基于贪心的邻域搜索在 PFSP 中的综合应用

武稔隽 (学号: 1120203529)

摘　要: 基于现代化工业生产常遇的生产问题，抽象出置换流水车间模型，即求解最优组合使得最大完成生产时间跨度达到最小值，从而最优化生产计划。给出了多种经典算法的实现，并在朴素的智能算法之上，提出了一种基于贪心策略的邻域搜索算法模块以增强传统算法，并从理论与实践的角度证明了其可行性与可靠性。通过数据可视化的角度，展示了这种增强算法的绝对优势，清晰明了的对比各种算法的优缺。

关键词: 启发式搜索 智能算法 贪心策略 邻域搜索

1 引言

1.1 问题背景介绍

现代工业的生产规模与日俱增，加工精细程度逐渐增强，因而工作的规模与复杂度几何式增长。而流水车间调度问题(FSP, Flow-Shop Scheduling Problem)是众多工作调度问题(JSP, Job Scheduling Problem)中最典型的一类问题。在 FSP 问题中，又属置换流水车间问题(PFSP, Permutation FSP)最为经典，也是最容易建模并且求解的一类组合优化问题，因此具有重要的研究价值。

1.2 数学语言建模

在 PFSP 中，调度方案用一组工序的排列来表示，假设一个 PFSP 问题中一共有 n 道工序，那么其对应的调度方案是：

$$\text{Solution} \leftrightarrow \pi = \{\pi_1, \pi_2, \ldots, \pi_n\}.$$

上述转化也成为译码/解码(Encoding/Decoding)转换。在获取了调度的数学符号表达方式后，假设该问题对应的最优解为：

$$\pi^* = \{\pi_1^*, \pi_2^*, \ldots, \pi_n^*\}.$$

最优解所代表的含义则是在给定约束下，使得整个工作的最大完成时间最小。最大完成时间指的是，从第一道工序在第一台机器上开始工作，直至最后一道工序完成在最后一台机器为止的这段时间跨度(Makespan)。

定义一共 m 台机器，记作$\{\rho_1, \rho_2, \ldots, \rho_m\}$，则在给定调度π的条件下，记工序$\pi_i$在机器$\rho_j$上的完成时刻为$C(\pi_i, \rho_j)$或者简记为$C_{i,j}$。

在 PFSP 问题中，每道工序的加工顺序相同，即按照$\{\rho_1, \rho_2, \ldots, \rho_m\}$的顺序加工；每道工序在每台机器上只加工一次；一道工序不能同时在不同的机器上加工；每个机器同时只能加工一道工序[1]。

因此很显然某调度中的第一道工序π_1一定在ρ_1上加工，并且即开始加工的时刻为 0 时刻，由上可得目标函数，即整个工作的最大完成时间为：

$$\text{Object Function} : C(\pi_n, \rho_m) = C_{n,m}.$$

所以可以写出最优解与目标函数之间的关系：

$$\pi^* = \underset{\pi}{\mathbf{argmin}} \left(C(\pi_n, \rho_m)\right).$$

约束条件可由如下几个式子用符号表示：

$$\begin{cases} t_{ij} = t(\pi_i, \rho_j), \text{表示工序}\pi_i\text{单独在机器}\rho_j\text{上加工所消耗的时间跨度} \\ C_{1,1} = t_{11} \\ C_{1,j} = C_{1,j-1} + t_{1j}, j = 2,3,\dots,m \\ C_{i,1} = C_{i-1,1} + t_{i1}, i = 2,3,\dots,m \\ C_{i,j} = \mathbf{max}\left(C_{i-1,j}, C_{i,j-1}\right) + t_{ij}, i = 1,2,\dots,n, j = 1,2,\dots,m \end{cases}$$

1.3 方法与结果简介

针对上述问题，采用了(Taillard 加速)NEH 启发式算法(Nawaz-Enscore-Ham Algorithm)、邻域/局部搜索(Local Search)算法、爬山法(Hill Climb)、模拟退火算法(Simulated Annealing)，以及遗传算法(Genetic Algorithm)。不难发现，在 PFSP 问题中，“邻域”的定义是模糊的，两个不同排列之间的距离很量化，因此在提出一种基于贪心的邻域确定与搜索算法后，将其运用在爬山法、模拟退火法以及遗传算法，结果表明，经过邻域搜素增强后的算法性能有相当幅度的提升。

本文后续部分组织如下。第 2 节详细陈述使用的方法，第 3 节报告实验结果，第 4 节对讨论本文的方法并总结全文。

2 算法设计

2.1 NEH算法

NEH 算法是一种启发式算法，由 Nawaz 等人与 1983 年提出[2]，而更为常用的实现方式则是由 Taillard 等人于 1990 年提出[3]。沿用 1.2 部分的记号，引入$q_{i,j}$表示从工序π_i在机器ρ_j上开始加工的时刻，直至整个工作结束时刻的时间跨度，即：

$$q_{n,j} = 0,$$
$$q_{i,j} = \mathbf{max}\left(q_{i,j+1}, q_{i+1,j}\right) + t_{ij},$$
$$i = n-1,\dots,1, j = m,\dots,1. \quad (2-1)$$

引入最早相对完成时间$f_{i,j}$，满足：

$$f_{i,0} = 0,$$
$$f_{i,j} = \mathbf{max}\left(f_{i,j-1}, C_{i-1,j}\right) + t_{ij},$$
$$i = 1,\dots,n, j = 1,\dots m. \quad (2-2)$$

NEH 算法的具体步骤为：

Step 1 : 初始化

Step 1.1 计算每个工序单独在各个机器上单独加工的时间总和，并且按加和结果从大到小排序。

Step 1.2 选出前两个工序，并按照部分时间跨度(Partial Makespan)最小原则排列。所谓的部分时间跨度，即在只考虑部分工序的情况下，完成这些工序所必须的最大完成时间跨度。

Step 2 :

For k in range (3, n + 1) :

Step 2.1 根据 1.2 部分的边界条件与递推公式求出$C_{i,j}$ ($i = 1,\dots,k-1, j = 1,\dots,m$)。

Step 2.2 根据$(2-1)$求出 $q_{i,j}$($i = k-1,\dots 1, j = m,\dots,1$)。

Step 2.3 根据$(2-2)$求出 $f_{i,j}$($i = 1,\dots,k, j = 1,\dots,m$)。

Step 2.4 求出工序π_k在i位置插入后，新的部分总时间跨度$M_i = \max_j\left(f_{i,j} + q_{i,j}\right)$，其中下标满足约束条件 $i = 1,\dots,k, j = 1,\dots,m$。

Step 2.5 跟踪记录最小部分时间跨度值M_i^*，以及对应的插入位置i^*，并将工序π_k插入到位置i^*中。

不难发现，作为算法的主体部分，step2 的每一步的时间复杂度为：

$$\text{Time Complexity of Step 2 (k)} = O(km) + O(km) + O(km) + O(km) + O(k) = O(km).$$

而 Step1 涉及到排序，因此时间复杂度是$O(n\log n)$。因此，整个算法的时间复杂度为：

$$\text{Time Complexity of NEH} = O(n\log n) + \sum_{k=3}^{n} O(km) = O(n^2 m).$$

考察 NEH 算法的空间复杂度，分析可知，一共需要三个 n × m 大小的数组来存取中间过程量，因此总的空间复杂度为：

$$\text{Space Complexity of NEH} = O(mn).$$

相比于其他的启发式算法，如 Palmer 启发式算法，其时间复杂度仅为$O(n\log n) + O(nm)$，NEH 算法消耗的时间长，但是时间换取了准确率，NEH 算法在几乎所有传统启发式搜索中的性能是最优越的[3]。

