船舶调度问题的建模与优化

运筹学大作业　计算机图灵实验班　孙浩翔　2023202305

1　问题分析与建模

本文中，我们将探讨船舶调度问题的建模与优化.

1.1　建立数学模型

首先，我们对船舶调度问题进行建模. 一共有 N 条船，每条船具有到达时间 r_i ，长度 l_i ，需要服务时间 p_i ，吃水深度 d_i .

M 个泊位各具有长度 b_j 和水深 $f_j(t)$ ，在该问题中，取 $amp = 2$ ，$T = 1440$ ，有

$$f_j(t) = D_j + amp \times \sin\frac{2\pi t}{T}$$

先来考虑最简单的情形，假设只有一个 0 号泊位，我们用 t_i 表示第 i 条船实际开始服务的时间，于是我们有长度约束 $l_i \le b_0$ ，时间约束 $t_i \ge r_i$ ，水深约束

$$d_i \le D_0 + amp \times \min_{t_i \le t \le t_i + p_i} \sin\frac{2\pi t}{T}$$

想要最小化所有船只的等待总时间，也就是想最小化所有 t_i 的和，因为

$$z = \sum_{i=1}^{N}(t_i - r_i) = \sum_{i=1}^{N} t_i - \sum_{i=1}^{N} r_i$$

首先我们给出 t_i 的范围. 注意到 $\forall x \in [a, a+p]$, $\sin\dfrac{2\pi}{T}x \ge y_0$ 的 a 有如下的性质：

$$\forall x \in [a, a+p], \frac{2\pi}{T}x \in \bigcup_{k\in\mathbb{Z}} [\arcsin y_0 + 2k\pi, \pi - \arcsin y_0 + 2k\pi]$$

当然如果 $y_0 \le -1$ ，上述条件恒成立；如果 $y_0 \ge 1$ ，上述条件不可能成立.

回到原问题，于是我们说，存在某一个 $k \in \mathbb{Z}$ ，使得

$$[t_i, t_i + p_i] \subset [kT + \frac{T}{2\pi}\arcsin y_i, \Big(k + \frac{1}{2}\Big)T - \frac{T}{2\pi}\arcsin y_i]$$

其中 $y_i = \dfrac{d_i - D_0}{amp} \in [-1, 1]$. 对于超出 $[-1, 1]$ 的，按照可行区域为 $\mathbb{R}$ 或者 $\varnothing$ 处理.

下面我们来讨论 t_i 的显式范围. 从上面的讨论中，我们可以发现，t_i 的可行域 H_i 是一段区间的周期重复. 如果我们设 $w_i = \dfrac{T}{2\pi}\arcsin y_i = \dfrac{T}{2\pi}\arcsin\dfrac{d_i - D_0}{amp}$ ，那么

$$H_i = \{x : x \in \bigcup_{k\in\mathbb{Z}} [kT + w_i, \Big(k + \frac{1}{2}\Big)T - w_i - p_i] \ \wedge \ x \geq r_i\}$$

于是我们的问题变为：最小化 t_i 的和，其中 $t_i \in H_i$ ，且各线段 $[t_i, t_i + p_i]$ 不重叠. 这一问题仍然不是传统优化算法容易解决的问题，不过我们可以很容易对某一个解进行估价. 事实上，设 $\{q_i\}$ 为 $1, 2, \cdots, N$ 的一个排列，定义为第 i 号船只的优先级. 在给定优先级（先后顺序）的前提下，我们可以确定最优的情形，从而定义最优情形为这个排列的价值.

我们来进一步阐述这个过程：首先，对于一个给定的港口 j 和一艘给定的船 i ，我们设港口从 $minT$ 时间后可用，于是船只可以进入该港口的最早时间更新为 $minT = \max\{minT, r_i\}$. 我们考察

$$\{x : x \in \bigcup_{k\in\mathbb{Z}} [kT + w_i, \Big(k + \frac{1}{2}\Big)T - w_i - p_i] \ \wedge \ x \geq minT\}$$

中的最小元素 t_i ，它就是这艘船 i 在这个港口 j 最早可以开始停泊的时间.

