺序，因此就算 $block_i$ 与 $block_j$ 在同一流水线等级，只要 $\grave{ind}_i > \grave{ind}_j$ 也是符合违反读后写数据依赖的，如图 2.10 所示：

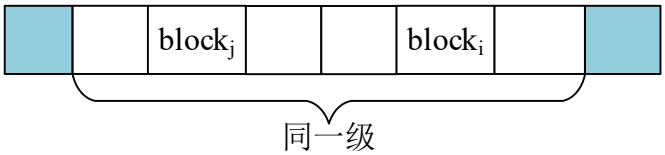


图 2.10 违反读后写数据依赖的情况

于是将这两种情况进行合并就只需要满足 $block_i$ 在整条染色体的位置索引 ind_i 大于 $block_j$ 的位置索引 ind_j，即：

$$Violated_4 = C\left(ind_i > ind_j\right) \tag{2-7}$$

其中 ind_i 表示 $block_i$ 整条染色体的位置索引。

通过对以上三种数据依赖约束情况的建模，代入 matlab 程序便可以求解得到违反三种数据依赖类型的基本块对数，记为 VB_{sj}。并为其添加惩罚因子 α，作为遗传算法的成本损

失函数的一项，不断迭代，将其优化为 0。

（2）控制依赖约束

根据文中相关解释，控制依赖是指当从某个基本块出发的路径，只有部分路径通过下游某个基本块时，两基本块之间构成控制依赖。即是在一个程序流程图中，当一个基本块出现分支后，开始产生控制依赖；直到基本块产生的分支全部融合到下游某个基本块时，控制依赖结束，则产生分支的基本块与分支经过的所有基本块都构成控制依赖关系。如 D 题图 A-2 中所示，B1 基本块产生的分支经过 B2，B5，B6，B7，B8 直到 B3 分支融合才结束，则 B1 与 B2，B5，B6，B7，B8 都具有控制依赖关系。

定义控制依赖顺序为具有控制依赖关系的基本块要满足的流水线级数顺序，即 A 排布的流水线级数需要小于或等于 B 排布的流水线级数($A \le B$)，chrom 为整条染色体包含所有基本块号的随机排列。只要 chrom 中 A 的索引在 B 的索引前面即 $IndexA < IndexB$，后续对 chrom 进行切分，则 A 和 B 有可能分到同一级，级数相同 $A = B$，也可能分到不同级则 A 的级数在 B 的前面 $A < B$，即最后得到的流水线每级的基本块排布都满足控制依赖顺序 $A \le B$；所以只要保证在 chrom 中 $IndexA < IndexB$ 就能保证 $A \le B$。如公式(2-8)所示：

$$IndexA < IndexB = \begin{cases} A < B & \text{A的级数} < \text{B的级数} \\ A = B & \text{A与B在同一级流水线上} \end{cases} \tag{2-8}$$

满足控制依赖的两种情况如图 2.11、图 2.12 所示：

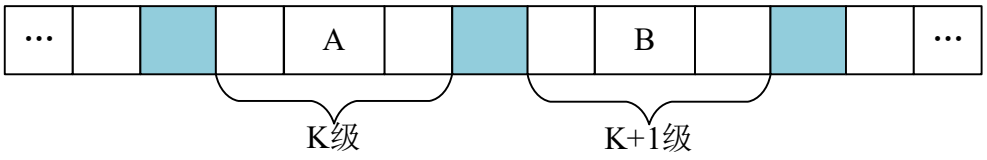


图 2. 11　A 与 B 分处不同级数时

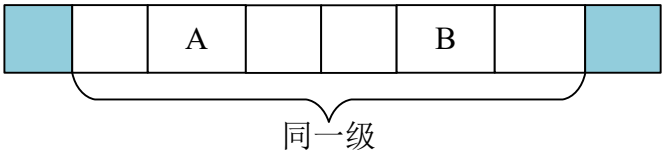


图 2.12　A 与 B 同处一级

实现时，利用两层循环遍历 attachment3 矩阵。如图 2.13 所示：

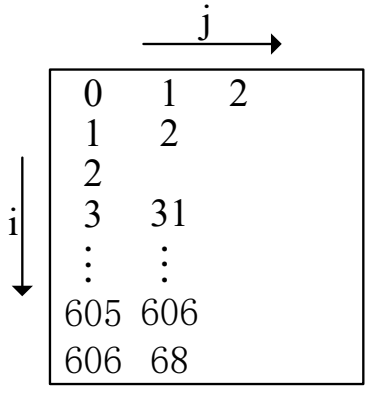


图 2.13 控制依赖遍历方式

从上图可以看出外层循环由 i 遍历矩阵行，内层循环由 j 遍历矩阵列，对于 attachment3 矩阵中每行数据大于 2 的行，取出此行第 1 列元素（父节点）在 chrom 中的索引 *Index*1，遍历 attachment3 第 i 行的其余列（每个子节点），依次取出子节点索引 *Index*2，比较 *Index*1 与 *Index*2，如果 *Index*1 > *Index*2 则 violate_control 加 1，这里 violate_control 是指不满足控制依赖顺序的对数，算法如下所示：

