

**课程实验报告**

|  |  |
| --- | --- |
| 课程名称 | 自动规划 |
| 实验名称 | 指定实验1 车辆搬运 |

|  |  |
| --- | --- |
| 学 号： | 58122307 |
| 姓 名： | 何锦诚 |
| 学 院： | 人工智能学院 |
| 专 业： | 人工智能 |
| 指导教师： | 张志政 |
| 实验日期： | 2024-2025秋季学期 |

## **用 TypeScript 完成车辆搬运规划算法的设计与实现**

本项目使用 TypeScript 和 React 实现了一个交互式的车辆重排系统。系统已经部署在 Vercel 平台上，可以通过访问 https://auto-planning-jackson.vercel.app/ 直接体验。系统提供了直观的用户界面，支持用户自定义车道数量、车位数量，并可以设置初始状态和目标状态。通过选择不同的算法（BFS或A\*），系统能够自动计算最优的移动序列。

1. 规划问题定义
2. 流的描述：本规划问题是一个车辆重排问题，通过TypeScript和React实现了一个交互式的车辆重排系统。系统支持用户自定义车道数量、车位数量，并可以设置初始状态和目标状态。系统提供了两种算法（BFS和A\*）来求解最优移动序列。
3. 动作描述：
   1. 在同一车道内移动：车辆可以在同一车道内向左或向右移动，前提是移动路径上没有其他车辆阻挡
   2. 跨车道移动：车辆可以移动到其他车道的空位，前提是原车道右侧没有阻挡，且目标车道目标位置右侧没有阻挡
4. 初始环境：系统支持用户通过交互界面设置初始状态，包括：
   1. 车道数量（默认为2）
   2. 每个车道的车位数量（默认为3）
   3. 每个位置上的车辆编号（0表示空位）
5. 目标环境：用户可以通过交互界面设置目标状态，包括：
   1. 与初始状态相同的车道和车位数量
   2. 期望的车辆分布状态
6. 规划生成机制（规划算法）介绍：系统实现了两种规划算法：
   1. 广度优先搜索（BFS）：
      1. 使用队列进行状态空间搜索
      2. 保证找到最短路径（最少移动次数）
      3. 使用Set记录已访问状态避免重复搜索
   2. A\*算法：
      1. 使用启发式函数估计到目标状态的代价
      2. 启发式函数考虑：
         1. 曼哈顿距离：车辆到目标位置的直线距离
         2. 车道变换惩罚：跨车道移动增加额外代价
         3. 阻挡惩罚：路径上有其他车辆阻挡增加代价
      3. 使用优先队列选择最优状态进行扩展
7. 规划程序设计与实现
8. 介绍
   1. 编程语言：TypeScript + React
   2. 运行环境：Web浏览器，支持现代浏览器（Chrome、Firefox、Safari等）
9. 运行情况说明
   1. 输入输出说明：
      1. 输入：
         1. 车道数量（laneCount）
         2. 每个车道的车位数量（spaceCount）
         3. 初始状态（initialState）：二维数组表示车辆分布
         4. 目标状态（targetState）：二维数组表示期望的车辆分布
      2. 输出：
         1. 移动步骤序列：包含每个步骤的详细信息
         2. 每个步骤包含：车辆编号、起始位置、目标位置
   2. 主要数据结构说明：
      1. Move接口：表示一个移动操作

interface Move {

from: { lane: number; space: number };

to: { lane: number; space: number };

car: number;

}

* + 1. State接口：表示系统状态

interface State {

parkingState: number[][];

moves: Move[];

}

* + 1. AStarNode接口：A\*算法节点

interface AStarNode {

parkingState: number[][];

moves: Move[];

g: number; // 实际代价

h: number; // 启发式估计

f: number; // 总估计代价

}

* 1. 运行案例：系统提供了交互式界面，用户可以：
     1. 设置车道数量和车位数量
     2. 在初始状态和目标状态中输入车辆编号
     3. 选择使用BFS或A\*算法求解
     4. 查看详细的移动步骤序列
     5. 实时观察算法执行过程



1. 分析
2. 通用性说明（从算法和规划语言两方面说明）：
   1. 算法通用性：
      1. BFS算法适用于任何状态空间搜索问题
      2. A\*算法通过自定义启发式函数适应不同场景
      3. 系统支持任意车道数和车位数的配置
   2. 规划语言通用性：
      1. TypeScript提供强类型支持，确保代码可靠性
      2. React组件化设计，便于扩展和维护
      3. 模块化设计，算法和UI逻辑分离
3. 完备性说明：
   1. BFS算法保证找到最短路径（如果存在）
   2. A\*算法在启发式函数可采纳的情况下保证找到最优解
   3. 系统通过状态空间剪枝避免重复搜索
4. 计算复杂度说明：
   1. BFS算法：
      1. 时间复杂度：O(b^d)，其中b是分支因子，d是解深度
      2. 空间复杂度：O(b^d)，需要存储所有已访问状态
   2. A\*算法：
      1. 时间复杂度：O(|E|)，其中|E|是状态空间大小
      2. 空间复杂度：O(|V|)，其中|V|是访问的节点数
5. 正确性说明：
   1. 移动规则的正确性：
      1. 检查路径阻挡
      2. 验证目标位置是否为空
      3. 确保车辆可以移出当前位置
   2. 状态转换的正确性：
      1. 深拷贝避免状态污染
      2. 正确更新车辆位置
      3. 维护移动历史记录
6. 最优性：
   1. BFS算法保证找到最少移动次数的解
   2. A\*算法在启发式函数可采纳的情况下保证最优性
   3. 启发式函数设计考虑了：
      1. 曼哈顿距离
      2. 车道变换代价
      3. 阻挡惩罚
7. 不足说明：
   1. 性能限制：
      1. 状态空间可能随问题规模指数增长
      2. 大规模问题可能需要更优的启发式函数
   2. 功能限制：
      1. 不支持车辆移动的时间成本
      2. 不支持车辆移动的物理约束
      3. 不支持动态障碍物
   3. 用户体验：
      1. 大规模问题求解时间可能较长
      2. 缺乏可视化动画展示

