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	河南师范大学
参赛队号	22104760020
队员姓名	1.桑梓航
	2.秦梦真
	3.包伊莎

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　　PISA 架构芯片资源排布问题

摘　　要：

在当前日益复杂的国际形势下，科技是衡量一个国家综合实力的重要标准，芯片技术在科技领域中最为重要，芯片对于任何一个国家来说都是刚需。芯片对于科技创新的意义重大，它时时刻刻都在改变人们的生活，不管什么设备都离不开芯片，芯片技术的提高已成为一个国家的重要战略目标。可编程的交换芯片的诞生是芯片技术发展的一个重要里程碑。作为当前主流的可编程交换芯片之一的架构，PISA（Protocol Independent Switch Architecture）既有和固定功能交换芯片相当的处理速率，又具有可编程性，在 5G 网络中具有广阔的应用场景。在 PISA 架构编程模型中，通过编译 P4 程序可生成芯片上可执行的机器码。编译器在编译 P4 程序时，会将 P4 程序划分为各个基本块，再将各基本块排布到流水线各级中。由于芯片各类资源有限，高资源利用率意味着芯片能力的提高，需要进行资源的排布，使得占用的流水线级数尽量短。

针对问题一：首先，对附件 3 的数据，利用基本块间的邻接关系，得到基本块之间的有向图，将图可视化，结果显示 **365 号**基本块为资源排布的起始块。本文通过 BFS 算法得出全部基本块的遍历序列。之后，结合附件 1 和附件 2 的数据分别总结出各基本块所需资源个数，以及变量被基本块的读写情况。然后，分析题目中的约束条件，**建立拉格朗日乘子法的模型**，结合附件分析的结果，并且以 BFS 算法分析出的序列，根据贪心策略设计**流水线资源排布算法，**算法初步得到流水线排布的总级数为 **51 级**。最后，分别采用**遗传算法**和 **SVM** 两种方法对流水线排布算法进行优化，并与未优化算法进行对比，得到 SVM 优化得到最短级数为 **38 级**，遗传算法优化得到最短流水线级数为 **36 级**。

针对问题二：进一步提出了执行流程对资源排布的影响，不在一条执行流程上的基本块，可以共享 HASH 资源和 ALU 资源，在同一流水线级的同一执行流程的基本块的 HASH 资源与 ALU 资源不超过每级资源限制，该基本块即可排布到当级流水线。问题二修改了问题一中约束条件（2）、（3）、（5）的约束。本文首先设计**执行流程查找算法**，找到从基本块 365 开始的所有执行流程，并进一步修改问题一的流水线资源排布算法，修改问题二提出约束的判断，将问题一中的约束条件（2）、（3）、（5）在同一流水线级上资源的限制，更改为每级中同一条执行流程上资源的限制。最后通过**遗传算法**对新的流水线排布算法进行优化，得到最短流水线级数为 **32 级**。

关键词：PISA；P4 语言；资源排布算法；SVM；遗传算法

目录

第一章 问题背景与问题重述

1.1 问题背景

当今世界，芯片行业是信息技术产业的基础，是支撑国民经济发展与保障国家安全战略性、基础性以及先导性的产业，其行业发展程度是衡量一个国家科技发展水平的核心指标之一。在复杂紧张的国际形势下，芯片已成为各个大国的“兵家必经之地”。近年来，在美国垄断芯片的紧张环境下，中国的芯片发展突飞猛进，自给率逐年提高。国产芯片大多应用在消费类领域。其中，在通信、工业、医疗以及军事的大批量应用中，对芯片的稳定性、可靠性以及资源利用率要更高。为了提高芯片的研发效率，诞生了可编程的交换芯片。PISA 是当前主流的可编程交换芯片架构之一，由通用的逻辑单元和流水线组成。

2014 年，随着 Nick McKeown 教授等人发表论文《P4: Programming Protocol-Independent Packet Processors》[1]，P4 语言正式走入了大众视野，并引起 SDN 界极大反响和广泛关注。P4 是一种以协议无关性、目标无关性以及现场可重配置能力为目标的专用编程语言，可以解决 OpenFlow 存在的编程能力不足和可拓展性差等问题[3]-[4]。当 P4 高级编程语言运行在新一代高性能可编程芯片上时，数据包的完整处理流程都可以实现自定义，从而赋予了网络所有者进行快速迭代开发的能力。其中 P4 语言的运行需要以 PISA，即协议无关可编程架构为依托，PISA 架构的运行原理如图 1-1 所示。通过 P4 和 PISA 架构实现的可编程数据平面具有软件重用、数据隐藏、库创建、硬件和软件组件分离、软件便捷升级和快速调试等优点[5]。可以说，P4 的出现改变了网络芯片的设计方式，并且开辟了一个设计完全可编程芯片的样本模型和思路[2]。

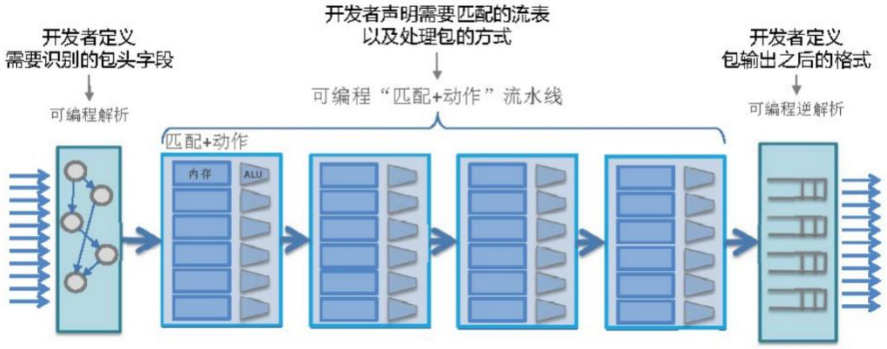

图 1-1 PISA 架构的运行原理图

上图的 PISA 架构主要由可编程的报文解析器 (PARSER)、可编程的匹配动作单元模块（MATCH-ACTION UNIT，简称 MAU）和报文组装模块（DEPARSER）构成[6]。在流水线入口，可编程的报文解析器负责对数据包进行预处理和解析。可编程的匹配动作单元模块主要用于各种查找表操作，芯片内部包含多级匹配动作单元并以流水线的方式组合而成[7]。每级匹配动作单元内都包含一定数量的哈希资源、SRAM 资源、TCAM 资源和 ALU 资源等。哈希资源可以用于实现各自查找表算法，SRAM 资源可以用来实现精确匹配表的存储与查找，TCAM 资源则可以用来实现模糊匹配表的存储与查找，ALU 资源可以实现内部的相关控制，具备逻辑与或的运算能力和加减的运算能力，从而实计数 (Counter)、流量计(Meter) 以及寄存器 (Register) 的功能[10]。报文组装模块则将查表结果作用于报文之上，增加一些新的报文头部，或删除一些报文头部，或修改已有报文头部中

的一些字段，并将处理完成的报文提交给缓存管理模块入队，调度模块则根据选择的调度算法将报文调度出队，经过下行流水线处理最终由出口端口转发给下一跳设备。

PISA 架构最大的优点是可以根据用户需求进行开发与定值，并且各条流水线能够协同工作，在不损失整体性能的大前提下，可以让系统整体的效率大幅提升。因此，PISA 架构芯片在未来有很广阔的应用场景。在 PISA 架构中，基本块是源程序的一个程序片段，对源程序进行基本块划分的操作，可以将源程序划分为若干基本块。基本块是一个最大化的指令序列，程序执行只能从这个序列的第一条指令进入，从这个序列的最后一条指令退出。

PISA 架构的编程模型流程图如图 1-2 所示：

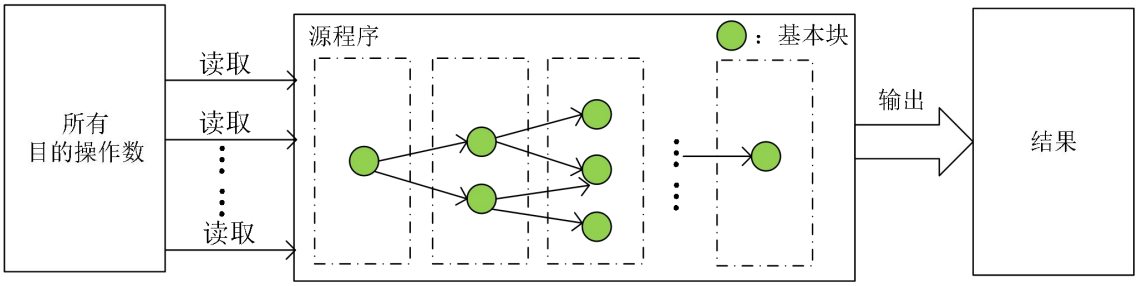

图 1-2 PISA 架构的编程模型流程图

在 PISA 架构编程模型中，用户使用 P4 语言描述报文处理行为得到 P4 程序，再由编译器编译 P4 程序，进而生成芯片上可以执行的机器码[7]-[8]。当编译器在编译 P4 程序时，会首先将 P4 程序划分为一系列的基本块，再将各基本块排布到流水线各级当中，即需要确定每个基本块排布到流水线哪一级。因此，将基本块的排布问题称作 PISA 架构芯片资源排布问题。

1.2 问题重述

基于上述问题背景与题目提供的数据，本文需要研究解决以下问题：

问题一：根据给定的资源约束条件以及依赖关系进行资源排布，以占用流水线级数尽量最短为优化目标，编写资源排布算法，输出基本块的排布结果。

PISA 架构资源排布即将 P4 程序[9]流程图中的各节点（即各基本块）在满足约束条件下排布到流水线各级当中。其约束条件主要包括两个方面：一方面是来自于 P4 程序本身，P4 程序中变量的读写顺序使得基本块之间存在数据依赖，并且当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存在的控制依赖。另一方面的约束条件来自于芯片的资源约束。在本题中，芯片中的资源包括 TCAM、HASH、ALU、QUALIFY 四类。在各级流水线中，不可以违背芯片的资源限制。

在问题一中，根据附件 “attachment1.csv”、“attachment2.csv”和“attachment3.csv”所给的数据，结合本问题中的数据依赖和控制依赖条件，为了提高芯片资源利用率，编写合理的资源排布算法，使占用的流水线技术尽量最短，最后输出基本块的排布结果。其中问题 1 中的约束条件为：流水线中每级的四类芯片资源最大分别为：1、2、56、64；折叠两级的 TCAM 资源、HASH 资源加起来的最大值为 1 和 3；有 TCAM 资源的偶数级数量不能超过 5；每个基本块只能排布到一级。

问题二：根据更改的资源约束条件和附录 A 中的依赖关系，编写算法，优化流水线级数，使之尽量最短，最终得到基本块的排布结果。

在本问中，考虑到不在一条执行流程上的基本块，可以共享 HASH 资源和 ALU 资源。

若该类基本块中任意一个的 HASH 资源与 ALU 资源均不超过每级资源限制，则可将其排布到同一级上。

在问题二中，根据本题所提供的 3 个数据表格，在问题一的基础上更改约束关系，控制资源分配，通过编写合适的算法，最终获得所需的基本块排布结果。其中与问题一中的资源约束条件不同的是：流水线每级中同一条执行流程上的基本块的 HASH 资源以及 ALU 资源之和最大分别为 2、56；对于折叠两级的 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。

第二章 模型假设与符号说明

2.1 模型假设

考虑到现实问题，本文做出以下假设：
（1）假设除了题目中的变量，没有其他变量会对资源排布产生影响；
（2）假设题目提供的数据是真实可靠的；
（3）假设如果需要的流水级数超过 32 级，则从第 32 开始的级数不考虑折叠资源限制；
（4）假设在优化过程中，不改变现有资源约束条件。

2.2 符号说明

本文中用到的变量符号及其说明如下表 2-1 所示：

表 2-1 变量符号及其说明

序号	符号	符号说明
1	λ	不定乘子
2	x	基本块的块号
3	n	流水级的级数
5	$g_T(n)$	TCAM 资源个数限制函数
6	$g_H(n)$	HASH 资源个数限制函数
7	$g_A(n)$	ALU 资源个数限制函数
8	$g_Q(n)$	QUALIFY 资源个数限制函数
9	$Count$	计算偶数级个数的函数
10	$W(x)$	更新权重的函数
11	k	边被迭代过的次数

