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	陆军工程大学
参赛队号	22910040028
队员姓名	1.陈智博
	2.崔智超
	3.杨宁

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　**基于模拟退火和粒子群混合的的 PISA 架构芯片资源排布算法设计**

摘　　要：

本文依据赛题给出的约束，建立了芯片资源优化问题，分析了其等价问题及其对偶问题，研究了芯片资源利用率与基本块排布的关系，以建模分析和计算机仿真相结合的方式，设计了 PISA 芯片资源排布算法，针对赛题提出了两个问题，分别给出了实际解决方案和仿真结果。

针对问题 1，研究了排布顺序的影响，确定了正向排布顺序，以控制依赖和数据依赖为基础，基于逻辑合并构造了决策矩阵，建立了决策约束排布规则，确保求解结果在可行域内，基于决策矩阵的有向无环图 DAG 的拓扑排序定义了优先度指标，给出了基于粒子优先度更新的基本块优选算法，为避免局部最优，设计了基于模拟退火的全局最优保护机制，在满足约束的条件下，流水线技术可达 39 级，四种资源的最大利用率最大，TCAM 资源利用率可达 58.33%，HASH 资源利用率可达 98.44%，ALU 资源的利用率可达 60.04%，QUALIFY 资源的利用率可达 20.35%。

针对问题 2，使用深度优先搜索算法 (DFS) 计算了每级流水线包含的执行流程数，从而确定了每级流水线 HASH 和 ALU 资源占用量，构建了问题 2 的资源约束判定方法，设计了基于模拟退火和粒子群混合优化的 CRSA2 算法，以占用的流水线最短为优化目标，在满足约束的条件下，流水线技术可达 34 级，得到了众多非劣解。定义各级流水线 HASH 资源利用率均值为问题 2 的资源利用率，从众多非劣解中确定了最优芯片资源排布。

论文最后讨论了模型的优缺点和进一步的改进方向。

关键词：图论；决策矩阵；排布优先度；模拟退火；粒子群

目录

1. 问题重述

1.1 问题背景

为了提高芯片的研发效率，使其可以适应多种网络协议，可编程的交换芯片被认为是未来芯片的发展方向。PISA（Protocol Independent Switch Architecture 是当前的主流芯片架构，它在具有可编程性的同时保证了与固定功能交换芯片相当的处理速率，在未来网络中具有广阔的应用场景。当编译器对程序进行编译时，会将完成的代码分割为若干基本块，各基本块被排布到流水线各级中。高资源利用率的资源排布算法对于编译器设计至关重要。但在实际的 PISA 架构芯片的设计中，为了减少连线的复杂度，往往对流水线各级的资源、以及流水线各级之间的资源有着多种多样的约束条件，这一系列复杂的资源约束条件使得资源排布问题尤为困难。水线各级之间的资源有着多种多样的约束条件，这一系列复杂的资源约束条件使得资源排布问题尤为困难。然而，芯片的各类资源均有限，越高的资源利用率意味着能够越好的发挥芯片的能力，让芯片支持更多的业务，因此，高资源利用率的资源排布算法对于编译器设计至关重要。

作为 PISA 架构芯片设计中的一种关键算法，芯片资源排布算法是提高芯片资源利用率，更好发挥芯片的能力的重要过程，本题以 PISA 架构芯片资源排布算法为例，探讨算法的设计思路与实现方法，以实现具体场景下的最优设计。

1.2 问题提出

问题 1：总和资源约束下的芯片资源排布算法设计以流水线每级资源和一定为资源约束下求解资源排布问题，称为总和资源约束下的芯片资源排布问题。占用的流水线级数尽量短为优化目标，在满足上述数据依赖、控制依赖、以及如下资源约束条件下，给出资源排布算法，输出基本块排布结果。

给定如下资源约束：

（1）流水线每级的 TCAM 资源最大为 1；

（2）流水线每级的 HASH 资源最大为 2；

（3）流水线每级的 ALU 资源最大为 56；

（4）流水线每级的 QUALIFY 资源最大为 64；

（5）约定流水线第 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，折叠的两级 TCAM 资源加起来最大为 1，HASH 资源加起来最大为 3。注：如果需要的流水线级数超过 32 级，则从第 32 开始的级数不考虑折叠资源限制；

（6）有 TCAM 资源的偶数级数量不超过 5；

（7）每个基本块只能排布到一级。

问题 2：共享资源条件下的芯片资源排布算法设计对于不在一条执行流程上的基本块，可以共享 HASH 资源和 ALU 资源，此时的优化问题称为共享资源条件下的芯片资源排布问题。以占用的流水线级数尽量短为优化目标，在满足上述数据依赖、控制依赖、以及如下资源约束条件下，给出资源排布算法，输出基本块排布结果。

给定如下资源约束：

（1）流水线每级的 TCAM 资源最大为 1；

（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；

（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；

（4）流水线每级的 QUALIFY 资源最大为 64；

（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。

（6）有 TCAM 资源的偶数级数量不超过 5；

（7）每个基本块只能排布到一级。

1.3 研究基础

经查，本文目前没有查找到关于芯片资源排布算法相关研究及专利。

由于本题目具有线性资源约束和高维非线性约束，具有 0-1 整数规划问题特征。本题的优化目标与约束条件与车间生产规划（Job Shop Schedulin）十分相近 [1]。因此将上述两种问题的部分相关研究基础整理如下。

线性规划问题中所有未知量均为整数的问题称为整数规划问题（Integer programming）。0-1 整数规划问题是一类整数规划，是指全部待决策变量只为 0 或 1。一般这种 0-1 整数规划问题包含多种约束条件，如果采用常规的枚举算法，对每个约束条件进行判断，最坏的情况可能在多项式时间内无法解决问题，即 NP-hard 问题 [2]。因此常用的 0-1 整数规划问题的解法为隐式枚举法，隐枚举法是在枚举法基础上的优化，能够在达到最优解之前，只需要检查所有可能的变量组合的一部分即可。在判断某种情况是否可行时，会先将搜索的解代入目标函数进行计算，如果目标函数的结果大于目前的最优解再进一步进行判断是否为可行解，如果时可行解就将该结果值作为新的最优解；否则丢弃改解，继续搜索 [3]。

车间生产调度问题是经典的排布问题，该问题可概述为多个作业在多台机器上处理 [4]、[5]。每个作业由一系列任务组成，这些任务必须按照给定的顺序执行，并且每个任务必须在特定的机器上处理。问题是如何安排机器上的任务，以最小化调度的长度，即所有任务完成所需的时间。作业车间问题由几个约束条件：

（1）在作业的前一个任务完成之前，作业的任何任务不能启动。

（2）一台机器一次自能做一项任务。

（3）一项任务，一旦开始，必须运行到完成。

可以看出车间调度问题与芯片资源排布算法在约束和优化目标上十分相近，均有优先约束和资源约束。不同之处在于，调度问题的任务执行流程是单入多出的树形结构，而芯片资源排布问题是多入多出的图形结构。因此如何处理芯片多种执行顺序的约束关系是解决问题的关键。如果能够将排布优先顺序确定，芯片资源排布问题就完全转化为车间生产调度问题。可以用元启发式算法进行求解 [6]、[7]。常用的算法有粒子群算法模拟退火算法，禁忌搜索算法和遗传算法等。

2. 模型假设和已知条件

2.1 模型假设

假设 1：假设基本块中的指令并行执行。

假设 2：假设每个基本块只会写同一个变量一次（即基本块中不存在多条指令对相同变量赋值）。

假设 3：假设从第 32 开始的级数不考虑折叠资源限制。

假设 1-3 出自赛题约定。

假设 4：假设占用的流水线级数越短，芯片资源利用率越高。

假设 5：假设流水线级数上的一个资源在同一级只能服务于同一基本块。

假设 5 仅限于问题 1。

假设 6：假设不在一条执行流程上的基本块，可以共享 HASH 资源和 ALU 资源。

假设 7：假设不考虑资源并行度导致的资源浪费。

假设 6-7 仅限于问题 2。

2.2 符号系统

D: 基本块的排列矩阵

m: 流水线总等级数

n: 基本块总个数

A：基本块优先度更新算法

$N = \{n_i\}, (i = 0, 1, 2, ..., n-1)$: 基本块集合

$D = \{d_{i,j}\} \in {}^{m \times n}$: 基本块的排列矩阵

$C = \{c_{i,j}\} \in {}^{n \times 4}$: 基本块占用的各类资源数

G: 有向无环图

$V = \{v_{i,j}\}$: 顶点

E: 图中的边

$\text{hash}_u sed : HASH$

$\text{hash}_c ost(\text{n}_i)$: 基本块所占用的 HASH 资源。

$\text{hash}_p ath(i) : iHASH$

$\text{alu}_u sed : HASH$

$\text{alu}\ _c ost(\text{n}_i)$: 基本块所占用的 HASH 资源。

$\text{alu}\ _p ath(i) : iHASH$

A(G): 邻接矩阵

$P = \{P_{i,j}\}$：可达矩阵

$K = \{k_{i,j}\} \in {}^{n \times n}$：控制关系矩阵

L: 等级列表

N_p: 已排基本块集合

T^0: 初始温度

λ: 退火衰减速率

t: 迭代次数

T^t: 当前温度

3. 问题 1: 总和资源约束下的芯片资源排布算法设计

本章在问题 1 的条件下，以控制依赖和数据依赖为基础，构造了决策矩阵，建立了决策约束排布规则，以问题 1 的资源为约束，以占用的流水线级数最短为优化目标，设计了一套基于模拟退火和粒子群混合的资源排布算法 CRSA1，实现了问题 1 的目标。

3.1 问题分析及优化模型

问题 1 需要在满足资源约束、数据依赖、以及控制依赖的条件下，以占用的流水线级数尽量短为目标，给出资源排布算法。

研究问题 1 的约束条件及优化目标，分析问题 1 存在以下基本求解思路：

求解思路 1： 问题 1 给出的资源约束均为线性约束，优化目标流水线级数也可以表示为具有凸性的目标函数形式，从表面上，仅需要将数据依赖、控制依赖转化为具有凸形式的函数，该问题即可转化为具有凸形式的 0-1 整数规划问题，利用 CVX 等工具箱即可获得全局最优解。但是由于数据依赖与控制依赖的相互作用，该非线性依赖难以转化为线性形式。因此从该思路求解存在困难。

求解思路 2： 考虑到控制依赖与数据依赖两种约束，基本块之间存在执行顺序约束，因此可以考虑从图论的基本结论出发进行求解。通过将芯片资源排布顺序建模成 AOE 网络（Activity On Edge Network），顶点表示该基本块放入流水线中，有向边表示基本块放入的先后顺序，有向边上的权值表示完成将该基本块所消耗的资源。通过求出网络的关键路径即可获得 AOE 网络中基本块排布在流水线的顺序，以及资源消耗的总量。该方法的难点在于折叠级数资源限制难以表达，同时多条关键路径的搜索也较为困难。