控制依赖约束算法
Input：包含所有基本块的整条染色体 chrom，附录 3 矩阵 attachment3
Output：chrome 中不满足控制依赖顺序的对数 violate_control
1 [r,c] = size(attachment3);
2 **For** i ← 1 **to** r **do**
3 **If** attachment3(i,:) > 2 **then** #每行数据大于 2 才产生分支
4 Index1 ← 找到附录 3 中第 i 行父节点在 chrome 中的索引;
5 **For** j ← 1 **to** c **do**
6 Index2 ← 找到附录 3 中第 i 行子节点在 chrome 中的索引;
7 **If** Index1>Index2 **then**
8 violate_control += 1;

上述算法统计了 chrom 中违反控制依赖的基本块对数 VB_{kz}，为其添加惩罚因子 β，作为遗传算法的成本损失函数的一项，不断迭代，将其优化为 0。

（3）各类资源约束

题中对流水线每级资源的使用量做了一定限制，并对特殊的折叠级数与偶级数的资源使用情况也做了相应规定。这些资源约束可大致分为三类。

约束 1-4：流水线每级资源约束

对流水线某一级的所有基本块 PCB{i}，根据附件一 attachment1.csv 统计出该级流水线对四类资源 TCAM、HASH、ALU、QUALIF 的使用情况。之后对其分别累加计数，求出资源超限量并存储于 violate_r1, violate_r2, violate_r3, violate_r4 中；设置标志位 flag 表示不合格的流水线，初始值为 0。对于某一级 PCB{i}，若有一类资源使用情况超过限制则置标志位 flag 为 1。Judge_resource 算法如下所示：

约束 1-4 算法
Input：流水线的一级 PCB ,附录 1 数据矩阵 attachment1
Output：超出资源 n 的数量 violate_rn，不合格的流水线标志 flag
1 flag,violate_r1,violate_r2,violate_r3,violate_r4 ← 0,-1,-2,-56,-64;
2 **for** i ← 1 **to** length(PCB) **do**
3 r ← 在 attachment1 矩阵第一列中寻找 PCB(i)的索引;
4 violate_rn ← 对每个 r 索引统计四类资源使用;
5 **if** 四类资源中有一个使用数量>0 **then**
6 flag = 1;

为易于表述，本文摘要将问题一中的四种资源超限量也分别记为 VR_1，VR_2，VR_3，VR_4。

约束 5：流水线折叠级数资源约束

由题目已知流水线第 0 级与第 16 级，第 1 级与第 17 级，……，第 15 级与第 31 级为

折叠级数。假设现在已知流水线总级数 NPC，则判断折叠级数存在的条件为 pc<16&&(pc+16)<NPC，其中 pc 为级数遍历变量。算法实现如下所示：

约束 5 算法
Input：某级流水线 PCB , 基本块所需资源数据 attachment1，流水级总级数 NPC
Output：违反约束 5 的流水线等级对数 violated_pc5
1 初始化 violated_pc5 ← 0;
2 **For** pc ← 1 **to** length(PCB)/2
3 **if** pc<16&&(pc+16)<NPC
4 对这两个流水线等级代入约束 1-4 的 Judge_resource 算法进行是否违反相关资源判断
5 如果违反则对 violated_pc5 进行+1

最后将通过算法得到的违反约束 5 的流水线等级对数 violated_pc5 记为VPC_5。

约束 6：偶数级数量约束

传入流水线总级数 NPC，对于流水线的每一级 PCB{i}，利用 Judge_resource 算法得出每个 PCB 的 violate_r1（TCAM 资源），计算违反偶数级数量约束的超限数 violated_pcnum，所得值为负则表示未超限。算法如下所示：

约束 6 算法
Input：流水线一级 PCB ,附件一数据 attachment1，总级数 NPC
Output：偶数级资源约束的超限数 violated_pcnum
1 violated_pcnum ← -5; # 初始化为限制数 -5
2 **for** i ← 1 **to** NPC i += 2 **do** # 步长为 2 只统计偶数级
3 [~, violate_r1, ~, ~, ~] ← 约束 1-4 算法(PCB{i},attachment1);
4 **if** violate_r1 > -1 **then** # >-1 表示 TCAM 资源被使用
5 violated_pcnum+=1；

本文将偶数级资源约束的超限数 violated_pcnum 记为VPC_6。

约束 7：基本块所在级数唯一约束

基本块排布染色体模型，基本块的编号唯一，则优化后对整条染色体自动切割时，每个基本块必然会被分配到流水线中唯一的某一级，因此对约束 7 自动满足。

通过以上对数据依赖约束、控制依赖约束以及资源约束，本文分别得到了违反三种数据依赖类型的基本块对数VB_{sj}，违反控制依赖的基本块对数VB_{kz}，四种资源超限量分别记为VR_1、VR_2、VR_3、VR_4，违反约束 5 的流水线等级对数记为VPC_5以及偶数级资源约束的超限等级数VPC_6。因此成本损失函数定义如公式(2-9)所示：

$$F_{\cos t} = \mathrm{NPC} + \alpha * VB_{sj} + \beta * VB_{kz} + \theta * \sum_{i=1}^{4} VR_i + \gamma * \sum_{j=5,6} VPC_j \tag{2-9}$$