第三章 问题一模型的建立与求解

3.1 问题分析

问题一要求我们针对给定的 7 个资源约束条件和附录中的依赖关系进行资源排布，以占用流水线级数尽量短为优化目标，给出资源排布结果。本文对附件中所给数据的预处理过程如下：首先，对“attachment1.csv”文件进行处理，得到使用 TCAM、HASH、ALU、QUALIFY 四类资源所对应的基本块，这有利于对资源进行统一分配。其次，对“attachment2.csv”文件的处理主要解决基本块的数据依赖问题，整理了每一个变量被基本块读写的操作情况，有利于更加直观的看出基本块之间的数据依赖关系，是否存在读后写、写后读、写后写等操作。再次，对“attachment3.csv”进行处理，该操作主要解决基本块间的控制依赖关系。本文将该文件中的邻接表转换为邻接矩阵，再将邻接矩阵转为图，并得到图的可视化结果。之后，本节先使用**贪心策略**设计 PISA 资源排布算法。在上述结果中，本文寻找图的起始点（即入度为 0 的点），以该结点作为广度优先搜索（BFS）的起始点，接着对图进行广度优先遍历。最后，将遍历的结果与问题 1 中约束条件建立数学模型，设计流水线资源排序算法并结合进行优化算法，进行各个基本块的流水级排序，得到本问的结果。

为了提高资源的利用率，本节使用两个优化模型来对比贪心算法的结果，以获得更短的流水线级数。

3.2 数据预处理

为了解决问题，本小节对三个文件进行了数据预处理，以便作为资源优化排布的输入。具体的处理过程如下：

3.2.1 对基本块资源信息的分析

附件“attachment1.csv” 的内容是各基本块使用的资源信息，文件 1 内容如表 3-1 所示：

表 3-1 各基本块使用的资源信息

BLOCK	TCAM	HASH	ALU	QUALIFY
0	0	0	2	0
1	0	0	0	0
2	0	0	0	0
…	…	…	…	…
604	0	0	7	0
605	0	0	0	0
606	0	0	25	1

从表 3-1 可观察出，每个基本块对四类资源的占用情况，如 BLOCK_0 需要占用 ALU 资源 2 个，BLOCK_1 需要占用 TCAM 资源 1 个等，因此对资源信息的归纳尤为重要。本文罗列出了 TCAM、HASH、ALU、QUALIFY 四类资源被各个基本块使用的情况。

对附件“attachment1.csv”的预处理结果如下：

(1) 使用 TCAM 资源的基本块情况如图 3-1 所示：

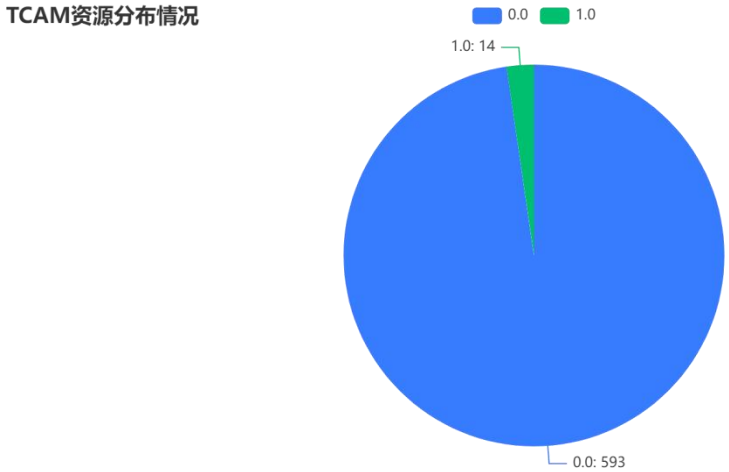

图 3-1 使用 TCAM 资源的基本块情况

如图 3-1 所示，第一列为基本块号，第二列是各个基本块需要 TCAM 资源的个数，共有 14 个基本块使用 TCAM 资源，每个基本块占用 TCAM 资源的个数均为 1。

（2）使用 HASH 资源的基本块情况如图 3-2 所示：

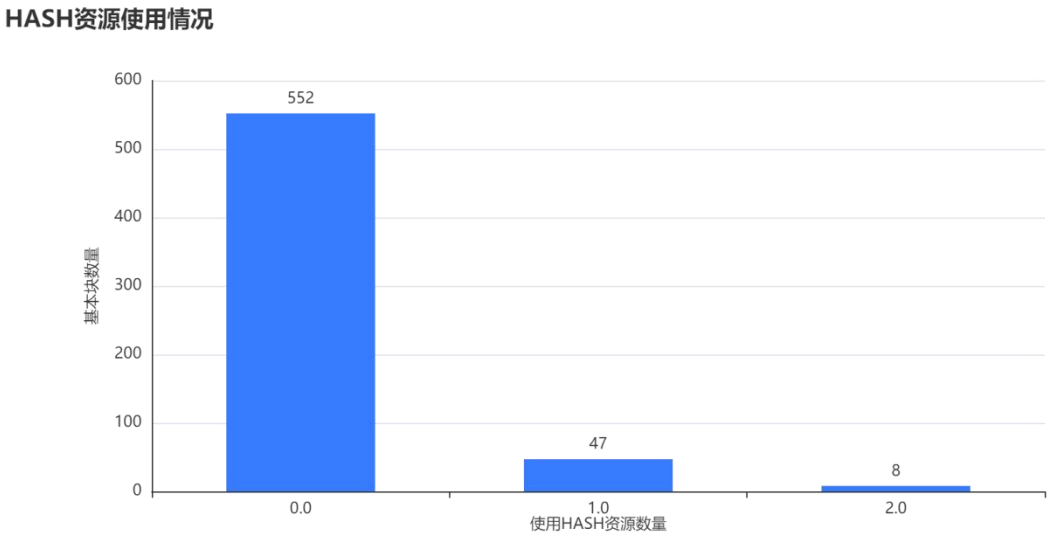

图 3-2 使用 HASH 资源的基本块情况

观察图 3-2 可知，共有 55 个基本块占用 HASH 资源，其中 BLOCK_23、BLOCK_54、BLOCK_56、BLOCK_66、BLOCK_162、BLOCK_164、BLOCK_581、BLOCK_583 占用 HASH 资源的个数均为 2，其余基本块使用 HASH 资源的个数均为 1。

（3）由于使用 ALU 资源的基本块情况统计结果较多，本文将完整的结果放置在附录中，部分结果如表 3-2 所示：

表 3-2 使用 ALU 资源的基本块情况的部分结果

BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数
0	2	25	6	56	4	588	4
4	10	29	11	57	9	591	1
5	6	31	2	58	3	593	1
7	3	33	9	61	2	594	1
8	4	35	2	64	1	595	9
10	7	37	6	66	2	596	1
12	5	39	6	…	…	597	2
14	2	41	5	…	…	599	1
15	3	43	6	…	…	600	6
16	2	45	6	578	2	601	1
19	16	47	1	579	3	602	1
21	2	49	8	581	11	603	18
22	1	51	2	583	11	604	7
23	3	54	5	586	4	606	25

结合表 3-2，根据使用 ALU 资源的基本块情况的完整结果（见附录）可知，共有 360 个基本块占用 ALU 资源，各个基本块占用 ALU 资源个数的范围为[1，11]，[13，22]，[25]，其中 BLOCK_417 和 BLOCK_606 占用的资源个数最多，均为 25。

（4）使用 QUALIFY 资源的基本块情况部分结果如表 3-3 所示：

表 3-3 使用 QUALIFY 资源的基本块情况部分结果

BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数
4	3	62	1	88	2	574	3
7	4	64	1	89	5	578	1
8	4	66	1	90	6	579	1
10	3	67	1	97	2	585	6
12	1	74	1	98	5	588	7
14	2	75	1	99	1	589	1
19	4	76	8	…	…	591	1
21	3	77	1	…	…	593	1
28	1	78	1	…	…	594	1
54	2	80	1	559	5	599	1
56	2	83	1	562	5	601	1
57	3	84	1	566	1	602	1
59	2	85	1	570	2	603	8
61	5	86	2	573	1	606	1

通过观察表 3-3 和占用 QUALIFY 资源的基本块情况的完整结果（见附录）可知，共有 192 个基本块占用 QUALIFY 资源，占用资源个数的范围为[1，9]。其中，使用资源数最少的基本块有 86 个，占用资源的个数为 1；占用资源数最多的基本块有 2 个，分别为 BLOCK_227 和 BLOCK_236，使用资源的个数为 9。

3.2.2 对基本块读写信息的分析

附件“attachment2.csv” 的内容是各个基本块读写的变量信息。表中第一列表示基本

块编号，第二列中的“W”表示写变量，“R”表示读变量，后续的列表示这个基本块写或者读的变量名称，若某一行从第 3 列及后续列没有任何元素时，就说明此编号的基本块没有读或写任何一个变量。

本节对附件“attachment2.csv”进行了处理，整理了每一个变量对基本块的操作情况，有利于更加直观的表示出每一个变量对基本块的访问情况，便于在后续代码中判别同一流水线级数内访问的基本块之间是否存在读后写，写后读，写后写这三种数据依赖关系。运用 MATLAB 软件编写代码，最后将本题的结果保存在稀疏矩阵内，由于数据内容较多，仅展示部分结果，全部结果保存在附件的**“q2.mat”**文件中，在文件**“q2.mat”**中，W_X 表示写操作，R_X 表示读操作。处理的部分结果如表 3-6、表 3-7 所示：

表 3-4 与变量进行写操作对应基本块的统计情况

变量	基本块号	变量	基本块号
X0	0，25，90，136,388,588	…	…
X1	0,57,58,88,90,209,335,370,377,391	X820	599
X2	78,101,128,139,…,535,549,550,567,603	X821	-
X3	78,101,128,139,…,535,549,550,567,603	X822	-
…	…	X823	604

在表 3-4 中，**符号“-”表示无基本块对相应的变量进行写操作**。由表 3-6 可知每个变量被基本块进行写操作的情况。

表 3-5 与变量进行读操作对应基本块的统计情况

变量	基本块号	变量	基本块号
X0	-	…	…
X1	358，360，365	X820	603
X2	-	X821	603
X3	-	X822	603
…	…	X823	-

和表 3-4 一样，表 3-5 中的**符号“-”表示没有基本块对相应的变量进行读操作**。由表 3-5 可知每个变量被基本块进行读操作的情况。

3.2.3 对基本块邻接信息的分析

文件“attachment3.csv”的内容表示各个基本块在流程图中的邻接基本块信息。本文将表格分析为不含边的邻接表，由于表格数据太多，仅展示前 5 行的数据，如表 3-6 所示。题目所给数据的前 5 行数据的邻接表如图 3-3 所示：

表 3-6 “attachment3.csv”文件的前 5 行数据

基本块编号	邻接基本块信息	
0	1	2
1	2	-
2		-
3	31	-
4	0	586

注：表 3-6 中，符号“-”表示无邻接基本块。

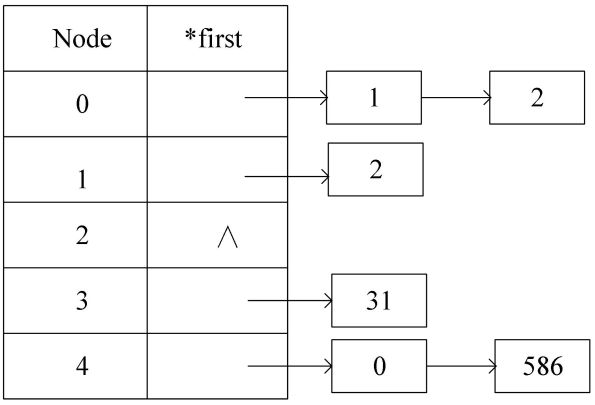

图 3-3 前 5 行数据的邻接表展示图

由题意分析得，把每个基本块看作一个结点，每一行的第一列为一个结点，其后的结点为此结点的后继结点。即：0 号基本块到 1 号基本块和 2 号基本块之间存在有向边连接（即 0 号基本块执行结束后可以跳转到 1 号基本块或 2 号基本块执行）。前 5 行数据的有向图如下图 3-4 所示：