在安排了优先级之后，我们从优先级最高的船开始考虑. 对于只有一个港口的情况，初始 $minT = 0$ ，然后按照上面的算法将优先级最高的船 i_1 停放在这个港口，得到最早的时间 t_{i_1} . 于是这个港口在这段时间被占用，$minT$ 更新为 $t_{i_1} + p_{i_1}$ ，再来考虑优先级为第二位的船 i_2 ，依此类推，最终这个排列的估值就是所有 t_i 的和.

我们首先给出该部分的 Python 代码. 请参考提交的 `subprocess1.py` ，代码中包含了注释，用来解释各段代码的作用.

1.2 模型的进一步求解

现在考虑多港口的解决方法. 事实上，一旦我们规定每条船的停靠港口，我们就可以对各个港口独立地进行上述计算. 我们定义 $s_i \in \{1, \cdots, M\}$ 为第 i 艘船停靠的港口编号. 对于一个给定的港口 j，我们只需要考虑所有停在这个港口的船只，即指标集 $\{i : s_i = j\}$，按照它们的优先级进行安排即可. 最终的估价 $g(q, s)$ 就是所有船的 t_i 之和.

现在我们将原问题转换为了如下的模型：给定 N 和 M ，需要找到 $1, 2, \cdots, N$ 的一个排列 q 和 $s = (s_1, \cdots, s_N) \in \{1, 2, \cdots, M\}^N$ ，使得估价函数 $g(q, s)$ 最小.

其中估价函数 $g(q, s)$ 的计算代价或者说时间复杂度是 $O(NM \log N)$ 的. 注意在我们的程序中，还需要进行合规性检验，这里我们直接设定不可能满足条件的 q, s 的

$g(q,s)$ 为 `float('inf')` 即可. 接下来，我们首先在 `algorithm.py` 中对之前的结果进行封装，这样可以使我们直接关心主要问题.（`algorithm.py` 中的这部分删除了注释）

不难发现，解空间的大小是 $N! \times M^N$ 量级的，一一枚举并不现实. 这里我们采用遗传算法进行优化.

1.2.1 变量的连续化与统一化

如果我们要处理排列 q ，需要用到部分映射交叉等交叉技术，并不容易实现. 于是我们可以考虑 $u=(u_1,\cdots,u_N)\in(0,1]^N$ 为一个实数向量，从小到大进行排序，考虑它们的下标，就可以生成一个排列 q. 类似地我们考虑 $v=(v_1,\cdots,v_N)\in(0,1]^N$ ，对于某个 v_i ，考虑它落在哪一个长度为 $1/M$ 的子区间即可，即 $s_i=\lceil Mv_i\rceil$. 这样一来，两个因素 q,s 在形式上就统一化了. 换言之，如果我们定义 $x=(x_1,\cdots,x_{2N})\in(0,1]^{2N}$ ，前 N 个元素按照 q 来解读，后 N 个元素按照 s 来解读，我们说这样的 x 就能唯一得到一种排列方案，其估价为 $\varphi(x)=g(q,s)$.

从而我们可以重新叙述原问题：给定 N 和 M ，找到 $x=(x_1,\cdots,x_{2N})\in(0,1]^{2N}$ ，使得 $\varphi(x)$ 最小. 其中 $\varphi(x)$ 的计算代价约为 $O(NM\log N)$.

1.2.2 代码实现

这种情形下，通过下面的代码理论上就可以实现优化：

```
N = int(input()) # 船只数量
M = int(input()) # 港口数量
solver = Solver(N=N, M=M)


def phi(x):
    u = x[:N]
    v = x[N:]
    tmp = []
    s = [0] * (N + 1)
    for idx in range(N):
        tmp.append([idx + 1, u[idx]])
    tmp = sorted(tmp, key=lambda x: x[1])
    q = [0] + [_[0] for _ in tmp]
    for idx in range(N):
        s[idx + 1] = ceil(M * v[idx])
```

```
    return solver.g(q, s)[0]


lb = [1e-5] * (2 * N)
ub = [1] * (2 * N)
from sko.GA import GA
ga = GA(func=phi, n_dim=2 * N, size_pop=50,
        max_iter=800, prob_mut=0.001, lb=lb,
        ub=ub, precision=1e-7)
best = ga.run()
print(best)
```

实际运行时，得到的结果均为 `float('inf')`，显然不合理.