其中 NPC 表示流水线等级数，α、β、θ、γ表示每项的惩罚系数，且只有当各项优化至 0 时，才是优化流水线等级数的正式开始。

适应度值评分函数就是对成本损失函数取倒数，即：

$$F_{fit} = \frac{1}{F_{\cos t}} \tag{2-10}$$

2.2.4 筛选子代

本文通过对题中各类约束条件进行分析并建模，建立了一个适应度评价函数。当第 n 代子代形成后，使用该函数对其进行评分，筛除适应度值低的个体，并保留相应适应度值高的母代，准备下一次迭代或是输出最优结果。由题意分析可知，相关约束条件主要有：资源约束、控制依赖以及数据依赖，遗传算法如下所示：

遗传算法
Input：[基本块序号,N-1 个插值]，P_c，P_m
Output：最优的个体（基本块分布）
1　PFIH 插值生成 N 个初始个体
2　通过 $F_{\cos t}$ 函数求得个体的成本损失值
3　**For** i = 1 **to** 子代数量
4　　通过自然选择筛出适应度值相近的两个母代
5　　**If** rand(1)<P_c　**then**
6　　母代进行杂交
7　　**If** rand(1)<P_m　**then**
8　　子代某块基因变异
9　　**If** 子代 $F_{\cos t}$ 值满足要求　**then**
10　　接受
11　**End For**
12　更新种群个体
13　判断是否达到要求，若无则继续循环

最终遗传算法收敛后的结果模型，如图 2.14 所示：

…	23	45	67	671	…	456	…	653	…	524	…	623	656	673	621

图 2.14 收敛结果示例

如上所示，经过优化后的子代在均满足上述约束条件的情况下，多余的标志位会不断聚集在染色体后端，且其中不包含基本块。之后将多余标志位删除，便可得到最终的流水线基本块排布，如图 2.15 所示：

…	23	45	67	671	…	456	…	653	…	524	…	623

图 2.15 流水线基本块排布示例

2.3 问题一的结果及其分析

通过 matlab 对遗传算法编程求解即可得到流水线最优等级；
遗传算法的迭代曲线如图 2.16 所示：

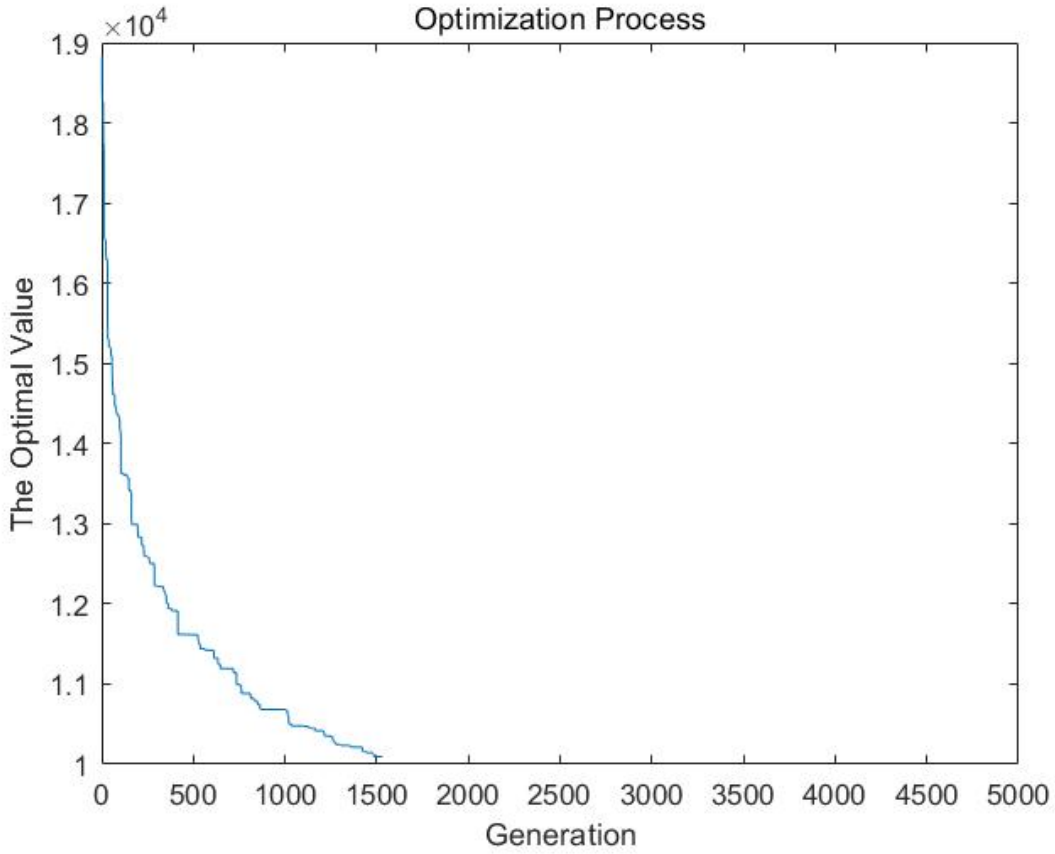


图 2.16 遗传算法迭代曲线

从上图可以看出，随着迭代次数的增加，最优值也逐渐减少，即成本损失函数中的各类约束正在不断优化。但是由于时间原因，程序并未达到设置的迭代次数，此处显示的是迭代次数 1500 的结果，最终迭代曲线以及最终的基本块排布方案见附件 result1.csv。

