代码逻辑与创新性说明

运行的结果在测试服务器上的文件夹路径：`/mnt/home/modelcomp_07/DOCS2025-VRPTW/solutions/results0806`

1. 概述：并行多策略子问题分解HGS求解器

我们所设计的算法是**并行多策略子问题分解HGS求解器**，其主体框架是混合遗传搜索（Hybrid Genetic Search, HGS）[1]，这是一种在组合优化问题中表现稳健的元启发式算法。然而，传统的混合遗传搜索算法在处理大规模的VRPTW问题时，容易陷入局部最优，后期搜索效率降低。

为了克服这一瓶颈，我们设计并实现了两个关键的创新模块：

- 其一是**基于并行算法组（Parallel Algorithm Portfolios, PAPs）思想的多策略协同求解框架。**该框架通过同时运行多个参数各异的HGS求解器实例，并令其周期性地共享精英解，来提升算法整体的搜索鲁棒性与广度。
- 其二是**质心聚类的问题分解与并行求解机制（Barycenter Clustering Decomposition）。**该机制本质上是一种受精英解引导的大邻域搜索（Large Neighborhood Search, LNS），它通过以下方式显著增强了算法的全局探索能力和计算效率：
    - **智能分解**：并非随机破坏解的结构，而是基于当前最优解的地理特征，将原问题智能地分解为多个高度相关的子问题。
    - **并行加速**：利用现代多核CPU架构，同时对所有子问题进行并行求解，将原本耗时巨大的深度邻域搜索过程转化为高效的并行计算任务。
    - **优质解注入**：将子问题的优化结果合并成一个全新的高质量解，并注入回主混合遗传算法的种群中，为算法进化提供新的、结构更优的基因。

下面，我们将详细阐述这两个创新模块的具体代码逻辑。

2. 核心创新点：并行多策略与子问题分解

2.1 创新点一：并行多策略

2.1.1 并行算法组与并行求解

考虑到实际比赛的服务器配置是8 CPU核心，因此我们采用了并行算法组的思想来提高CPU的利用率，并行算法组的基本概念如下：

在求解复杂的组合优化问题时，任何单一的算法（或其特定参数配置）都难以在所有类型的问题样例上取得最佳性能。近年来，一种被称为“并行算法组”（Parallel Algorithm Portfolios, PAPs）的高性能并行计算范式受到了广泛关注 。PAPs的核心思想是，与其依赖单一算法，不如将多个具有性能差异性的“成员算法”构成一个算法集合 。当求解问题时，算法组内的所有成员并行、独立地运行 ，最终的求解性能由其中表现最佳的成员决定 。

根据上述介绍可以看到，该概念十分适合本次竞赛的服务器配置。为了实现这一概念，我们需要明确可以调节的参数有哪些，以及如何构建并行算法组，这些将会在下面的章节中详细介绍。而代码方面，并行算法组的代码实现逻辑大致如下：

- 在`main.py`中通过`--parallel_mode`参数启用后，系统会初始化一个`ParallelMultiStrategyHGS`实例，它扮演着算法组“管理者”的角色。
- 此管理者会根据`--num_strategies`参数，为每一个可用的CPU核心创建一个`StrategyWorker`线程。
- 每个`StrategyWorker`都是一个功能完备但参数各异的HGS求解器实例。通过从`parameter_configs.py`中加载不同的`HGSStrategyConfig`，我们为每个工作者赋予了不同的“性格”（如激进的、保守的、重探索的、重开发的等），这就构成了一个具有内在性能多样性的动态并行算法组。

2.1.2 可调配的超参数

遗传算法参数

`repair_probability`: 修复概率, 控制不可行解被修复的概率

种群参数

`min_population_size`: 最小种群大小，控制种群的最小规模
`generation_size`: 代际大小，每代产生的新解数量
`lb_diversity`: 多样性下界，控制种群多样性的上下界
`ub_diversity`: 多样性上界，控制种群多样性的上下界
`nb_elite`: 精英解数量，每代保留的最优解数量

邻域参数

`nb_granular`: 粒度邻居数，控制邻域搜索的粒度

2.1.3 同步与精英解注入机制

传统的PAPs中，各成员算法在运行过程中通常不进行任何信息交互，相互独立地执行 。我们的设计在此基础上进行了一个关键的创新：引入了**周期性的同步与精英解注入机制**，将原本单纯并行的“竞赛模式”升级为高效的“协同模式”。其核心目标是让算法组内的所有成员能够实时共享搜索成果，利用团队的集体智慧来加速求解。