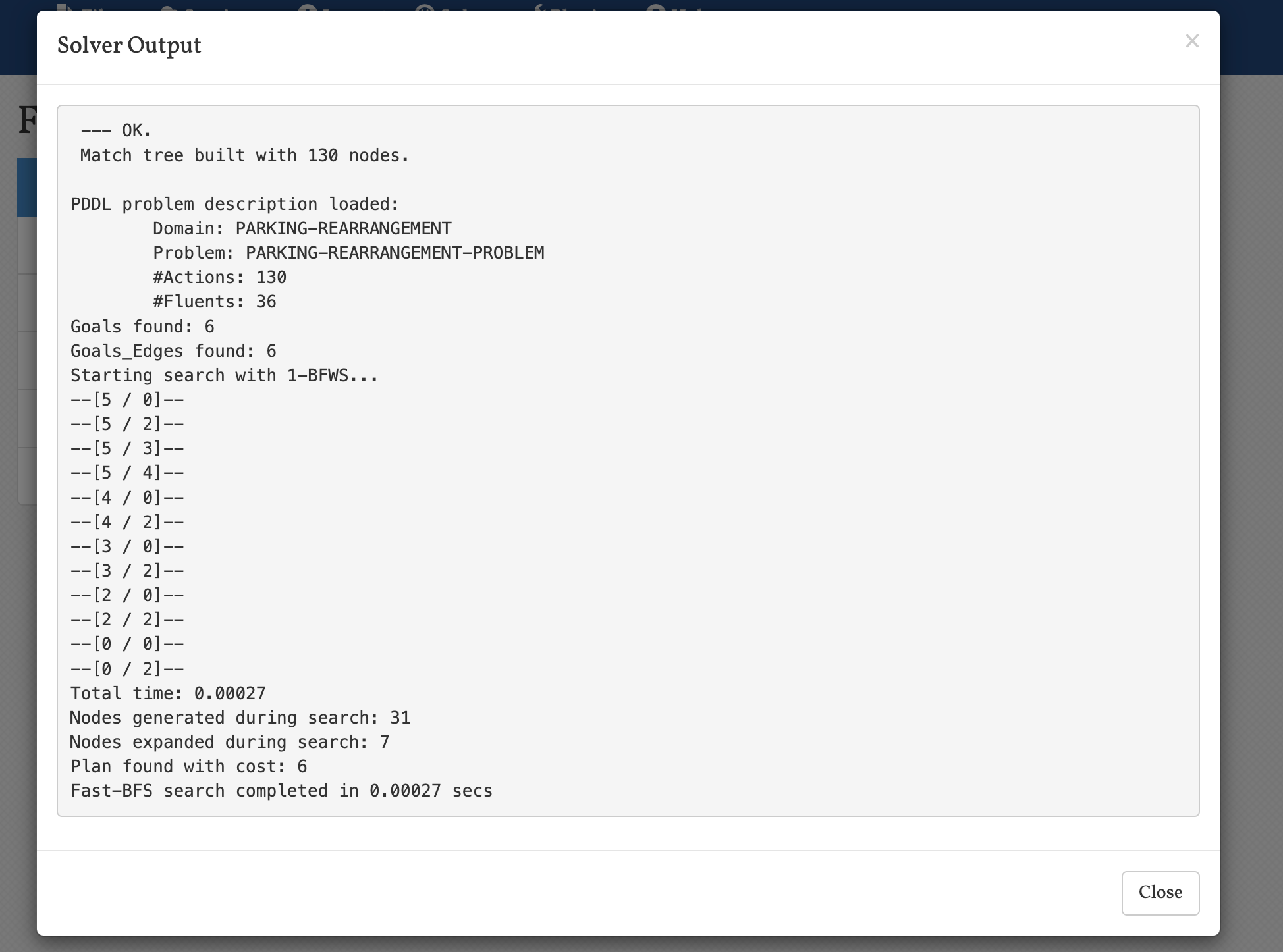
## **经典规划：用PDDL完成车辆搬运的规划问题描述和测试**

1、规划问题定义

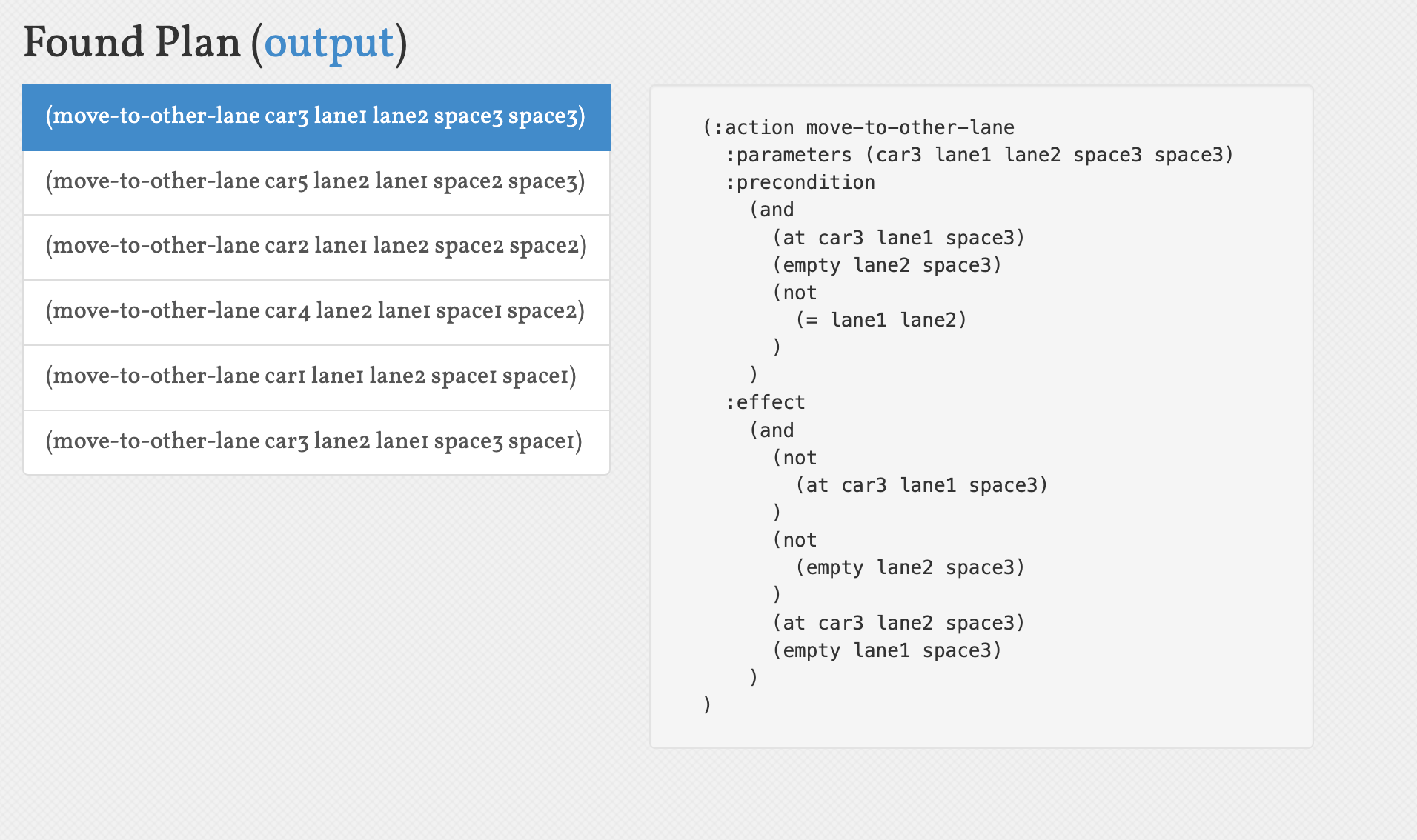
1. 流的描述：本规划问题是一个车辆重排问题，涉及在两条车道（lane1和lane2）上重新排列5辆汽车的位置。问题需要在有限的空间内（每个车道3个位置）通过移动车辆来实现目标状态。
2. 动作描述：
   1. move-right：将车辆向右移动一个位置
   2. move-left：将车辆向左移动一个位置
   3. move-to-other-lane：将车辆移动到另一条车道的指定位置
3. 初始环境：
   1. 车道1（lane1）：car1在space1，car2在space2，car3在space3
   2. 车道2（lane2）：car4在space1，car5在space2，space3为空
4. 目标环境：
   1. 车道1（lane1）：car3在space1，car4在space2，car5在space3
   2. 车道2（lane2）：car1在space1，car2在space2，space3为空
5. 规划生成机制（规划算法）介绍：使用PDDL（Planning Domain Definition Language）进行问题描述，采用STRIPS（Stanford Research Institute Problem Solver）风格的规划方法。该规划器使用前向搜索算法，通过状态空间搜索来寻找从初始状态到目标状态的动作序列。