三、问题二的分析与求解

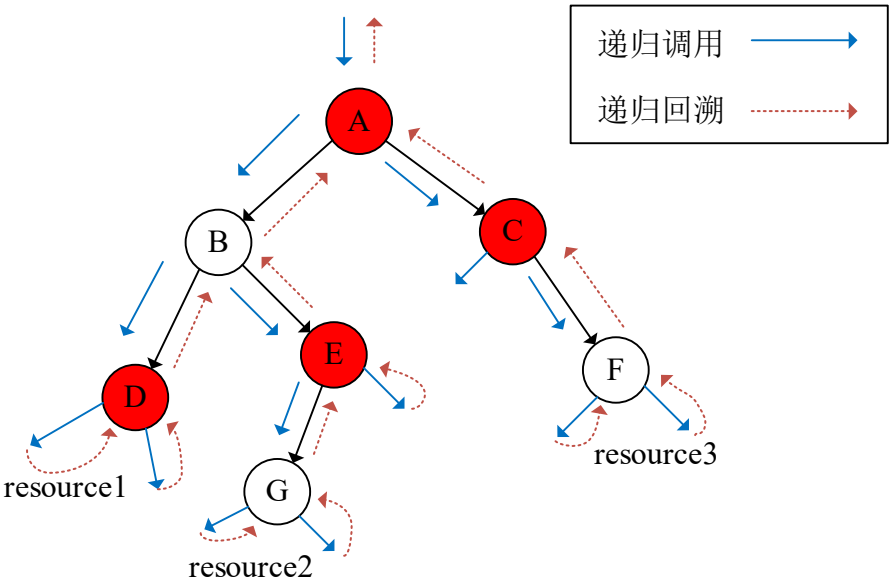
3.1 问题二的分析

问题二中，将问题一资源约束（2）（3）（5）进行了细微的修改。问题二中增加了同一执行流程这一限制条件，因此统计某一流水线等级某资源所需数并不是简单将流水线某一级上的该资源所需量求和。通过分析，本文将依次统计某一流水线等级上所有同一执行流程的资源需求数，并将需求最大的资源数作为同一执行流程的资源需求数。因此问题二的约束条件相对于问题一有所放松，因此理论上能够求解出流水线等级数更少的基本块排布方案。

3.2 问题二的求解

3.2.1 同一流程资源统计建模

求解问题二最关键的就是如何对同一执行流程进行有效的判定，由已知定义，在程序流程图中，由一个基本块出发可以到达另一个基本块则两基本块在一条执行流程上，反之不在一条执行流程上。因此我们建立同一流程资源统计算法模型 SameProcess，对流水线的各等级 PCB，依次遍历该级的基本块 PCB{i}，并把 PCB 中存在的基本块号在邻接矩阵 M_NEIGH_{k*k} 对应位置中标红，邻接矩阵 M_NEIGH_{k*k} 是由上文数据依赖约束建模中通过 attachment3.csv 生成；对标注后的邻接矩阵 M_NEIGH_{k*k}，遍历邻接矩阵行，对于有出度且标红的行首元素，通过 DFS 深度优先搜索算法搜索其产生的每条执行流程，对于处在同一条执行流程且被标红的基本块统计其资源需求数 resource，然后求出每条执行流程上资源需求最大的资源数，如图 3.1 所示。其中标注通过设置一个与邻接矩阵同大小的标志矩阵 $M\_Adjacency_{k*k}$ 实现，初始化为 0。同一流程资源统计建模算法如下所示：



Resource=max(resource1,resource2,resource3)

图 3.1 深度优先搜索

同一流程资源统计算法 SameProcess

Input: 流水线的一级 PCB , 邻接矩阵 M_NEIGH_{k*k}, 标志矩阵 M_Adjacency$_{k*k}$
Output: 同一流程最大资源数 MaxResource
1 [r,c] = size(M_NEIGH_{k*k}), M_Adjacency$_{k*k}$ ← 0;
2 **for** i ← 1 **to** length(PCB) **do**
3 **for** j ← 1 to r **do**
4 **if** M_NEIGH_{k*k}(j,1) == PCB(i) **then** #矩阵第 1 列中存在 pcb 里的元素
5 M_Adjacency$_{k*k}$(j,1) ← 1; #标注
6 **for** k ← 2 **to** c **do** #遍历 j 行其余列进行标注
7 **if** M_NEIGH_{k*k}标注的第 j 行第 k 列在 PCB 中 **then**
8 M_Adjacency$_{k*k}$(j,k) ← 1; #标注
9 **for** i ← 1 to r **do**
10 **if** M_Adjacency$_{k*k}$(i,1) == 1 **then** #对矩阵中标注的起始节点进行深搜
11 resource ← DFS(M_Adjacency$_{k*k}$(i,1)) ;
12 **if** resource > MaxResource **then**
13 MaxResource = resource ;

3.2.2 针对约束 2、3 的 DFS 建模

对于问题二中的约束 2、3，都是使用同一流程资源统计建模中的模型算法，区别在于通过 DFS 深度优先搜索算法进行搜索时，对每个基本块子节点的资源判断不同，约束 2 需要统计 HASH 资源个数，约束 3 需要统计 ALU 资源个数，此处分别用 r2、r3 表示，具体 DFS 算法如下所示：