此外，NEH 算法因其启发式的特点，在很多现代智能算法解决 PFSP 问题中，常被用做初始化算法。而在此之前，常被用做初始化的启发式算法是 Palmer 算法。本次实验中没有单独地比较采取 NEH 初始化是否会获得更佳的结果，下图 2-1 1[1]经过 Palmer 初始化的爬山法(Hill Climb)与朴素爬山法之间的对比，诠释了启发式初始化的优越性。值得一提的是，Taillard 在他的这篇论文中，还提到了使用禁忌搜索(Taboo Search)的思路解决 PFSP 问题，其结果略优于 NEH 启发式算法，但是缺点很明显，即太过消耗算力[3]。

图 2-1 1 表明了随着爬山法迭代次数的增加，算法整体相对误差的收敛趋势以及收敛值。可以发现，经过 Palmer 初始化后的爬山算法，误差率有明显的降低，而且收敛速度也明显更快。图 2-1 1 的数据是在 Reeves 数据集[2]上测试的结果，这是由 Reeves 在 1995 年提出的一类问题数据集[4]。

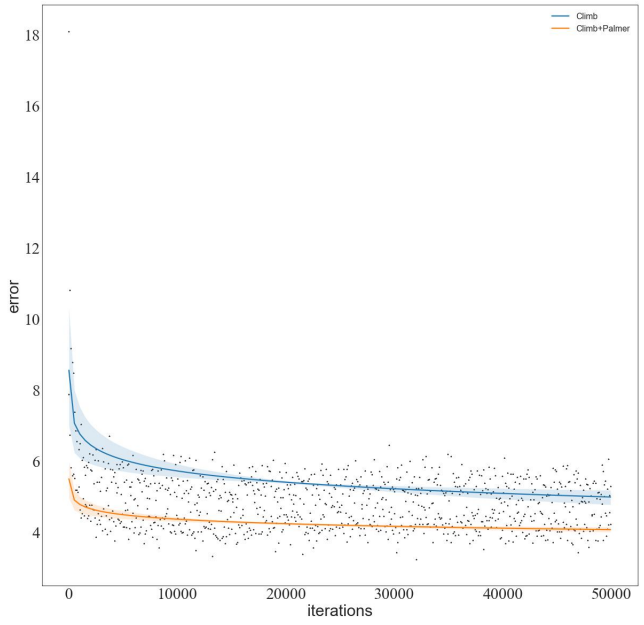

图 2-1 1

1 之所以没有采用 NEH 初始化进行直接对比，是因为这是之前的完成了一半的工作的遗留数据，与这次工作所依赖的编程语言与环境都不同。

2 该数据集在附件中的名字是 instance_rec.txt。

关于 NEH 自身算法的性能，会在第三部分展开叙述，此处仅对算法理论部分进行阐释。

另一个值得注意的细节是，在实际执行 NEH 算法的过程中，会经常遇到多个最小部分时间跨度的情况，这个时候就需要进行“打破”这个“平衡”，这类算法称为 Tie-Breaking Mechanisms [5]。本文根据 Fernandez 等人提出的伪代码，使用 cython 实现并应用到后续的算法中。Tie Breaking 机制的伪代码如下图 2-1 2。

$\pi' \longleftarrow$ Sort in decreasing order of sum of processing times p_{ij};
$\pi \longleftarrow \pi'_1$;
for $k = 2$ **to** n **do**
  $r \longleftarrow \pi_k{}'$;
  Determine the values of e_{ij}, q_{ij} and f_{il} from Taillard's acceleration (see equations 2, 3, and 4);
  Determine minimal makespan resulting from inserting job r in all possible positions of π;
  $bp \longleftarrow$First position where the makespan is minimal;
  $tb \longleftarrow$ Number of positions with minimal makespan (i.e. number of ties);
  $ptb \longleftarrow$ Array (of length tb) with the positions where the makespan is minimal;
  it_{bp} is the idletime corresponding to the bp and set to a very large number;
  if $tb > 1$ **and** $k < n$ **then**
    for $l = 1$ **to** tb **do**
      $it'' \longleftarrow 0$;
      if $ptb[l] = k$ **then**
        for $i = 2$ **to** m **do**
          $it'' \longleftarrow it'' + f_{i,k} - e_{i,k-1} - t_{i,r}$;
        end
      else
        $f'_{1,ptb[l]} \longleftarrow f_{1,ptb[l]} + p_{1,ptb[l]}$;
        for $i = 2$ **to** m **do**
          $it'' \longleftarrow it'' + f_{i,ptb[l]} - e_{i,ptb[l]} + p_{i,ptb[l]} - t_{i,r} + \max\{0, f'_{i-1,ptb[l]} - f_{i,ptb[l]}\}$;
          $f'_{i,ptb[l]} \longleftarrow \max\{f'_{i-1,ptb[l]}, f_{i,ptb[l]}\} + p_{i,ptb[l]}$;
        end
      end
      if $it_{bp} > it''$ **then**
        $bp \longleftarrow ptb[l]$;
        $it_{bp} \longleftarrow it''$;
      end
    end
  end
  $\pi \longleftarrow$ Array obtained by inserting job r in position bp of π;
end

图 2-1 2 Tie Breaking Mechanism 的伪代码[5]

2.2 基于贪心的邻域搜索算法

众所周知，在著名的搜素算法，如爬山法和模拟退火法中，邻域这个概念是非常重要的。而最容易处理的最优化问题往往是把连续的凸函数作为目标函数，因此邻域只需一个中心以及微小量δ即可确定。但是在 FSP 问题中，目标函数的定义域是离散的排列，亦或者是离散多维向量，因此很难确定邻域的范围。本文基于 Zhao 等人于 2021 年的工作[6]以及 Ruiz 等人于 2005 年的工作[7]，实现了基于贪心策略的邻域搜索算法。

算法的核心分为两部分，第一部分是基于贪心的邻域搜索， 第二部分是对贪心过程的迭代法。

现在来介绍基于贪心的邻域搜索。算法的核心思想如下：

Step 1 在原始排列的基础上随机交换两道工序

Step 2 从当前排列中任意取出一道工序。

Step 3 将这个工序插回剩余排列中的任意位置。

Step 4 比较前后两次排列的最大完成时间跨度，如果后一种时间跨度更小，则继续 Step 2，否则结束。

不难发现上述算法被称作“贪心”的原因所在。首先，每次只改变一个工序的相对位置，因此可以理解

为在邻域之中。其次，由于循环不变性，最终搜索得到的结果一定不会比当前值更差，从而近似达到邻域搜索的目的。算法的伪代码如下：

```
procedure IterativeImprovement_Insertion (π)
    improve := true;
    while (improve = true) do
        improve := false;
        for i := 1 to n do
            remove a job k at random from π (without repetition)
            π' := best permutation obtained by inserting k in any possible positions of π;
            if C_max(π') < C_max(π) then
                π := π';
                improve := true;
            endif
        endfor
    endwhile
    return π
end
```

图 2-2 1 基于贪心的邻域搜索的伪代码[7]

然后是第二部分，即迭代过程，此处类似于模拟退火法，采用了 Metropolis 准则，而初始化温度的选择则是参考了 Osman 等人[8]于 1983 年提出的基于模拟退火的初始化温度方法。但是为了体现与模拟退火算法的不同，本文在每轮邻域搜索之前，首先采用准 NEH 算法(Semi-NEH)算法进行全域启发式搜索，从而修正只是用邻域搜索的缺陷。所谓的 Semi-NEH 算法，就是从原排列中随机的取出 k 道工序，并按照 NEH 插入的规则将这 k 道工序插回，并随后进行邻域搜索。这部分的伪代码如图 2-2 2 所示。

```
procedure IteratedGreedy_for_PFSP
    π := NEH_heuristic;
    π := IterativeImprovement_Insertion(π);
    π_b := π;
    while (termination criterion not satisfied) do
        π' := π;                                  % Destruction phase
        for i := 1 to d do
            π' := remove one job at random from π' and insert it in π'_R;
        endfor
        for i := 1 to d do                        % Construction phase
            π' := best permutation obtained by inserting job π_R(i) in all possible positions of π';
        endfor
        π'' := IterativeImprovement_Insertion(π');   % Local Search
        if C_max(π'') < C_max(π) then             % Acceptance Criterion
            π := π'';
            if C_max(π) < C_max(π_b) then         % check if new best permutation
                π_b := π;
            endif
        elseif (random ≤ exp{−(C_max(π'') − C_max(π))/Temperature}) then
            π := π'';
        endif
    endwhile
    return π_b
end
```