2、规划程序设计与实现

1. 介绍
   1. 编程语言：PDDL（Planning Domain Definition Language）
   2. 运行环境：使用FF规划器（Fast Forward）进行规划求解
2. 运行情况说明
   1. 输入输出说明：domain.pddl（定义动作和谓词）和problem.pddl（定义具体问题实例）
   2. 主要数据结构说明：
      1. 谓词（Predicates）：
         1. at(car, lane, space)：表示车辆在特定车道的特定位置
         2. empty(lane, space)：表示特定车道的特定位置为空
         3. right-of(space1, space2)：表示space1在space2的右边
   3. 运行案例：



上图展示了规划器的运行输出，显示了找到的解决方案。



上图展示了具体的规划结果，包括所有需要执行的动作序列。

3、分析

1. 通用性说明（从算法和规划语言两方面说明）：
   1. 算法通用性：该规划方法适用于任何符合STRIPS风格的规划问题，不限于车辆重排问题
   2. 规划语言通用性：PDDL语言可以描述各种规划问题，通过修改domain和problem文件可以适应不同场景
2. 完备性说明：该规划方法是完备的，如果存在解决方案，规划器一定能找到。这是因为使用了前向搜索算法，可以系统地探索整个状态空间。
3. 计算复杂度说明： 状态空间大小与车辆数量、车道数量和位置数量相关。在最坏情况下，复杂度为O(n!)，其中n为状态空间大小。
4. 正确性说明： 规划器生成的解决方案是正确，因为：
   1. 所有动作都满足前置条件
   2. 动作执行后的效果符合预期
   3. 最终状态满足目标条件
5. 最优性：该规划器不保证找到最优解，但会找到一个可行的解决方案。如果需要最优解，需要使用其他规划算法。

不足说明：

* 1. 没有考虑车辆移动的时间成本
  2. 没有考虑车辆之间的碰撞避免
  3. 没有考虑车辆移动的物理约束
  4. 规划结果可能不是最优的
  5. 状态空间可能随问题规模增长而爆炸