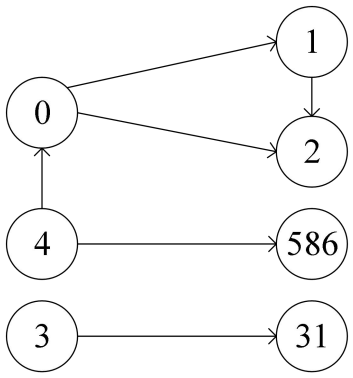

图 3-4 前 5 行数据的有向图

将邻接表转换为邻接矩阵，把邻接矩阵转换成图，对所有基本块之间的连接信息展示出来，找到树的顶端（即只有出度无入度的顶点）。以这个顶点作为广度优先遍历的起始端，得到广度优先遍历的结果。流水级的排列顺序可以先按照广度优先遍历的结果进行排列，然后，在此基础上再进行资源分配及条件约束。**广度优先遍历(Breath First Search)的结果包含了基本块之间的先后顺序，有利于解决控制依赖的问题。**

3.3 模型建立

为了更好的解决问题，本文对约束条件的建模如下：

3.3.1 拉格朗日乘子法

拉格朗日乘子法[11]（Lagrange multipliers）是一种寻找多元函数在一组约束下的极值的方法。通过引入拉格朗日乘子，可将有 d 个变量与 k 个约束条件的最优化问题转化为具有 $d+k$ 个变量的无约束优化问题求解。

假设：

（1）求极值的目标函数 (objective function)为 $f(x,y)$ 。

（2）约束条件为$\phi(x,y)=M$。

设$g(x,y)=M-\phi(x,y)$，定义一个新函数$F(x,y,\lambda)=f(x,y)+\lambda g(x,y)$，则用偏导数方法列出下列方程：

$$\begin{cases}\dfrac{\partial F}{\partial x}=0\\ \dfrac{\partial F}{\partial y}=0\\ \dfrac{\partial F}{\partial \lambda}=0\end{cases} \tag{3-1}$$

求出x，y，λ的值，代入即可得到目标函数的极值。扩展为多个变量的式子为：

$$F(x_1,x_2,\cdots\lambda)=f(x_1,x_2,\cdots)+\lambda g(x_1,x_2,\cdots) \tag{3-2}$$

则求极值点的方程为：

$$\begin{cases}\dfrac{\partial F}{\partial x_i}=0 \quad , \quad i=1,2,\cdots\\ \dfrac{\partial F}{\partial \lambda}=g\left(x_1,x_2,\cdots\right)=0\end{cases} \tag{3-3}$$

另外，可以将这种把约束条件乘以λ（即不定乘子）后加到待求函数上的求极值方法推广到变分极值问题及其它极值问题当中。可选x值的集合被一个或多个约束所限制，这就对解有重要影响，此问题为有约束的。我们可以应用拉格朗日乘子法来研究。

由问题分析可知，本问是一类条件约束的优化问题。在求解优化类问题时，对条件约束建模是问题求解的必要条件。常用的方法是拉格朗日乘子法与卡罗需-库恩-塔克（KKT）条件法，当约束条件含有等式约束，一般采用拉格朗日乘子法，而含有不等式约束条件时使用 KKT 条件。本问中，我们选用 KKT 法，即在满足现有规则的条件下，一个非线性规划问题能有最优解的一个必要和充分条件，属于广义化拉格朗日乘数的结果。本问的求解目标函数如下：

$$n=\min_{x} f(x) \tag{3-4}$$

其中，x表示基本块的块号，n表示流水级的级数(n从 0 开始)。对目标函数的不等式约束如下：

$$s.t.\ \begin{cases}g_T(n)\le 1\\ g_H(n)\le 2\\ g_A(n)\le 56\\ g_Q(n)\le 64\end{cases} \quad , n=0,1,2\ldots\ldots \tag{3-5}$$

其中，$s.t.$表示受限于，$g_T(n)$，$g_H(n)$，$g_A(n)$，$g_Q(n)$，分别表示 TCAM，HASH，ALU，QUALIFY 四类资源的个数限制函数。将题目中的约束条件（5）转换为公式，如下所示：

$$s.t.\ g_f(n)=\begin{cases} g_T(n)+g_T(n+16)\le 1 \\ g_H(n)+g_H(n+16)\le 5 \\ n\le 16 \end{cases} \tag{3-6}$$

题目约束条件（6）中需要对 TCAM 资源的偶数级数量不超过 5，本文对其建立的模型公式如下：

$$s.t.\ g_e(n)=Count(\sum_{i=1}^{2t-1} n)\le 5 \tag{3-7}$$

在公式（3-7）中，*Count* 函数为计算偶数集个数的函数。

则关于不等式约束的拉格朗日函数 L 的表达式如下：

$$L(x,n)=f(x)+\sum_{k=1}^{q} g_k(n) \tag{3-8}$$

其中，k 表示第 k 个不等式约束方程。

3.4 问题求解

3.4.1 广度优先遍历

为了避免数据依赖问题，本文采用广度优先遍历算法。广度优先遍历算法（Breath First Search，简称 BFS）是典型的遍历算法，类似于层次遍历。其原理如图 3-5，红色箭头方向代表遍历结点的顺序。

：

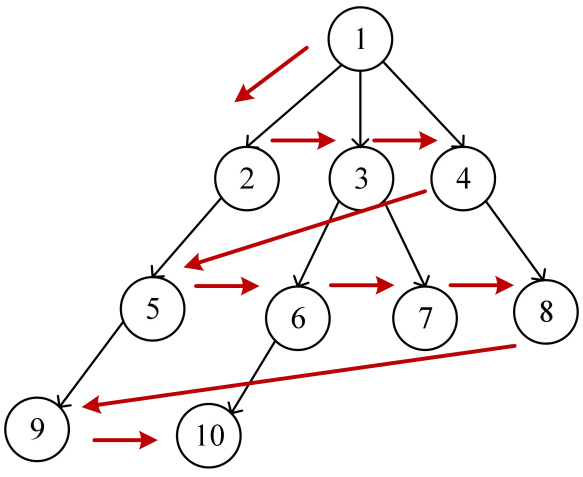

图 3-5 广度优先遍历示意图

广度优先遍历的顺序为 1，2，3，4，5，6，7，8，9，10。

为了优化存储空间，把邻接矩阵保存为稀疏矩阵，得到的邻接矩阵存储在附件内**“q3.mat”**的 Neighbor 变量中，另一个变量 Nbor 内存放的是完整的邻接矩阵。下面仅展示变量 Neighbor 的部分稀疏矩阵结果：

表 3-7 变量 Neighbor 的部分稀疏矩阵结果

坐标点	标记变量	坐标点	标记变量
（5，1）	1	…	…
（60，1）	1	（603，604）	1
（63，1）	1	（418，605）	1
（68，1）	1	（605，606）	1
（79，1）	1	（605，607）	1
…	…	（606，607）	1

注：该表格是 MATLAB 结果，序列坐标从 1 开始，如（60，1）实际对应基本块 59 指向基本块 0。

上述表格的结果表示基本块之间的指向关系。例如，坐标点（5，1）=1 代表 5 号基本块指向 1 号基本块，（606，607）=1 代表 606 号基本块指向 607 号基本块，这样所有的 y 坐标为 1 的坐标点，对应的 x 坐标就是基本块 1 的所有入度。即基本块 5，60，63，68，79 等均为基本块 1 的入度数。

3.4.2 邻接矩阵转换为树

由于问题中 607 个基本块转换为图的结果过于大，在此只显示部分数据，具体数据存放在附件内**“tree.mat”**文件中，通过调用 MATLAB 可视化语句 **view(bg1);**，将邻接矩阵转换的树可视化。本文根据显示的结果得到广度优先搜索算法的输入结点为 Node366，即第 365 个基本块（MATLAB 中序列从 1 开始）。图的起始部分如图 3-6 所示（部分图展示）：

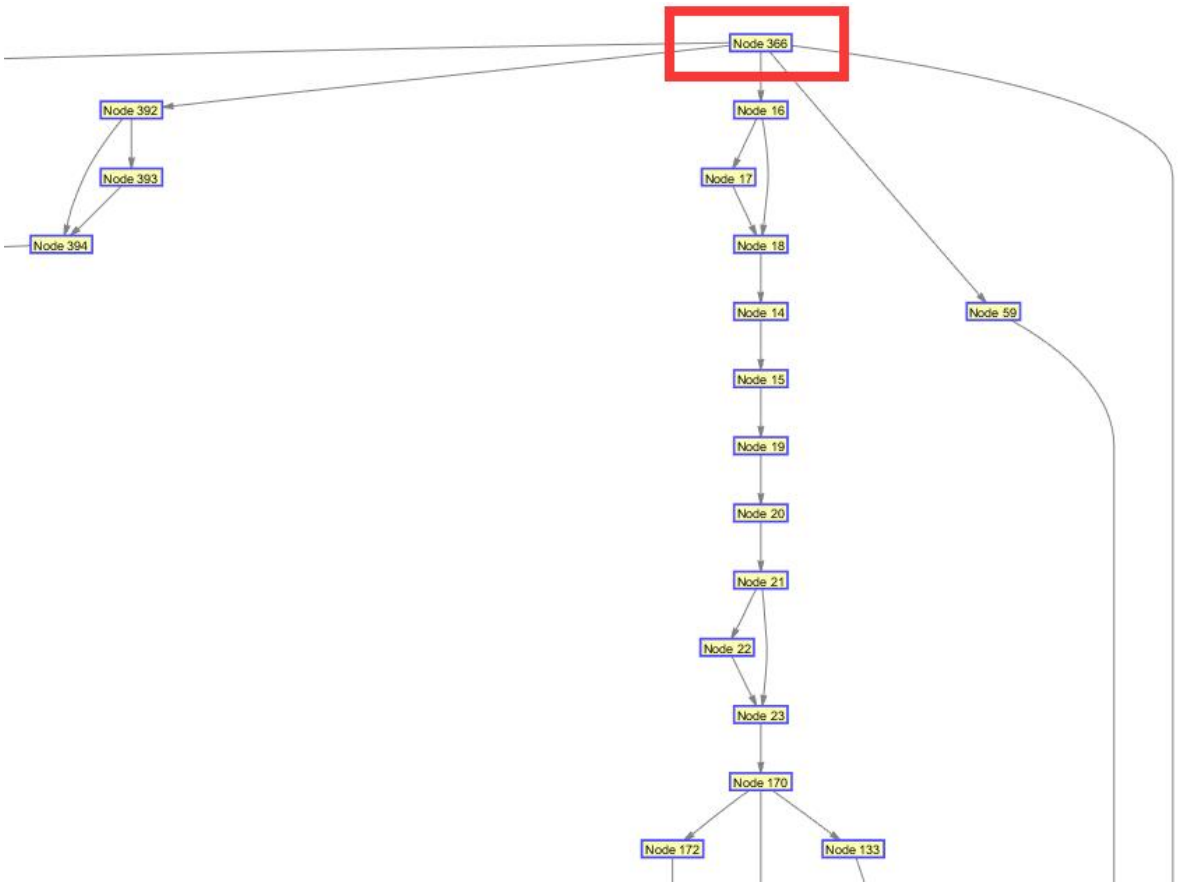

图 3-6 图起始部分的部分结果展示图

可视化的结果得到了 Node366（即基本块 365）为入度为 0 的点，将其作为广度优先遍历的第一个结点。得到广度优先遍历的结果部分展示如下：

365→0→15→58→377→391→1→2→16→17→59→……→65→322→283→66→67

将 BFS 算法得到的基本块序列，分层放入流水线的层级中。至此，基于 3.2 小节数据预处理部分得到的结果，增加数据依赖以及题目中所给的 7 个约束条件进行求解。

3.4.3 基于贪心策略的流水线资源排布算法的设计

贪婪算法又叫贪心算法。当求解问题时，总是做出当前看来最好的选择，不从整体上加以考虑，所得到的结果仅仅是某种意义的局部最优解。

贪婪算法没有固定的算法框架，算法设计的关键是贪心策略的选择。贪心算法不是对所有问题都能得到整体最优解，选择的贪心策略必须是具备无后效性，即某个状态以后的过程不会影响以前的状态，只与当前状态有关。其求解过程如下：

