下面进行一步步的优化，整体思路是，整体思路是：在保持已有的调度和计算流程的基础上，逐步引入更真实的基础数据、道路网络信息、评估体系和可视化手段，从而让实验和后续研究更完整、结果更能落地。

1. 建立更真实的基础数据

1.1 医院与医生的基础数据

1. 数据维度

• 地理位置（经纬度或城市坐标系）；

• 基础资源：可用床位总数、呼吸机、手术室、ICU数量、烧伤科设备等；

• 医护人员：医生、护士的人数或班次信息，可以用于更精细的调度；

• 医院等级、专科信息：例如三甲、综合性、专科医院等；

• 历史收治率/科室负载数据（如果有真实数据，可以模拟医院在不同时间段的资源消耗情况）。

2. 数据获取

• 公开数据或模拟数据：如果没有真实医院数据，可以基于一些已公开的“各区域医院床位数”等统计信息，随机生成或仿真；

• 结构化存储：将这些信息放在一个CSV或JSON文件中（如 hospitals.csv），在代码中批量读取、注册到 Scheduler 中。

3. 整合进当前代码

• 你可以封装一个 DataLoader 类或一个读取函数，比如 load\_hospitals\_from\_csv(file\_path) -> List[Hospital]；

• 在 example.py 或其他入口脚本中调用 scheduler.register\_hospital(h) 时，循环读取并注册。

1.2 救护车的基础数据

1. 数据维度

• 起始位置：可细化到具体的急救站或医院位置；

• 车辆类型：普通急救车、重症监护型救护车（ICU车）、甚至直升机等；

• 载客容量、随车医护、特殊设备（ECMO、呼吸机、监护仪）等；

• 运营班次或工作时间（可选，如果要考虑车辆的轮班机制）。

2. 数据获取与存储

• 同样可以用CSV/JSON维护一个列表，如 vehicles.csv；

• 读取后转换为 Vehicle 对象并注册到 scheduler。

3. 动态更新（可选）

• 如果想模拟现实，还可在运行过程中随机改变某些车辆的状态（车辆故障、设备损耗等），让模型更真实。

1.3 病人的基础数据

1. 模拟或真实事件记录

• 若没有真实事故数据，可自行模拟：例如高峰时段、节假日、大规模事故的病人出现频率；

• 可以将病人的属性（伤情等级、特殊需求、地理位置）存储到 CSV/JSON 中，模拟持续不断“出现”的病人队列。

2. 批量或实时添加

• 你可以在 BaseScheduler 或一个独立模块中实现“事件生成器”，定时或批量地 add\_patient(patient) 触发调度，逼近真实场景。

2. 融入城市路网进行路径和转运时间的精准计算

2.1 路网数据的获取与存储

1. OpenStreetMap

• 可以使用 Python 的 OSMnx、pyrosm 等库来下载或加载某个城市范围的路网数据；

• 路网通常包含道路拓扑（节点和边），每条路段的限速、长度、方向等信息。

2. 自定义或简化网络

• 如果暂时不想使用外部库，也可手动创建一个简单的路网拓扑（如网格状）并存储在 JSON/CSV 文件中，记录每条边的长度和通行时间；

• 后期再升级到更真实的 OSM 数据。

2.2 从“欧氏距离”到“路网最短路径”

1. 替换 distance\_to

• 目前 Location.distance\_to(other) 用的是欧氏距离，后续需要改为“查询路网最短路径距离”。

• 可以写一个 RoadNetwork 类或 RoutePlanner 类，里面实现 get\_shortest\_path\_distance(start: Location, end: Location) -> float。

• 这样，在 DistanceCalculator 里就不再直接调用欧氏距离，而是通过 route\_planner.get\_distance(start, end) 来得到真实路网距离。