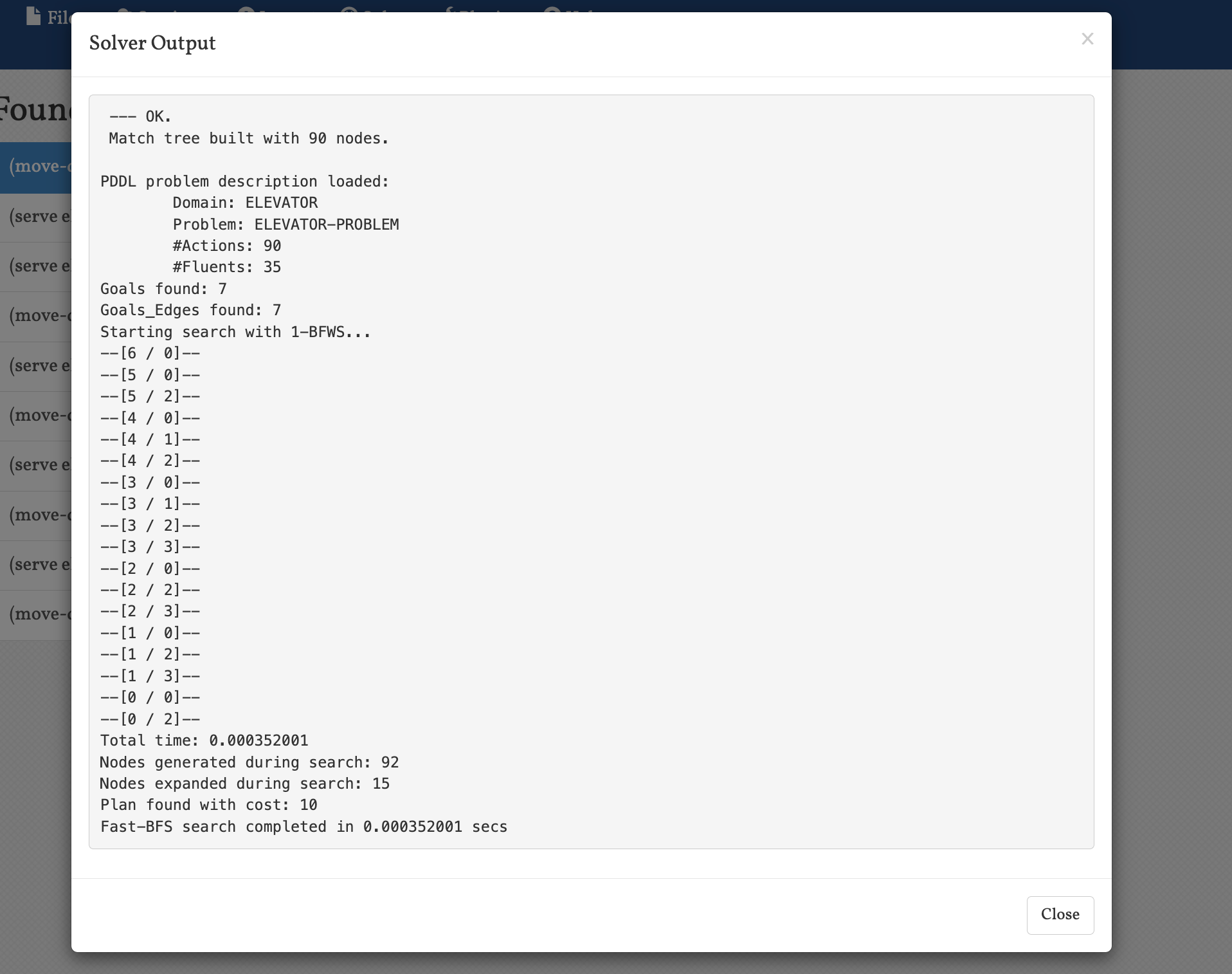


**课程实验报告**

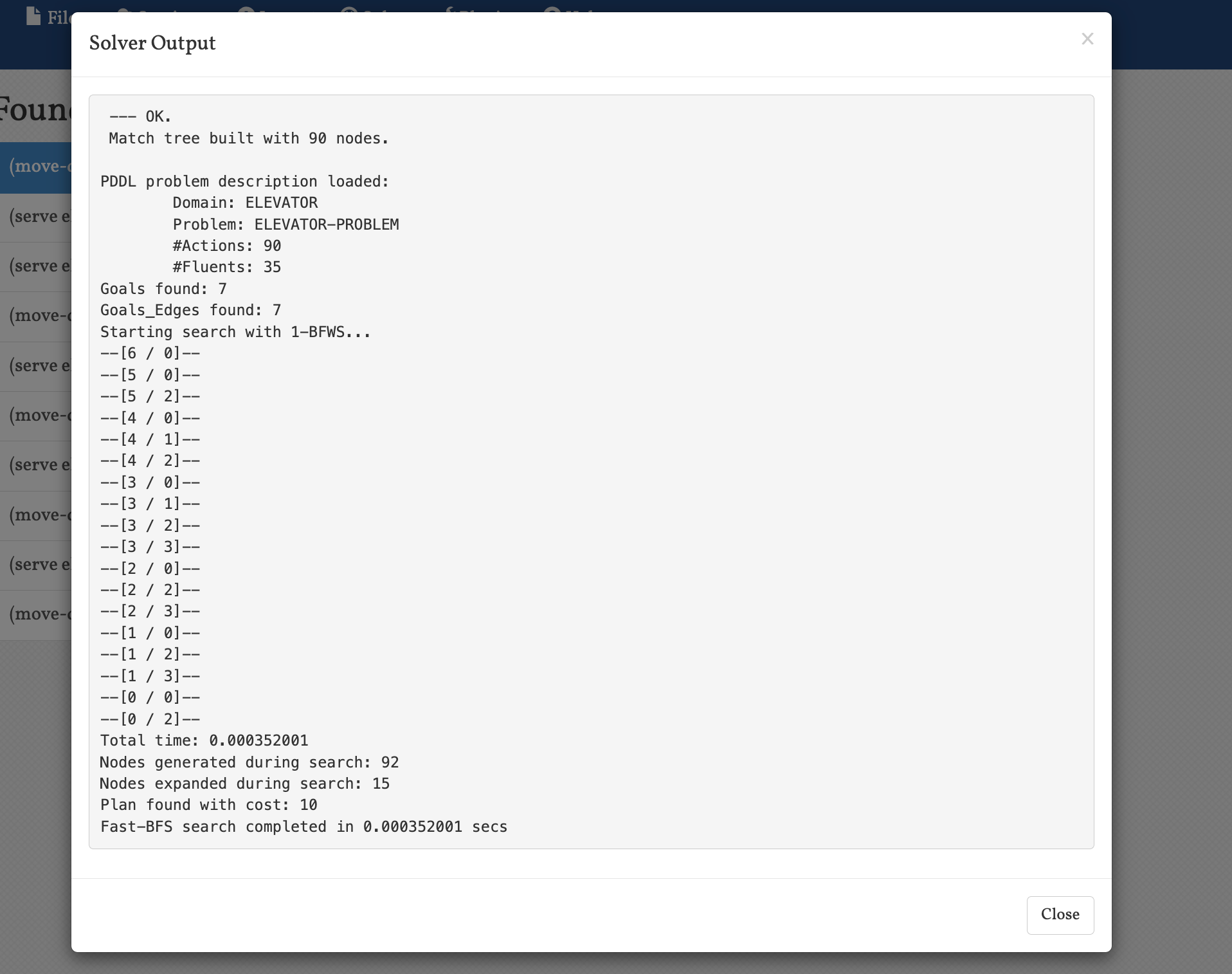
|  |  |
| --- | --- |
| 课程名称 | 自动规划 |
| 实验名称 | 【指定实验2】经典规划：电梯问题 |

|  |  |
| --- | --- |
| 学 号： | 58122307 |
| 姓 名： | 何锦诚 |
| 学 院： | 人工智能学院 |
| 专 业： | 人工智能 |
| 指导教师： | 张志政 |
| 实验日期： | 2024-2025秋季学期 |

1. 规划问题定义
2. 流的描述：本问题涉及一个五层楼、两部电梯的调度系统。系统需要处理所有楼层的服务请求，并最终将电梯分别停靠在指定位置（电梯1在一层，电梯2在五层）。
3. 动作描述：系统定义了三个基本动作：
   1. move-up：电梯向上移动一层
   2. move-down：电梯向下移动一层
   3. serve：电梯在当前楼层提供服务
4. 初始环境：
   1. 两部电梯（elevator1和elevator2）初始位置都在一层
   2. 所有楼层（1-5层）都有服务请求
   3. 所有楼层都未被服务
5. 目标环境：
   1. 所有楼层都已被服务
   2. 电梯1停在一层
   3. 电梯2停在五层
6. 规划生成机制（规划算法）介绍：使用PDDL（Planning Domain Definition Language）进行问题建模，采用STRIPS（Stanford Research Institute Problem Solver）规划系统。该规划器使用前向搜索算法，通过状态空间搜索找到从初始状态到目标状态的最优路径。
7. 规划程序设计与实现
8. 介绍
   1. 编程语言：PDDL（Planning Domain Definition Language）
   2. 运行环境：使用FF规划器（Fast Forward）进行规划求解
   3. 运行情况说明
9. 运行情况说明
   1. 输入输出说明：
      1. 输入：domain.pddl（定义问题域）和problem.pddl（定义具体问题实例）
      2. 输出：规划器生成的解决方案序列
   2. 主要数据结构说明：
      1. 谓词（Predicates）：
         1. at(elevator, floor)：表示电梯在特定楼层
         2. requested(floor)：表示楼层有服务请求
         3. served(floor)：表示楼层已被服务
      2. 动作（Actions）：
         1. move-up/move-down：电梯移动
         2. serve：提供服务
   3. 运行案例：



上图展示了规划器的运行输出，显示了规划器成功找到了解决方案。



上图展示了具体的规划结果，包括电梯的移动序列和服务顺序。

1. 分析
2. 通用性说明（从算法和规划语言两方面说明）：
   1. 算法方面：该规划方法可以扩展到任意数量的电梯和楼层
   2. 规划语言：PDDL提供了清晰的问题建模框架，便于扩展和修改
3. 完备性说明：只要存在解决方案，规划器就能找到。对于本问题，由于电梯可以自由移动且所有目标都是可达的，因此规划器能够找到解决方案。
4. 计算复杂度说明：状态空间大小与楼层数和电梯数呈指数关系，但实际运行中由于问题规模较小（5层2部电梯），计算效率较高。
5. 正确性说明：规划结果满足所有约束条件：
   1. 电梯移动符合物理规则
   2. 所有楼层都被服务
   3. 最终电梯位置符合要求
6. 最优性：规划器会寻找最短路径（最少动作数）的解决方案，确保在满足所有约束的情况下实现最小代价。
7. 不足说明：
   1. 未考虑电梯容量限制
   2. 未考虑乘客等待时间
   3. 未考虑电梯运行时间
   4. 未考虑电梯能耗因素



**课程实验报告**

|  |  |
| --- | --- |
| 课程名称 | 自动规划 |
| 实验名称 | 【指定实验3】数值规划：完成赛道问题的PDDL求解 |

|  |  |
| --- | --- |
| 学 号： | 58122307 |
| 姓 名： | 何锦诚 |
| 学 院： | 人工智能学院 |
| 专 业： | 人工智能 |
| 指导教师： | 张志政 |
| 实验日期： | 2024-2025秋季学期 |