建立数学模型来描述问题；

（1）把求解的问题分成若干个子问题；

（2）对每一子问题求解，得到子问题的局部最优解；

（3）把子问题的局部最优解合成原来问题的一个解。本文基于贪心策略设计 PISA 架构芯片资源排布算法。贪心算法又称贪婪算法，在对问题求解时，总是做出在当前看来是最好的选择，即通过贪心选择来达到目的。

算法 1 按照广度优先遍历基本块的顺序将基本块输入，从该基本块可以放入的最低级流水线开始，在读写关系约束条件下，将每个基本块放入流水线，计算流水线是否超出资源限制，若超出，则将该基本块放入下一级流水线，直至可以成功放入。

算法伪代码如下，时间复杂度为$O(n^3)$：

算法 1:基于贪婪策略的流水线资源排布算法

输入：每个基本块包含的四种资源数:resources
广度优先遍历图得到的节点顺序:BFS

输出：排布后的流水线每级包含的基本块:S

```
def PISA(resources, BFS)
S          //流水线每级包含的基本块
S_resource   //流水线每级的资源
S_n        //当前最后一级流水线
START[]    //所有基本块的起始流水线
for i in BFS[0]: //按广度优先顺序遍历每个基本块
  for j in START[i] to S_n: //从标记的起始流水线开始放入基本块
    S_resources[j]+=resources[i][1:] //将基本块资源加入当前流水线资源量
    If 不符合约束条件
      S_resources[j] -= resources[i][1:]  //将已经加入的资源量删去
    else:
      把基本块放入 S[i]
      break
    if 如果已有流水线级全都不能放该基本块
      while True
        开辟新的基本块
        S_resources[S_n] += resources[i][1:]
        If 不符合约束
          S_resources[S_n] -= resources[i][1:] //将已加入的资源量删去
        else:
          就把基本块放入开辟的基本块
          break
  //到此，当前基本块已经放入流水线，判断其读写关系
  if 该基本块有写操作数
    for 遍历未放入流水线的其他基本块
      if 与该块有写后写操作和写后读操作
        更新 START 内容为这个基本块位置+1
  if 该基本块有读操作数
    for 遍历未放入流水线的其他基本块
      if 与该块有读后写操作
        更新 START 内容为这个基本块位置+1
return S
```

贪心算法，作为暴力求解算法是一般的求解思路，其部分结果如下，完整结果见附录。

表 3-8 未优化部分结果

级数	分配的基本块编号——贪婪算法
0	365 377 015 379 016 378 001 392 017 393 002 013 371 587 014 037 018 406 405 590 389 384 038 019 020 381 372 373 022 021 474 068 479 484 481 171 496 493 487 490 477 497 471 133 437 436 030 363 411 455 409 412 364 137 454 073 422 413 011 003 439 459 451 430 468 427 592 425 448 421 032 006 145 419 457 009 125 442 159 122 106 114 165 163 110 161 118 186 070 605 179 176 151 157 185 148 182 154 345 528 079 511 520 361 524 514 353 527 517 354 507 352 027 349 082 347 533 129 565 543 558 561 536 552 024 551 540 548 053 547 026 055 582 095 584 141 580 226 094 096 188 092 229 218 221 569 394 568 577 052 572 042 400 166 040 087 598 368 398 044 327 046 034 396 503 205 036 210 310 305 308 195 357 197 048 190 192 337 312 339 313 050 231 060 063 287 290 292 320 317 402 342 259 252 242 240 269 272 238 244 262 281 264 246 249 275 266 278 255 065 282 322
1	000 059 586 404 383 385 388 469 475 499 491 169 376 410 452 431 432
2	058 407 591 380 485 472 473 489 132 172 134 138 071 072 467 509
3	391 589 494 488 588 483 492 029 433 447 521
4	370 482 434 486 495 135 136 344 525
5	390 480 375 470 456 415 423 355 350 515 351
6	382 386 478 374 435 408 498 139 420 416 126 124 173 529 346
7	387 476 428 450 429 512
8	449 453 414 440 466
9	170 127 438 441 426 446 518 202
10	012 424 444 128 460 465 445 007 008 119 115 348
11	031 464 144 005 461 462 418 123 417 085
12	463 594 004 593 162 111 107 102 120 010 149 152 174 539 556 212 025 566 596
13	164 108 103 606 155 146
……	……
37	189 329 194 303 333 359 328 297 299 294 501 403 343
38	196 298 291 288 289 286 319 250 256 253
39	049 285 316
40	203 318 234
41	204 235 236 257 245 251 247 258 254 280
42	273 241 239 268 277
43	265 243 271 274
44	276 237 321
45	270 283
46	267
47	279
48	263
49	261
50	066
51	067

3.5 问题的优化

如表 3-8 所示是未使用优化算法的结果，可以看出流水线级数占用较多，不利于提高芯片资源的利用率。3.4.3 节是用一般算法思想进行求解，为了使算法前后对比，表现出优化算法的优势。我们选择深度学习的支持向量机（SVM）以及群智能优化算法中的遗传算法（GA 算法），二者均是对含约束问题的求解。

3.5.1 优化模型的选择

（1）支持向量机

支持向量机（Support Vector Machine, SVM）[12]是一类按监督学习（supervised learning）方式对数据进行二元分类的广义线性分类器（generalized linear classifier），其决策边界是对学习样本求解的最大边距超平面（maximum-margin hyperplane）。在 SVM 算法中，通过引入拉格朗日乘子 λ 可得到其拉格朗日函数和对偶问题，主要用于解决带约束的优化问题。

优化问题与对偶问题的转换关系如下：

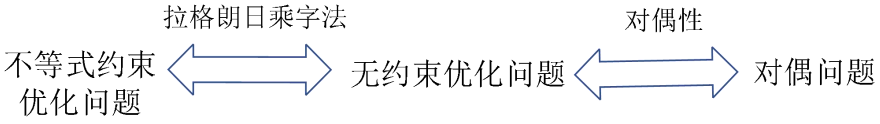

图 3-7 优化问题与对偶问题的转换关系

由拉格朗日乘子法，这里有 n 个样本，于是可为每条约束引入拉格朗日乘子 λ_i，拉格朗日函数为：

$$\ell(\omega,b,\alpha)=\frac{1}{2}\|\omega\|^2+\sum_{i=1}^{n}\lambda_i\left(1-y_i\left(\omega^T x_i+b\right)\right) \tag{3-9}$$

其对应的对偶问题可定义为：

$$\begin{aligned}&\max_{\alpha}\min_{\omega,b}\frac{1}{2}\|\omega\|^2+\sum_{i=1}^{m}\alpha_i(1-y_i(\omega^{\mathrm{T}}x_i+b))\\&s.t.\alpha_i\geq 0,i=1,2,...,m\end{aligned} \tag{3-10}$$

上述公式的最优解满足 KKT 条件。

（2）遗传算法

遗传算法（Genetic Algorithms，简称 GA）[13]是一种基于自然选择原理和自然遗传机制的搜索（寻优）算法，它是模拟自然界中的生命进化机制，在人工系统中实现特定目标的优化。遗传算法的实质是通过群体搜索技术，根据适者生存的原则逐代进化，最终得到最优解或准最优解。它必须做以下操作：初始群体的产生、求每一个体的适应度、根据适者生存的原则选择优良个体、被选出的优良个体两两配对，通过随机交叉其染色体的基因并随机变异某些染色体的基因后生成下一代群体，按此方法使群体逐代进化，直到满足进化终止条件。其实现方法如下：

Step1. 根据具体问题确定可行解域，确定一种编码方法，能用数值串或字符串表示可

行解域的每一解。

Step2. 对每一解应有一个度量好坏的依据，其度量函数叫做适应度函数，适应度函数应为非负函数。

Step3. 确定进化参数群体规模 M 、交叉概率 p_c 、变异概率 p_m 、进化终止条件。

为便于计算，一般来说，每一代群体的个体数目都取相等。群体规模越大、越容易找到最优解，但由于受到计算机的运算能力的限制，群体规模越大，计算所需要的时间也相应的增加。进化终止条件指的是当进化到什么时候结束，它可以设定到某一代进化结束，也可能根据找出近似最优是否满足精度要求来确定。遗传算法自从被提出来，得到了广泛的应用，特别是在函数优化、生产调度、模式识别、神经网络、自适应控制等领域，遗传算法发挥了很大的作用，提高了一些问题求解的效率。

遗传算法的基本理论图如图 3-8 所示：

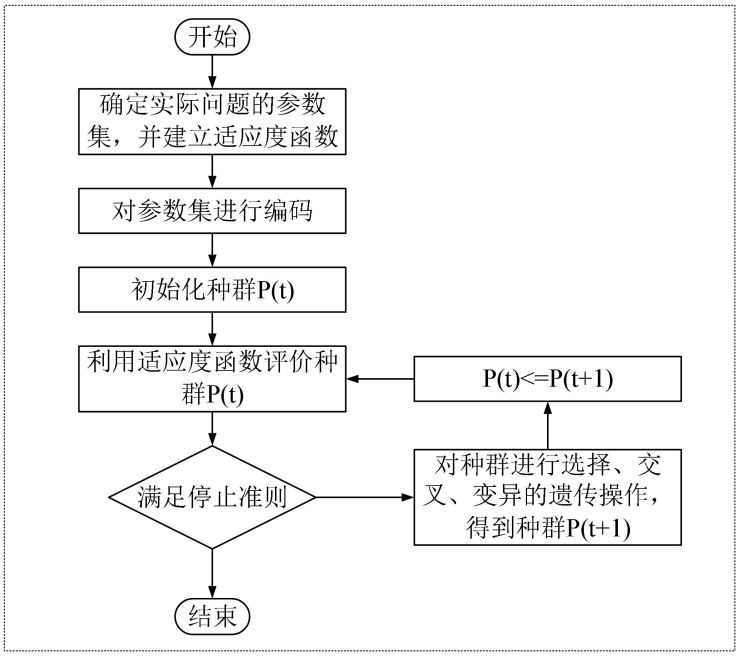

图 3-8 遗传算法基本理论图

通过分析问题可知，本题属于典型的约束条件下的多目标优化问题，而解决此类问题的算法有许多，遗传算法、粒子群算法、模拟退火算法等。本题选择最经典遗传算法进行优化求解。遗传算法的主要优化步骤如下：

A. 种群的初始化

由 3.4.2 小节求出的结果，按层次展现的部分图如下：

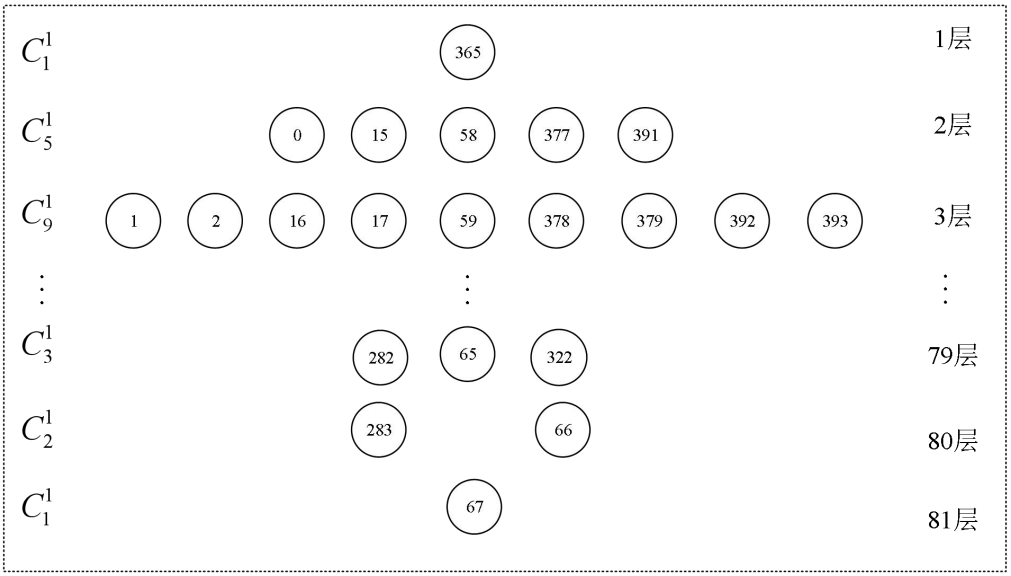

图 3-9 部分结果图

为了减少流水线的级数，且 BFS 的结果是为了解决控制依赖，那么对 BFS 的结果作为随机种群的初始化。由图可以看出，第一层的基本块 365，第二层的基本块是 15,58,377,391。解决控制依赖的话，第一层的随机顺序为 C_1^1，第二层的随机顺序有 C_5^1，以此类推。如果使用穷举法，则可能的情况有 $C_1^1 * C_5^1 * C_9^1 \ldots\ldots C_2^1 * C_1^1$ 种，则输入太过庞大。我们在各层随机初始化种群作为输入。