DFS 算法

Input：邻接矩阵的元素M_NEIGH_{k*k}(i,j)简记为节点 node，attachment1
Output：同一执行流程上的资源需求数 resource
1 if 节点都被搜完 then
2 return resource ;
3 for node 的每个子节点 nodeSon do
4 if nodeSon 还没被搜 then
5 标记 nodeSon 为 ture ; #表示递归搜索
6 resource ← 统计 nodeSon 在 attachment1 中使用的资源数累加
7 DFS(nodeSon);
8 标记 nodeSon 为 false ; #表示递归回溯
9 end
10 end

3.2.3 针对约束 1-4 的模型

因为约束 1 和约束 4 同问题一，所以结合问题一的资源约束算法中的约束算法 1 和约束算法 4，同时满足问题二中约束 2 要求的流水线每级中同一条执行流程上的基本块的 HASH 资源之和最大为 2 和约束 3 要求的流水线每级中同一条执行流程上的基本块的 ALU 资源之和最大为 56，而约束 2 直接通过 SameProcess 算法求解，可得到问题二约束 1-4 的算法，如下所示；输出为统计的不合格的流水线标志 flag 和资源 n 使用的超限量 violate_rn。

问题二中约束 1-4 资源统计算法 JudgeResource2
Input：PCB ,M_NEIGH_{k*k}, M_Adjacency$_{k*k}$,attachment1 Output：超出资源 n 的数量 violate_rn，不合格的流水线标志 flag 1 flag,violate_r1,violate_r2,violate_r3,violate_r4 ← 0; #同问题一资源约束中定义 2 **for** i ← 1 **to** length(PCB) **do** 3 r2 ← SameProcess(PCB ,M_NEIGH_{k*k},M_Adjacency$_{k*k}$); 4 r3 ← SameProcess(PCB ,M_NEIGH_{k*k},M_Adjacency$_{k*k}$); 5 **for** i ← 1 **to** length(PCB) **do** 6 r ← 在 attachment1 矩阵第一列中寻找 PCB(i)的索引; 7 r1, r4 ← 对每个 r 索引统计 r1 和 r4 资源使用; 8 violate_rn = rn - Reslimit; # n 取 1,2,3,4; Reslimit 取 1,2,56,64 9 **if** 四类资源中有一个资源使用数量>0 **then** 10 flag = 1;

为易于表述，本文摘要将问题二中的四种资源超限量也分别记为$VR_1^{'}$，$VR_2^{'}$，$VR_3^{'}$，$VR_4^{'}$。

3.2.4 针对约束 5 的模型

因为问题二约束 5 要求折叠的两级对于 TCAM 资源约束不变，对于 HASH 资源，每级分别计算同一条执行流程上的基本块占用的 HASH 资源，再将两级的计算结果相加结果不超过 3；则可以利用问题一的资源约束 5 算法来保证 TCAM 的资源约束，同时对本问的 HASH 资源约束进行建模；输入流水线的一级 PCB，邻接矩阵 M_NEIGH_{k*k}，标志矩阵 $M\_Adjacency_{k*k}$ 和 attachment1 通过前面约束 1-4 建模的 JudgeResource2 算法求出资源超限 violate_r2，最后得出违反约束 5 的流水线折叠级对数 violated_pc5，具体算法如下所示：

折叠级同一流程资源约束算法
Input：PCB ,M_NEIGH_{k*k}, M_Adjacency$_{k*k}$,attachment1 Output：违反约束 5 的流水线折叠级对数 violated_pc5 1 violated_pc5 ← 0; 2 **for** i ← 1 **to** length(PCB)/2 **do** 3 判断如果折叠级数存在，if pc<16&&(pc+16)<NPC； 4 violate_r2 ← JudgeResource2(PCB{i} ,M_NEIGH_{k*k},M_Adjacency$_{k*k}$,attachment1); 5 violate_r2 += JudgeResource2(PCB{i+16} ,M_NEIGH_{k*k},M_Adjacency$_{k*k}$,attachment1); 6 violate_r1 通过问题一的相应算法进行求解； 7 if violate_r2 > -1||violate_r1>0 then #判断条件参见注释 8 violated_pc5 += 1；

为方便表述，最后将通过算法得到的违反约束 5 的流水线等级对数 violated_pc5 记为 $VPC_5^{'}$。

注释：因为约束 2 已经限制了同一流程每级 HASH 资源为 2；若 violate_r2 初值为-2，则代表每级有两个 HASH 资源可使用，则折叠的两级有 4 个资源可使用即 violate_r2 值为-4 时，但约束 5 要求了同一流程折叠两级资源需求和最大为 3，此时 violate_r2 为-1，所以

violate_r2 > -1 时，即折叠两级资源需求大于了 3，则此对折叠级违反约束 5，进行统计并作为遗传算法的一个惩罚项进行优化。

因为问题二中的约束 6、7 和问题一相同，可以直接使用问题一中的约束 6、7：偶数级数量约束算法，这里不再赘述。

3.2.5 适应度评分函数构建

通过以上对资源约束的建模，本文分别得到了四种资源超限量分别记为$VR_1^{'}$、$VR_2^{'}$、$VR_3^{'}$、$VR_4^{'}$，违反约束 5 的流水线等级对数记为$VPC_5^{'}$以及偶数级资源约束的超限等级数$VPC_6^{'}$。因此成本损失函数定义如公式(3-1)所示：

$$F_{\cos t} = \mathrm{NPC} + \theta^{'} * \sum_{i=1}^{4} VR_i^{'} + \gamma^{'} * \sum_{j=5,6} VPC_j^{'} \tag{3-1}$$