该机制的优势在于：

- **加速全局收敛**：任何一个成员发现的突破性解（精英解）都能被迅速传播，引导其他成员跳出当前的局部最优，向更优的解空间区域靠拢。

- **维持种群健康度**：精英解的注入为其他成员的种群带来了高质量的“新鲜血液”，极大地丰富了遗传多样性，从而使整个算法组能更持久地保持进化活力。

整个过程由`ParallelMultiStrategyHGS`类中的`_run_parallel_with_sync`和`_perform_synchronization`方法调度，具体步骤如下：

- **周期性暂停 (Pause)**: 主控制器按预设的频率（目前实现为固定的时间间隔，意图与`sync_frequency`参数关联）触发同步事件。一旦触发，它会通过线程事件（`pause_event`）向所有`StrategyWorker`发送暂停信号，使所有并行的HGS求解过程暂时冻结。
- **信息收集 (Collect)**: 主控制器遍历所有已暂停的`StrategyWorker`，收集它们各自当前找到的最优解（`current_best`）。
- **精英解识别 (Identify Elite)**: 收集到的所有解会被提交给`SolutionSynchronizer`模块。该模块负责对这些来自不同策略的解进行集中比较，并识别出当前全局最优的解，我们称之为“精英解”（Elite Solution）。
- **精英解注入 (Inject)**: 主控制器将这个全局最优的“精英解”广播给**所有**的`StrategyWorker`。每个`StrategyWorker`接收到这个解后，会将其放入一个专用的注入队列（`inject_queue`）。
- **恢复与进化 (Resume & Evolve)**: 主控制器解除所有`StrategyWorker`的暂停状态。每个工作者在恢复运行时，会检查其注入队列。如果队列中有新的精英解，它会立即将这个高质量的“外来基因”添加到自己的种群（`population`）中。

2.1.4 算法组的构建

传统上，要人工构造一个高性能的并行算法组，需要大量的领域知识和反复的实验调参，门槛极高 。为此，论文[2]中提出了一种名为**AutoPAP**的智能汇聚自动构造方法，其核心是基于演化优化的思想，以一种贪心、迭代的方式来构建算法组 。我们借鉴了这一先进的构造思想来设计我们的部分核心策略。

AutoPAP的核心构造流程如下：

1. **初始化**: 算法组（即种群P）从一个空集合开始 。
2. **候选生成**: 在每一代（次迭代）中，系统会调用一个复杂的变异算子（如基于SMAC的序列模型优化方法 ），从庞大的算法配置空间中搜索并生成`n`个新的、有潜力的候选算法配置 。为了高效搜索，AutoPAP还采用了“分而治之”的策略，在不同基础算法的子配置空间内并行进行搜索 。
3. **性能增益评估**: 对这`n`个候选算法，系统会评估每一个加入到**当前算法组P**后所能带来的**性能增益**（Marginal Gain）。这个评估是在一个固定的训练问题集上完成的 。
4. **贪心选择与汇聚**: 系统会从`n`个候选中，选择那个能使当前算法组P性能提升最大的算法，并将其永久性地加入到P中 。这一“取最优而纳之”的步骤，体现了“汇聚构造”的思想。
5. **迭代构建**: 重复步骤2至4，直到算法组P的规模达到预设的大小`k` 。

该方法在理论上被证明可以达到`(1 - 1/e)`的近似最优构造效果 ，确保了其构建出的算法组具有坚实的理论性能保障。

在我们的实践中，我们借鉴了这一智能构造思想，利用该方法生成了部分核心的高性能参数配置，并结合了针对VRPTW问题特性的专家经验，手工设计了其他具有互补性的策略，共同构成了最终用于求解的、包含8个成员的并行算法组。

2.2 创新点二：子问题分解

此创新点的实现逻辑主要分为 **“分解”**、**“求解”** 和 **“合并”** 三个阶段，由 `GeneticAlgorithm.py` 中的主循环逻辑调度，并调用 `decomposition.py` 中的核心函数完成。

2.2.1 触发与分解阶段 (Decomposition)

我们尝试了两种触发子问题分解的机制：一种是在遗传算法的主循环中（`GeneticAlgorithm.py` 的 `run` 方法），我们设置了 `decomposition_frequency` 参数（例如，每4000次迭代），周期性地触发分解机制；另一种是并行多策略每10s同步一次然后把最优解更新，连续5次同步没有改进的话就触发分解改进。我们的实验表明后者效果更好一些。

分解逻辑 (`decomposition.py` 中的 `barycenter_clustering_decomposition` 函数):