从理论上来说，$x = (x_1, \cdots, x_{2N})$ 可以衍生出所有可能出现的安排，也即涵盖了可行解空间. 然而，可行解在 $(0, 1]^{2N}$ 上的分布实际上是相当稀疏的，因为大量方案是无法满足要求的. 换言之，在遗传算法的计算中，大量的解都是 `float('inf')`. 于是我们退而求其次，仅对一组排列（优先级）来计算可行方案. 依照优先级从高到低，一艘船选取所有可行港口中开始时间最早的，停靠在这一港口.

2 模型的修正与优化

上面我们解释了为什么精准的优化模型不容易用遗传算法解决. 于是我们退而求其次，采用一种贪心的思路来选取，从而保证 $1, \cdots, N$ 的一个排列可以确定一组可行解.

下面我们来详细地阐述新的算法. 对于一组给定的排列（优先级）q，从优先级最高的船 i_1 开始. 我们记录所有港口的可行时间 $minT_j$. 初始情况下所有 $minT_j$ 均为 0.

我们枚举 j，按照之前的方法，计算 i_1 如果停靠在 j 时的最早开始服务时间 $t'_{i_1 j}$. 然后，我们选取 $j_1 = \arg\min_{1 \le j \le M} t'_{i_1 j}$ 作为 i_1 停靠的港口，将 t_{i_1} 设定为 $t'_{i_1 j_1}$，并将 $minT_{j_1}$ 更新为 $t_{i_1} + p_{i_1}$. 依此类推，可以得到所有 t_i，将他们的和定为 q 的价值 $h(q)$.

这时我们的优化问题变为：给定 N 和 M，找到 $1, 2, \cdots, N$ 的一个排列 q，使得 $h(q)$ 最小，其中 $h(q)$ 的计算代价为 $O(NM + N \log N)$.

然后，我们将估价函数 $h(q)$ 封装到 `algorithm2.py` 中，进行求解. 具体细节请见 `algorithm2.py`. 简要地说，我们封装了评估器 $h(q)$，然后调用 `deap` 库进行优化，其中种群大小设为 100，交叉概率设为 0.6，突变概率设为 0.05，迭代次数设为 800，使用锦标赛策略（大小为 2）. 这里使用的是 `cxPartialyMatched` 来处理排列的交叉，

使用 np.random.permutation 来生成随机排列，使用 mutShuffleIndexes 来进行变异，使用 selTournament 来进行选择. 优化代码如下：

```
N = int(input()) # 船只数量
M = int(input()) # 港口数量
solver = Solver(N=N, M=M)

# 这里的 x 从 0 开始
h = lambda x: solver.h([0] + [_ + 1 for _ in x])

import numpy as np
from deap import base, creator, tools, algorithms

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
toolbox = base.Toolbox()
toolbox.register("indices", np.random.permutation, N)
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indices
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=1 / N)
toolbox.register("select", tools.selTournament, tournsize=2)
toolbox.register("evaluate", lambda x: (h(x), ))

print("Start optimizing ...")
population = toolbox.population(n=100)
result, log = algorithms.eaSimple(
    population, toolbox, cxpb=0.6, mutpb=0.05, ngen=800,
    stats=None, halloffame=None, verbose=None
)
best = tools.selBest(result, 1)[0]
```

3 实验

通过以下的方式来运行程序：`python algorithm2.py`，运行时需将数据放置在同一目录下，首先输入船数 N，然后输入港口数 M（需要换行）. 程序将自动从 `ports.txt` 和 `shipsN.txt` 中读取数据，并且输出.