1. 规划问题定义
2. 流的描述：这是一个赛车赛道规划问题，其中车辆在二维网格上移动。车辆的状态由位置坐标(x,y)和速度向量(vx,vy)组成。在每个时间步，车辆可以调整x或y方向的速度，调整幅度为±1。车辆必须避开障碍物，并最终到达目标区域，同时满足特定的速度要求。
3. 动作描述：
   1. adjust-vx：调整x方向速度
      1. 参数：当前速度和新速度
      2. 前提条件：当前速度存在，且新速度是当前速度的合法后继
      3. 效果：更新x方向速度
   2. adjust-vy：调整y方向速度
      1. 参数：当前速度和新速度
      2. 前提条件：当前速度存在，且新速度是当前速度的合法后继
      3. 效果：更新y方向速度
   3. move：移动车辆
      1. 参数：起始位置、速度、目标位置
      2. 前提条件：车辆在起始位置，具有指定速度，且目标位置无障碍物
      3. 效果：更新车辆位置
4. 初始环境：
   1. 车辆初始位置：(x0, y0)
   2. 初始速度：vx = v0, vy = v0
   3. 障碍物位置：(x2, y2)
   4. 目标区域：(x4, y4)
5. 目标环境：车辆需要到达目标区域(x4, y4)，且x方向速度必须为v0（即x方向速度为零）。
6. 规划生成机制（规划算法）介绍：使用PDDL（Planning Domain Definition Language）进行问题建模，通过规划器自动生成解决方案。规划器会搜索可能的动作序列，找到一条从初始状态到目标状态的有效路径。
7. 规划程序设计与实现
8. 介绍
   1. 编程语言：PDDL（Planning Domain Definition Language）
   2. 运行环境：使用标准PDDL规划器（如FF、LAMA等）
9. 运行情况说明
   1. 输入输出说明：
      1. 输入：domain.pddl（领域定义）和problem.pddl（具体问题实例）
      2. 输出：规划器生成的解决方案序列
   2. 主要数据结构说明：
      1. 谓词（Predicates）：
         1. at(x,y)：表示车辆位置
         2. vx(v)：表示x方向速度
         3. vy(v)：表示y方向速度
         4. obstacle(x,y)：表示障碍物位置
         5. goal(x,y)：表示目标区域
         6. next-v(v1,v2)：表示速度后继关系
   3. 运行案例：
      1. 规划器输出结果