图 2-2 2 迭代部分的伪代码[7]

下面进行算法核心部分的时间复杂度的分析。

对于算法的第一部分来说，时间复杂度最长的情况是最极端的，即内部 for 循环每次都能恰好获得最新解，并且持续到下一个 while 循环当中去，周而复始的不断循环下去，直至遍历完整个值域，这是一种非常极端的情况，在实际样例中是几乎不可能出现的。2.1 部分对 NEH 算法的时间复杂度分析已经证明了找到一个工序的最佳插入位置需要消耗$O(km)$的时间复杂度，而最差的情况是遍历整个解空间，所以本算法的第一

部分的最差时间复杂度是：

$$\text{Time Complexity of Local Search (Worst)} = O(n!\,nm).$$

为了防止出现这样的最差情况，尽管这是不可能发生的，在具体实现基于贪心的迭代算法时，在这部分引入了一个新的变量 local_optimum 用来防止这种情况的发生。其具体的工作原理很简单，即用来检测 while 循环重复执行的次数，并在超过一定的阈值之后强行切断循环。

同时考虑最佳的时间消耗情况，这也是最靠近现实的情况，即在 for 循环数次，或者 while 循环几次后，算法终止，这样一来就可以理解为常数次循环，而在算法分析中，本文采取这种分析结果，即：

$$\text{Time Complexity of Local Search (Expected)} = O(kmn).$$

下面来考察算法的第二部分的时间复杂度。第二部分可以分为几个模块：初始化、while 循环、基于贪心的抽放 k 道工序、基于贪心算法的邻域搜索、判断比较部分。

首先是初始化模块，初始化包含一个 NEH 初始化以及一次邻域搜索，即$O(n^2m) + O(kmn)$。

对于随机的抽、放 k 道工序，可以理解为 Semi-NEH 算法，因此复杂度也是$O(n^2m)$。

假设一共循环 r 次，那么总共的时间复杂就是：

$$\begin{aligned}\text{Time Complexity of Iterated Greedy (Expected)} &= O(n^2m) + O(kmn) + O(rn^2m) + O(rkmn)\\ &= O(rn^2m) + O(rkmn).\end{aligned}$$

该算法的空间复杂度仅为线性的，一般情况下邻域搜索算法应该额外开辟空间，但是本基于贪心的搜索算法仅依靠迭寻找新的邻值并更新当前最优值，即：

$$\text{Space Complexity of Iterated Greedy} = O(n).$$

2.3 爬山法

爬山法作为最著名的算法之一，此处不做冗余介绍。但是爬山法的重要性不容忽视，爬山法搭建出许多智能算法的大致框架，而且邻域搜索正是爬山法中重要的思想之一。下面是本文所采用的经过基于贪心的邻域搜索增强后的爬山法的伪代码(Latex 版本请见附件)。

Algorithm 3 Hill Climbing

```
π := NEH_Heuristic
π := Local_Search(π)
iter := 0
while iter < MAX_ITERATIONS do
    i, j := Random_Integers(0, n)
    Swap (π_i, π_j)
    π' := Local_Search(π)
    if C_π' < C_π then
        π := π'
    end if
    iter := iter + 1
end while
```

图 2-3 1 爬山法伪代码

爬山法的时间复杂度分析如下：

2.2 部分已经分析过，爬山法也可以分为初始化部分和迭代部分两部分，不同的是，爬山法的迭代部分比较简单，随机取两个下标并且交换两个元素的位置最多可以在$O(n)$的时间内完成。相似地，假设一共迭代了 r 次，则爬山法的时间复杂度为：

$$\text{Time Complexity of Hill Climbing} = O(n^2m) + O(kmn) + O(rn) + O(rkmn)$$

$$= O(n^2m)+O(rkmn)$$

一般来说，爬山法的需要的空间复杂度就是邻域的大小，但是由于上述修正后的爬山法采用的是贪心策略搜索邻域最小值，所以爬山法的空间复杂度应该是线性的，即：

$$\text{Space Complexity of Hill Climbing } = O(n).$$

2.4 模拟退火算法

模拟退火作为当今最火热的只能算法之一，其原理和背景更是毋用多述。其基本思想为先从一个较高的初始温度出发，逐渐降低温度，直到温度降低到满足热平衡条件为止。在每个温度下，进行多轮搜索，每轮搜索时对旧解添加随机扰动生成新解，并按一定规则接受新解[9]。这个规则通常采用 Metropolis 准则，而降温的算法则有很多选择。同样的，关于模拟退火算法的初始温度，本文选择 Osman 等人给出的初始化算法。

这里给出经过基于贪心的邻域搜索增强后的模拟退火算法的伪代码(Latex 版本请见附件)：

Algorithm 4 Simulated Annealing

```
π := NEH_Heuristic
π := Local_Search(π)
iter := 0
temperature := Osman_Initialization(π)
while iter < MAX_ITERATIONS do
  i, j := Random_Integers(0, n)
  Swap (π_i, π_j)
  π' := Local_Search(π)
  diff := C_π' − C_π
  acceptance_criterion := exp(−diff/temperature)
  probablity := Random_Real (0, 1)
  if C_π' < C_π then
    π := π'
  else if probablity < acceptance_criterion then
    π := π'
  end if
  iter := iter + 1
  if iter % ITER_PER_EPOCH == 0 then
    temperature := temperature * annealing_rate
  end if
end while
```

图 2-4 1 模拟退火算法伪代码

模拟退火法的时间复杂度分析如下。

从伪代码中不难看出来，模拟退火法与爬山法的最大的区别在于，模拟退火法采用了 Metropolis 准则，而这些判断的时间复杂度均为常数性质的，所以模拟退火的时间复杂度与爬山法几乎相同，假设共经历 r 次降温，则时间复杂度为

$$\text{Time Complexity of Hill Climbing } = O(n^2m) + O(rkmn)$$

模拟退火的空间复杂度与爬山法也几乎没有区别，即

$$\text{Space Complexity of Simulated Annealing } = O(n).$$

2.5 遗传算法

遗传算法作为人类从自然进化总结而来的智能算法，原理显得有些微妙，但是大致的思路仍然可以理解为是基于爬山法的思想。遗传是一种随机全局搜索优化方法，它模拟了自然选择和遗传中发生的复制(Selection)、交叉(Crossover)和变异(Mutation)等现象，从任一初始种群（Population）出发，通过随机选择、交叉和变异操作，产生一群更适合环境的个体，使群体进化到搜索空间中越来越好的区域，这样一代一代不断繁衍进化，最后收敛到一群最适应环境的个体（Individual），从而求得问题的优质解[10]。

基于遗传算法的改进有三点，第一点即对“任一初始种群”的修正，一般认为，即使是蓝藻这种最古老的生物，其基因的排列依然具有良好的适应性，即个体具有不错的适应度(Fitness)，因此有必要采取一定的初始化措施，而不是放任其随即乱序初始化。

第二点在于变异方式的选择，变异的方式分为交换(Interchange)、逆序(Reverse)以及插入(Insertion)等。朴素的遗传算法事先决定好变异方式，但是在实际的生物学研究[11]中，变异方式往往是复杂的、并行的、随机的，甚至是有规律[12]的。因此应当将这些不同的变异方式混合在一起，以一定概率决定哪些方式将共同作用于当前代(Generation)。当然，也可以采用爬山法的思想，这一从采用哪些变异方式取决于上一次变异的结果，从而得到“智能”变异的效果。本文实现了前者，即混合变异算法(Mixed Mutation)。