其中 NPC 表示流水线等级数，$\theta^{'}$、$\gamma^{'}$表示每项的惩罚系数，且只有当各项优化至 0 时，才是优化流水线等级数的正式开始。

适应度值评分函数就是对成本损失函数取倒数，即

$$F_{fit} = \frac{1}{F_{\cos t}} \tag{3-2}$$

3.3 问题二的结果及其分析

通过 matlab 对遗传算法编程求解即可得到流水线最优等级；

遗传算法的迭代曲线如图 3.2 所示：

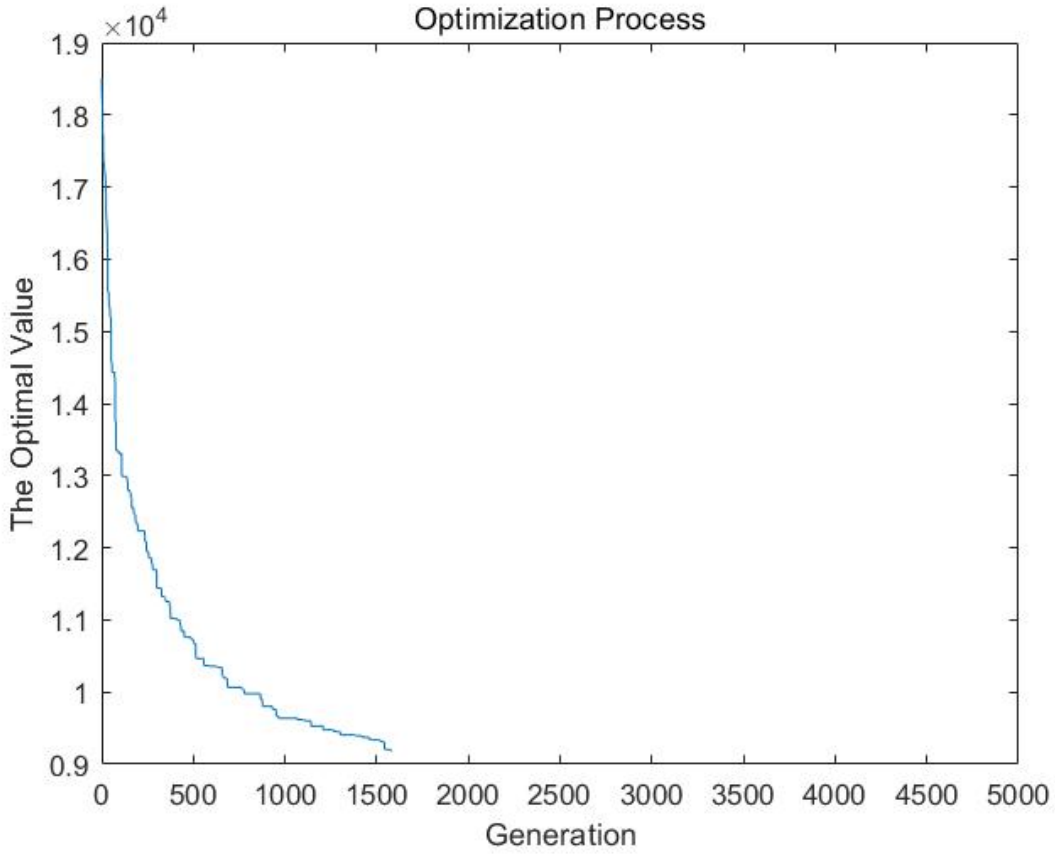


图 3.2 遗传算法迭代曲线（问题二）

从上图可以看出，随着迭代次数的增加，最优值也逐渐减少，即成本损失函数中的各类约束正在不断优化。但是由于时间原因，程序并未达到设置的迭代次数，此处显示的是迭代次数 1500 的结果。从结果可知，它相对于问题一收敛速度更快，在迭代次数 1500 左右时，最优值已经降到了 0.9×10^4。问题二最终迭代曲线以及最终的基本块排布方案见附件 result2.csv。

四、模型评价

4.1 模型优点与创新点

4.1.1 模型优点

1、遗传算法不易陷入局部最小的陷阱，可以对全局进行优化；
2、迭代时从种群出发，而不是单个个体，具有潜在的并行性；
3、使用概率方式来进行迭代，过程相较具有随机性；
4、模型的扩展性强，易与其他算法进行结合；

4.1.2 创新点

1、在初始种群的设置上，本文将基本块放置在染色体上，并在其后加入 N-1 个标志值(即大于最大序号 606 的值)，进行随机分布。这样即可得到含有分级信息的初始种群，再通过遗传算法对排列进行迭代优化；

2、利用了遗传算法的扩展性，通过对各类约束条件进行分析建模，使其不断迭代，自适应搜索求得最优解。

4.2 模型缺点及改进

1、遗传算法的收敛速度与初始种群的质量有很大关系。一个性能更加优异的初始种群，能够让遗传算法更迅速地迭代到优化结果附近。而本文中初始种群的设置仅是将序号进行了随机重组，因时间限制未能深入考虑做更多优化，所以算法收敛速度较慢；

2、程序中有些矩阵的大小存在冗余，后续可对矩阵大小进行优化，以加快算法的收敛速度；

参考文献

[1]Bosshart P, Gibb G, Kim H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 99-110.