求解思路 3： 在芯片资源排布问题中由于每次排布一个芯片后，下一次排布关系需要重新计算，因此该问题是典型的 NP-Hard 问题。而题目中共有 607 个基本块，共有 2^{607} 种组合，因此通过枚举法进行求解甚至很难找到可行解。但是可以考虑使用启发式算法解决。该问题可以理解为一种特殊的车间调度问题，全部基本块构成的 P4 程序是待加工的工件，607 个基本块是工件的 607 道工序，每级流水线资源即为车间设备资源。但是在传统的车间调度问题中，工件的生产工序是一个单向的树型结构，而芯片的资源排布顺序是一个多入多出的图型结构。因此，常用的元启发式算法搜索解空间范围过大，时间复杂度和空间复杂度仍然过高。

综合考虑求解思路 1、2、3，总结出求解问题 1 应将非线性约束转化为基本块排列规则，构建可排列基本块集合，再考虑资源约束，筛选出可行基本块集合，通过设计基于粒子适应度更新的基本块选择算法，选取当前最优基本块，通过模拟退火算法，随机跳出当前局部最优，搜索解空间。

3.1.1 芯片资源优化模型

基本块集合为 $N = \{n_i\}, (i = 0, 1, 2, \ldots, n-1)$, 基本块的排列矩阵为 $\mathbf{D} = \{d_{i,j}\} \in C^{m\times n}$, m 为流水线总级数，n 表示共有 n 个基本块待排列组合。$d_{i,j} = 1$ 表示基本块 j 排布在流水线第 i 级。

$$\mathbf{D} = \begin{bmatrix} d_{0,0} & d_{0,2} & \cdots & d_{0,n-1} \\ d_{1,1} & d_{1,2} & \cdots & d_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ d_{m-1,1} & d_{m-1,2} & \cdots & d_{m-1,n-1} \end{bmatrix} \tag{3.1}$$

基本块占用的各类资源数表示为 $\mathbf{C} = \{c_{i,j}\} \in C^{n\times 4}$，矩阵 $\mathbf{C}$ 从第 1 列到第 4 列分别为基本块占用的 TCAM、HASH、ALU 和 QUALIFY 资源。例如 $c_{i,2} = 2$ 表示基本块 i 占用的 HASH 资源为 2。

$$\mathbf{C} = \begin{bmatrix} c_{0,1} & c_{0,2} & c_{0,3} & c_{0,4} \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ \cdots & \cdots & \cdots & \cdots \\ c_{n-1,1} & c_{n-1,2} & c_{n-1,3} & c_{n-1,4} \end{bmatrix} \tag{3.2}$$

优化目标为：

$$(P3) : \min(m) \tag{3.3}$$

约束条件表示为：

$$\sum_{j=0}^{n-1} d_{i,j} c_{j,1} \leq 1, i = 0, 1, ..., m-1 \tag{3.4}$$

$$\sum_{j=0}^{n-1} d_{i,j} c_{j,2} \leq 2, i = 0, 1, ..., m-1 \tag{3.5}$$

$$\sum_{j=0}^{n-1} d_{i,j} c_{j,3} \leq 56, i = 0, 1, ..., m-1 \tag{3.6}$$

$$\sum_{j=0}^{n-1} d_{i,j} c_{j,4} \leq 64, i = 0, 1, ..., m-1 \tag{3.7}$$

$$\sum_{j=0}^{n-1} d_{i,j} c_{j,1} + \sum_{j=1}^{n} d_{(i+16),j} c_{j,1} \leq 1, i = 0, 1, ..., 15 \tag{3.8}$$

$$\sum_{j=0}^{n-1} d_{i,j} c_{j,2} + \sum_{j=1}^{n} d_{(i+16),j} c_{j,2} \leq 1, i = 0, 1, ..., 15 \tag{3.9}$$

$$\sum_{i} \sum_{j=0}^{n-1} d_{i,j} c_{j,1} \leq 5, i = 0, 2, 4, ..., m-1 \tag{3.10}$$

$$\sum_{i=0}^{m-1} d_{i,j} = 1, j = 0, 1, 2, ..., n-1 \tag{3.11}$$

公式 3.4表示流水线每级的 TCAM 资源最大为 1；公式 3.5表示流水线每级的 HASH 资源最大为 2。公式 3.6表示流水线每级的 ALU 资源最大为 56。公式 3.7表示流水线每级的 QUALIFY 资源最大为 64。公式 3.8、3.9表示流水线第 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，折叠的两级 TCAM 资源加起来最大为 1，HASH 资源加起来最大为 3。公式 3.10表示有 TCAM 资源的偶数级数量不超过 5；公式 3.11表示每个基本块只能排布到一级。

3.1.2 原问题的等价问题及对偶问题

由于 m 为未知量，无法直接作为优化目标。因此将问题转化为等价问题，将各层剩余资源最小化为优化目标，此时优化问题转化为等价问题：

$$(P2): \quad \max \ \min_{k} \sum_{j=1}^{n-1} d_{i,j} c_{j,k} \tag{3.12}$$

通过设计满足非线性约束的可行排布规则，逐层优化，求解等价问题（P2），通过迭代优化的方法可以获得最优或次优可行解。

进一步分析原问题的对偶问题，若将约束条件表示为 $g(D)$，构造拉格朗日函数：

$$L(k,\lambda) = \sum_{j=1}^{n-1} d_{i,j} c_{j,k} + \lambda g(D) \tag{3.13}$$

此时则原问题（P1）可转化为如下对偶问题：

$$(P3): \quad \min_{k} f(k) = \min_{k} \max_{\lambda} L(k,\lambda) \tag{3.14}$$

由于存在重要不等式：

$$\min_{k} \max_{\lambda} L(k,\lambda) \geqslant \max_{\lambda} \min_{k} L(k,\lambda) \tag{3.15}$$

Proof.

$$\max_{\lambda} L(k,\lambda) \geqslant L(k,\lambda) \geqslant \min_{x} L(k,\lambda) \tag{3.16}$$

当对偶问题的对偶间隙为零时，对偶问题的最优解就是原问题的最优解 [9]。所以有时候对偶问题比原问题更容易求解时，可以选择求解对偶问题（P4）。

$$(P4): \quad \min_{k} f(k) = \min_{k} \max_{\lambda} L(k,\lambda) \tag{3.17}$$

本文通过求解等价问题（P2）问题设计芯片资源排布算法。

3.2 CRSA1 算法

为解决原问题的等价问题（P2），本节设计了芯片资源排布算法 CRSA 1（Chip Resource Scheduling Algorithm for Question 1），如图 1所示：

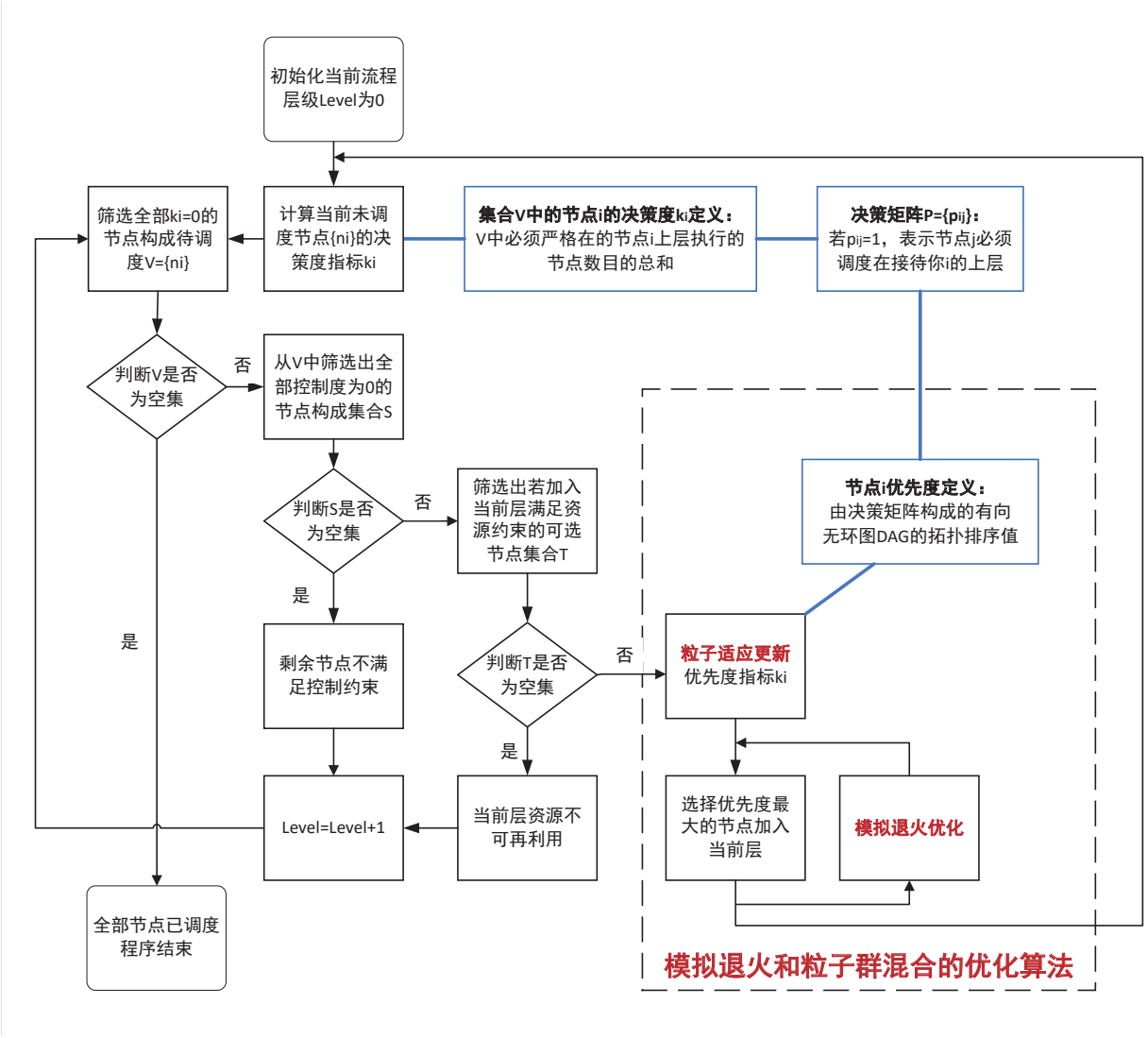

图 1: CRSA1 算法的流程图。

首先计算当前未调度基本块列表 $V = \{n_i\}$ 的决策度指标 k_i，筛选出全 $[k_i = 0$ 的基本块作为待调度基本块列表 $V_i = \{n_i\}$。若 V 为空集，表示全部基本块均已被调度，此时程序结束。若 V 不为空，为避免仍有控制关系的基本块进入当前层导致解不可行，则需要从 V 中筛选出全部控制度为 0 的基本块构成 S。若 S 为空集，则剩余基本块均存在对当前层基本块有控制约束，无法放入当前层。若 S 不为空，则需要进一步筛选出加入当前层后仍使当前层满足资源约束的基本块，这些基本块构成集合 T。此时，由于集合 T 可能存在多个可选的解，因此存在优化空间。为此，本文提出了一种基于模拟退火和粒子群混合的优化算法 CRSA1。

CRSA1 算法首先将基本块作为粒子，决策空间为基本块的选择与否。定义优先度指标作为粒子的适应度，表征其被选择的紧迫性。基于图论基本原理给出了优选度严格非增的证明，证明了以优选度作为选择指标具有明确的更新方向，因此在将优先度作为适应度指标下，算法必然会选择当前最优节点。