下面是运行示例，其中每个 `port` 后面是一个 `list`，里面存储了所有要在这个港口停泊的船的编号以及其开始停泊的时间：

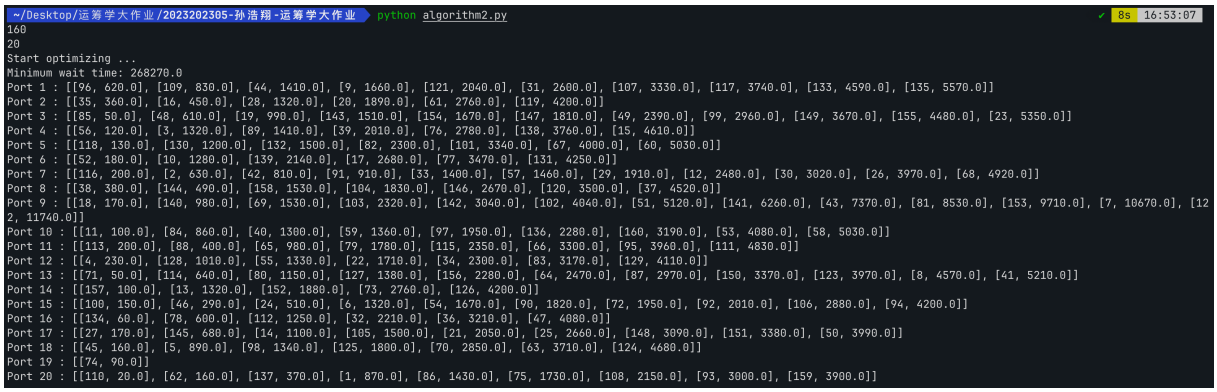

图 1：$N = 160$，$M = 20$ 的情形

下表中给出了在提供的数据集上的运行结果，考虑到遗传算法的随机性，我们运行三次优化代码取最佳值：

运行条件	最短总等待时间
$N = 20$，$M = 20$	950
$N = 40$，$M = 20$	9520
$N = 80$，$M = 20$	31450
$N = 160$，$M = 20$	268270

4 数据可视化

通过 `matplotlib` 库，我们可以实现要求的数据可视化，具体见 `algorithm2.py` 的数据可视化部分，下面仅展示优化结果. （图中在展示船舶停靠时长时，为了验证没有出现重叠，展示的是 $[start, end - 5]$，因此中间短暂断开的线段实际上是连在一起的）