B. 目标函数

目标函数作为适应度函数（即，*fitness*），是评价种群迭代好坏的一个重要标准。模型建立当中的 *n* 作为目标函数，即流水线的级数，*n* 的值越小，迭代的效果越好。

当一次迭代结果优于父代结果的时候，让子代替换父代，并且赋予更高的遗传概率，有利于选优出来的子代有更高的概率保存在种群中。本文中我们加入了给图边赋值权重来影响遗传概率，记录下子代的遍历序列，给图内包含子代遍历序列的边赋值更高的权重。当有更优的 *fitness* 出现时，我们更新其权重。更新权重的对应函数如下：

$$W(x) = \frac{\sum_{i=1}^{k} W(i)}{k} \quad i = 0,1,\ldots\ldots \tag{3-9}$$

i 表示基本块，*k* 表示该边被迭代过的次数，取累计次被迭代过的平均值，权重越小，认为该边被访问迭代的次数越多。注：初次被迭代的权重值赋 1。

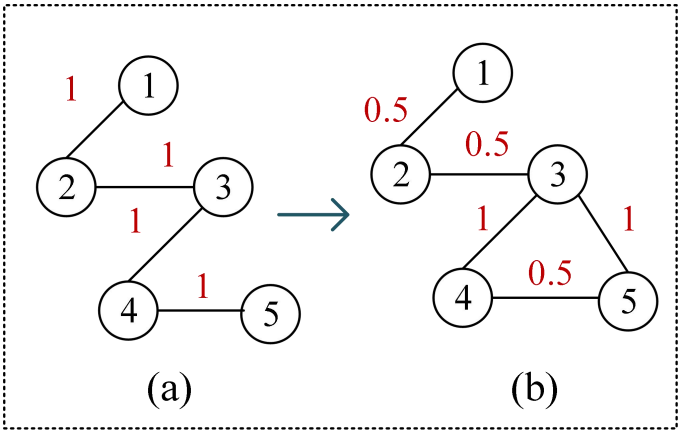

图 3-10 种群迭代优化的权重更新图

如图 3-10 所示，当随机生成的种群(a)，其对应的值为 1→2→3→4→5，经过计算得到的适应度函数 *fitness*，则初次遍历过的基本块之间的连线权值赋值为 1。迭代一次得到的结果为(b),其对应的值为 1→2→3→5→4，得到了优于（a）的 *fitness*，那么 1→2→3，4→5 的连接边的新的权重由公式（3-9）可得新的权重为 0.5。依次迭代，当迭代次数结束后，按层次搜索，寻找同层内各结点之间的最小权重值，按权重值从小到大依次确定每层的遍历顺序，最后将每层的结果顺序连接，即可得到最优的流水线级数。

3.5.2 算法求解的结果

（1）支持向量机

其流水级线数结果如表 3-9 所示，其完整结果见附录：

：

表 3-9 SVM 结果

级数	基本块号——SVM
0	365 015 391 002 017 016 378 393 392 001 379 013 371 587 014 590 384 389 018 037 406 405 019 038 381 372 020 022 373 474 479 496 477 068 490 481 493 171 487 484 497 437 471 133 436 409 030 412 411 455 363 071 364 137 413 454 422 073 003 011 459 439 451 430 427 425 421 468 592 032 448 006 145 457 419 009 125 110 122 159 163 118 161 114 442 106 165 186 070 605 179 148 154 151 176 185 182 157 345 528 079 520 353 517 511 361 527 524 507 514 354 027 352 349 082 347 533 565 129 561 558 543 536 552 551 024 540 548 547 053 026 055 582 584 580 095 141 094 218 188 226 092 096 221 229 394 569 052 572 577 568 400 040 166 042 598 087 368 398 327 044 046 396 034 503 205 036 210 357 197 048 308 195 305 310 190 192 337 312 339 313 050 231 060 063 292 287 290 320 317 402 342 259 264 275 278 249 269 272 238 246 244 240 262 266 281 242 252 255 322 065 282
1	058 059 586 404 388 385 383 469 475 499 376 169 132 410 452 432 431 074
2	000 380 407 591 485 473 374 492 029 172 134 138 072 447
3	377 589 491 588 434 375 483 489 136 433 139 467 344
4	370 480 486 495 408 135 456 449 423 466 355 350 351
5	390 482 470 170 428 429 438 414 415 440 441 124 113 105 202
6	382 386 494 435 498 450 127 444 128 420 446 416 126 008 417 173 109 542
7	387 472 453 426 464 119 115 604 010 069 075
8	488 021 424 111 107 102 117 108 121 174 152 146 443 155 149 560
9	478 031 012 418 460 445 144 465 007 103 180 078 081 080 348 557
10	476 462 461 005 463 593 123 004 594 183 101 346 596
11	162 112 177 085 539 550 556 212 566
12	458 160 104 606 535
……	……
22	510 562 537 093 325 168 208 302 207 209 303 356 360 230 297 502 284 260
23	213 091 051 097 504 329 306 358 311 335 293 299 294 341 403 253 250 256 248
24	039 334 298 285 291 286 289 319 343
25	045 288 316
26	043 234
27	369 033 047 332 235 270 236 243 258 280
28	056 366 035 189 309 333 359 328 501 237 271 277
29	395 401 399 397 194 304 279 245 268 274
30	196 273 241 321
31	049 276 283
32	203 318 265
33	307 204 257 247 254 251
34	193 267 239
35	191 261
36	198 228 263
37	066
38	067

（2）遗传算法

GA 算法的参数初始化设定：初始化种群的个数为 1000，GA 算法的迭代次数为 100。则 GA 算法的迭代次数如下：

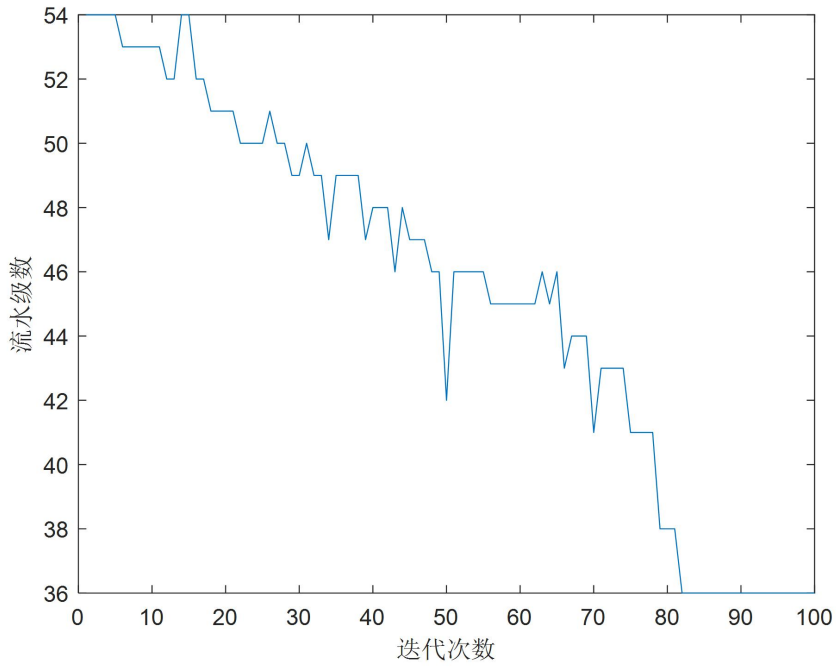

图 3-11 优化过程

由图 3-11 可知，流水级数随着迭代次数的增加，逐渐减小，最后在 36 级时趋于平稳状态，该级数就是所求的占用最短的流水线级数。

遗传算法求解的部分结果如下表所示，其完整结果见附录：

表 3-10 GA 结果

级数	分配的基本块编号——GA 算法
0	365 377 015 017 392 379 001 393 016 002 378 013 014 371 587 406 384 018 389 590 405 037 038 019 381 020 372 021 373 022 474 484 487 493 481 490 496 068 171 479 477 497 133 436 437 471 030 363 411 412 409 137 364 071 413 454 422 073 011 003 439 451 459 430 427 592 468 425 448 421 032 006 145 457 419 009 125 161 114 165 159 118 106 442 110 122 163 070 605 186 148 182 157 179 185 151 176 154 345 079 528 511 361 507 527 514 517 354 524 520 353 352 027 082 349 347 533 565 129 543 558 536 561 552 024 551 548 540 053 547 026 055 141 095 580 584 582 229 221 188 092 094 218 096 226 394 569 052 577 568 572 400 040 042 166 368 087 598 398 044 046 327 396 034 503 036 205 210 310 048 308 197 357 305 195 190 192 337 312 339 313 050 231 060 063 287 292 290 320 317 402 342 259 272 255 249 238 281 242 269 262 252 246 278 266 264 244 240 275 322 065 282
1	058 059 390 586 404 385 383 388 469 472 499 475 376 375 132 410 134 138 072 452 431 432 074 512
2	000 370 386 591 407 380 491 473 169 492 434 483 374 455 031 467 518
3	391 387 589 482 588 495 486 029 408 172 433 456 144 447 521
4	382 494 489 470 136 135 170 428 423 429 463 593 594 004 124 121 069 112 117 113 075
5	480 435 498 449 438 424 466 525
6	488 453 450 127 415 414 441 444 446 108 109
7	485 139 128 420 012 464 426 465 445 460 008 416 126 007 119 115 173 105 529 596
8	476 440 418 005 462 123 158 111 102 120 604 180 344 509 202
9	478 461 417 116 010 155 174 146 355 515 350 351 539 556 212 566
10	164 107 104 443 152 149
11	160 103 606 078 080 081 348
12	162 183 184 175 099
13	458 177 153 181 178 531 532 505 523 346 028 538 545
……	……
23	187 324 330 578 595 090 369 288 235 236 243 258 280
24	573 331 326 043 602 366 367 200 035 206 359 333 500 501 276 245 268 277
25	325 045 309 502 267 241 271 274
26	047 194 332 270 237 321
27	196 307 328 265 283
28	049 279
29	054 304 203 318
30	056 204 254 251 247
31	395 401 399 397 193 341 403 257 239
32	191 343 273
33	198 228 263
34	261
35	066
36	067

第四章 问题二模型的建立与求解

4.1 问题分析

问题二在第一问约束条件的基础上，进一步引入了执行流程对基本块在流水线排布的影响，如图 4-1 所示。

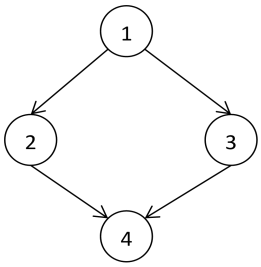

图 4-1 执行流图示例

基本块 2 和基本块 3 不在一条执行流程上，则两个基本块可以共享 HASH 和 ALU 资源。若基本块 2 和基本块 3 中任意一个的 HASH 资源与 ALU 资源均不超过每级资源限制，则基本块 2 和 3 即可排布到同一级。

基于执行流程的定义，针对问题 1 中的约束条件做出变化，即：对约束条件（2）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；对约束条件（3）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；对约束条件（5）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。

解决问题 2 的关键在于如何设计方案区分基本块是否存在于同一条执行流程，之后再修改算法一中约束条件，判断同一条执行流程上的基本块单独计算其 HASH 资源与 ALU 资源是否超过每级资源限制。