第三点在于每次经过复制、交换和变异操作之后，朴素遗传算法会直接进行最有个体筛选并且以此为依据决定是否保流这代群体，这也是不太适合的。而改进措施至少有两个，其一，就是在变异操作结束以后，将种群中较为优秀的子群，即最大完成时间跨度较小的几道工序取出来，对其进行基于贪心的邻域搜索，并将结果代替原来的个体放回种群；其二，就是参考模拟退火算法，模拟自然选择中的偶然进化，即采取 Metropolis 准则，给予看似失败的种群重新获得进化的可能性。本文基于贪心邻搜索强化域的思想，采取了第一种改进方法。

当然对于复制这一步，也有轮盘赌(Roulette)、锦标赛(Tournament)、随即遍历(Stochastic Uniform)等选择方法，本文采用的是锦标赛选择法。

下面给出遗传算法的伪代码(Latex 版本请见附件)：

Algorithm 5 Genetic Algorithm

```
population := Initialize_Chromes()
population := Local_Search (population)
generation := 0
while generation < MAX_GENERATION do
  new_population := Select_Chromes(population)
  new_population := CrossOver_Chromes(new_population)
  new_population := Mute_Chromes (new_population)
  new_population := Local_Search (new_population)
  if max (C_new_population) < max (C_population) then
    population := new_population
  end if
  generation := generation + 1
end while
```

图 2-5 1 遗传算法的伪代码

遗传算法的时间复杂度分析如下。

类似的道理，将算法拆解为初始化部分、复制、交叉、选择等部分。然后分别考察其时间复杂度。

首先是初始化部分，假设种群的个体数量是 p，那么如果采用 NEH 初始化的方式，这部分的复杂度为：$O(pn^2m) + O(pkmn)$.

如果采取的是锦标赛选择，则复杂度是$O(pn)$的[15]。

交叉部分的复杂度显而易见是$O(n)$的。

变异部分的复杂度比较复杂，因为本文采取了 Mixed 策略，但是从期望上看，此步亦是线性的。

综上，假设遗传算法一共进化 r 代，那么时间复杂度为：

$$\text{Time Complexity of Genetic Algorithm } = O(pn^2m) + O(rpkmn).$$

遗传算法的空间主要花费在存储 population 上，而额外的空间复杂度则是出现在交叉这一环节中，这会导致一个二维数组的额外空间复杂度，即：

$$\text{Space Complexity of Genetic Algorithm } = O(pn).$$

3 实验

3.1 实验设置

3.2 硬件设置

本文实验所采用的是华为 Matebook 14 笔记本。CPU 是 Intel 酷睿 10 代 i5 芯片，GPU 是 Nvidia 的 MX350 商务本显卡。

3.2.1 软件设置

本文实验是基于 Windows 10 企业版进行的，使用的语言是 python 和 cython。

Python 的语言环境为 Anaconda，运行的 IDE 是 Jupyter Lab (Python3 Kernel)。

Cython 的语言环境为 Anaconda 下安装的 python，对应的 C 编译器是 Visual Sudio 2022 Community 中 C/C++ developer 插件自带的 MingW64，所有的头文件和库文件均保存在原始的默认路径。

3.2.2 具体算法参数设置

为了时间考虑，本次实验对于每一个实验样例都设有时间限制，一旦超出时间规定时限，直接停止迭代。一般来说，设置的时间阈值分为 10000ms，20000ms 和 40000ms。

模拟退火的参数设置如下，初始温度由上文所述的特殊算法计算而来。而退火系数则设为 0.99，最大迭代次数有 10000 与 500000 两档。

遗传算法的参数设置如下，种群规模为 30 个个体，交换率为 0.6，变异率为 0.15，最大迭代次数有 10000 与 500000 两档。

3.3 实验结果与分析

3.3.1 算法结果可视化

将上述的五种算法在报告给出的 10 个样例上运行，结果如表 1 所示。

表 1 运行结果

算法	Instance 0	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5
NEH	7038	8564	7376	7399	8003	7835
Iterated Greedy	7038	8366	7166	7312	8003	7720
Hill Climbing	7038	8366	7166	7312	8003	7720
Simulated Annealing	7038	8366	7166	7312	8003	7720
Genetic Algorithm	7038	8366	7166	7312	8003	7720

算法	Instance 6	Instance 7	Instance 8	Instance 9	Instance 10
NEH	1550	2013	1132	2019	3313
Iterated Greedy	1431	1950	1109	1909	3277
Hill Climbing	1431	1950	1109	1902	3277
Simulated Annealing	1431	1950	1109	1902	3277
Genetic Algorithm	1531	1970	1139	1909	3277

由于篇幅有限，这里只展示报告要求的 10 个样例中的部分样例的可视化，所使用的工具是 Plotly 库所提

供的绘画甘特图。注意，此处为了更加清晰地观察各个工序，将完成时间的单位设为了分钟。

图 3-2-1 1 Instance 1 结果可视化

图 3-2-1 2 Instance 6 结果可视化

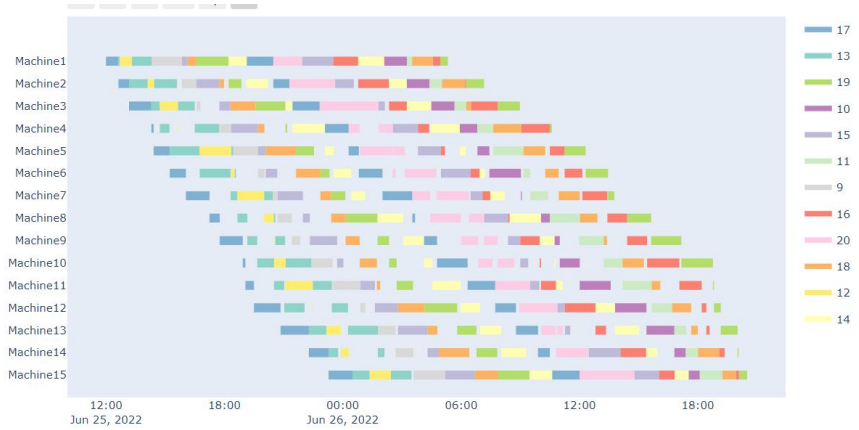

图 3-2-1 3 Instance 7 结果可视化

图 3-2-1 4 Instance 10 结果可视化

3.3.2 邻域搜索增强的对比分析

本文的结果对比分析部分以图为主，表格为辅，意在简洁形象的表明方法之间的优劣。相比于密麻缭乱的数据表格，图像更能清楚的表达数据内在的联系。本文声明，下文所用一切数据均在附件之中，是此次实验进行的结果，如果有更进一步观察数据表格的需求，可在附件中自行查看。

在进行各种对比分析之前，首先声明一下所有数据结果和数据集的来源。首先是来自于报告所要求的 10 个样例，构成附件中的 instance_bit.txt 文件，其次是上文已经提及的 Rec 数据集，在附件中的名字是 instances_rec.txt。

然后要介绍的就是 Vallada 等人于 2014 年提出的新数据集 VRF-Small 和 VRF-Large[13]，其各自有 240 个样例，VRF-Small 主要包含规模比较小的样例，而后者则涉及规模数上百的样例。Vallada 在论文中给出了 benchmark 的上限(Upper Bounds)和下限(Inferior Bounds)，上限是直到 2014 年为止，最领先的智能算法在数据集上的成果；而下限指的是根据分支界限法等蛮力算法求得的理论最优解。由于迄今为止，仍有大部分数据集没有算法可以接近最优解，所以下限标准显得过于遥远而不利于相互之间的比较。所以，本文实验的所有关于 VRF 数据集的相对误差，都是相对于上限而言的。

Taillard 数据集是 Taillard 于 1993 年提出的 benchmark[14]，共 120 个样例，均属于中小规模的样例，可以和 VRF-Small 配合使用，其在附件中的名字是 instances_et.txt。