它将主问题 $\mathbf{G}$ 分解为 k 个子问题 $\mathbf{G_1}, \ldots, \mathbf{G_k}$，每个子问题代表 $\mathbf{G}$ 的一个同构小规模问题（例如，客户数量为100）。具体来说，每一个子问题 $\mathbf{G_i} = (\mathbf{V_i^{'}}, \mathbf{E_i})$，其中顶点集 $\mathbf{V_i^{'}} = \{0\} \cup \mathbf{V_i}$，边集 $\mathbf{E_i} = \{(p,q) | p,q \in \mathbf{V_i^{'}}, p \neq q\}$，并且满足 $\mathbf{V} = \bigcup_{i=1}^{k} \mathbf{V_i}$, $\mathbf{V_i} \cap \mathbf{V_j} = \emptyset$, $\forall i \neq j (1 \leq i, j \leq k)$。 为了确定对 $\mathbf{V_i}$ 的划分，我们使用**质心聚类分解**[3]方法。该方法将HGS得到的精英解 $\mathbf{x_0}$ 中所有路径划分到 k 个簇中，并为每个簇构建一个子问题。一条路径 $\mathbf{R} = (n_1, n_2, \ldots, n_L)$ 的质心定义为：

$$(\overline{x}, \overline{y}) = \frac{1}{L-1} \sum_{i=1}^{L-1} (x_i, y_i), \tag{1}$$

其中 (x_i, y_i) 是节点 n_i 的笛卡尔平面坐标。 为了将质心相近（即地理空间具有相关性）的路径聚集在一起，并保持每个簇中的客户数量接近 $\left\lceil \frac{|\mathbf{V}|}{k} \right\rceil$，我们使用**均衡k-means算法**，将路径的质心视为待聚类点。在每次迭代中，根据所有路径与簇中心之间的距离，按升序将路径分配给对应的簇。当某个簇的客户数量超过 $\left\lceil \frac{|\mathbf{V}|}{k} \right\rceil$ 时，便不再向该簇添加更多路径。在将所有路径重新分配到各自的簇之后，每个簇 $\mathbf{S}_i = \{\mathbf{R}_{i,1}, \mathbf{R}_{i,2}, \ldots, \mathbf{R}_{i,K_i}\}$ 的质心更新如下：

$$(\overline{x_i}, \overline{y_i}) = \frac{1}{\sum_{j=1}^{K_i} (L_{i,j} - 1)} \sum_{j=1}^{K_i} (L_{i,j} - 1)(\overline{x_{i,j}}, \overline{y_{i,j}}) \tag{2}$$

其中 K_i 是簇 $\mathbf{S}_i$ 中的路径数量，$(\overline{x_{i,j}}, \overline{y_{i,j}})$ 是 $\mathbf{S}_i$ 中第 j 条路径 $\mathbf{R}_{i,j}$ 的质心。 重复为新簇分配路径的过程，直到所有簇的质心不再发生变化。最终，$\mathbf{V_i}$ 由簇 $\mathbf{S_i}$ 中的所有客户组成。下面是分解的代码逻辑：

1. **精英解选择**: 算法启动时，我们选取当前种群中的全局最优解 (`self._best`) 作为“精英解”。该精英解代表了当前搜索到的最高质量的路径规划模式。
2. **路径质心计算**: 遍历精英解中的每一条非空路径（Route），计算其包含的所有客户点的几何质心（Barycenter）。这个质心可以看作是该路径服务的地理核心区域。
3. **K-Means聚类**: 以所有路径的质心坐标为特征，我们使用 `KMeans` 算法将这些路径聚成 k 个簇 (`num_subproblems` 参数控制)。这一步的巧妙之处在于，它能自动地将地理上邻近、服务区域重叠的路径分在同一组，为形成结构紧凑、内部关联性强的子问题奠定了基础。

4. **动态均衡调整 (核心优化)**: 为了避免极端聚类结果导致某些子问题规模过大而另一些过小，我们引入了一个**带动态中心更新的均衡化循环**。相较于现有文献的**质心聚类分解**[3]仅设置子问题数量 k 而忽略规模均衡，我们的创新在于引入了该动态均衡调整机制，有效解决了子问题规模可能失衡的问题。
    - 如果某个簇的客户总数超过了预设的`max_customers_per_cluster`上限，算法会识别出该“过载簇”中离其簇中心最远的一条路径。
    - 然后，算法会为这条路径寻找一个新簇，即计算它到所有其他簇中心的距离，并将其移动到距离最近的那个簇。
    - **关键修正**: 在移动路径后，我们会**立即重新计算**发生变动（移出和移入）的两个簇的新质心。这一动态更新确保了后续的平衡调整始终基于最准确的簇几何中心，使平衡过程更稳定、更合理。
5. **子问题生成**: 根据最终的路径分组结果，为每个簇创建一个独立的、完整的`ProblemData`对象。这包括：
    - 提取该簇中所有路径涉及的客户点和仓库点。
    - 构建新的、更小的距离和时间矩阵。
    - 生成新旧客户索引的映射关系（`subproblem_mappings`），这是后续合并解的关键。