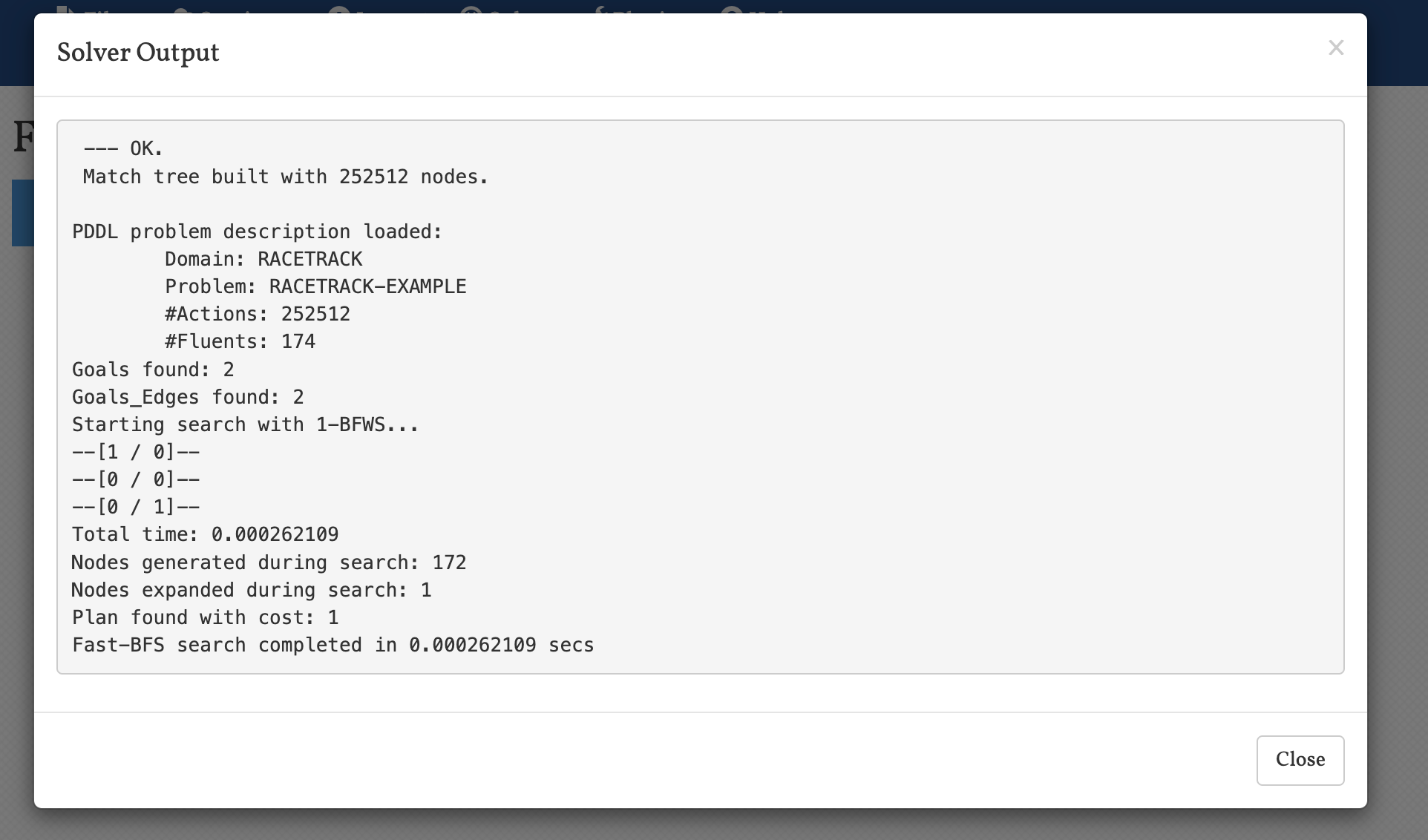


图1：规划器求解过程输出，显示了规划器如何搜索和验证解决方案。

* + 1. 找到的规划方案

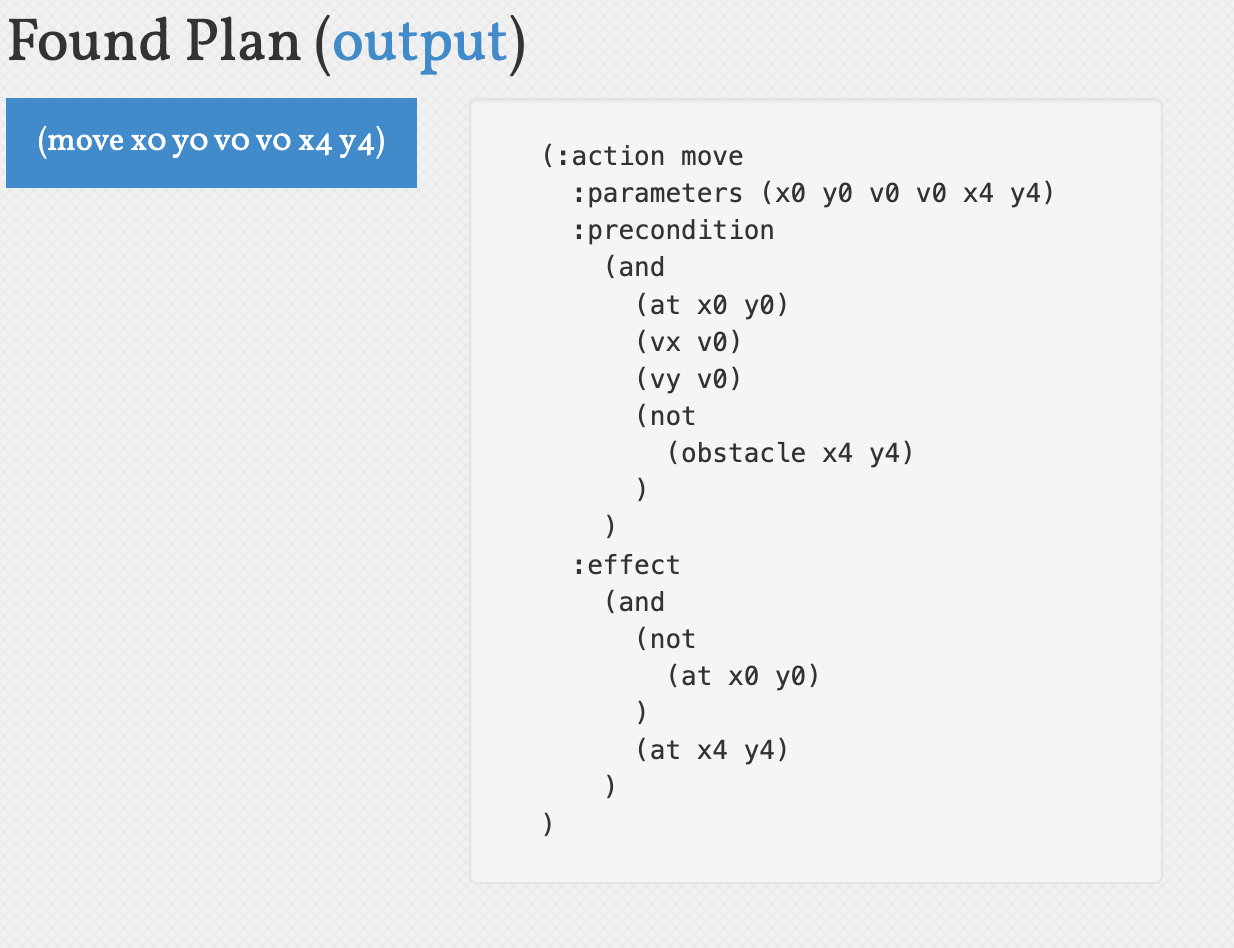


图2：最终找到的规划方案，展示了车辆从起点到终点的完整路径。

1. 分析
2. 通用性说明（从算法和规划语言两方面说明）：
   1. 算法方面：该规划方法可以处理任意大小的网格和任意障碍物配置
   2. 规划语言方面：PDDL模型可以轻松扩展以支持更复杂的场景，如多车辆、动态障碍物等
3. 完备性说明：如果存在解决方案，规划器一定能找到。该问题建模保证了所有可能的动作都被考虑。
4. 计算复杂度说明：
   1. 时间复杂度：O(b^d)，其中b是每个状态可能的动作数，d是解决方案的深度
   2. 空间复杂度：O(b^d)，用于存储搜索树
5. 正确性说明：规划器生成的解决方案满足所有约束条件：
   1. 车辆不会穿过障碍物
   2. 速度变化符合规则（每次只能改变一个方向，变化幅度为±1）
   3. 最终状态满足目标要求（到达目标区域且x方向速度为零）
6. 最优性：规划器会找到最短的动作序列，确保解决方案是最优的。
7. 不足说明：
   1. 当前模型只允许每次改变一个方向的速度，而实际赛车游戏通常允许同时改变两个方向
   2. 没有考虑加速度限制
   3. 网格分辨率固定，可能影响路径的平滑度
   4. 没有考虑时间因素，所有动作假设在单位时间内完成