为了避免算法进入全局最优，设计了基于模拟退火的保护机制。即根据 Metropolis 准则，有概率不接受基于粒子适应度更新的基本块选择结果，随机从集合 T 中选择其他基本块。

在选择某基本块加入当前层后，更新决策度及粒子适应度，迭代寻找可行基本块。

3.2.1 排布顺序分析

在进行资源排布时，首先需要确定排布顺序。通过题中所给"attachment3.csv" 可以确定基本块在源程序的执行顺序，确定每个基本块执行后跳转的目的基本块，进而构建起基本块的流程图。例如，可以寻找到从某顶点出发没有边的顶点为程序结束终点，同理，可以寻找到程序起点。因此，考虑按照程序执行流程进行资源排布。

存在两种顺序排布方法，分别是由程序开始到程序结束的顺序排布和由程序结束到程序开始的排布，如图所示。

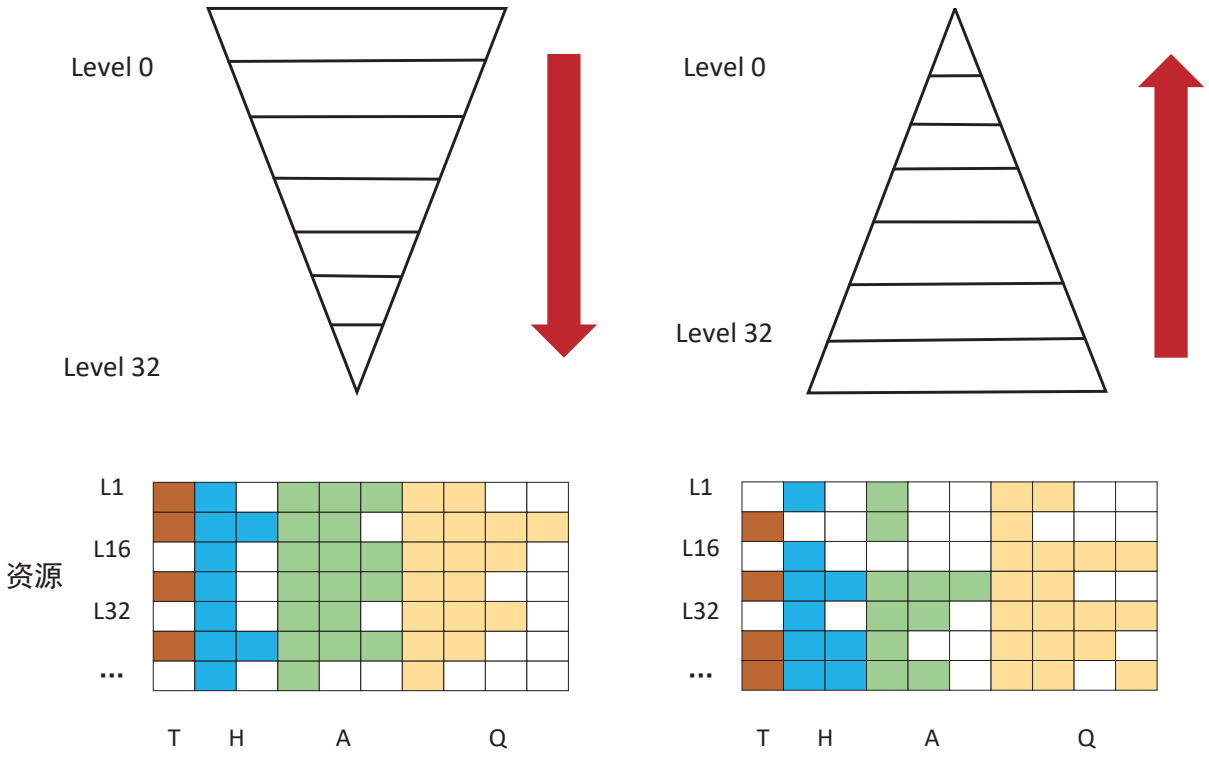

图 2: 排序图。

从程序开始到程序结束的顺序排布点数量形状类似于正金字塔，尽可能多的点被排在金字塔的第层（即 level=0 的 j 节点）。此时更多的资源被分配在底层。而由上到下的正向排布是从流水线的低级（0 级）开始，根据控制关系向下筛选基本块堆叠的方法，这种排布方式最终形成一个矩型流水线，即每级流水线的基本块数量相近，同时该方法容易引起控制约束冲突。由下到上的逆向排布是从流水线的最高级开始，向上寻找基本块，最后形成一个金字塔型流水线。

因此，使用顺序排布时容易导致决策冲突，从而使得最终解不在可行域内。因此，需要设计一种排布规则以确保解可行。相反，逆向排布是从高层级进行排布，优先排布程序终点。这种排布顺序能够使得决策冲突较少，但容易得到劣解。

综上所述，本文采用正向排布顺序，并设计排序规则以确保解在可行域内。

3.2.2 基于逻辑合并的决策关系构建

在本文 CRSA1 算法的正向顺序排布种，需要根据控制依赖及数据依赖依次选择基本块加入当前层。收到新增的正向顺序限制，新加入的基本块只能排布在当前层或下一层。为

便于决策，本节基于逻辑合并构建了基本块选择的决策关系矩阵 $J = j_{ij}$, 若 j_{ij} 为 1，表示基本块 n_i 只能排布在基本块 n_j 的上级。

首先对题目 1 中给出的非线性约束关系进行分析：

1、判断基本块之间是否存在控制依赖

（1）将流程图转换成有向图

将基本块抽象成一个节点，当基本块 A 执行完可以跳转到基本块 B 执行时，在 A 和 B 之间增加一条有向边，这样 P4 程序即可表示为一个有向无环图。在有向图中，边是单向的：每条边连接的两个顶点都是一个有序对，它们的邻接性是单向的。

我们定义所有边的长度均为 1，即 P4 程序流程图是一个有向无权图。根据“attachment3.csv”文件，我们可以得到各基本块之间的邻接关系。各节点的邻接关系即可转化成邻接矩阵用来表示此有向图。

邻接矩阵定义：设 $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ 是一个简单图，它有 n 个节点 $\mathbf{V} =\{v_1, v_2, v_3, \ldots, v_n\}$，则 n 阶方阵 $\mathbf{A}(\mathbf{G}) = \{a_{i,j}\}$ 称为 $\mathbf{G}$ 的邻接矩阵。

$$a_{ij} = \begin{cases} 1 (v_i, v_j) \in \mathbf{E} \\ 0 (v_i, v_j) \notin \mathbf{E} \end{cases} \tag{3.18}$$

邻接矩阵只能直观得到相邻的节点，但不相邻节点之间也可能存在控制依赖关系，因此我们进一步由邻接矩阵计算得出可达矩阵。

可达矩阵定义：设 $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ 是一个简单图，它有 n 个节点 $\mathbf{V} =\{v_1, v_2, v_3, \ldots, v_n\}$，则 n 阶方阵 $\mathbf{P} = \{p_{i,j}\}$ 的可达矩阵。

可达矩阵 $\mathbf{P}$ 与邻接矩阵 $\mathbf{A}$ 的关系为：

$$\mathbf{B} = (\mathbf{A} + \mathbf{I})^n = \mathbf{I} + \mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^n \tag{3.19}$$

将 $\mathbf{B}$ 中的非零元素改为 1，而零元素不变，变换后的矩阵即为可达矩阵 $\mathbf{P}$。

（2）判断有无控制依赖

控制依赖是程序控制流导致的一种约束。控制依赖定义为：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，两基本块构成控制依赖。也就是说，在有向图中，不可达点之间必然没有控制依赖关系。一对可达点之间没有控制依赖关系必须满足条件：起点的所有邻接点必须可达终点。（终点为邻接点时，不考虑自身）。

根据以上两个条件和邻接矩阵、可达矩阵，我们得到各基本块之间的控制依赖关系矩阵 $\mathbf{K} = \{k_{i,j}\}$。

当 $k_{i,j} = 0$ 时，节点 i 与节点 j 之间有控制依赖关系，即基本块 i 的流水线级数必须小于等于基本块 j 的流水线级数。

2、判断基本块之间是否存在数据依赖

数据依赖是 P4 程序的语句或代码块间由于数据的特定流向所造成的一种约束。具体来说，当 P4 程序的两个基本块中的代码段 S1 和 S2 先后执行时，可能会出现三种数据依赖，包括“读后写”、“写后读”、“写后写”。当存在“读后写”的数据依赖时，S1 所在的基本块排布的流水线级数要小于或等于 S2 所在的基本块排布的级数。而存在“写后读”、“写后写”的数据依赖时，S1 所在的基本块排布的流水线级数要严格小于 S2 所在的基本块排布的级数。因此，我们需要判断基本块之间的三种不同数据依赖。

对“attachment2.csv”文件进行处理，将其分为读取变量文件，和写入变量文件，再构建各基本块对于每个变量读写关系的字典，称为读写关系字典。

读写关系字典定义：设 **dict**= {′key′ : value} 是一个读写关系的字典，它的键“key”表示基本块号，值表示该基本块读写的变量名。

根据上节计算出的可达矩阵，我们可以获得任意两个可达点中代码段执行的先后顺序，结合读写关系字典，我们可以计算出读写关系矩阵。

控制依赖	读后写	写后写	写后读	目标决策变量
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1
1	1	0	0	0
1	0	1	1	1
1	0	1	0	1
1	0	0	1	1
1	0	0	0	0
0	1	1	1	1
0	1	1	0	1
0	1	0	1	1
0	1	0	0	0
0	0	1	1	1
0	0	1	0	1
0	0	0	1	1
0	0	0	0	0

表 1: 决策关系矩阵

读写关系矩阵定义：n 阶 01 方阵 $\mathbf{W_w} = \left(w_{ij}^{w}\right)$，$\mathbf{W_r} = \left(w_{ij}^{r}\right)$，$\mathbf{R_w} = \left(r_{ij}^{w}\right)$，其中 $w_{ij}^{w} = 1$ 表示基本块 i 和基本块 j 之间存在“写后写”数据依赖，$w_{ij}^{r} = 1$ 表示基本块 i 和基本块 j 之间存在“写后读”数据依赖，$r_{ij}^{w} = 1$ 表示基本块 i 和基本块 j 之间存在“读后写”数据依赖。

根据获取的三种读写关系矩阵，我们可以很容易得到任意两个基本块根据数据依赖得到的流水线排布级数的关系：

（1）当 $r_{ij}^{w} = 1$ 时，基本块 i 的流水线级数小于等于基本块 j 的流水线级数。

（2）当 $w_{ij}^{w} = 1$ 或 $w_{ij}^{r} = 1$，基本块 i 的流水线级数必须严格小于基本块 j 的流水线级数。

根据图论理论和题目给出的各基本块读写的变量信息、以及各基本块在流图中的邻接基本块，可以计算出任意两个基本块之间的同级依赖和上级依赖。将题目附件中“Attachement.2”和“Attachement.3”构建成四张有向无环图，分别是控制依赖关系图、“读后写”依赖关系图、“写后写”依赖关系图、“写后读”依赖关系图，分别表示基本块之间的依赖关系。如图所示，图 $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ 中，节点 a 指向节点 b，则表示为节点 a 对节点 b 有依赖关系。通过依赖图，我们可以表示出任意两个基本块的依赖关系以及流水线上下级关系。对图 $\mathbf{G}$ 进一步处理，可以获得依赖关系矩阵 $\mathbf{K} = (k_{ij})$，当 $k_{ij} = 0$ 时，表示节点 i 与节点 j 之间存在依赖关系。