首先，将使用经过邻域搜索增强的爬山法与朴素爬山法进行对比。图 3-2-2 1 描述的是二者在 VRF-Large 数据集 240 个样例运行结果相对误差率的频率分布直方图。不难看出，在数据规模比较大，达到几百数量级时，使用邻域搜索强化后的爬山法对应的分布峰更高，更加稳定，同时峰偏左，相对误差更低。

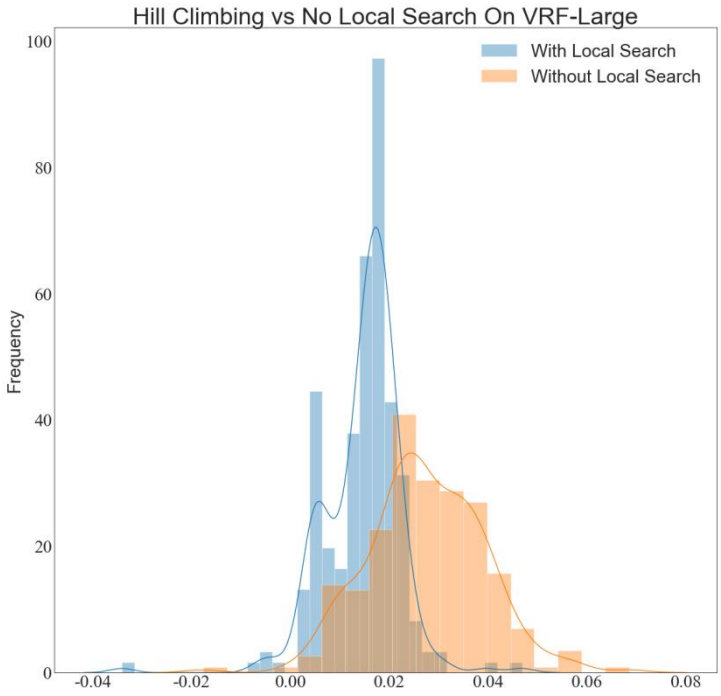

图 3-2-2 1 朴素爬山法与增强爬山法在 VRF-Large 数据集上的对比

观察一下二者在面对小规模样例时的表现，如图 3-2-2 2 所示。可以发现，蓝色在曲线接近一半的样例上已经做到了零偏差，而橙色曲线，即没有使用邻域增强的爬山法

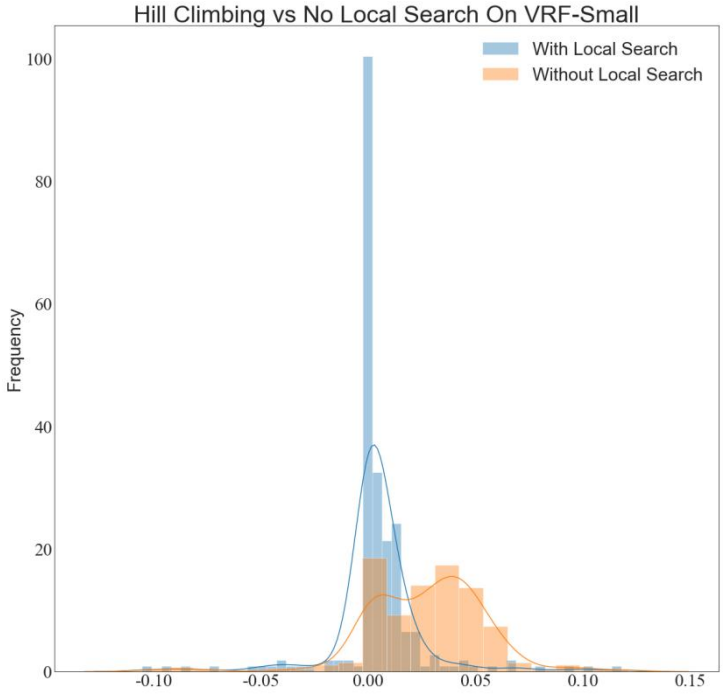

图 3-2-2 2 朴素爬山法与增强爬山法在 VRF-Small 数据集上的对比

接下来考虑模拟退火的邻域增强效果。图 3-2-2 3 是二者在 VRF-Large 数据集 240 个样例运行结果相对误差率的频率分布直方图。不得不说，效果的差异非常之大，没有邻域搜索的帮助，模拟退火在面对规模较大的样例的时候经常会出现无法左右跳动无法收敛的情况。所以说对于模拟退火来说，邻域搜索增强是十分有效的。

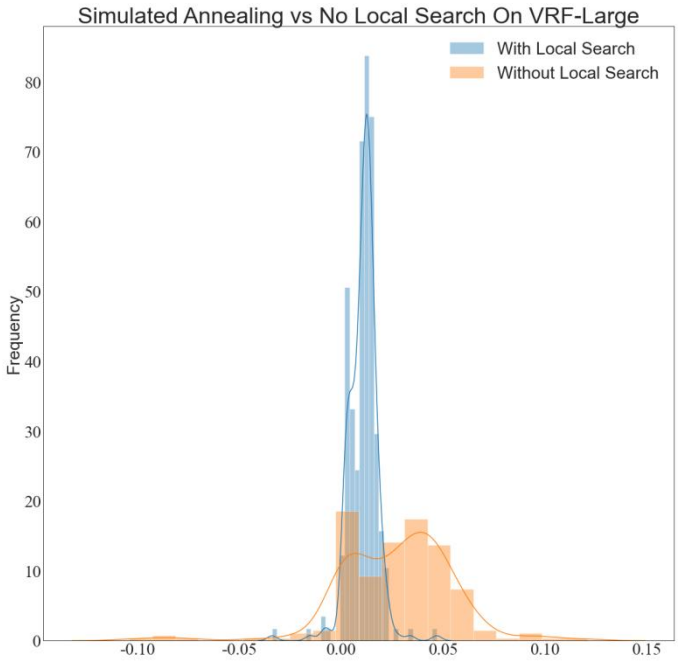

图 3-2-2 3 朴素模拟退火与增强模拟退火在 VRF-Large 数据集上的对比

考察一下二者在小数据集上的情况，如图 3-2-2 4 所示。可以发现面对小规模样例，朴素模拟退火的表现稍微好一些，但是仍然有着细微的偏差。

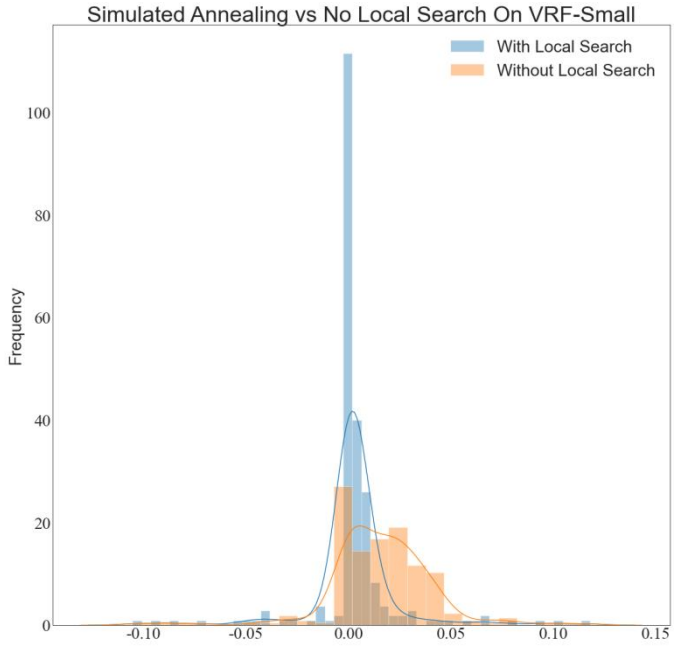

图 3-2-2 4 朴素模拟退火与增强模拟退火在 VRF-Small 数据集上的对比

最后考察遗传算法的邻域搜索增强效果。

由于实验的硬件条件有限，没办法等到遗传算法完全收敛后在停止记录数据。因此图 3-2-2 5 所描述的是相同运行时间下增强后的遗传算法与朴素版本之间的区别。值得注意的是，增强后的遗传算法由于迭代过程要多算一步，所以相同时间内迭代的次数少，但是却得到如此惊人的结果。