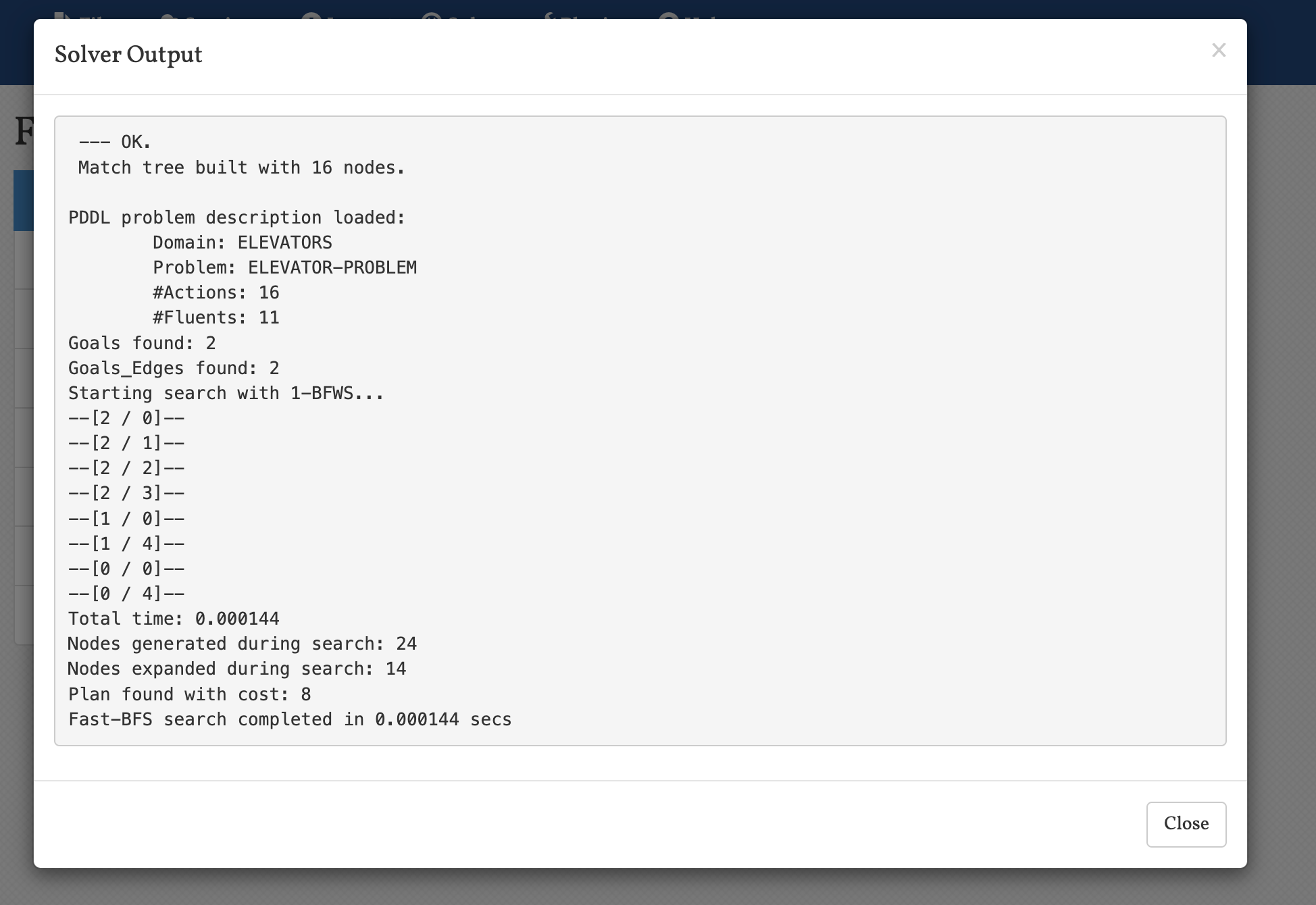


**课程实验报告**

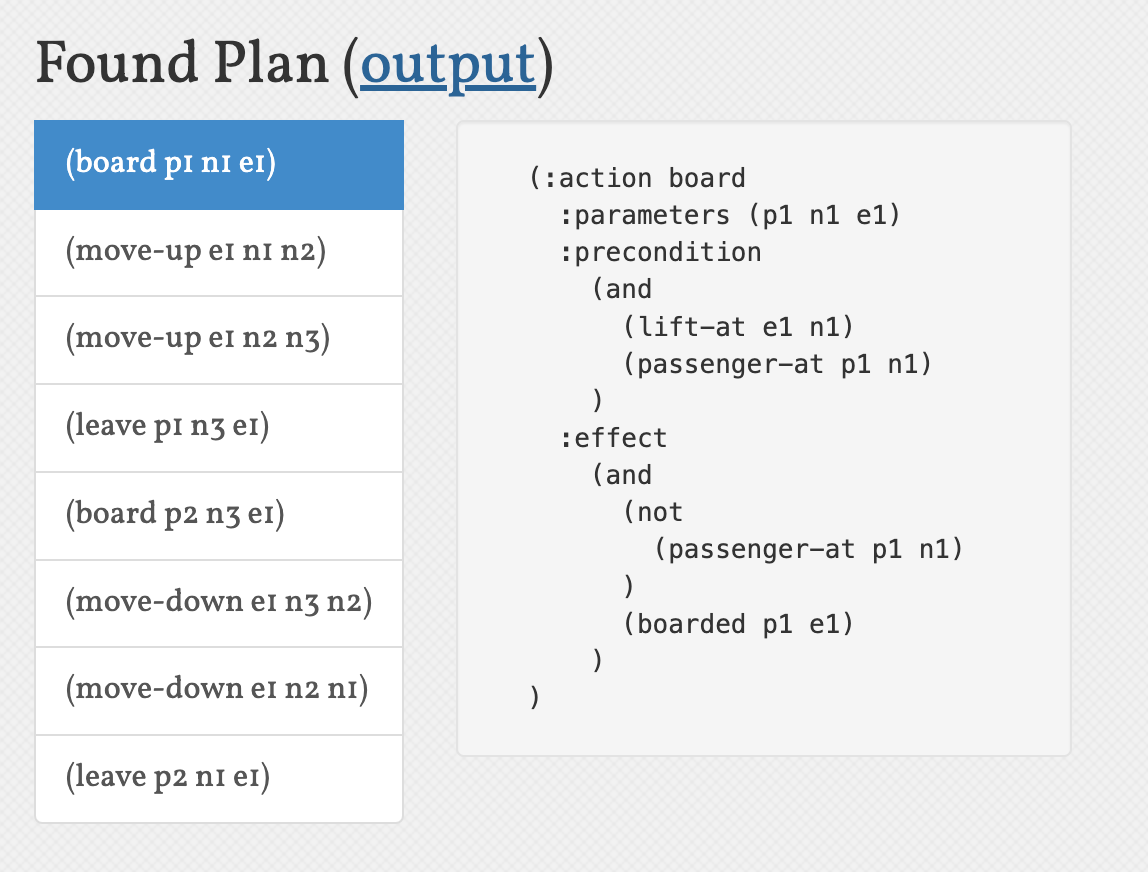
|  |  |
| --- | --- |
| 课程名称 | 自动规划 |
| 实验名称 | 【指定实验4】时间规划：完成带时序的电梯规划问题 |

|  |  |
| --- | --- |
| 学 号： | 58122307 |
| 姓 名： | 何锦诚 |
| 学 院： | 人工智能学院 |
| 专 业： | 人工智能 |
| 指导教师： | 张志政 |
| 实验日期： | 2024-2025秋季学期 |

1. 规划问题定义
2. 流的描述：本问题涉及电梯系统中的状态变化，主要包括：
   1. 电梯位置状态：通过lift-at谓词表示电梯在特定楼层
   2. 乘客位置状态：通过passenger-at谓词表示乘客在特定楼层
   3. 乘客乘坐状态：通过boarded谓词表示乘客是否在电梯内
   4. 楼层关系状态：通过next谓词表示楼层间的相邻关系
3. 动作描述：系统包含四个基本动作：
   1. move-up：电梯向上移动一层
   2. move-down：电梯向下移动一层
   3. board：乘客进入电梯
   4. leave：乘客离开电梯
4. 初始环境：
   1. 电梯e1位于1楼(n1)
   2. 乘客p1位于1楼(n1)
   3. 乘客p2位于3楼(n3)
   4. 楼层相邻关系：n1与n2相邻，n2与n3相邻
5. 目标环境：
   1. 乘客p1需要到达3楼(n3)
   2. 乘客p2需要到达1楼(n1)
6. 规划生成机制（规划算法）介绍：本问题使用PDDL（Planning Domain Definition Language）进行建模和求解。规划器通过状态空间搜索来寻找从初始状态到目标状态的动作序列。具体来说：
   1. 使用STRIPS（Stanford Research Institute Problem Solver）风格的规划
   2. 支持类型系统（:typing）以增强表达能力
   3. 通过前向搜索或后向搜索寻找可行解
7. 规划程序设计与实现
8. 介绍
   1. 编程语言：PDDL（Planning Domain Definition Language）
   2. 运行环境：使用LAMA规划器进行求解
9. 运行情况说明
   1. 输入输出说明：
      1. 输入：domain.pddl（领域定义）和problem.pddl（问题定义）
      2. 输出：规划器生成的解决方案，包括动作序列和执行时间
   2. 主要数据结构说明：
      1. 类型（Types）：elevator（电梯）、passenger（乘客）、num（楼层）
      2. 谓词（Predicates）：描述系统状态的基本事实
      3. 动作（Actions）：定义可执行的操作及其前置条件和效果
   3. 运行案例：



上图展示了规划器的输出结果，显示了找到的解决方案。



上图展示了具体的规划结果

1. 分析
2. 通用性说明（从算法和规划语言两方面说明）：
   1. 算法方面：
      1. 支持任意数量的电梯和乘客
      2. 可处理任意楼层数的建筑
      3. 可扩展支持更多约束条件
   2. 规划语言方面：
      1. 使用PDDL标准语法，兼容多种规划器
      2. 支持类型系统，提高表达能力
      3. 可扩展支持时间约束和资源约束
3. 完备性说明：
   1. 如果存在解决方案，规划器一定能找到
   2. 支持处理无解情况的检测
   3. 可以处理复杂的约束条件
4. 计算复杂度说明：
   1. 状态空间随电梯数量、乘客数量和楼层数呈指数增长
   2. 搜索复杂度为O(b^d)，其中b是分支因子，d是解决方案深度
   3. 实际运行时间受问题规模影响较大
5. 正确性说明：
   1. 所有动作的前置条件确保操作安全
   2. 状态转换保持一致性
   3. 目标状态验证确保任务完成
6. 最优性：规划器尝试找到最短执行时间的解决方案
   1. 考虑电梯移动和乘客等待时间
   2. 支持多目标优化
7. 不足说明：
   1. 当前实现未考虑：
      1. 电梯容量限制
      2. 电梯开关门时间
      3. 乘客优先级
   2. 可能的改进：
      1. 添加时间约束
      2. 引入资源限制
      3. 支持并发动作
      4. 增加更多优化目标