4.2 模型建立

第二问建立的模型需要在第一问模型的基础上修改约束条件，将问题 1 中的约束条件（2），（3），（5）在同一流水线级上资源的限制更改为每级中同一条执行流程上资源的限制。本问的求解目标函数如下：

假设 $h(x)$ 为判断基本块是否在同一条执行流程的函数

$$h(x,y)=\begin{cases}0\\1\end{cases}\quad x,y=0,1....606\ \ and\ x\neq y \tag{4-1}$$

在上述公式中，用 0 和 1 来表示基本块 x 和基本块 y 是否在同一条执行流程上，0 表示否，1 表示是。

$$s.t.\ \begin{cases} g_T(n) \le 1 \\ g_Q(n) \le 64 \end{cases},\ n = 0,1,2\ldots\ldots \tag{4-2}$$

如公式（4-2）是问题 1 中的（1），（4）的受限条件，而更改后的（2），（3），（5）的约束条件如下：

$$s.t.\ \begin{cases} g_H(x) + g_H(y) \le 2 \\ g_A(x) + g_A(y) \le 56 \\ h(x,y) = 1 \end{cases},\ n = 0,1,2\ldots\ldots \tag{4-3}$$

对于折叠级的 TCAM 资源约束如下：

$$s.t.\ g_f(n) = \begin{cases} g_T(n) + g_T(n+16) \le 1 \\ n \le 16 \end{cases} \tag{4-4}$$

而对折叠级的 HASH 资源需要考虑同条执行流程，其约束条件如下：

$$s.t.\ g_f(n,x,y) = \begin{cases} g_T(x) + g_T(y+16) \le 1 \\ h(x,y) = 1 \end{cases} \tag{4-5}$$

4.3 模型求解

由第一问求解的结果可知，遗传算法能得到更短的流水级数，对资源排布算法的优化效果更好。因此，本文在遗传算法的基础上进行优化求解。

第二问求解思路延续第一问的思路进行改进。首先，设计同一执行流程查找算法，寻找所有的执行流程，作为流水线资源排布算法的判断依据；然后，修改流水线资源排布算法的条件，若同一级内同一执行流程中的基本块的 HASH 资源与 ALU 资源均不超过每级资源限制，则可以放入该流水线级；最后，使用遗传算法优化求解，以得到最小流水线级数作为优化目标进行求解。

4.3.1 执行流程查找算法的设计

设计执行流程查找算法，寻找所有的执行流程，作为流水线资源排布算法的判断依据。算法 2 总体思路为，选定起始节点，从第一问中对图遍历结果可知，初始节点设为 365，按照深度优先遍历查找每条路径，直至没有后继节点。算法伪代码如下：

算法 2: 执行流程查找算法
输入：图的邻接表，以字典形式保存 graph 保存路径的栈:stack 起始节点:start 截止点:end **输出：**所有执行路径 V_s
1. findAllRoutes(graph,stack,start, end) 2. 将起始点添到栈中 3. If 找到截止点 4. 将路径保存，栈顶出栈 5. else: 6. For 栈顶元素的每一个邻接点 7. if 该点不在栈中: 8. findAllRoutes(stack,nextPoint, end) //函数迭代 9. 栈顶出栈

4.3.2 流水线资源排布算法的设计

算法 3 和算法 1 整体思路相似，按照广度优先遍历基本块的顺序将基本块输入，从该基本块可以放入的最低级流水线开始，在读写关系约束条件下，将每个基本块放入流水线。先判断流水线资源是否超出除 2，3，5 约束外的资源限制，再判断该级在同一执行流程的基本块是否符合 2，3，5 约束，若不符合则将该基本块放入下一级流水线，直至可以成功放入。

算法 3:考虑执行流程约束的流水线资源排布算法

输入：每个基本块包含的四种资源数:resources
广度优先遍历图得到的节点顺序:BFS

输出：排布后的流水线每级包含的基本块:S

```
def PISA(resources, BFS)
S          //流水线每级包含的基本块
S_resource   //流水线每级的资源
S_n        //当前最后一级流水线
START[]    //所有基本块的起始流水线
for i in BFS[0]:   #  按广度优先顺序遍历每个基本块
    for j in START[i] to S_n:   #  从标记的起始流水线开始放入基本块
        S_resources[j]+=resources[i][1:] #将基本块资源加入当前流水线资源量
        If 资源不符合 2，3，5 约束外的资源约束条件
            S_resources[j] -= resources[i][1:]   #将已经加入的资源量删去
        else if  该级在同一执行流程的基本块不符合 2，3，5 约束
        else:
            把基本块放入 S[i]
            break
            if 如果已有流水线级全都不能放该基本块
                while True
                    开辟新的基本块，更新资源
          If 不符合约束
            删去已加入的资源量
                  else:
            就把基本块放入开辟的基本块
                      break
    # 到此，当前基本块已经放入流水线，判断其读写关系
    if 该基本块有写操作数
      for 遍历未放入流水线的其他基本块
              if 与该块有写后写操作和写后读操作
          更新 START 内容为这个基本块位置+1
    if 该基本块有读操作数
      for 遍历未放入流水线的其他基本块
              if 与该块有读后写操作
          更新 START 内容为这个基本块位置+1
return S
```

4.3.3 问题二的结果

问题二的结果如下：

表 4-1 GA 结果

级数	结果第二问
0	365 058 015 393 378 016 002 392 017 379 001 013 014 587 371 384 389 037 018 405 406 590 038 019 372 020 381 022 373 376 169 474 490 171 493 068 479 496 487 481 477 484 437 471 436 497 133 134 409 030 412 363 071 364 138 137 422 454 072 073 413 011 003 459 439 430 451 421 427 592 448 425 032 468 145 006 457 419 009 125 114 163 118 165 110 106 161 159 122 442 605 186 070 148 151 176 182 179 157 185 154 345 528 079 517 354 524 353 361 511 514 527 520 507 027 352 347 349 082 533 129 565 536 558 543 561 552 024 551 548 540 053 547 026 055 580 582 584 141 095 094 226 221 096 218 188 092 229 394 569 052 572 577 568 166 400 042 040 087 368 598 398 044 046 327 503 396 034 205 036 210 048 308 310 357 197 195 305 190 192 312 337 313 339 050 231 060 063 290 287 292 320 317 402 342 259 244 272 269 278 242 262 246 252 264 240 281 255 266 238 249 275 065 282 322
1	000 059 390 586 404 385 388 383 469 472 499 475 374 132 172 455 410 452 432 431 467 074
2	391 370 386 591 380 407 485 473 489 375 411 433 447
3	377 387 589 482 021 588 483 495 423
4	382 491 486 492 135 415
5	480 434 470 420 416 126 173
6	488 408 029 435 498 456 107 417
7	494 136 170 428 429 124 111 115 121 109 069 112 604 117 113 105 075 344
8	476 449 453 414 438 012 426 466 119 116 103 355 350 351 202
9	478 450 127 464 440 424 031 128 441 144 446 005 007 008 104 348
10	139 444 418 462 463 460 461 445 465 004 594 593 158 120 010 149 155 146 606 443 152 346 539 556 212 566
……	……
25	304 328 257 241 321
26	535 553 193 270 283
27	550 555 191 276
28	198 228 267
29	546 261
30	541 054 098 089 263
31	549
32	330 090

第五章 模型总结

5.1 模型的优点

对第一问的求解采用贪婪策略、支持向量机、遗传算法三种模型进行求解。由结果分析对比，遗传算法得到最短的流水线级数。因此，第二问我们选择遗传算法求解。

（1）贪婪算法：做决策所需的计算复杂度较低。

（2）支持向量机：将问题转化为凸优化问题，思想简单，将样本与决策面的间隔最大化，有较好的效果。

（3）遗传算法：变异机制避免算法陷入局部最优，搜索能力强；引入自然选择，个体的选择具有随机性；优化结果与初始条件无关；算法独立于求解域；有较强的鲁棒性。

5.2 模型的缺点

（1）贪婪算法：容易得到局部最优解。

（2）支持向量机：难以解决多分类问题；对大规模训练样本难以实施。

（3）遗传算法：收敛速度慢；编程复杂；算法内交叉率、变异率需要依靠经验等其他方法缺点；对初始种群的优劣依赖性较强。

参考文献

[1] 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.

[2] 刘莹,曹畅,杨建军,张帅.P4技术的边缘计算相关应用研究[J].信息通信技术,2020,14(04):45-50.

[3] 何国锋. OpenFlow 在下一代数据中心网络的应用研究[J]. 互联网天地, 2013 (3): 71-74.

[4] Afek Y, Bremler-Barr A, Shafir L. Network anti-spoofing with SDN data plane[C]//IEEE INFOCOM 2017-IEEE conference on computer communications. IEEE, 2017: 1-9.

[5] 赵敏,田野.P4与POF协议无关可编程网络技术比较研究[J].网络新媒体技术,2018,7(1):54-58.

[6] 尼克•麦克欧文,金昶勳,高荣新.用P4对数据平面进行编程[J].中国计算机学会通讯,2016,12(7):12-20.

[7] 林耘森箫, 毕军, 周禹. 基于 P4 的可编程数据平面研究及其应用[J]. 计算机学报, 2019, 42(11): 2539-2560.

[8] Gebara N, Lerner A, Yang M, et al. Challenging the stateless quo of programmable switches[C]//Proceedings of the 19th ACM Workshop on Hot Topics in Networks. 2020: 153-159.

[9] Paolucci F, Civerchia F, Sgambelluri A, et al. P4 edge node enabling stateful traffic engineering and cyber security[J]. Journal of Optical Communications and Networking, 2019, 11(1): A84-A95.

[10] 刘熙. 基于可编程数据平面的 DLB 算法实现[J]. 电子技术与软件工程, 2021.

[11] 郭志军.拉格朗日乘子法在有约束条件的最优化问题研究[J].邢台学院学报,2013,28(04):170-171.

[12] 张波.支持向量机法在隧道支护技术优化中的应用[J].施工技术(中英文),2022,51(13):75-80.

[13] 徐艳凯. 基于遍历搜索与遗传算法的 L 公司生产线平衡研究[D].兰州理工大学,2017.

附录

1、各个基本块对 ALU 资源占用个数情况的完整结果如下所示：

BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数
0	2	78	2	139	11	204	2	273	3	336	1	414	9	478	3	542	1
4	10	81	2	140	13	206	3	274	1	340	1	415	6	480	3	544	1
5	6	84	1	142	1	207	7	276	3	341	2	416	1	482	3	546	7
7	3	85	2	143	4	208	1	277	1	343	10	417	25	483	1	549	7
8	4	88	5	144	7	209	2	279	3	344	3	418	2	485	3	550	2
10	7	89	2	146	2	211	2	280	1	346	3	420	2	486	1	553	7
12	5	90	3	147	4	214	11	283	1	348	3	423	1	488	3	554	7
14	2	91	1	149	2	215	6	285	2	351	3	424	3	489	1	555	7
15	3	93	1	150	4	216	1	286	2	356	1	426	3	491	3	557	1
16	2	97	1	152	2	217	2	288	2	358	1	429	3	492	1	559	1
19	16	98	2	153	4	219	1	289	2	359	2	433	2	494	3	560	1
21	2	99	2	155	2	220	2	291	2	360	17	434	10	495	1	562	1
22	1	100	1	156	4	222	1	293	2	362	1	435	7	498	1	563	4
23	3	101	2	158	11	223	1	294	2	364	1	436	1	500	1	564	1
25	6	102	1	160	11	224	1	295	1	366	1	438	18	501	1	567	2
29	11	103	2	162	11	225	3	296	2	369	2	440	2	504	2	570	7
31	2	104	7	164	11	227	7	297	2	370	3	441	2	506	1	571	4
33	9	105	2	167	1	232	2	298	4	373	2	443	8	508	1	573	2
35	2	107	2	168	2	234	14	299	2	374	2	444	1	509	1	574	6
37	6	108	7	169	21	237	2	300	1	375	6	445	2	510	1	575	9
39	6	109	2	170	2	239	1	301	2	376	8	446	1	512	1	576	1
41	5	111	2	174	1	241	2	303	7	377	2	447	1	513	1	578	2
43	6	112	7	175	16	243	2	304	2	378	4	449	22	515	1	579	3
45	6	113	2	177	1	245	2	306	7	380	1	450	2	516	1	581	11
47	1	115	2	178	15	247	1	307	2	382	1	453	22	518	1	583	11
49	8	116	7	180	1	248	1	309	2	383	2	455	1	519	1	586	4
51	2	117	2	181	15	250	5	311	2	385	1	456	3	521	1	588	4
54	5	119	2	183	1	251	1	314	1	386	1	458	1	522	1	591	1
56	4	120	7	184	20	253	5	315	1	387	1	460	8	523	1	593	1
57	9	121	2	187	2	254	1	316	2	388	6	461	2	525	1	594	1
58	3	123	7	189	2	256	5	318	5	390	2	462	2	526	1	595	9
61	2	124	2	191	3	257	3	319	1	391	2	463	1	529	1	596	1
64	1	127	1	193	3	258	1	321	2	392	10	464	1	530	1	597	2
66	2	128	3	194	4	261	3	323	2	403	6	465	2	531	6	599	1
69	7	130	1	196	4	263	3	324	1	404	8	466	1	532	3	600	6
71	1	132	2	198	2	265	3	329	7	405	1	467	1	534	7	601	1
72	6	134	1	199	2	267	3	330	1	408	7	470	2	535	7	602	1
75	1	135	1	201	10	268	1	333	9	409	1	472	3	537	7	603	18
76	2	136	6	202	1	270	3	334	1	410	1	473	1	538	7	604	7
77	7	138	2	203	2	271	1	335	2	411	1	476	3	541	7	606	25