[2]Bosshart P, Daly D, Gibb G, et al. P4: Programming protocol-independent packet processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.

[3]司守奎，孙兆亮. 数学建模算法与应用[M]. 北京：国防工业出版社，329-333，2011。

[4]Mantawy, A. H , Abdel-Magid, et al. Integrating genetic algorithms, tabu search, and simulated annealing for the unit commitment problem[J]. Power Systems IEEE Transactions on, 1999.

附　录

遗传算法主函数 main.m

```
%%用来调用所有函数
clear, clc;
close all;

%% input data
attachment1_date = csvread('attachment1.csv',1,0);
attachment2_date = readmatrix('attachment2.csv','OutputType','string');
attachment3_date = xlsread('attachment3.csv');
W = xlsread('W.xlsx');
R = xlsread('R.xlsx');
M_acess = matrix_change(attachment3_date);
M_aug_conflict = cM_aug_conflict(W,R);
M_constrain_shuju = compute_constrain_shuju(M_acess,M_aug_conflict);


% tw1 means the earliest start time, tw2 means the latest start time
block_number = size(attachment1_date,1);
process_class_number = 100; % number of process_class
% depot_time_window1 = attachment1_date(1,5);
% depot_time_window2 = attachment1_date(1,6);
% time_window1 = attachment1_date(2:end, 5); % tw, start time
% time_window2 = attachment1_date(2:end, 6); % tw, end time
% service_time = attachment1_date(2:end, 7); % service time

%% GA parameters
population = 100; %种群大小，越大越难以收敛
generation = 1;
maximum_generation = 5000; %迭代次数
probability_crossover = 0.9; %交叉概率
probability_mutation = 0.05; %变异概率
rate_gap = 0.9; %代沟
length_chrom = block_number + process_class_number - 1; %染色体长度=基本块数
+流水线级数-1

%% initialize population
%随机生成法
chroms = InitPop(population,length_chrom);
% init_chrom = createInitChrom(block_number, time_window1);
% chroms = createInitPopulation(population, length_chrom, block_number,
init_chrom);
```

```
% display some values of the initial random population
[PCB, NPC, violate_num_pc,
violate_r11,violate_r22,violate_r33,violate_r44, ...
    sviolate_blocks_shuju,sviolate_blocks_kongzhi] =
decode(chroms(1,:),block_number,attachment1_date,attachment3_date,M_const
rain_shuju);
disp('The initial population:');
disp(['Number of process_class: ', num2str(NPC), ', Number of violated
process_class: ',...
    num2str(violate_num_pc), ', Number of Violated blocks_shuju: ',
num2str(sviolate_blocks_shuju),...
    ', Number of Violated blocks_kongzhi: ',
num2str(sviolate_blocks_kongzhi)]);
fprintf('\n');
% calsulate the value of objective function
ObjV =
calObj(chroms,block_number,attachment1_date,attachment3_date,M_constrain_
shuju);
% the minimal value
pre_objective_value=min(ObjV);

% Display the change of the objective function value
% figure;
% hold on;
% box on;
% xlim([0, maximum_generation]);
% title('Optimization Process');
% xlabel('Generation');
% ylabel('The Optimal Value');


iter_time = [];


%% the loop
while generation <= maximum_generation

    tic;

    % calculate the fitness value
    ObjV =
calObj(chroms,block_number,attachment1_date,attachment3_date,M_constrain_
shuju);
```

```
    % draw the line
    %line([generation - 1, generation], [pre_objective_value, min(ObjV)]);
pause(0.0001);
    pre_objective_value = min(ObjV);
    FitnV = Fitness(ObjV);
    % select
    SelCh = Select(chroms,FitnV,rate_gap);
    % crossover
    SelCh = Recombin(SelCh,probability_crossover);
    % mutation
    SelCh = Mutate(SelCh,probability_mutation);



    % local search
    % SelCh = neighborhoodSearch(SelCh, block_number, time_window1,
time_window2, depot_time_window2, service_time, distance_matrix);
    %
    chroms = Reins(chroms,SelCh,ObjV);
    % Delete duplicate chromsomes
    chroms = DealRepeat(chroms);
    ObjV =
calObj(chroms,block_number,attachment1_date,attachment3_date,M_constrain_
shuju);
    [minObjV,minInd]=min(ObjV);
    disp(['Generation:' num2str(generation)]);

[bestPCB,bestNPC,bestviolate_num_pc,bestviolate_r11,bestviolate_r22,bestv
iolate_r33,bestviolate_r44, ...

bestsviolate_blocks_shuju,bestsviolate_blocks_kongzhi]=decode(chroms(minI
nd(1),:),block_number,attachment1_date,attachment3_date,M_constrain_shuju
);
    disp(['Number of process_class: ', num2str(bestNPC), ', Number of violated
process_class: ',...
    num2str(bestviolate_num_pc), ', Number of Violated blocks_shuju: ',
num2str(bestsviolate_blocks_shuju),...
    ', Number of Violated blocks_kongzhi: ',
num2str(bestsviolate_blocks_kongzhi)]);
    fprintf('\n');
    generation = generation + 1;

    iter_time(generation) = toc;
end
ObjV =
```

```
calObj(chroms,block_number,attachment1_date,attachment3_date,M_constrain_
shuju);
[minObjV,minInd]=min(ObjV);

disp('Optimal Result:')
bestChrom=chroms(minInd(1),:);
[bestPCB,bestNPC,bestviolate_r11,bestviolate_r22,bestviolate_r33,bestviol
ate_r44, ...

bestsviolate_blocks_shuju,bestsviolate_blocks_kongzhi]=decode(bestChrom,
block_number,attachment1_date,attachment3_date,M_constrain_shuju);
disp(['Number of process_class: ', num2str(bestNPC), ', Number of violated
process_class: ',...
    num2str(bestviolate_num_pc), ', Number of Violated blocks_shuju: ',
num2str(bestsviolate_blocks_shuju),...
    ', Number of Violated blocks_kongzhi: ',
num2str(bestsviolate_blocks_kongzhi)]);

% check the missing element in the final result
missing_element = judgeFullElement(bestPCB, block_number);

%drawMap(bestVC, vertexs);



% close all
```