全面考虑控制依赖矩阵及数据依赖中的读写依赖约束、读后写依赖约束、写后读依赖约束、写后写依赖约束，将非线性约束的逻辑关系表示如表 1所示。

通过逻辑合并，本节生成决策关系矩阵。

3.2.3 排布优先度构建

明确基本块之间的依赖关系后，即可利用决策关系矩阵排布基本块。但由于在排布基本块的算法中可能存在有多种备选基本块的情况，因此需要设计准则选择优选基本块。为此，本节建立了基本块数据关系的有向无环结构图 DAG，利用图论基本结论对基本块的等级进行排序，给出了基本块的等级关系，为优选基本块提供依据。

在图论中，拓扑排序（Topological Sorting）是一个有向无环图（DAG, Directed Acyclic

Graph）的所有顶点的线性序列。且该序列必须满足下面两个条件：

（1）每个顶点出现且只出现一次。

（2）若存在一条从顶点 A 到顶点 B 的路径，那么在序列中顶点 A 出现在顶点 B 的前面。

有向无环图（DAG）才有拓扑排序，非 DAG 图没有拓扑排序一说。

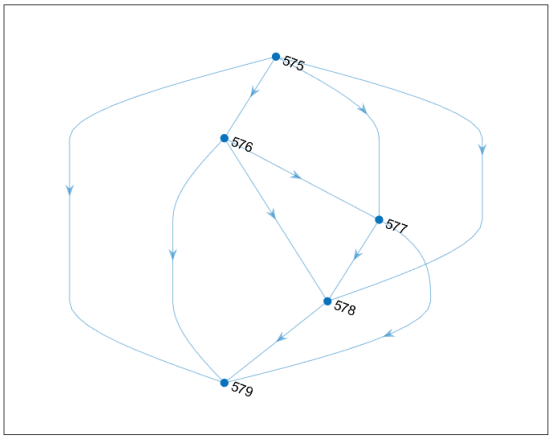

图 3: 流水线的部分子图

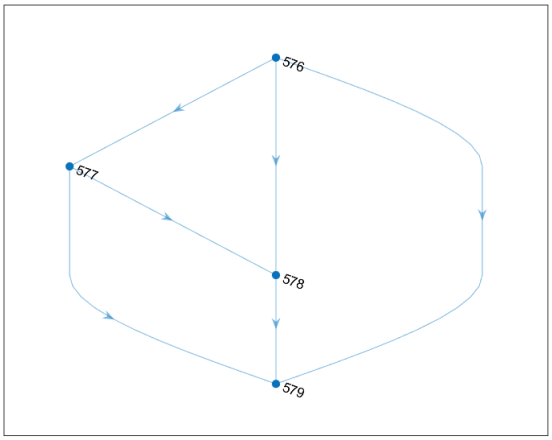

图 4: 去掉起点 575 后的拓扑关系图

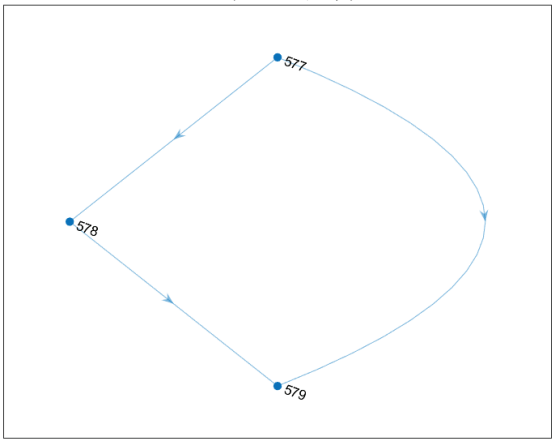

图 5: 去掉顶点 576 后的拓扑关系图

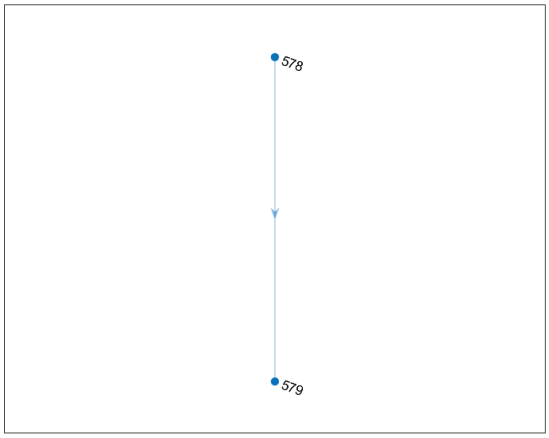

图 6: 去掉顶点 577 后的拓扑关系图

从原始流水线映射成的有向无环图中选取部分子图如图 3所示。拓扑排序过程如下：

1、从图中选择一个即入度为 0 的顶点并输出，所以第 1 级为顶点 575。2、从图中删除顶点 575 和所有以它为起点的有向边。3、重复 1 和 2 直到当前的 DAG 图为空或当前图中不存在无入度的顶点为止。于是，得到拓扑排序后的结果是 575，576，577，578，579。DAG 的拓扑结构变化如图 4、5、6所示。

由于可能同时出现多个入度为 0 的顶点，此时随机选择一个顶点作为当前等级的结果。因此，一个有向无环图可以有一个或多个拓扑排序序列。

参照拓扑排序的思想，我们计算一个等级列表，此列表同一级别允许包含多个顶点。也就是当同时出现多个入度为 0 的顶点时，不再随机选择一个顶点，而是将它们放入同一等级。算法总结见算法 1。

Algorithm 1 基本块等级列表计算算法

Input: 决策矩阵 J, 基本块集合 N
Output: 等级列表 L

```
i=0
while length(N) do
    根据决策矩阵计算 N 中各个基本块的出入度。
    将入度为 0 的基本块放入第 i 级等级列表。
    从 N 中删除入度为 0 的基本块。
    i=i+1
end while
```

在排布基本块时，由于控制依赖和读写依赖的存在，某些基本块的流水线等级必须小于其他基本块。也就是说在完成 P4 程序的过程中，必须按照等级列表的先后顺序排布基本块。

3.2.4 基于粒子优先度更新的基本块优选算法

为实现从集合 T 中选择可能对后续节点决策约束最小的节点作为优选节点，定义了优选度 y_i 作为选择指标，给出了基于优先度的粒子适应度更新的基本块优选算法，将具有最小优先度的基本块作为优选块。粒子的优先度表征了其被选择的紧迫性。由于在控制图中随节点减少，节点的边只会减少而不会增加，因此，我们给出定理 1：

Theorem 1. 优先度严格非增定理：$y_i^{t+1} \leqslant y_i^t$.

Proof. 根据子图的定义，若令图 G' 为 G 的子图，即 $G' \in G$，则必有子图的边的个数 $V' \in V$[8], 其中 t 表示第 t 表示第 t 次迭代次数。

因此，随着算法迭代，节点 n_i 的适应度必然严格非增。即适应度值的更新具有固定的方向，但速度未知。因此在将优先度作为适应度指标下，算法必然会选择当前最优节点。基本块粒子优先度更新算法如下：

Algorithm 2 基本块优先度更新算法

Input: 决策矩阵 J, 基本块集合 N, 已排基本块集合 Np
Output: 基本块优先度

```
i=0
从集合 N 中删除 Np 中的元素，生成集合 N'
while length(N') do
    根据决策矩阵计算 N' 中各个基本块的出入度。
    将入度为 0 的基本块放入第 i 级等级列表。
    从 N 中删除入度为 0 的基本块。
    i=i+1
end while
return 基本块优先度
```

3.2.5 基于模拟退火的全局最优保护机制

在基于粒子优先度更新的基本块优选算法中，每次迭代都能够选择局部最优基本块，但是在芯片资源排布的连续优化问题中，可能会导致陷入局部最优解。为解决上述问题，本

节引入了基于模拟退火的基本块优选保护策略，使得所提出的算法 CRSA 具有跳出局部最优陷阱的能力，有利于提高求得全局最优解的可靠性。算法流程如下：

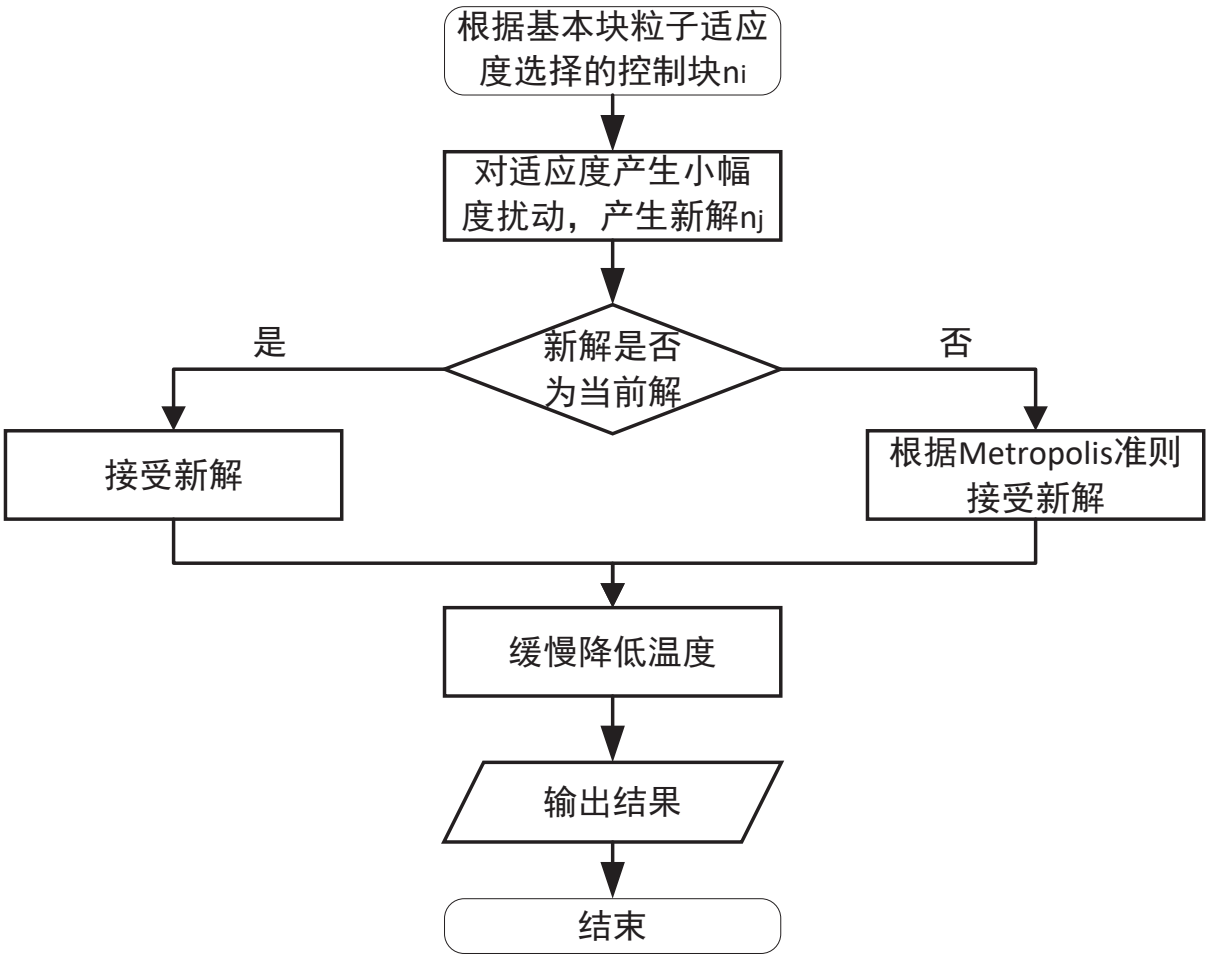

图 7: 模拟退火算法流程图。

令初始温度 T_0 为 1000，退火衰减速率为 $\lambda = 0.98$，采用指数式下降，则在第 $t+1$ 次基本块选择，当前温度为 $T^{t+1} = \lambda T^t$，令 a 为服从均匀分布 $a \sim U(0,1)$，退火准则为：