2. 动态交通状况

• 你的 traffic\_factor 逻辑可以继续沿用：将“路网最短路径距离 / 平均速度” 乘以 traffic\_factor，得到最终时间。

• 进一步可以细分到路段，如果你想更精细地模拟拥堵。

2.3 路径规划算法

1. Dijkstra / A\*

• 如果你使用路网数据，可以借助一些常见路径搜索算法（Dijkstra, A\*, Floyd-Warshall, etc.）来获得最短路径；

• OSMnx 底层也封装了常见的最短路径算法，可以直接调用。

2. 更新车辆实时位置

• 当车辆在路网中移动时，你可以在模拟中定时更新车辆所在的路网节点或经纬度，从而做出更精准的再调度。

3. 建立通用的调度结果测算与评估方法

3.1 关键评估指标

1. 响应时间：从病人被发现到车辆到达现场的时间；

2. 运输时间：从车辆接到病人到达医院的时间；

3. 医院资源利用率：如医院床位、关键设备的使用占比；

4. 优先级满足率：高优先级（红级）病人是否能在一定时间门槛内得到救治；

5. 任务完成率：在给定时限内完成多少比例的救援；

6. 车辆调度效率：车辆空驶率或空驶时长。

3.2 评估框架

1. 在代码中留评估接口

• 例如建立一个 Evaluation 类，把上面的指标都实现对应的计算方法；

• 在每一次调度或调度结束后，调用 evaluation.calculate\_metrics(scheduler) 来得到各项指标。

2. 统一输出或日志记录

• 可以将评估结果（如平均响应时间、医院资源使用率等）记录到日志或CSV中，方便后续分析；

• 如果要可视化，也可以把这些指标传给可视化模块进行图表展示。

3. 对比不同算法或参数

• 你可以在 GreedyScheduler 的基础上，再实现一个 HeuristicScheduler、RLBasedScheduler 等，然后用同一套评估方法进行对比实验；

• 也可以对比在 交通拥堵 vs. 交通畅通、多车辆 vs. 少车辆 等不同情境下的调度效果。

4. 增加可视化模块，动态展示调度结果

4.1 地图层面的可视化

1. 采用现有可视化库

• Python中常用的有 folium（基于Leaflet，做网页地图可视化）、geoplotlib、geopandas + matplotlib 等；

• 或者前端基于 Mapbox、Leaflet 等，再通过后端接口提供数据。

2. 展示内容

• 医院：在地图上以标记显示位置、可用床位、资源状态；

• 车辆：动态在地图上移动，若有多个车辆，可以用不同颜色/图标区分；

• 病人：初始位置标记，连接线条显示车辆去接病人的轨迹，或者使用动画方式展示车辆移动过程。

3. 静态 vs. 动态

• 静态：在所有运送结束后，渲染一张包含所有路线和时间戳的地图；

• 动态：在模拟过程中，定时刷新车辆、病人位置，可以看到资源调度的过程。

4.2 时间过程可视化

1. 甘特图或时序图

• 以车辆为主轴，展示每个车辆在不同时段执行的任务（去接哪位病人、送往哪个医院、返回时间等）；

• 这对于调度流程分析很直观，可以在 matplotlib 或 plotly 里绘制甘特图。

2. 指标曲线

• 随着模拟时间的推移，动态展示系统指标（响应时间、医院剩余床位等）的变化趋势；

• 可以用 matplotlib 或 plotly 的折线图来实现实时或离线绘制。

5. 总体整合思路

1. 数据层

• 构建 hospitals.csv, vehicles.csv, patients.csv 或者在数据库中管理；

• 提供统一的 load\_\* 函数来批量导入；

• 为路网提供 road\_network.osm（OSM 文件）或本地 JSON/Graph 数据。

2. 业务逻辑层

• 在 BaseScheduler 中进一步抽象调度流程；

• 在 DistanceCalculator/RoutePlanner 中实现对路网的调用；

• 增强 GreedyScheduler 或开发更多算法，利用同一个框架评估。

3. 评估与可视化层

• 提供 Evaluation 类或模块来计算指标；

• 提供 Visualization 模块或脚本：

• 地图（folium + geojson）

• 甘特图/时序图（matplotlib/plotly）

• 指标曲线

4. 迭代式开发

• 建议先从最核心的数据读取与调度评估开始，做好单元测试；

• 再接入路网真实距离；

• 最后把可视化加入进来，或并行开发一个可视化的Demo进行效果展示。

6. 小结

你的代码框架已经包含了伤员-车辆-医院的核心调度逻辑，也留出了交通因子的接口。后续扩展主要集中在四个方面：

1. 真实或模拟基础数据（医院、车辆、病人）；

2. 城市路网（从欧氏距离到真实路网距离/路径）；

3. 调度评估指标与对比（建立统一的测算与对比框架）；

4. 可视化（地图+图表），辅助分析与展示。

以上优化方向能够让你的调度模型更贴近实际应用场景，也能在学术或工程上产生更高的价值。随着功能不断完善，你可以进行更多高级算法（如多目标优化、强化学习）的尝试，并在同一系统中进行数据的收集与评估。祝研究顺利进行，期待看到更完善的系统演示与成果。