2.2.2 并行求解阶段 (Parallel Computing)

当子问题列表生成后，算法利用Python的`concurrent.futures.ProcessPoolExecutor`创建一个进程池，将每个子问题分配给一个独立的CPU核心进行求解。

求解逻辑 (`GeneticAlgorithm.py`中的`solve_subproblem_worker`函数):

- **并行独立求解**: 每个工作进程（worker）调用一个独立的HGS求解器（`solve_subproblem`），在设定的迭代次数内（`subproblem_iters`）对分配到的子问题进行优化。
- **资源隔离**: 由于是在不同进程中运行，各子问题的求解过程互不干扰，可以最大化地利用计算资源。
- **高效探索**: 这种方式等同于在HGS的单次迭代内，同时对解空间中 k 个不同的、高质量的“大邻域”进行了深度搜索，其效率远高于串行执行。

2.2.3 合并与注入阶段 (Merging & Integration)

在所有子问题求解完成后，主进程收集各个子问题的最优解，并进行合并。

合并逻辑 (`GeneticAlgorithm.py`的`run`方法内):

1. **索引映射转换**: 利用分解阶段保存的`old_to_new_map`，我们将每个子问题解中的客户索引转换回原始问题的全局索引。
2. **新解的构建**: 将所有从子问题中优化过的、并已转换索引的路径汇集起来，构建成一个对应于原始问题的、全新的完整解（`merged_solution`）。
3. **注入与进化**: 这个新合并的解，凝聚了所有子空间优化的精华，通常具有非常高的质量和新颖的结构。我们将其立即进行局部搜索优化（`_improve_offspring`），然后添加到HGS的种群中。
    - 如果这个新解优于当前的全局最优解，它将成为新的精英解，不仅直接提升了求解质量，也为下一次的分解提供了更高质量的基因。

- 即使它不是最优解，作为一个高质量的新个体，它也能极大丰富种群的多样性，帮助算法跳出局部最优，从而引导后续的交叉、变异操作向更有希望的区域进化。

3. 创新性总结

我们的算法设计其核心创新性体现在上述两个关键的模块：

并行多策略模块

1. **从单一求解器到协同算法组的范式革新**：我们没有遵循传统的、试图寻找一套“最优”参数的思路，而是引入了**并行算法组（PAPs）的思想。通过同时运行多个具有不同参数配置的HGS求解器，算法的整体性能不再受限于任何单一策略，从而极大地提升了求解的鲁棒性和搜索广度。**这种“专家团队”式的架构，确保了无论面对何种特性的问题实例，总有至少一个策略能进行有效探索。
2. **动态信息共享与协同进化机制**：相较于传统PAPs中各成员“各自为战”的独立运行模式，我们设计了关键的**同步与精英解注入机制。**这在算法组内建立了一套高效的信息共享通道。任何成员发现的突破性进展都会被迅速广播，引导整个团队向更优的解空间协同进化，有效避免了在无效区域的重复计算。

子问题分解模块

1. **将大邻域搜索思想与聚类算法有机结合**：我们没有采用传统的随机性破坏算子，而是通过基于精英解路径质心的**均衡K-Means聚类**，实现了一种**有指导、有逻辑的邻域划分。**这使得“破坏”和“修复”的过程不再盲目，而是聚焦于地理上高度相关的区域，大大提高了搜索的有效性。
2. **以并行计算克服大邻域搜索的效率瓶颈**：大邻域搜索虽然效果强大，但其计算开销巨大。我们通过将大的邻域（子问题）分解，并利用多核CPU进行**并行求解**，巧妙地将时间复杂度转化为空间（计算资源）复杂度，在可接受的时间内完成了对解空间的大范围、深层次探索。
3. **实现了HGS与分解并行框架的动态协同**：整个分解、求解、合并的过程被设计成一个周期性的“强力变异”算子，嵌入在遗传算法的框架中。它依赖于HGS演化出的精英解，同时其产出的高质量新解又反哺HGS种群，形成了一个**动态、自适应的进化循环**，有效平衡了算法的“探索”与“开发”。

综上所述，我们的算法通过并行多策略和子问题分解并行求解机制，能够比传统HGS更高效地探索复杂的VRPTW解空间，在保证解质量的同时，显著提升了寻找更优解的潜力和速度。

参考文献

[1] Vidal, Thibaut, et al. "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows." *Computers & operations research* 40.1 (2013): 475-489.

[2] 刘晟材,杨鹏,唐珂. "近似最优并行算法组智能汇聚构造" 中国科学:技术科学 53 (2023): 280-290.

[3] Santini, Alberto, et al. "Decomposition strategies for vehicle routing heuristics." *INFORMS Journal on Computing* 35.3 (2023): 543-559.