2、各个基本块对 QUALIFY 资源占用个数情况的完整结果如下所示：

BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数	BLOCK	资源占用个数
4	3	98	5	213	1	332	1	417	4	534	1
7	4	99	1	214	7	333	6	423	1	535	6
8	4	100	2	215	5	338	1	428	2	537	1
10	3	101	1	227	9	340	1	431	1	538	4
12	1	126	5	228	2	343	4	432	2	539	2
14	2	127	1	230	1	350	2	433	4	541	1
19	4	128	6	232	4	353	1	440	5	544	5
21	3	130	1	233	4	354	2	441	5	545	2
28	1	131	1	234	8	355	1	443	3	549	6
54	2	138	2	235	8	358	1	449	8	553	1
56	2	139	5	236	9	359	1	452	2	554	7
57	3	142	1	250	3	360	1	453	2	555	1
59	2	143	4	253	3	362	5	455	1	556	1
61	5	145	3	256	3	364	3	456	5	559	5
62	1	167	1	260	2	365	5	458	1	562	5
64	1	168	2	283	1	366	1	460	2	566	1

66	1	169	3	284	1	367	1	461	5	570	2
67	1	170	4	295	1	369	1	462	5	573	1
74	1	172	1	296	2	373	2	463	1	574	3
75	1	173	4	298	2	374	3	469	7	578	1
76	8	189	8	300	1	375	3	475	2	579	1
77	1	191	1	301	2	376	5	498	1	585	6
78	1	193	1	302	2	385	3	499	1	588	7
80	1	198	2	314	1	390	3	501	1	589	1
83	1	199	4	323	1	395	1	502	2	591	1
84	1	200	1	324	3	397	1	504	4	593	1
85	1	201	8	325	2	399	1	505	6	594	1
86	2	203	5	326	1	401	1	508	4	599	1
88	2	204	5	328	1	403	8	529	4	601	1
89	5	206	1	329	4	407	1	530	1	602	1
90	6	211	4	330	1	414	4	531	1	603	8
97	2	212	1	331	1	416	1	532	1	606	1

3、问题一的 3 个完整结果如下：

级数	分配的基本块编号——贪婪算法
0	365 377 015 379 016 378 001 392 017 393 002 013 371 587 014 037 018 406 405 590 389 384 038 019 020 381 372 373 022 021 474 068 479 484 481 171 496 493 487 490 477 497 471 133 437 436 030 363 411 455 409 412 364 137 454 073 422 413 011 003 439 459 451 430 468 427 592 425 448 421 032 006 145 419 457 009 125 442 159 122 106 114 165 163 110 161 118 186 070 605 179 176 151 157 185 148 182 154 345 528 079 511 520 361 524 514 353 527 517 354 507 352 027 349 082 347 533 129 565 543 558 561 536 552 024 551 540 548 053 547 026 055 582 095 584 141 580 226 094 096 188 092 229 218 221 569 394 568 577 052 572 042 400 166 040 087 598 368 398 044 327 046 034 396 503 205 036 210 310 305 308 195 357 197 048 190 192 337 312 339 313 050 231 060 063 287 290 292 320 317 402 342 259 252 242 240 269 272 238 244 262 281 264 246 249 275 266 278 255 065 282 322
1	000 059 586 404 383 385 388 469 475 499 491 169 376 410 452 431 432
2	058 407 591 380 485 472 473 489 132 172 134 138 071 072 467 509
3	391 589 494 488 588 483 492 029 433 447 521
4	370 482 434 486 495 135 136 344 525
5	390 480 375 470 456 415 423 355 350 515 351
6	382 386 478 374 435 408 498 139 420 416 126 124 173 529 346
7	387 476 428 450 429 512
8	449 453 414 440 466
9	170 127 438 441 426 446 518 202
10	012 424 444 128 460 465 445 007 008 119 115 348
11	031 464 144 005 461 462 418 123 417 085
12	463 594 004 593 162 111 107 102 120 010 149 152 174 539 556 212 025 566 596
13	164 108 103 606 155 146
14	160 069 604 121 105 113 112 117 109 180 443 075 599
15	158 116 177 178 181
16	458 104 183 156 184 175 531 505 028 560 545
17	153 530 076 077 532 078 081 080 538 542 211 057 214 086
18	147 506 099 101 083 084 563 564 557 130 140 219 091
19	150 522 523 534 131 554 227 215 093 217 220 142 225 224 097 595 336 295 248
20	526 508 100 362 537 023 222 216 323 567 143 167 602 326 340 300 061
21	516 544 535 555 223 314 301 233
22	513 559 550 553 324 573 338 296
23	510 562 546 578 331 500 232
24	519 213 549 541 054 098 089 325 064 502
25	187 051 330 090 600 062

26	367 200 206 309
27	583 304
28	307
29	581
30	056 585 575
31	571 570 574 395 579 401 597 399 397 191
32	041 576 088 201 193
33	039 601 199 208 209 198 228
34	045 168 358
35	369 043 603 504 334 311 315
36	033 366 035 302 207 047 356 306 332 360 335 230 293 341 284 260
37	189 329 194 303 333 359 328 297 299 294 501 403 343
38	196 298 291 288 289 286 319 250 256 253
39	049 285 316
40	203 318 234
41	204 235 236 257 245 251 247 258 254 280
42	273 241 239 268 277
43	265 243 271 274
44	276 237 321
45	270 283
46	267
47	279
48	263
49	261
50	066
51	067

级数	基本块号——SVM
0	365 015 391 002 017 016 378 393 392 001 379 013 371 587 014 590 384 389 018 037 406 405 019 038 381 372 020 022 373 474 479 496 477 068 490 481 493 171 487 484 497 437 471 133 436 409 030 412 411 455 363 071 364 137 413 454 422 073 003 011 459 439 451 430 427 425 421 468 592 032 448 006 145 457 419 009 125 110 122 159 163 118 161 114 442 106 165 186 070 605 179 148 154 151 176 185 182 157 345 528 079 520 353 517 511 361 527 524 507 514 354 027 352 349 082 347 533 565 129 561 558 543 536 552 551 024 540 548 547 053 026 055 582 584 580 095 141 094 218 188 226 092 096 221 229 394 569 052 572 577 568 400 040 166 042 598 087 368 398 327 044 046 396 034 503 205 036 210 357 197 048 308 195 305 310 190 192 337 312 339 313 050 231 060 063 292 287 290 320 317 402 342 259 264 275 278 249 269 272 238 246 244 240 262 266 281 242 252 255 322 065 282
1	058 059 586 404 388 385 383 469 475 499 376 169 132 410 452 432 431 074
2	000 380 407 591 485 473 374 492 029 172 134 138 072 447
3	377 589 491 588 434 375 483 489 136 433 139 467 344
4	370 480 486 495 408 135 456 449 423 466 355 350 351
5	390 482 470 170 428 429 438 414 415 440 441 124 113 105 202
6	382 386 494 435 498 450 127 444 128 420 446 416 126 008 417 173 109 542
7	387 472 453 426 464 119 115 604 010 069 075
8	488 021 424 111 107 102 117 108 121 174 152 146 443 155 149 560
9	478 031 012 418 460 445 144 465 007 103 180 078 081 080 348 557
10	476 462 461 005 463 593 123 004 594 183 101 346 596
11	162 112 177 085 539 550 556 212 566
12	458 160 104 606 535
13	164 116 175 181 529 025
14	158 120 184 178 521 599
15	153 076 530 077 531 532 505 509 028 538 545 211 057 214 086

16	150 506 525 084 563 083 564 130 023 553 140 219
17	147 512 516 131 554 555 227 583 216 142 336 295 061
18	156 515 513 546 581 222 215 224 220 225 567 167 600 340 300 064
19	518 519 362 541 054 585 323 223 217 143 571 098 089 574 579 570 597 314 301 233 062
20	523 099 526 544 549 187 575 324 573 576 595 201 338 296
21	522 508 100 534 559 578 041 330 331 090 601 088 326 602 199 603 367 200 206 315 232 500
22	510 562 537 093 325 168 208 302 207 209 303 356 360 230 297 502 284 260
23	213 091 051 097 504 329 306 358 311 335 293 299 294 341 403 253 250 256 248
24	039 334 298 285 291 286 289 319 343
25	045 288 316
26	043 234
27	369 033 047 332 235 270 236 243 258 280
28	056 366 035 189 309 333 359 328 501 237 271 277
29	395 401 399 397 194 304 279 245 268 274
30	196 273 241 321
31	049 276 283
32	203 318 265
33	307 204 257 247 254 251
34	193 267 239
35	191 261
36	198 228 263
37	066
38	067

级数	分配的基本块编号——GA 算法
0	365 377 015 017 392 379 001 393 016 002 378 013 014 371 587 406 384 018 389 590 405 037 038 019 381 020 372 021 373 022 474 484 487 493 481 490 496 068 171 479 477 497 133 436 437 471 030 363 411 412 409 137 364 071 413 454 422 073 011 003 439 451 459 430 427 592 468 425 448 421 032 006 145 457 419 009 125 161 114 165 159 118 106 442 110 122 163 070 605 186 148 182 157 179 185 151 176 154 345 079 528 511 361 507 527 514 517 354 524 520 353 352 027 082 349 347 533 565 129 543 558 536 561 552 024 551 548 540 053 547 026 055 141 095 580 584 582 229 221 188 092 094 218 096 226 394 569 052 577 568 572 400 040 042 166 368 087 598 398 044 046 327 396 034 503 036 205 210 310 048 308 197 357 305 195 190 192 337 312 339 313 050 231 060 063 287 292 290 320 317 402 342 259 272 255 249 238 281 242 269 262 252 246 278 266 264 244 240 275 322 065 282
1	058 059 390 586 404 385 383 388 469 472 499 475 376 375 132 410 134 138 072 452 431 432 074 512
2	000 370 386 591 407 380 491 473 169 492 434 483 374 455 031 467 518
3	391 387 589 482 588 495 486 029 408 172 433 456 144 447 521
4	382 494 489 470 136 135 170 428 423 429 463 593 594 004 124 121 069 112 117 113 075
5	480 435 498 449 438 424 466 525
6	488 453 450 127 415 414 441 444 446 108 109
7	485 139 128 420 012 464 426 465 445 460 008 416 126 007 119 115 173 105 529 596
8	476 440 418 005 462 123 158 111 102 120 604 180 344 509 202
9	478 461 417 116 010 155 174 146 355 515 350 351 539 556 212 566
10	164 107 104 443 152 149
11	160 103 606 078 080 081 348
12	162 183 184 175 099