解码函数 decode.m

```
%%获取各个约束函数

% input:
% chrom ---------- one single chroms
% block_number --------- number of blocks
% a, b ----------- time window
% L -------------- latest arrival time to depot

% output:
% PCB ------------- bolcks of every process_class
% NPC ------------- number of process_class

% seperate a chromosome to several routes

function [PCB, NPC, violate_num_pc,
violate_r11,violate_r22,violate_r33,violate_r44, ...
    sviolate_blocks_shuju,sviolate_blocks_kongzhi] = decode(chrom,
```

```
block_number,attachment1_date,attachment3_date,M_constrain_shuju)
  violate_num_pc = 0;
  violate_r11 = 0;
  violate_r22 = 0;
  violate_r33 = 0;
  violate_r44 = 0;
  sviolate_blocks_kongzhi = 0;
  sviolate_blocks_shuju = 0;
  PCB = cell (block_number, 1);
  count = 1;
  % 选取最后的几个点，作为分割位置
  location0 = find(chrom > block_number); % 找到这几个切割点

  % 将基本块分配到流水线级数
  for i = 1:length(location0)
    if i == 1
      % create a new route, include the number of depot
      route = chrom(1:location0(i));
      % delete the number of depot at the end
      route(route == chrom(location0(i))) = [];
    else
      route = chrom(location0(i - 1):location0(i));
      route(route == chrom(location0(i - 1))) = [];
      route(route == chrom(location0(i))) = [];
    end
    PCB{count} = route; % add the route to set
    count = count + 1;
  end

  % for the last route
  if isempty(location0)
      route = chrom;
  else
      route = chrom(location0(end) : end);
      route(route == chrom(location0(end))) = [];
  end
  PCB{count} = route;
  % delete empty routes, and get the number of process_class
  [PCB,NPC]=deleteEmptyRoutes(PCB);

  % 计算违反数据依赖和控制依赖的基本块个数，以及超过最大级数的个数
  for j=1:NPC
    route = cell(1,1);
    route{1} = PCB{j}; % pick up one route
```

```
    % check the violated resource
    route = route{1};
    [flag, violate_r1, violate_r2, violate_r3, violate_r4] =
Judge_resource(route,attachment1_date);
    [violate_blocks_shuju23] = Judge_shuju23(route,M_constrain_shuju);
    if flag == 1
        violate_num_pc = violate_num_pc+1;
        violate_r11 = violate_r11+violate_r1;
        violate_r22 = violate_r22+violate_r2;
        violate_r33 = violate_r33+violate_r3;
        violate_r44 = violate_r44+violate_r4;
    end
  end
  % check the violated kongzhi & shuju constrain
  violate_blocks_kongzhi = Judge_kongzhi(chrom,attachment3_date);
  violate_blocks_shuju2 = Judge_shuju(chrom,M_constrain_shuju,2);
  violate_blocks_shuju3 = Judge_shuju(chrom,M_constrain_shuju,3);
  violate_blocks_shuju4 = Judge_shuju(chrom,M_constrain_shuju,4);
  violated_pc5 = Judgecontrain5(PCB,attachment1_date,NPC);
  violated_pc6 = Judgecontrain6(PCB,attachment1_date,NPC);
  violate_num_pc = violated_pc5 + violated_pc6;
  sviolate_blocks_shuju =
sviolate_blocks_shuju+violate_blocks_shuju4+violate_blocks_shuju23+violat
e_blocks_shuju2+violate_blocks_shuju3;
  sviolate_blocks_kongzhi = sviolate_blocks_kongzhi+violate_blocks_kongzhi;

end
```

数据依赖约束算法 Judge_shuju.m

```
% check the shuju constrains
% Input:
% chrom:整条染色体

% Output:
% violate_blocks_shuju2:不满足写后读数据依赖的对数
% violate_blocks_shuju3:不满足写后写数据依赖的对数
% violate_blocks_shuju4:不满足读后写数据依赖的对数

function [violate_blocks_shuju4] =
Judge_shuju(chrom,M_constrain_shuju,type)
    violate_blocks_shuju4 = 0;
    [Mr_constrain,Mc_constrain] = find(M_constrain_shuju==type);
    for i = 1:length(Mr_constrain)
        index1 = find(chrom == (Mr_constrain(i)));
        index2 = find(chrom == (Mc_constrain(i)));
```

```
        if index1 > index2
            violate_blocks_shuju4 = violate_blocks_shuju4 + 1;
        end
    end
end
```