$$p = \exp\left(-\frac{T^{iter}}{T^t}\right) \tag{3.20}$$

其中 T_s^{iter} 为一个与实验次数 $iter$ 相关的数，在每次实验中为常数，随着试验次数的增加而变大，初始值 $T_s^0 = 1$，$T_s^{iter+1} = 1.005 * T_s^{iter}$。

3.3 模型求解

实验 1 结果：将最大迭代次数设置为 100，执行 CRSA1 算法，仿真结果如下所示：

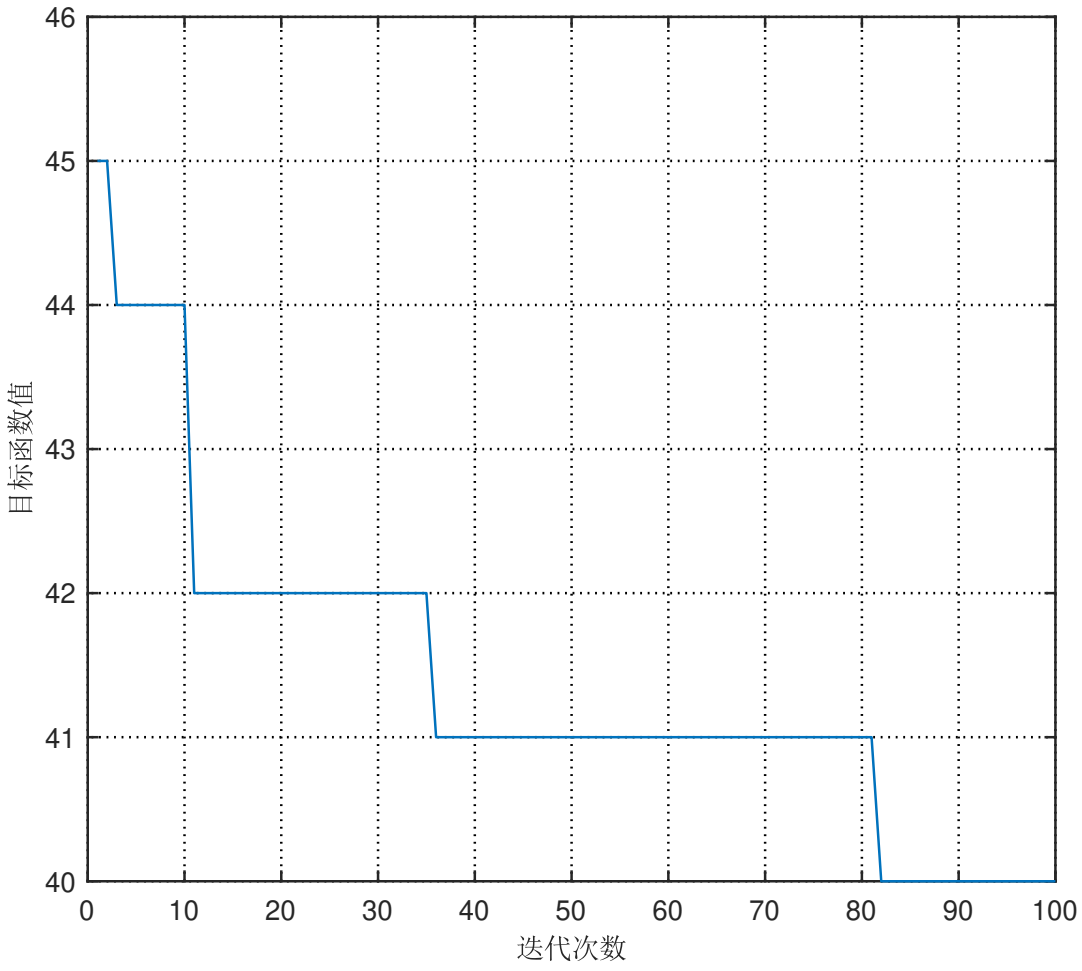

图 8: CRSA1 性能曲线

从图中可以看到，CRSA1 的最佳性能在 30 次迭代后即迅速下降到 40 级次优解。在 82 次迭代后即寻找到一个全局最优解 39 级。虽然所提出的 CRSA1 算法使用了基于启发式搜索算法，存在迭代不稳定的情况，但是与解空间共 2^{607} 中组合情况相比，CRSA1 仅通过不到 100 次迭代就实现了接近最优的解的搜索，性能非常惊人。

在此基础上，计算了此时资源排布的资源利用率，表示如下表:

TCAM	HASH	ALU	QUALIFY
0.5833	0.9844	0.6004	0.2035

表 2: 四种资源的利用率

可以看到，在该排布方案下 HASH 资源的利用率达到了 98.44%。证明所找到的解已经是或非常接近最优解。

3.4 资源排布图

在上述资源排布算法下输出的基本块排布结果如下:

流水线级数	分配的基本块编号														
0	1	2	3	6	9	11	12	13	15	16	17	18	20	22	24
0	26	27	30	32	34	36	37	38	40	42	44	46	48	50	52
0	53	55	59	60	63	65	68	70	73	79	82	87	92	94	95
0	96	106	110	114	118	122	125	129	132	133	134	137	138	141	145
0	146	148	151	154	155	157	159	161	163	165	166	171	176	179	182
0	185	186	188	190	192	195	197	202	205	208	210	212	218	221	226
0	229	231	238	239	240	242	244	246	248	249	252	255	259	262	264
0	266	269	272	275	278	281	282	287	290	292	305	308	310	312	313
0	317	320	322	327	334	335	337	339	342	345	347	349	352	353	354
0	357	361	363	365	368	371	372	373	379	381	384	389	392	393	394
0	396	398	400	402	406	409	412	413	419	421	422	425	427	430	436
0	437	439	442	448	451	454	457	459	468	471	474	477	479	481	484
0	487	490	493	496	497	500	501	503	507	511	514	517	520	523	524
0	525	527	528	533	536	539	540	543	547	548	551	552	556	558	561
0	565	566	568	569	572	576	577	578	580	582	584	587	590	592	596
0	598	601	605												
1	14	19	58	135	136	152	209	211	325	358	364	377	391	410	509
1	512	530	542	571	599	600									
2	147	158	169	170	172	336	370	378	382	385	386	387	390	405	515
2	518	529													
3	21	144	153	162	376	380	383	388	467	508	550	573	575		
4	139	149	150	160	374	375	447	567	595	602					
5	28	156	164	213	362	505	506	510	513	516	521	522	526	531	532
5	538	544	545	555	557	559	560	562	563	564					
6	519	553	554	583											
7	51	140	214	541	570	579	581	585	597						
8	33	207	230	260	284	302	311	431	432	475	499	546	549	574	603
9	61	143	216	219	223	404	407	534							
10	54	71	142	167	215	220	222	224	225	227	294	295	300	301	
10	314	315	323	324	326	330	331	338	340	452	535	537			
11	56	62	64	168	187	189	199	200	201	217	232	250	253	256	296
11	299	303	356	367	395	397	399	401	502	504					
12	39	285	286	289	291	298	306	307	316	319	360				
13	194	293	304												
14	49	196	206	233	247	251	254	297	309	318	469	473	483	486	489
14	492	495	498												
15	341	343	485	488											
16	193	472													
17	4	31	480												
18	478	482													
19	491														
20	191	198	203	204	228										
21	234	235	236	237	241	243	245	258	268	271	274	277	280	283	288

表 3: 芯片资源排布方案

流水线级数	分配的基本块编号														
21	403														
22	265														
23	470	476													
24	270	494													
25	257	267													
26	273														
27	276														
28	261														
29	263	321	411	434	435	440	441	453	455	461	462				
30	279	408	428	438	449	456	463	464	594						
31	72	174	177	180	183	414	423	429	433	443	444	445	446	450	458
31	460	465	466	593											
32	74	111	119	126	173	415	416	417	418	420	424	426	604		
33	41	103	127	175	178	181	369								
34	35	43	45	47	102	107	115	328	329	332	366	606			
35	5	29	66	67	75	76	104	120	123	333	344	350	355		
36	7	80	86	91	108	112	113	116	117	121	124	184	348		
37	0	8	10	69	78	81	99	100	105	109	128	351	359	586	588
37	589	591													
38	25	57	85	93	97	346									
39	23	77	83	84	88	89	90	98	101	130	131				

表 4: 芯片资源排布方案（接上表）

3.5 小结

本章建立了芯片资源优化问题及其对偶问题，分析了排布顺序的影响，确定了正向排布顺序，以控制依赖和数据依赖为基础，基于逻辑合并构造了决策矩阵，建立了决策约束排布规则，确保求解结果在可行域内，基于决策矩阵的拓扑排序定义了优先度指标，给出了基于粒子优先度更新的基本块优选算法，为避免局部最优，设计了基于模拟退火的全局最优保护机制，实现了问题 1 的目标，在满足约束的条件下，流水线技术可达 39 级，四种资源的最大利用率最大，TCAM 资源利用率可达 58.33%，HASH 资源利用率可达 98.44%，ALU 资源的利用率可达 60.04%，QUALIFY 资源的利用率可达 20.35%。因此，可认为该排布方案即为最优资源排布方案。

4. 问题 2：共享资源条件下的芯片资源排布算法设计

本章在问题 2 的条件下, 使用深度优先搜索算法 (DFS) 重新计算每级流水线 HASH 和 ALU 资源占用量，确定了新的资源约束判定方法，以占用的流水线最短为优化目标，设计了基于模拟退火和粒子群混合优化的 CRSA2 算法，实现了问题 2 的目标。

4.1 问题分析与模型建立

问题 2 需要在满足资源约束、数据依赖、以及控制依赖的条件下，以占用的流水线级数尽量短为目标，给出资源排布算法。

问题 2 与问题 1 的关键区别在于流水线每级占用的 HASH 和 ALU 资源计算方式的不同。由于不在一条执行流程上的基本块，可以共享 HASH 资源和 ALU 资源。因此本级流水线所占用的资源为占用资源最多的执行流程所占用的资源。由于共享资源的存在，流水线每级可排布的基本块增多。