13	458 177 153 181 178 531 532 505 523 346 028 538 545
14	150 076 530 077 506 508 100 563 085 564 560 211 086 576
15	156 510 101 084 083 534 542 130 023 554 057 025 214 051
16	147 526 557 537 131 550 583 227 219 574 570 579 597 336 061
17	522 362 535 553 216 093 567 601 088 201 603 315 340
18	519 544 581 199 208 302 207 209 356 303 360 314 230 301 284 260
19	516 562 555 140 091 585 097 571 599 168 329 306 358 335 338 299 294 296 248
20	513 559 222 142 575 033 504 334 311 293 300 298 232 291 286 289 319 253 256
21	546 213 215 220 225 224 041 600 167 189 297 295 064 316 250
22	549 541 323 223 217 143 098 089 039 233 285 062 234
23	187 324 330 578 595 090 369 288 235 236 243 258 280
24	573 331 326 043 602 366 367 200 035 206 359 333 500 501 276 245 268 277
25	325 045 309 502 267 241 271 274
26	047 194 332 270 237 321
27	196 307 328 265 283
28	049 279
29	054 304 203 318
30	056 204 254 251 247
31	395 401 399 397 193 341 403 257 239
32	191 343 273
33	198 228 263
34	261
35	066
36	067

4、问题二的完整结果如下：

级数	结果第二问
0	365 058 015 393 378 016 002 392 017 379 001 013 014 587 371 384 389 037 018 405 406 590 038 019 372 020 381 022 373 376 169 474 490 171 493 068 479 496 487 481 477 484 437 471 436 497 133 134 409 030 412 363 071 364 138 137 422 454 072 073 413 011 003 459 439 430 451 421 427 592 448 425 032 468 145 006 457 419 009 125 114 163 118 165 110 106 161 159 122 442 605 186 070 148 151 176 182 179 157 185 154 345 528 079 517 354 524 353 361 511 514 527 520 507 027 352 347 349 082 533 129 565 536 558 543 561 552 024 551 548 540 053 547 026 055 580 582 584 141 095 094 226 221 096 218 188 092 229 394 569 052 572 577 568 166 400 042 040 087 368 598 398 044 046 327 503 396 034 205 036 210 048 308 310 357 197 195 305 190 192 312 337 313 339 050 231 060 063 290 287 292 320 317 402 342 259 244 272 269 278 242 262 246 252 264 240 281 255 266 238 249 275 065 282 322
1	000 059 390 586 404 385 388 383 469 472 499 475 374 132 172 455 410 452 432 431 467 074
2	391 370 386 591 380 407 485 473 489 375 411 433 447
3	377 387 589 482 021 588 483 495 423
4	382 491 486 492 135 415
5	480 434 470 420 416 126 173
6	488 408 029 435 498 456 107 417
7	494 136 170 428 429 124 111 115 121 109 069 112 604 117 113 105 075 344

8	476 449 453 414 438 012 426 466 119 116 103 355 350 351 202
9	478 450 127 464 440 424 031 128 441 144 446 005 007 008 104 348
10	139 444 418 462 463 460 461 445 465 004 594 593 158 120 010 149 155 146 606 443 152 346 539 556 212 566
11	123 162 102 174 177 175 178 181 184 078 081 080 085 596 599
12	164 108 180 530 076 077 211 057 025 214 227 086
13	160 183 101 083 084 130 023 219 567
14	458 156 531 532 505 538 028 560 131 545 091 215 217 220 224 225 600 336 061
15	153 525 506 563 564 557 583 093 216 323 143 097 595 201 340 064 248
16	147 518 522 542 554 581 223 326 602 088 199 603 315 314 296 062
17	150 512 513 140 585 324 575 579 573 574 570 597 033 168 208 209 329 302 356 303 360 338 230 293 301 285 341 403 260 284
18	509 510 362 222 142 571 578 576 331 504 189 207 194 306 358 311 335 297 294 295 299 500 232 288 343 253 256 250
19	521 519 559 187 056 325 601 167 196 334 298 300 291 502 289 286 319
20	515 526 562 395 039 401 399 367 200 397 206 049 233 316
21	529 099 516 544 041 309 203 318 234
22	523 534 213 051 369 045 204 235 279 236 237 247 254 251 258 277
23	508 100 043 366 035 359 307 333 501 265 245 239 271 274 066
24	537 332 047 273 243 280 268 067
25	304 328 257 241 321
26	535 553 193 270 283
27	550 555 191 276
28	198 228 267
29	546 261
30	541 054 098 089 263
31	549
32	330 090

5、相关代码

（1）数据预处理代码：

```
%对M1文件处理  也是从0开始需要+1个  真实是需要-1个 得到所有资源数在BLOCK
clc;clear all;close all;
load('M1.mat');
[H,W]=size(M1);
TCAM=zeros(H,2);%第一列是第i-1块 第二列是需要多少个
HASH=zeros(H,2);
ALU=zeros(H,2);
QUALIFY=zeros(H,2);
for i=1:H  %i是行 j是列
    T=M1(i,2);
    H=M1(i,3);
    A=M1(i,4);
    Q=M1(i,5);
    if(T~=0)
        TCAM(i,1)=i;
        TCAM(i,2)=T;
    end
    if(H~=0)
        HASH(i,1)=i;
        HASH(i,2)=H;
    end
    if(A~=0)
        ALU(i,1)=i;
        ALU(i,2)=A;
    end
    if(Q~=0)
        QUALIFY(i,1)=i;
        QUALIFY(i,2)=Q;
    end

end
%得到有资源数的基本块
TCAM_num=find(TCAM(:,1)>=1);
HASH_num=find(HASH(:,1)>=1);
ALU_num=find(ALU(:,1)>=1);
QUALIFY_num=find(QUALIFY(:,1)>=1);

%得到所有的基本块及其对应的资源数
TCAM_BLOCK=TCAM(TCAM_num,:);
HASH_BLOCK=HASH(HASH_num,:);
ALU_BLOCK=ALU(ALU_num,:);
QUALIFY_BLOCK=QUALIFY(QUALIFY_num,:);
```

```
load('M2.mat');
%M2中，第一列是块号，第二列是读写 0写1读 3列之后mod999为读取的X变量
%获取每个变量的读以及写的基本块
[H,W]=size(M2);
W_xo=zeros(500,500);
R_xo=zeros(500,500);
%按行读取 R_xo  xo读取 W_xo  xo写入
for i=1:2:H  %一次扫两行 0号块读了X0 X1 4号块读了X5 X6
    for j=3:W
        if(M2(i,j)~=0)
            X_w=mod(M2(i,j),999);
            if(M2(i,2)==0)  %是写操作的话
               %用坐标表示(x,y) x表示第几个变量 y表示第几个块操作 变量和块都从0开始，坐标从1开始
               disp('写');
               disp([X_w+1,M2(i,1)+1]);
               W_xo(X_w+1,M2(i,1)+1)=1; %1表示存在
            end
        end
        if(M2(i+1,j)~=0)
            X_r=mod(M2(i+1,j),999);
            if(M2(i+1,2)==1)  %是读操作的话
                disp('读');
                disp([X_r+1,M2(i,1)+1]);
                R_xo(X_r+1,M2(i,1)+1)=1;
            end
        end

    end
end

W_X_sparse=sparse(W_xo);%稀疏矩阵
R_X_sparse=sparse(R_xo);%稀疏矩阵
save q2_data_每个变量对应的基本块;
%
% bgl = biograph(W_xo);
% view(bgl);
W_xo=logical(W_xo);
R_xo=logical(R_xo);
% R_xo(end+1,:)=0;
% R_xo=logical(R_xo);
% W_R_xo=W_xo & R_xo;
% WR_X_sparse=sparse(W_R_xo);
%遍历所有的点，存储WR_X 第一列为变量号，第二列为变量被写的操作块，第三列为变量被读的情况
[H,W]=size(W_xo);
[H1,W1]=size(R_xo);
% W_X(:,1)=1:1:824;
W_X=zeros(H,W);
R_X=zeros(H,W);
for i=1:H
    flag=find(W_xo(i,:)~=0);
    len=length(flag);
    W_X(i,1:len)=flag;
end
for i=1:H1
   flag=find(R_xo(i,:)~=0);
    len=length(flag);
    R_X(i,1:len)=flag;
end
save q2;

```

```
%构建结点之间的关系 用坐标表示点 输出矩阵方程 获得控制依赖 及各个点的前驱
%对数据 稍微处理了一下 块数依次+1 把0号块隔过去写1
% 如果3跟1有边 则 (3,1)=1 如果(1,3)有边则 (1,3) 为1
clc;clear all;close all;
load('M3.mat');
%数据意义： 0 1 2 0号块是1 2的前驱
[H,W]=size(M3);
Nbor=zeros(max(H,W),max(H,W));
k=2;%列扫描 从第二列开始
for i=1:H
     Num=M3(i,1); %访问基本块
      [m,k]=find(M3(i,:)==0,1,'first');
         Neighbor=M3(i,2:k-1);  %拿到临界点 去赋值坐标点
     %得到M3(i,1)的块他控制的基本块为Neighbor里的
     %若M3(i,1)=1 neghibor= 2 3 则 (2,3)=1 (2,4)=1 1 到2 3 有边
     if(Neighbor~=0)
     %有邻居 则赋值坐标结点
     if(Num==-1)
         x=Num+2;
     else
         x=Num+1;
     end
     len=length(Neighbor);
        for j=1:len
            if(Neighbor(1,j)==-1)
                 y=Neighbor(1,j)+2;
            else
                 y=Neighbor(1,j)+1;
            end
            Nbor(x,y)=1;
        end
     end
   Neighbor=0;
end
Neighbor=sparse(Nbor);%稀疏矩阵
save q3;
%获取树
load('q3.mat');
bg1 = biograph(Nbor);
view(bg1);
```

```
def PISA3(resources, BFS):
    """基于贪心策略，同时考虑数据依赖和控制依赖条件（读写条件）"""
    S = {0: []} # 初始化流水线每级包含的基本块
    S_resources = {0: [0, 0, 0, 0]} # 初始化流水线每级的资源 TCAM HASH ALU QUALIFY
    S_n = 0 # 当前最后一级流水线
    START = np.zeros(607, dtype=int) # START 保存所有基本块的起始流水线

    for idx, i in enumerate(BFS[0]): # 按广度优先顺序遍历
        d = 0 # 当前基本块 i 最终所在流水线级数
        # print("================================")
        # print("S_n",S_n)
        while True:
            if START[i] >= S_n + 1:
                S_n += 1
                S[S_n] = []
                S_resources[S_n] = [0, 0, 0, 0]
            else:
                break

        for j in range(START[i], S_n + 1): # 从第一级流水线开始，尝试放入基本块
            # 计算资源量
            S_resources[j] += resources[i][1:] # TCAM HASH ALU QUALIFY

            # 计算约束条件
            d_TCAM = [] # TCAM 的偶数级资源
            for k in range(S_n + 1): # 更新 d_TCAM
                if (k % 2 == 0) and (S_resources[k][0] != 0):
                    d_TCAM.append(k)
            if S_n >= 16 and S_n <= 31:
                x1 = x2 = x3 = x4 = 0
                if j < 16 and j + 16 <= S_n:
                    x1 = S_resources[j][0] + S_resources[j + 16][0]
                    x3 = S_resources[j][1] + S_resources[j + 16][1]
                if j > 16 and j <= 31:
                    x2 = S_resources[j][0] + S_resources[j - 16][0]
                    x4 = S_resources[j][1] + S_resources[j - 16][1]

            # 判断约束条件
            if ((S_resources[j][0] > 1)
                    or (S_resources[j][1] > 2)
                    or (S_resources[j][2] > 56)
                    or (S_resources[j][3] > 64)
                    or (S_n >= 16 and S_n <= 31 and (x1 > 1 or x3 > 3 or x2 > 1 or x4 > 3))
                    or (np.array(d_TCAM).shape[0] > 5)):
```

```
            # del S[j][-1] # 从这个流水线删除
            S_resources[j] -= resources[i][1:]  # TCAM HASH ALU QUALIFY
        else:
            S[j].append(int(i))  # 符合约束条件，就把基本块放进去
            d = j
            # print("jinru")
            # print(j)
```

完整代码见附件。