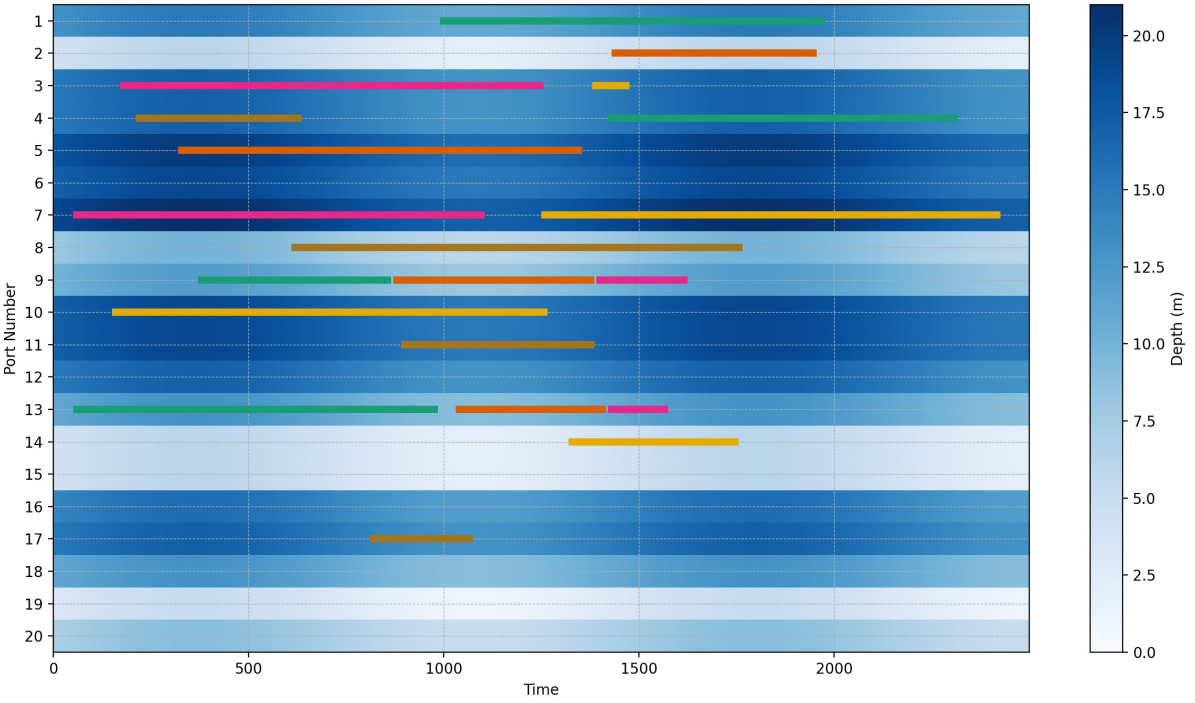

图 2：$N = 20$，$M = 20$ 的情形，优化结果为 950

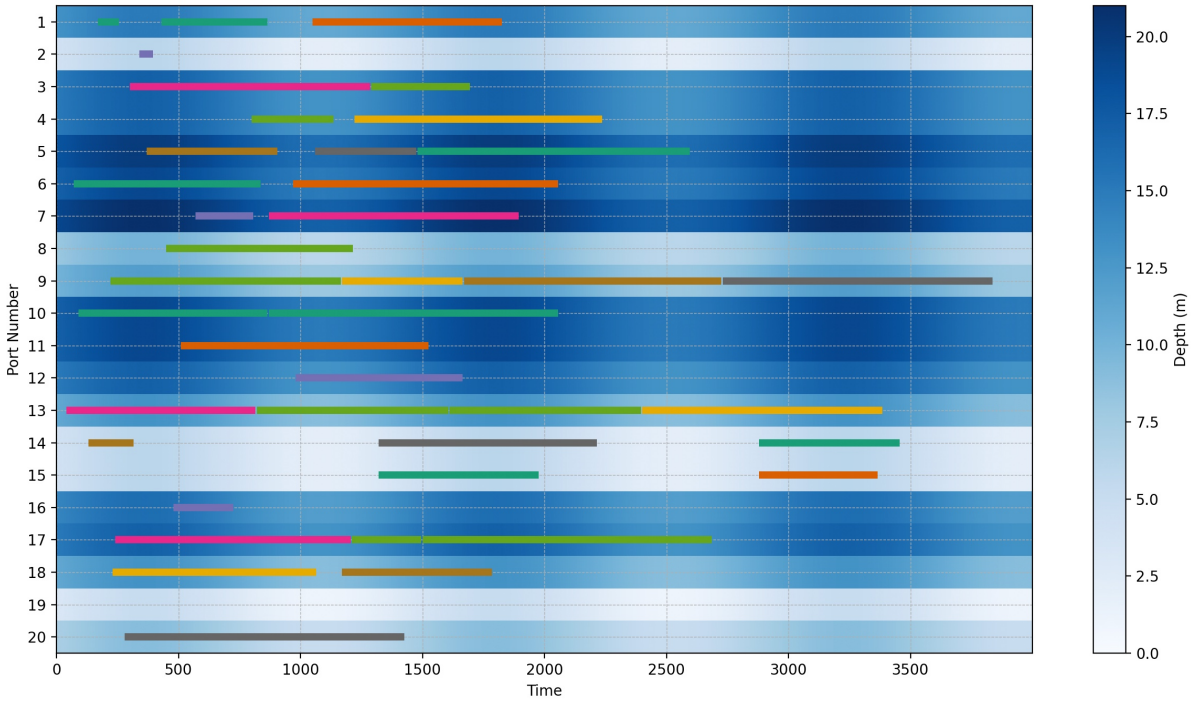

图 3：$N = 40$，$M = 20$ 的情形，优化结果为 9610