研究问题 2 的约束条件及优化目标，分析解决问题 2 需要修改 HASH 和 ALU 资源占用量计算算法，将原始流水线映射为有向无环图，根据顶点的出入度，确定流水线每级所包含顶点中的起点和终点结合。计算起点和终点之间的路径数量，也就是本级流水线所包含的执行流程数目。计算每条执行流程所占用的资源，取它们的最大值作为本级流水线占用的资源。

在设计问题 2 资源计算方式、构建问题 2 资源判定逻辑后，可参照算法 CRSA1，根据决策约束派不规则，基于粒子群与模拟退火混合的优化方法设计问题 2 的资源排布规则。

此时优化目标仍然为：

$$(P1) : \min(m) \tag{4.1}$$

某一级流水线的基本块的执行流程关系为 $Z_i = \{z_{i,p,q}\}, (i = 0, 1, ...m - 1)$，其中 $q = 1, 2, 3..., Q$，表示本级流水线包含 Q 条执行流程。$z_{i,p,q} = 1$ 表示基本块 q 在第 i 级流水线第 p 条执行流程上。

约束条件（1）（4）（6）（7）保持不变，仍为 3.4,3.7,3.10,3.11，其余变为：

$$\max(\sum_{j=0}^{n-1} z_{i,p,j} c_{j,2}) \leqslant 2,\ i = 0, 1, ..., m - 1 \tag{4.2}$$

其中 $q = 1, 2, 3..., Q$。

$$\max(\sum_{j=0}^{n-1} z_{i,p,j} c_{j,3}) \leqslant 56,\ i = 0, 1, ..., m - 1 \tag{4.3}$$

其中 $q = 1, 2, 3..., Q$。

$$\max(\sum_{j=0}^{n-1} z_{i,q,j} c_{j,2}) + \max(\sum_{j=1}^{n} z_{(i+16),q,j} c_{j,2}) \leqslant 1,\ i = 0, 1, ..., 15 \tag{4.4}$$

其中 $q = 1, 2, 3..., Q$。

公式 4.2表示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；公式 4.3表示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；公式 4.4表示折叠的两级，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。

4.2 问题 2 资源计算算法

由于不在一条执行流程上的基本块，可以共享 HASH 和 ALU 资源，因此问题 2 的关键在于流水线本级中一共包含多少条执行流程。每条执行流程上的基本块消耗的 HASH 和 ALU 资源总和即为本条执行流程消耗资源。消耗资源最多的执行流程所使用的 HASH 资源和 ALU 资源也就是本级流水线所消耗的资源。

4.2.1 HASH 资源计算算法

结合第一问的思路，我们根据“attacment3.csv”将流程线映射为有向无环图，基本块为顶点，基本块之间的指向关系为边。入度指有向图中某点作为图中边的终点的次数之和。出度指有向图中某点作为图中边的起点的次数之和。设本级流水线包含的顶点构成集合 **V**。通过计算流水线本级包含的顶点的出入度，可以确定执行流程的起点和终点。依据第一问得出的邻接矩阵和可达矩阵，确定本级包含顶点之间的关系。此时只有弧线两端顶点均在本级流水线中，才算有效出入度。入度为 0 的顶点构成起点集合 **S**，出度为 0 的顶点构成终点集合 **E**。然而，流水线本级包含的某些节点可能独自在一条执行流程上，此时它的出入度均为 0。我们将此类顶点定义为本级的“离散点”。这些离散点所在执行流程所消耗的资源也就是单个顶点所消耗的资源。起点和终点的交集就是离散点的集合。我们把离散点从起点和终点集合中去除。

接下来的关键就是确定起点和终点之间路径的条数。深度优先搜索算法（Depth First Search，简称 DFS）：一种用于遍历或搜索树或图的算法。沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点 v 的所在边都已被探寻过或者在搜寻时结点不满足条件，搜索将回溯到发现节点 v 的那条边的起始节点。整个进程反复进行直到所有节点都被访问为止。算法的核心为探索与回溯。回溯法（探索与回溯法）是一种选优搜索法，又称为试探法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。我们从原始流水线映射成的有向无环图中选取部分子图如图**??**所示。在图 3中，起点集合 **S** = {v_{575}}, 终点集合 **E** = {v_{579}}。

假设某级流水线中包含的基本块为图 3中的节点，那么我们使用 DFS 算法来确定节点 575 与节点 579 之间的路径数即执行流程数。

1. 从 v_{575} 出发，将 v_{575} 标记，并将其入栈。

2. 找到 v_{579}，将其标记并入栈。v_{579} 是终点，将栈中的元素从栈底往栈顶输出，即为一条路径（v_{575}—>v_{579}）。v_{579} 出栈，并取消标记，回溯到 v_{575}。

3. 找到 v_{576}，将其标记并入栈。继续找到 v_{579}，将其标记并入栈。v_{579} 是终点，将栈中的元素从栈底往栈顶输出，即为一条路径（v_{575}—>v_{576}—>v_{579}）。v_{579} 出栈，并取消标记，回溯到 v_{576}。

4. 找到 v_{578}，将其标记并入栈。继续找到 v_{579}，将其标记并入栈。v_{579} 是终点，将栈中的元素从栈底往栈顶输出，即为一条路径（v_{575}—>v_{576}—>v_{578}—>v_{579}）。v_{579} 出栈，并取消标记，回溯到 v_{578}。

5. v_{578} 除 v_{579} 外没有其他出度，v_{578} 出栈，并取消标记，回溯到 v_{576}。找到 v_{577}，将其标记并入栈。继续找到 v_{578}，将其标记并入栈。继续找到 v_{579}，将其标记并入栈。v_{579} 是终点，将栈中的元素从栈底往栈顶输出，即为一条路径（v_{575}—>v_{576}—>v_{577}—>v_{578}—>v_{579}）。v_{579} 出栈，并取消标记，回溯到 v_{578}。

……

当回溯到 v_{575} 且无新的邻接点时，栈空，结束遍历。得到图**??**中共有 8 条路径。

在确定路径时依据的是总流水线映射成的邻接矩阵和可达矩阵，也就是说，路径上的某些顶点可能并不在本级流水线中。因此，确定路径后，需要将路径上不在本级流水线上的节点去除，之后剩余节点占用资源相加，即为此条执行流程占用的资源。

本级流水线占用的 HASH 资源计算方法如算法 3所示。

Algorithm 3 本级流水线 HASH 资源占用量计算算法

Input: 决策矩阵 J、排布矩阵 D、流水线等级 level

Output: 本级流水线 HASH 资源占用量

1: 根据决策矩阵 **J** 确定流水线本级包含的顶点集合 $\mathbf{V} = \{v_i, v_j, \ldots, v_m\}$。
2: 初始化占用的 HASH 资源 $hash\_used = \max(hash\_cost(v_i))$，$i \in \mathbf{V}$，$hash\_cost(v_i)$ 表示顶点 v_i 所占用的 HASH 资源。
3: 计算本级顶点之间的出入度，入度为 0 的点构建为起点集合，出度为 0 的点构建为终点集合。
4: 计算起点集合和终点集合的交集得到离散点集合，并删除起点集合和终点集合的离散点。
5: 起点和终点两两组合，使用 DFS 方法确认它们之间的路径数，得到本级流水线包含的执行流程总数 N。
6: 对于每一条路径，去除不在本级流水线上的顶点后 < 对所有顶点占用的 HASH 资源求和，即为此条执行流程占用的 HASH 资源 $hash\_path(i), i = 1, 2, \ldots, N$。
7: 确定本级流水线占用的 HASH 资源 $hash\_used = Max(hash\_path(i), hash\_used)$(i=1,2,...,N)。

4.2.2 ALU 资源计算算法

ALU 资源与 HASH 资源的区别在于流水线的折叠级数对于 ALU 资源占用量无要求。因此 ALU 资源的共享仅导致约束条件（3）的变化。ALU 资源占用量计算算法如算法 4所示。

Algorithm 4 本级流水线 ALU 资源占用量计算算法

Input: 决策矩阵 J、排布矩阵 D、流水线等级 level

Output: 本级流水线 HASH 资源占用量

1: 根据决策矩阵 **J** 确定流水线本级包含的顶点集合 $\mathbf{V} = \{v_i, v_j, \ldots, v_m\}$。
2: 初始化占用的 ALU 资源 $alu\_used = \max(alu\_cost(v_i))$，$i \in \mathbf{V}$，$alu\_cost(v_i)$ 表示顶点 v_i 所占用的 ALU 资源。
3: 计算本级顶点之间的出入度，入度为 0 的点构建为起点集合，出度为 0 的点构建为终点集合。
4: 计算起点集合和终点集合的交集得到离散点集合，并删除起点集合和终点集合的离散点。
5: 起点和终点两两组合，使用 DFS 方法确认它们之间的路径数，得到本级流水线包含的执行流程总数 N。
6: 对于每一条路径，去除不在本级流水线上的顶点后，对所有顶点占用的 ALU 资源求和，即为此条执行流程占用的 ALU 资源 $alu\_path(i), i = 1, 2, \ldots, N$。
7: 确定本级流水线占用的 ALU 资源 $alu\_used = Max(alu\_path(i), alu\_used)$(i=1,2,...,N)。