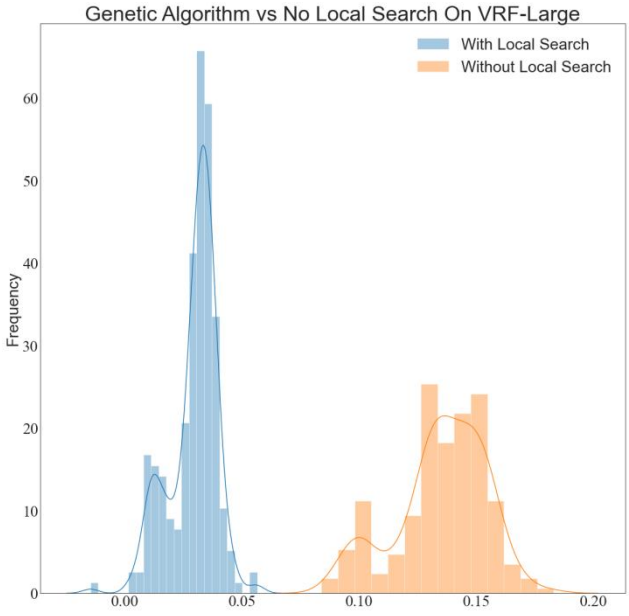

图 3-2-2 3 朴素遗传算法与增强遗传算法在 VRF-Large 数据集上的对比

上图可以看作是逐渐收敛过程中的切片，但是也足以表明二者之间的差距，很明显蓝色曲线有着更快速的收敛能力。考察一下二者在小规模数据集上的表现情况，如图 3-2-2 6 所示。显然。邻域增强后的遗传算法有着更强更迅速的收敛能力。

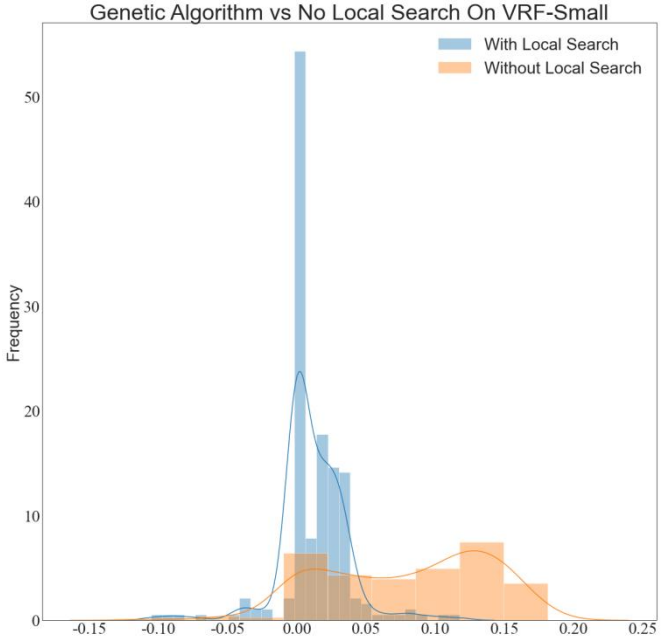

图 3-2-2 5 朴素遗传算法与增强遗传算法在 VRF-Small 数据集上的对比

根据上述分析，不难看出，邻域搜索增强在各个智能算法，如爬山法、模拟退火法和遗传算法中，都可以更快更好的完成 PFSP 问题的求解。

3.3.3 算法间对比分析

首先来看一看作为启发式搜索的获胜者 NEH，其与最入门的智能算法爬山法的比较，结果如图 3-2-3 1 所示，注意这是在 VRF-Large 数据集上比较的。图 3-2-3 1 描述的是各自算法在 240 个样例上运行结果的相对误差率的频率分布直方图。

不难发现，NEH 算法对应的曲线矮，说明结果方差很大，不稳定性强。而且橙色曲线对应的峰偏右，相对误差更高，说明 NEH 的准确率与爬山法的差距非常明显。也即再强的启发式搜索，也难以跨过智能搜索这道门槛，在搜索的过程中不断学习目标函数的特征，往往强过于事先整体的分析。

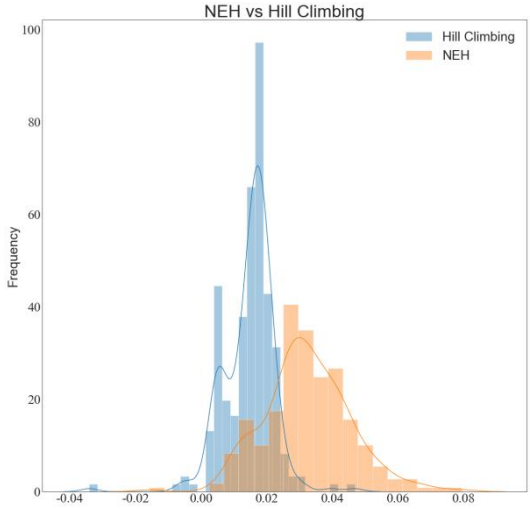

图 3-2-3 1 NEH 与爬山法在 VRF-Large 数据集上的对比

对于剩下的四种算法：贪心迭代(Iterated Greedy)，爬山法(Hill Climbing)，模拟退火法(Simlated Annealing)，遗传算法(Genetic Algorithm)，不妨直接混在一起直观的考察一下，如图 3-2-3 2 所示。

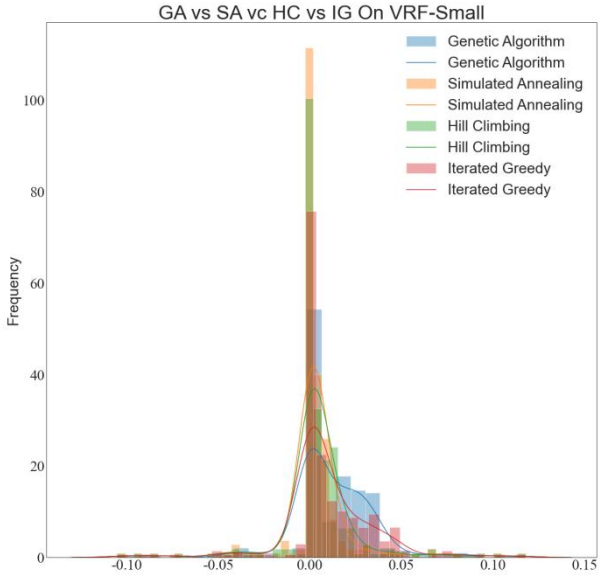

图 3-2-3 2 四种算法在 VRF-Small 上的表现对比

对于 VRF-Small 这种小规模样例的数据集来说，直接查看零偏差个数是一个很好的指标，如此从观察图 3-2-3 2，尽管颜色混杂，但是可以从峰值，抑或是下方的核密度曲线观察得知，

$$SA > HC > IG > GA.$$

前三者的大小关系毋庸置疑，关于为何遗传算法的准确率远低于预期，本文总结以下几方面原因：

1、遗传算法所需要收敛的时间比较长，而上述实验是按照等运行时间原则进行的，所以会显得乏力。

2、遗传算法对应的超参数非常多，由于时间和硬件设施有限，没有能话足够的资源去得到最优的参数配置。

3、遗传算法因其编码/解码过程，非常适合作业调度问题(JSP)，而 JSP 问题的编码非常的长，通常是工序数与机器数的乘积的规模。而 PFSP 因其特殊性，编码的长度仅为工序数。正如同现实中基因的遗传规律一样，交叉、变异等操作对于基因整体的变化是可以忽略不计的，即每一代与前后共三代种群之间是缓慢变化的。换做 JSP 问题，长编码对于这些变化可以起到很好的缓慢收敛的效果，而这些对短编码往往是巨大的改变，所以可能会导致几近随机变换那样跳动，从而大大干扰整体的收敛情况。