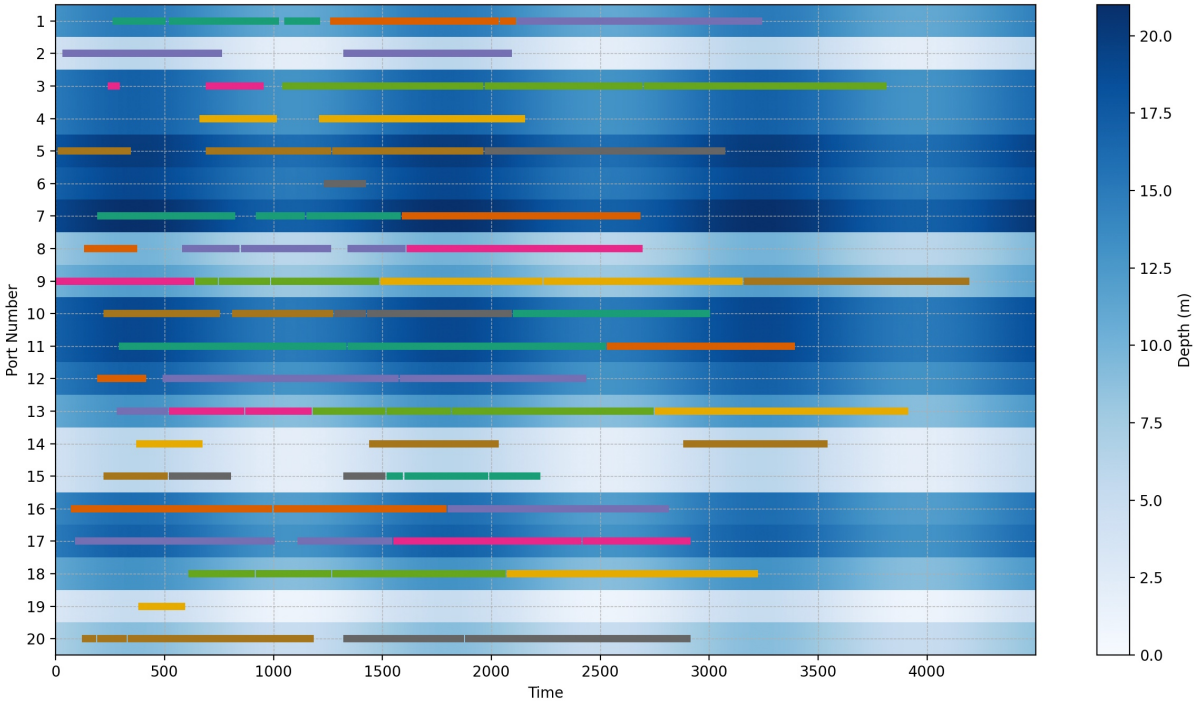

图 4：$N = 80$，$M = 20$ 的情形，优化结果为 32990

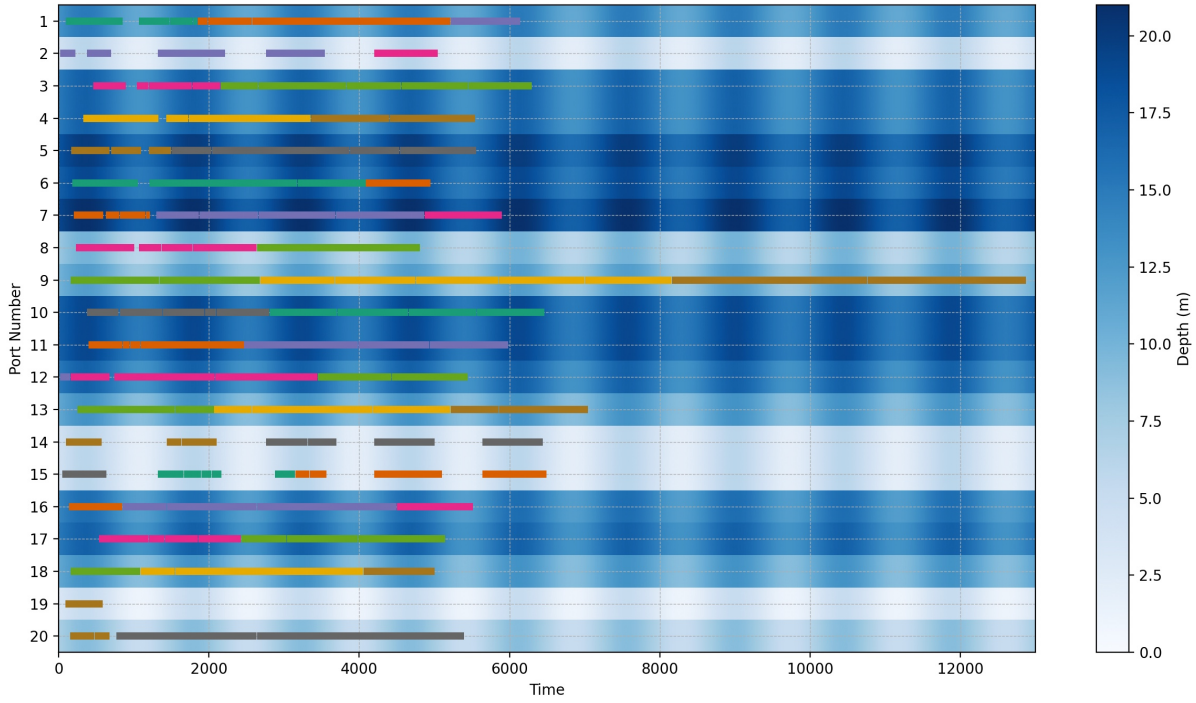

图 5：$N = 160$，$M = 20$ 的情形，优化结果为 287940