4.3 CRSA2 算法流程图。

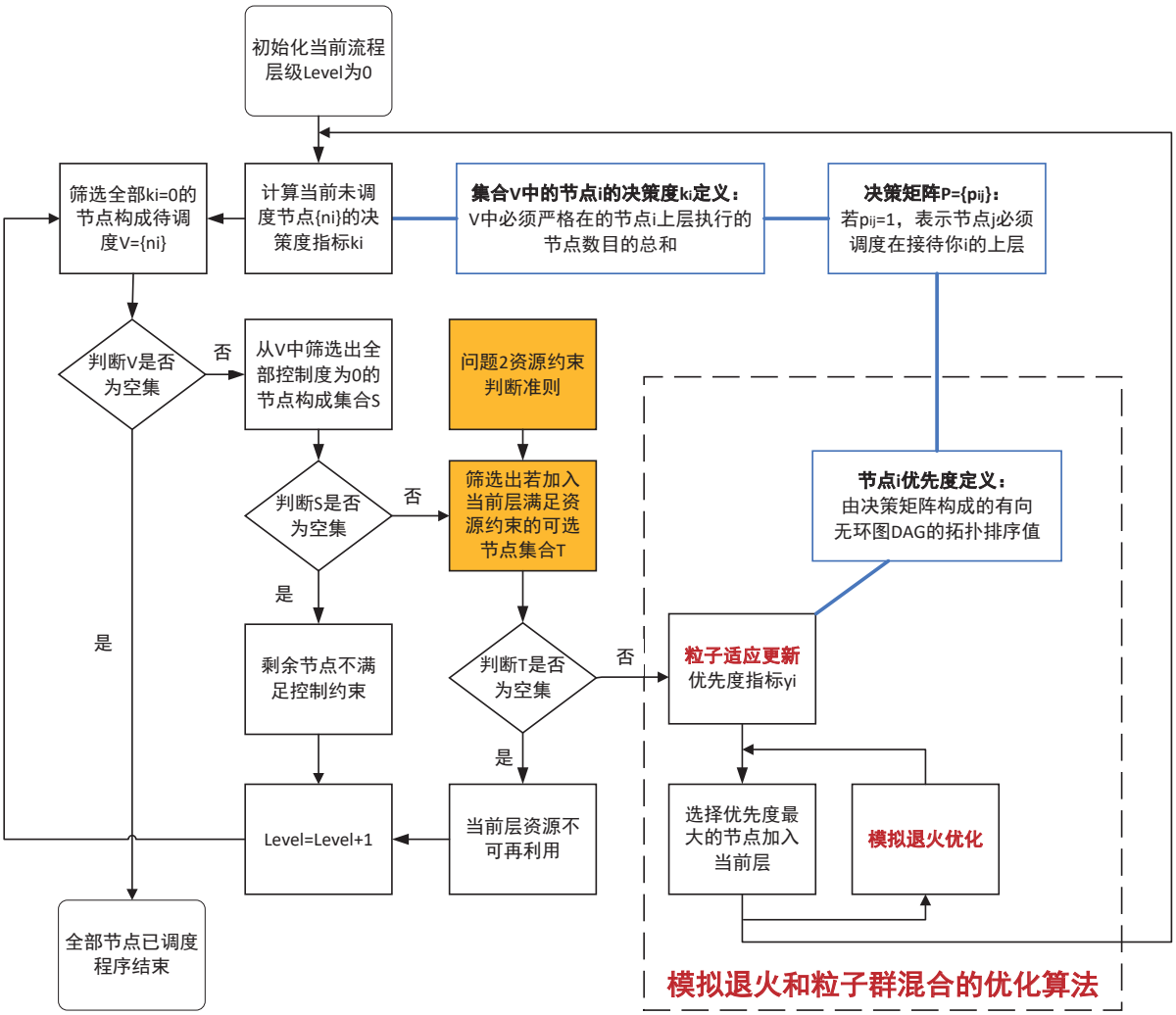

图 9: CRSA2

4.4 模型求解

将最大迭代次数设置为 100，执行 CRSA2 算法，仿真结果如下所示：

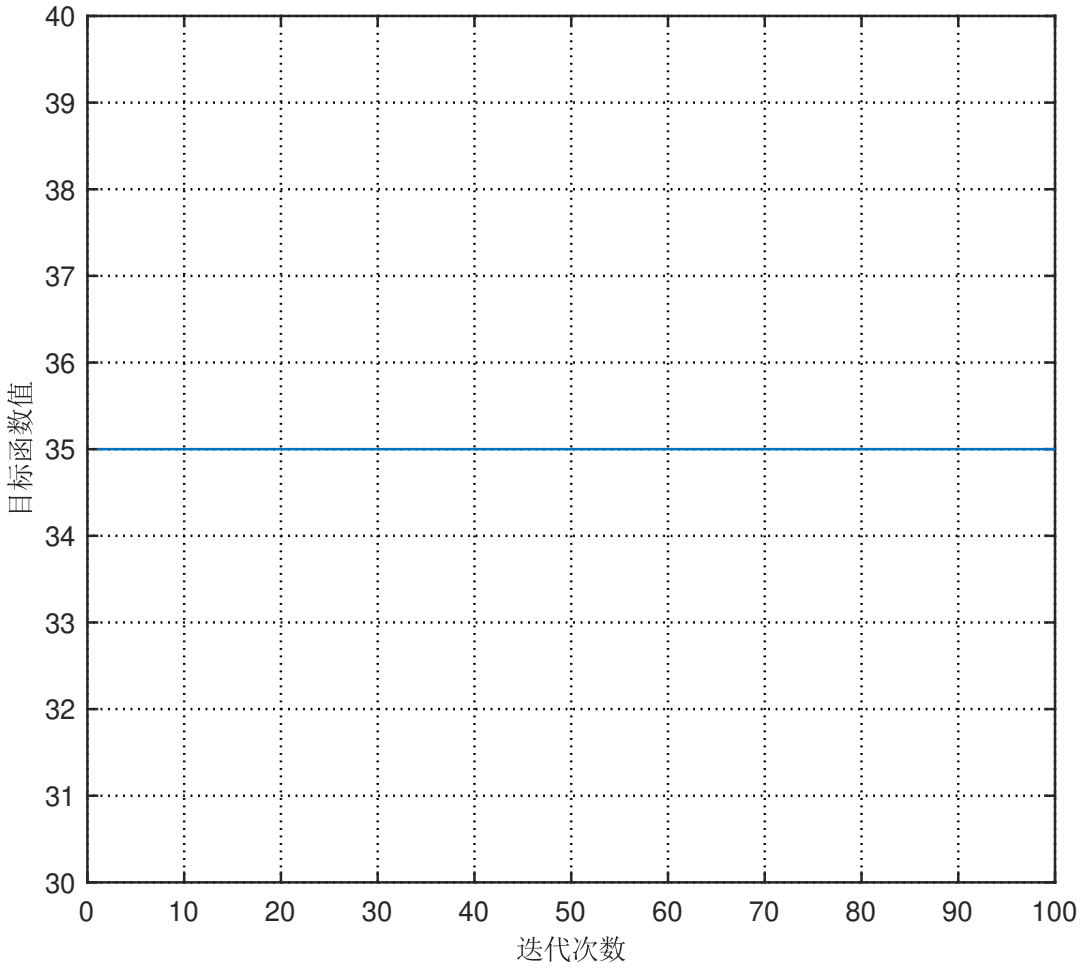

图 10: CRSA2 性能曲线

从图中可以看到，CRSA2 的性能保持在 34 层，没有得到有效的性能下降，仅得到了众多非劣解。由于无法确定 34 层是否是最优解，本文分析了所得到的众多非劣解的资源利用率，表示如表 5。

TCAM	HASH	ALU	QUALIFY
0.4588	0.7634	0.7324	0.2437

表 5: 四种资源各级流水线的平均利用率

4.5 资源排布图

由于仅得到众多非劣解，定义在共享资源约束下，资源利用率为各级流水线 HASH 资源利用率均值。因此，在 100 次迭代获得的非劣解中，选取部分方案进行比较，获得非劣解中资源利用率最高的解作为资源排布结果。

流水线级数	分配的基本块编号														
0	11	13	14	15	16	17	18	19	20	21	22	37	38	58	59
0	132	133	134	135	138	145	146	148	149	151	152	154	155	157	159
0	161	163	165	169	170	171	172	363	364	365	370	371	372	373	374
0	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389
0	390	391	392	393	530										
1	12	27	28	136	137	139	144	147	150	153	156	158	160	162	164
1	361	362	505	506	507	508	509	510	511	512	513	514	515	516	517
1	518	519	520	521	522	523	524	525	526	527	528	529	531	532	533
1	534	536	538	539	540	542	543	544	545	547	548	550	551	552	554
1	556	557	558	559	560	561	562	563	565	566	582	584			
2	553	564	567	568	569	581	583	585	601						
3	555	595	596	602											
4	541	570	571	572	573	574	575	576	577	578	579	597	598	599	600
4	603														
5	535	537													
6	3	32	36	40	42	44	46	51	52	53	54	55	140	141	142
6	143	166	167	188	211	212	213	214	215	216	217	218	219	220	221
6	222	223	224	225	226	227	229	230	231	296	301	302	305	308	310
6	311	312	313	314	315	323	324	325	326	327	330	331	394	396	398
6	400	546	549												
7	33	34	168	199	201	232	503	504							
8	56	187	189	190	192	195	197	200	303	306	395	397	399	401	404
8	405	406	407												
9	4	31	35	39	41	43	45	47	48	49	50	60	61	62	63
9	64	191	193	194	196	198	202	203	204	208	209	210	228	293	294
9	295	297	298	299	300	328	329	332	333	334	335	336	337	338	339
9	340	356	357	358	359	366	367	368	369	469	471	474	475	477	479
9	481	484	487	490	493	496	497	499	500	501	502				
10	307	360	470	472	473	476	478	480	482	483	485	486	488	489	491
10	492	494	495	498											
11	304														
12															
13	65	205	206	207	233	234	235	236	237	238	239	240	241	242	243
13	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258
13	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273
13	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288
13	289	290	291	292	309	316	317	318	319	320	321	322	342	402	403
13	434	437	450	451	452	454	455	456	580						
14	66	67	412	435	436	438	439	440	441	442	443	444	445	446	447
14	448	449	453	457	458	459	460	461	462	463	464	465	466	467	468
14	592	593	594												
15	341	343	408	409	410	411	413	414	415	416	418	419	420	421	422
15	423	424	425	426	427	428	429	430	431	432	433	586	587	588	589

表 6: 芯片资源排布方案

流水线级数	分配的基本块编号														
15	590	591													
16	29	30	68	71	72	73	127	173	174	176	177	179	180	182	183
16	185	186	417	604	605										
17	5	6	7	8	9	70	74	76	78	79	80	102	103	104	106
17	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121
17	122	123	124	125	126	128	175	178	181	184	606				
18	10	81	99	100	105										
19															
20															
21															
22															
23	69	75	344	345	346	347	348	349	350	351	352	353	354	355	
24															
25															
26															
27															
28															
29															
30															
31															
32															
33	0	1	2	23	24	25	26	57	77	82	83	84	85	86	87
33	88	89	90	92	94	95	96	98	101	129	130	131			
34	91	93	97												

表 7: 芯片资源排布方案 (接上表)

4.6 小结

本章在问题 2 的共享资源机制条件下, 使用深度优先搜索算法 (DFS) 计算了每级流水线包含的执行流程数，从而确定了每级流水线 HASH 和 ALU 资源占用量，构建了问题 2 的资源约束判定方法。设计了基于模拟退火和粒子群混合优化的 CRSA2 算法，以占用的流水线最短为优化目标，求解 CRSA2 算法，实现了问题 2 的目标。在满足约束的条件下，流水线级数可达 34 级，得到了众多非劣解。定义各级流水线 HASH 资源利用率均值为问题 2 的资源利用率，从众多非劣解中确定了最优芯片资源排布。

5. 模型的总结与评价

5.1 模型优点

1. 结合数学模型和计算机仿真模型，通过理论分析和仿真验证设计了 CRSA1 & 2 算法，结论可信度高。

2. 将芯片资源排布问题转化为其对偶问题，使得原问题能够求解。

3. 将两种高维非线性约束（控制依赖约束与数据依赖约束）逻辑合并为同一决策约束，能够有效降低算法复杂度。

4. 将粒子群优化算法与模拟退火算法的思想引入所提出的 CRSA1 & 2 算法中，使得寻找到的解保持在可行域内。同时，一般能够在多项式时间内找到最优解或次优解，解决芯片资源排布问题。

5. 当更换不同类别芯片时导致资源约束不同时，本文提出的 CRSA1 2 的核心算法逻辑无需发生改变，可移植性强。

5.2 模型缺点

1. 采用了启发式算法进行，有概率找到次优解。

2. 定义排布优先度作为基本块粒子适应度，计算复杂度较高。

5.3 改进方向

1. 后续可考虑将禁忌搜索引入本文所提出的芯片资源排布算法。

2. 本文设置的模拟退火机制采用了指数式下降方式，后续可根据实际情况建立基于经验公式的退温函数。

3. 考虑假设 7 中的影响因素，进一步考虑并行度作为优化目标之一，重新设计优化目标。

参考文献

[1] 马艳利, 陈明秀. 一类混合整数规划模型的共享车调度优化算法研究 [J]. 兰州文理学院学报 (自然科学版),2022,36(01):1-5.