四者之间更多的关系图见下。首先是图 3-2-3 3，描述了在 VRF-Large 数据集上的对比情况，注意此时的遗传算法可能还未收敛。不过，从图中就已经可以发现，基于贪心的迭代法在面对大规模样例时显得十分乏力。而由于四者是等时间运行的，所以轻便的爬山法因为多次迭代隐约与模拟退火法相当。

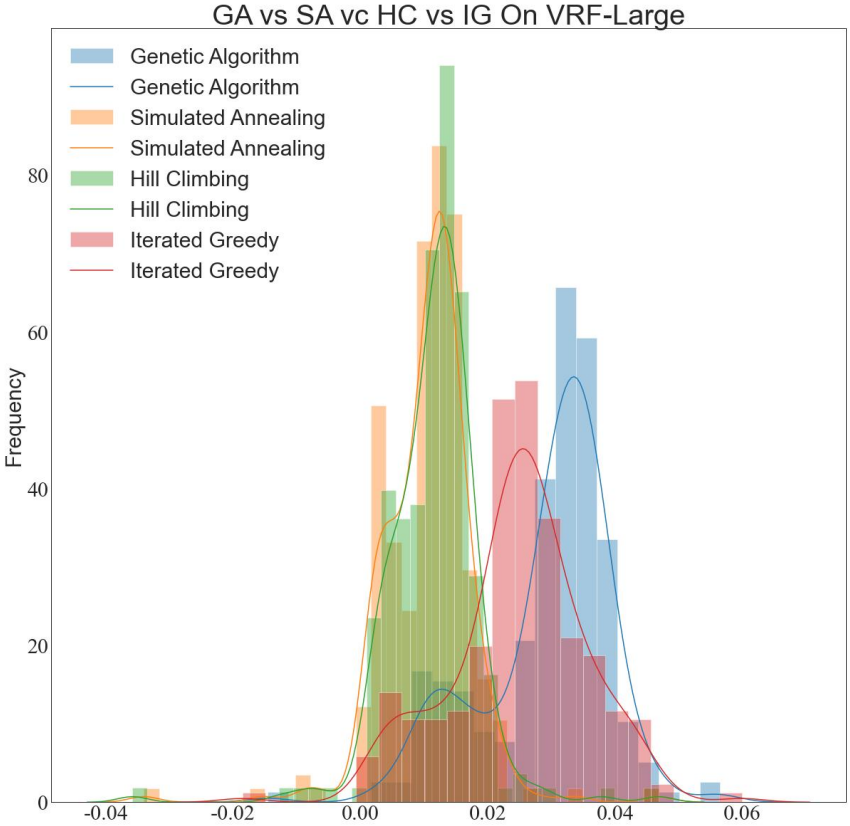

图 3-2-3 3 四种算法在 VRF-Large 上的表现对比

考察一下四者在 ET 数据集上的表现，如图 3-2-3 4 所示。可以发现，此时遗传算法的表现略优于基于贪心的迭代算法，而模拟退火和爬山法依然是保持着较好的情况。

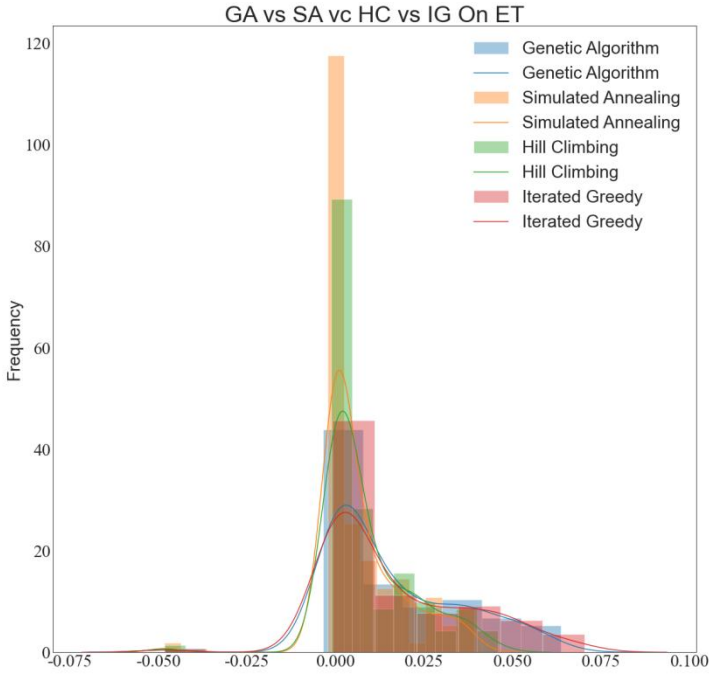

图 3-2-3 4 四种算法在 ET 数据集上的表现对比

最后一张图将五种算法放在一起考察，如图 3-2-3 5 所示。

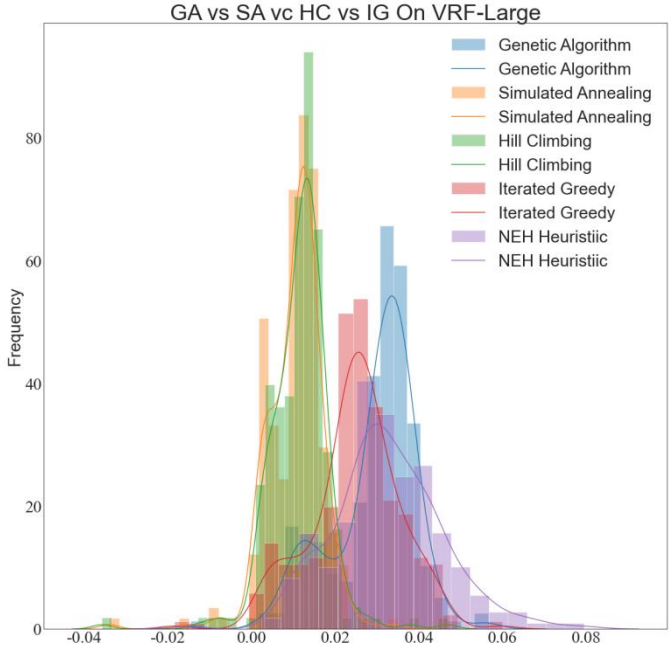

图 3-2-3 5 五种算法在 VRF-Large 上的表现综合

3.3.4 关于算法参数的对比分析

此处展示模拟退火算法中不同参数对于算法性能的影响。注意，由于本文中的模拟退火算法的温度初值是由 Osman 提出的算法给出，所以不属于可调参数，因此此处模拟退火算法可调参数仅为三个，即“同温度下每轮迭代的次数”、“降温系数”以及“最大循环次数”。由于最终的循环次数均在上万次，所以初值的

选择在一定范围内没有明显的作用，但是在数据规模比较大的样例上，会体现出少许的差距。

如图 3-2-4 1 所示，似乎降维系数在 0.9999 时出现了明显高于其他解的情况。所以可以尝试从降温系数为 0.9999 开始下一步调参。

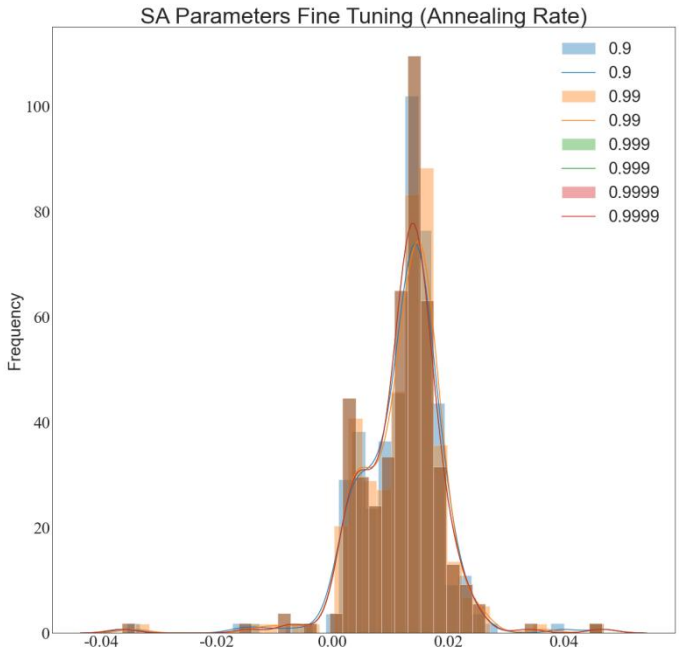

图 3-2-4 1 模拟退火法参数微调(降温系数)

考察一些每个温度下最优的迭代次数，如图 3-2-4 2 所示。

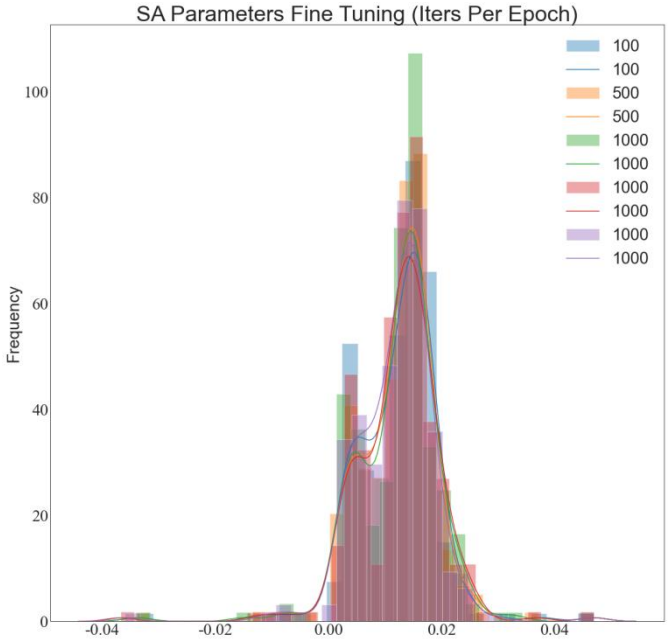

图 3-2-4 1 模拟退火法参数微调(每轮迭代次数)

4 总结

4.1 结论

本文为了解决置换流水车间调度问题(Permutation Flow-Shop Schedule Problem)，介绍了一种基于贪心策略的邻域搜索，并展开理论分析，说明其合理性。并将其运用到四种不同的算法之中进行实验，从实践的角度证明了基于贪心的邻域搜索作为增强算法的可行性与可靠性。同时，本文引入了 NEH 启发式算法作为一种非常优秀的初始化方法，并将其运用到实验中。此外，本文还针对不同的算法进行了深刻的理论分析，分析其具体的结构组成、时间复杂度以及空间复杂度，还对其有改进的地方进行讨论。在进行数据分析与对比分析的时候，本文采用了多种数据标注集作为实验样本，反复进行实验，尽最大可能保证数据完整性与鲁棒性，同时从多种角度分析了算法性能之间地差异。此外，本文采取了“以图代表”的思路，将数据可视化，清晰明了地将各个算法之间的各种特征展现出来。最后，本文对一些算法的参数进行了优化搜索。

经过上述内容的分析与展示，基本足够证明基于贪心策略的邻域搜索对于各种算法在 PFSP 问题上有着很好的增强作用，同时也从侧面阐明了部分智能算法的基本框架与模块化思想。

4.2 反思

本文的不足之处与缺陷之处很多，本文自行总结有如下几点：

1、方法仍然过于粗糙，没有从细节着手分析算法的来龙去脉与特征

2、理论分析不够深刻，很多分析只是停留表面，没有深入研究，看上去泛泛而谈。

3、实验过程中，没有准备好相应的硬件资源，导致遗传算法部分的实验没有拿到预期获得的数据，影响了对数据的分析，以及后续的讨论。

针对以上问题，本文会在之后的工作中，尽可能逐一解决，弥补本文的漏洞。比如，可以通过举例的方式从算法细节角度阐释算法、理解算法，也可以从更宏观的角度，进一步将算法模块化，并深入探究模块与模块之间的互相作用。另一方面，继续加强对于理论性质的研究，对于可能的设想尽可能付诸实践，并站在结果的角度深入透彻的分析理论的可行性。同时，将一些实验中留有遗憾的部分予以补齐，争取将本文意图阐明的故事的全貌展现出来。

参考文献:

[1] Li, Y., Wang, C., Gao, L. et al. An improved simulated annealing algorithm based on residual network for permutation flow shop scheduling[J]. *Complex Intell. Syst*. 7, 1173–1183 (2021). https://doi.org/10.1007/s40747-020-00205-9

[2] Nawaz, M., Enscore Jr., E., and Ham, I..A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem[J]. *OMEGA, The International Journal of Management Science,* 1983.11.1: 91-95.

[3] Taillard, É.D. (1990). Some efficient heuristic methods for the flow shop sequencing problem[J]. *European Journal of Operational Research,*1990: 47, 65-74.

[4] Reeves, Colin. (1995). A genetic algorithm for flowshop sequencing[J]. *Computers & Operations Research.* 1995:22. 5-13. 10.1016/0305-0548(93)E0014-K.

[5] Victor Fernandez-Viagas, Jose M. Framinan.On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem[J].*Computers & Operations Research,Volume 45,*2014 : 60-67,ISSN 0305-0548.

[6] Z. Zhao, M. Zhou and S. Liu. Iterated Greedy Algorithms for Flow-Shop Scheduling Problems: A Tutorial[J], *IEEE Transactions on Automation Science and Engineering,* doi: 10.1109/TASE.2021.3062994.

[7] Rubén Ruiz, Thomas Stützle.A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem[J]. *European Journal of Operational Research, Volume 177, Issue 3.*2007 : 2033-2049,ISSN 0377-2217.

[8] Osman, I., Potts, C., 1989. Simulated annealing for permutation flow-shop scheduling[J]. *OMEGA, The International Journal of Management Science* 17 (6), 551–557.

[9] 数学建模 andMATLAB.模拟退火算法超详细教程[EB/OL].*https://zhuanlan.zhihu.com/p/382426984*, 2021.06.20

[10] tigerqin1980.遗传算法入门详解[EB/OL].*https://zhuanlan.zhihu.com/p/100337680*, 2022.03.22

[11] T.J. Schrader. MUTAGENS. Editor(s): Benjamin Caballero.Encyclopedia of Food Sciences and Nutrition (Second Edition)[M]. *Academic Press,*2003, Pages 4059-4067, ISBN 9780122270550,

[12] Karczewski, K.J., Francioli, L.C., Tiao, G. et al. The mutational constraint spectrum quantified from variation in 141,456 humans[J]. *Nature* 581, 434–443 (2020). https://doi.org/10.1038/s41586-020-2308-7

[13] Eva Vallada, Rubén Ruiz, Jose M. Framinan.New hard benchmark for flowshop scheduling problems minimising makespan[J]. *European Journal of Operational Research, Volume 240*, Issue 3,2015 : 666-677,ISSN 0377-2217.

[14] E. Taillard.Benchmarks for basic scheduling problems[J].*European Journal of Operational Research,Volume 64,* Issue 2,1993: 278-285,ISSN 0377-2217

[15] 柯糖.智能优化算法——遗传算法[EB/OL].https://blog.csdn.net/weixin_42394252/article/details/124612862, 2022.05.08