[2] 高一龙. 若干综合调度问题的智能优化算法研究 [D]. 哈尔滨理工大学.

[3] 赵宇. 多目标柔性车间生产调度系统设计与实现 [D]. 电子科技大学.

[4] 曹楠. 基于遗传模拟退火算法的 U 型装配线 E 类平衡研究 [D]. 吉林大学.

[5] 李雪溶. 基于模拟退火算法的应急物流体系建设 [J]. 科技与创新,2022(13):62-64+68.

[6] 曹鹏, 刘敏. 基于改进的整数规划法结合零注入节点的 PMU 优化配置方法 [J]. 电力系统保护与控制,2021,49(16).

[7] 王联国, 洪毅, 赵付青, 余冬梅. 一种模拟退火和粒子群混合优化算法 [J]. 计算机仿真,2008(11):179-182.

[8] Luenberger D G, Ye Y. Linear and nonlinear programming[M]. Reading, MA: Addison-wesley, 1984.

[9] Diestel R, 于青林, 王涛, 等. 图论 [J]. 2013.

附录 A CRSA1 算法主程序

```
Level=60;%设置最大流水线级数
lv_r=abs(lv)+1;
D=zeros(Level,607);
%维护一个节点的奖励值列表，每个节点选择他都会给这个节点一个奖励值，
%这样在从几个节点中进行选择时就不是随机选，而是选择奖励值最高的节点。
node_list=[]; %已放置节点
award_list=gengxin_list(node_list,matrix_kongzhi);
TK=1; 模拟退火
T=1;
lanbda=0.98;
lambda2=1.03;

%% 循环
iter=1e2%最大迭代次数
V0=100;%初始迭代值
record_list=zeros(1,iter);
D_opt=D;
for ittimes= 1:iter
    length(base_nodelist);

    %维护节点列表
    node_list=[]; %已放置节点
    d_dq_list=[];%当前级待放置节点
    d_up_list=[];%需要放置在上级的节点列表
    D=zeros(Level,607);
    level = 1;%当前层数
    while length(node_list)<607 &&(level<=Level)

        %fprintf('当前层数为%d\n',level)
        d_dq_list=findc1(node_list,matrix_kongzhi);%出度备选矩阵
        %进一步筛选列表中需要没有对未选列表中有任何控制
        d_dq_list=shaixuan(d_dq_list,node_list,matrix_kongzhi);
        if length(d_dq_list)~=0
            %          判断 每个节点加入后层l是否满足资源约束，满足的点
               构成列表list m
            list_m=ziyuan(D,d_dq_list,C,caps,level);
            %此时应该用粒子群算法从List_(m)中随机选择一个节点
            %为了程序测试，我们这里就使用第一个值
            if length(list_m)~=0
                %temp_id=list_m(randi([1,length(list_m)],1,1));
                %award_list=gengxin_list(node_list,matrix_kongzhi);
                temp_id=findmax_award(list_m,award_list);%寻找最大奖励
                   值节点%加入噪声
        if tuihuo(T,TK)
                                D(level,temp_id)=1;
                                node_list=[node_list temp_id];
        end
            else
```

```
                level=level+1;
                %d_dq_list=list_m;
            end
        else %如果列表为空，则当前层资源不能再用了
            level=level+1;

        end
        T=T*lamda;
    end
    if level < V0
        V0=level;
        D_opt=D;
        fprintf('当前最优值为%d\n',V0)
    end
    record_list(ittimes)=V0;
Tk=lambda2*Tk;
end
```

附录 B 入度筛选子函数

```
function d_dq_list=findc1(node_list,matrix_kongzhi)
K=matrix_kongzhi;
l=1:607;
K([node_list],:)=[];
l([node_list])=[];
K(:,[node_list])=[];
res=sum(K,1);
%find(res==0);%找到所有入度为0的矩阵，此时编号为在K里的编号
d_dq_list=l(find(res==0));
end
```

附录 C 基本块粒子适应度计算代码

```
award_list=zeros(1,607);

for i=1:607
 award_list(i)=gengxin_rank(i,matrix_kongzhi);
end
end

function A=gengxin_rank(id,matrix_kongzhi)
M0=matrix_kongzhi;
Res={};
res=[];
k=1;
while length(res)<607
    su=sum(M0,1);
```

```
    res=find(su==0);
    Res{k}=res;


    k=k+1;
    M0(res,:)=zeros(length(res),607);
    %M0(:,res)=zeros(607,length(res));
end

for j = 1 : length(Res)
    if length(find(Res{j}==id))
        A=j;
        break
    end

end
end
```

附录 D 问题 1 资源约束判定子函数

```
list_m=[];
for i=1:length(d_dq_list)
    D_temp=D;
    D_temp(level,d_dq_list(i))=1;
    if zy_value(D_temp,C,level,caps)==1
        list_m=[list_m d_dq_list(i)];
    end
end

end




function val=zy_value(D_temp,C,level,caps)
A=D_temp(level,:)*C;
val=(A(1) <= caps(1)) && (A(2) <= caps(2)) && (A(3) <= caps(3)) && (A
   (4) <= caps(4)) && (sum(D_temp(1:2:level,:)*C(:,1))<=5);

if level>=17 && level<=32
    B=D_temp(level-16,:)*C;
    val=(A(1) <= caps(1)) && (A(2) <= caps(2)) && (A(3) <= caps(3)) &&
       (A(4) <= caps(4)) && (sum(D_temp(1:2:level,:)*C(:,1))<=5) && ((A
       (1)+B(1)) <= 1) && ((A(2)+B(2)) <= 3);
end

%有TCAM资源的偶数级数量不超过5;

%sum(D_temp(1:2:level,:)*C(:,1))
```

```
    %约束1：流水线每级的TCAM资源最大为1；
    %约束2：流水线每级的HASH资源最大为2；
    %约束3：流水线每级的ALU资源最大为56；
    %约束4：流水线每级的QUALIFY资源最大为64；
    %约束5：折叠的两级TCAM资源加起来最大为1
    %约束6：折叠的两级HASH资源加起来最大为3
    %约束7：每个基本块只能排布到一级


end
```

附录 E 粒子优先度更新子函数

```
% CRSA1算法_粒子优先度更新子函数
function award_list=gengxin_list(node_list,matrix_kongzhi)
 award_list=zeros(1,607);

 for i=1:607
  award_list(i)=gengxin_rank(i,matrix_kongzhi);
 end
 end

function A=gengxin_rank(id,matrix_kongzhi)
M0=matrix_kongzhi;
Res={};
res=[];
k=1;
while length(res)<607
    su=sum(M0,1);
    res=find(su==0);
    Res{k}=res;


    k=k+1;
    M0(res,:)=zeros(length(res),607);
    %M0(:,res)=zeros(607,length(res));
end

for j = 1 : length(Res)
    if length(find(Res{j}==id))
        A=j;
        break
    end

end
end
```

附录 F 粒子优先度优选子函数

```
%寻找奖励值最大的节点
list_m_award=award_list(list_m)+randi([-10,10],[1,length(list_m)]);%增
    加随机扰动
[value,location]=max(list_m_award);
A=list_m(location);
end
```

附录 G 模拟退火判定子函数

```
A=0
a=rand();
if a<exp(TK/T)
        A=1;
end
end
```

附录 H 问题 2 HASH 资源计算子函数

```
load('adjoin_matrix.mat');
linjie=adjoin_matrix;
load('keda.mat');
keda=P;
load('C.mat');
c=C(:,2);
G=digraph(linjie);%有向图对象
index=find(D_temp(level,:)==1);
%% 计算出入度，确定起始点
chu=zeros(1,length(index));
ru=zeros(1,length(index));
for i=1:length(index)
    for j=1:length(index)
        if keda(index(i),index(j))==1
            chu(i)=chu(i)+1;
        end
        if keda(index(j),index(i))==1
            ru(i)=ru(i)+1;
        end
    end
end
qidian=index(find(ru==0));
zhongdian=index(find(chu==0));
use=max(c(index));
qidian1=qidian;
qidian=setdiff(qidian,zhongdian);
zhongdian=setdiff(zhongdian,qidian1);
```

```
for i=1:length(qidian)
    for j=1:length(zhongdian)
%         [TR,lujing]=shortestpathtree(g,qidian(i),zhongdian,'
   OutputForm','cell')
        path=allpaths(G,qidian(i),zhongdian(j));
        for path_i=1:length(path)
            node=intersect(path{path_i},index);%确定路径上属于本层的点
            use1=sum(c(node));
            use=max([use,use1]);
        end
    end
end
end
```

附录 I 问题 2 ALU 资源计算子函数

```
load('adjoin_matrix.mat');
linjie=adjoin_matrix;
load('keda.mat');
keda=P;
load('C.mat');
c=C(:,3);
G=digraph(linjie);%有向图对象
index=find(D_temp(level,:)==1);
chu=zeros(1,length(index));
ru=zeros(1,length(index));
for i=1:length(index)
    for j=1:length(index)
        if keda(index(i),index(j))==1
            chu(i)=chu(i)+1;
        end
        if keda(index(j),index(i))==1
            ru(i)=ru(i)+1;
        end
    end
end
qidian=index(find(ru==0));
zhongdian=index(find(chu==0));
use=max(c(index));
qidian1=qidian;
qidian=setdiff(qidian,zhongdian);
zhongdian=setdiff(zhongdian,qidian1);
for i=1:length(qidian)
    for j=1:length(zhongdian)
%         [TR,lujing]=shortestpathtree(g,qidian(i),zhongdian,'
   OutputForm','cell')
        path=allpaths(G,qidian(i),zhongdian(j));
        for path_i=1:length(path)
            node=intersect(path{path_i},index);%确定路径上属于本层的点
```

```
            use1=sum(c(node));
            use=max([use,use1]);
        end
    end
end
end
```

附录 J 问题 2 资源约束判定子函数

```
A=D_temp(level,:)*C;
A(2)=hash(D_temp,level);
A(3)=alu(D_temp,level);
val=(A(1) <= caps(1)) && (A(2) <= caps(2)) && (A(3) <= caps(3)) && (A
   (4) <= caps(4)) && (sum(D_temp(1:2:level,:)*C(:,1))<=5);

if level>=17 && level<=32
    B=D_temp(level-16,:)*C;
    hash1=hash(D_temp,level-16);
    val=(A(1) <= caps(1)) && (A(2) <= caps(2)) && (A(3) <= caps(3)) &&
       (A(4) <= caps(4)) && (sum(D_temp(1:2:level,:)*C(:,1))<=5) && ((A
       (1)+B(1)) <= 1) && ((A(2)+hash1) <= 3)
end

%有TCAM资源的偶数级数量不超过5;

%sum(D_temp(1:2:level,:)*C(:,1))



end
```