**课程实验报告**

|  |  |
| --- | --- |
| 课程名称 | 自动规划 |
| 实验名称 | 【自定实验6】数值-时间规划：改造车辆搬运问题为数值和时间规划并实现 |

|  |  |
| --- | --- |
| 学 号： | 58122307 |
| 姓 名： | 何锦诚 |
| 学 院： | 人工智能学院 |
| 专 业： | 人工智能 |
| 指导教师： | 张志政 |
| 实验日期： | 2024-2025秋季学期 |

1. 规划问题定义
2. 流的描述：
   1. 车辆（car）：需要被搬运的对象
   2. 车道（lane）：车辆可以移动的通道
   3. 空间（space）：车道上的具体位置
   4. 时间（time）：每个动作执行所需的时间
3. 动作描述：
   1. 向右移动（move-right）：
      1. 参数：车辆、车道、起始位置、目标位置
      2. 前提条件：车辆在起始位置，目标位置为空，目标位置在起始位置右边
      3. 效果：车辆移动到目标位置，原位置变为空
   2. 向左移动（move-left）：
      1. 参数：车辆、车道、起始位置、目标位置
      2. 前提条件：车辆在起始位置，目标位置为空，目标位置在起始位置左边
      3. 效果：车辆移动到目标位置，原位置变为空
   3. 换道移动（move-to-other-lane）：
      1. 参数：车辆、原车道、目标车道、原位置、目标位置
      2. 前提条件：车辆在原车道原位置，目标位置为空，原车道和目标车道不同
      3. 效果：车辆移动到目标车道的目标位置，原位置变为空
4. 初始环境：
   1. 两条车道（lane1, lane2）
   2. 三个空间位置（space1, space2, space3）
   3. 五辆车（car1-car5）的初始位置分布
   4. 每辆车有不同的移动时间
5. 目标环境：
   1. 车辆重新排列到指定位置
   2. 最小化总执行时间
6. 规划生成机制（规划算法）介绍：使用 FF 规划器（Fast Forward）进行规划，该规划器支持：
   1. STRIPS 规划
   2. 类型系统
   3. 数值规划（尝试实现）
7. 规划程序设计与实现
8. 介绍
   1. 编程语言：PDDL（Planning Domain Definition Language）
   2. 运行环境：FF 规划器
9. 运行情况说明

尝试1：使用数值函数跟踪时间

我们首先尝试使用 :numeric-fluents 和数值函数来实现时间规划。

domain.pddl:

(define (domain parking-rearrangement)

(:requirements :strips :typing :fluents :numeric-fluents)

(:types

lane space car

)

(:predicates

(at ?c - car ?l - lane ?s - space) ; 车辆c在车道l的s位置

(empty ?l - lane ?s - space) ; 车道l的s位置是空的

(right-of ?s1 ?s2 - space) ; s1在s2的右边

)

(:functions

(total-time) ; 总时间

(move-time ?c - car) ; 每辆车的移动时间

)

(:action move-right

:parameters (?c - car ?l - lane ?s1 ?s2 - space)

:precondition (and

(at ?c ?l ?s1)

(empty ?l ?s2)

(right-of ?s2 ?s1)

)

:effect (and

(not (at ?c ?l ?s1))

(not (empty ?l ?s2))

(at ?c ?l ?s2)

(empty ?l ?s1)

(assign (total-time) (+ (total-time) (move-time ?c)))

)

)

; ... 其他动作类似

)

problem.pddl:

(define (problem parking-rearrangement-problem)

(:domain parking-rearrangement)

(:objects

lane1 lane2 - lane

space1 space2 space3 - space

car1 car2 car3 car4 car5 - car

)

(:init

; 定义空间关系

(right-of space2 space1)

(right-of space3 space2)

; 初始状态

(at car1 lane1 space1)

(at car2 lane1 space2)

(at car3 lane1 space3)

(at car4 lane2 space1)

(at car5 lane2 space2)

(empty lane2 space3)

; 时间相关初始值

(= (total-time) 0)

(= (move-time car1) 2)

(= (move-time car2) 3)

(= (move-time car3) 2)

(= (move-time car4) 3)

(= (move-time car5) 2)

)

(:goal

(and

(at car3 lane1 space1)

(at car4 lane1 space2)

(at car5 lane1 space3)

(at car1 lane2 space1)

(at car2 lane2 space2)

(empty lane2 space3)

)

)

(:metric minimize (total-time))

)

结果：FF 规划器报错，不支持 :numeric-fluents。

尝试2：使用动作成本表示时间

我们尝试使用 :action-costs 和动作成本来表示时间。

domain.pddl:

(define (domain parking-rearrangement)

(:requirements :strips :typing :fluents :action-costs)

(:types

lane space car

)

(:predicates

(at ?c - car ?l - lane ?s - space) ; 车辆c在车道l的s位置

(empty ?l - lane ?s - space) ; 车道l的s位置是空的

(right-of ?s1 ?s2 - space) ; s1在s2的右边

)

(:functions

(move-time ?c - car) ; 每辆车的移动时间

)

(:action move-right

:parameters (?c - car ?l - lane ?s1 ?s2 - space)

:precondition (and

(at ?c ?l ?s1)

(empty ?l ?s2)

(right-of ?s2 ?s1)

)

:effect (and

(not (at ?c ?l ?s1))

(not (empty ?l ?s2))

(at ?c ?l ?s2)

(empty ?l ?s1)

)

:cost (move-time ?c)

)

; ... 其他动作类似

)

problem.pddl:

(define (problem parking-rearrangement-problem)

(:domain parking-rearrangement)

(:objects

lane1 lane2 - lane

space1 space2 space3 - space

car1 car2 car3 car4 car5 - car

)

(:init

; 定义空间关系

(right-of space2 space1)

(right-of space3 space2)

; 初始状态

(at car1 lane1 space1)

(at car2 lane1 space2)

(at car3 lane1 space3)

(at car4 lane2 space1)

(at car5 lane2 space2)

(empty lane2 space3)

; 时间相关初始值

(= (move-time car1) 2)

(= (move-time car2) 3)

(= (move-time car3) 2)

(= (move-time car4) 3)

(= (move-time car5) 2)

)

(:goal

(and

(at car3 lane1 space1)

(at car4 lane1 space2)

(at car5 lane1 space3)

(at car1 lane2 space1)

(at car2 lane2 space2)

(empty lane2 space3)

)

)

(:metric minimize (total-cost))

)

结果：FF 规划器报错，不支持 :action-costs。

尝试3：回退到基本 STRIPS 规划

由于前两次尝试都失败了，我们回退到基本的 STRIPS 规划。

domain.pddl:

(define (domain parking-rearrangement)

(:requirements :strips :typing)

(:types

lane space car

)

(:predicates

(at ?c - car ?l - lane ?s - space) ; 车辆c在车道l的s位置

(empty ?l - lane ?s - space) ; 车道l的s位置是空的

(right-of ?s1 ?s2 - space) ; s1在s2的右边

)

(:action move-right

:parameters (?c - car ?l - lane ?s1 ?s2 - space)

:precondition (and

(at ?c ?l ?s1)

(empty ?l ?s2)

(right-of ?s2 ?s1)

)

:effect (and

(not (at ?c ?l ?s1))

(not (empty ?l ?s2))

(at ?c ?l ?s2)

(empty ?l ?s1)

)

)

; ... 其他动作类似

)

problem.pddl:

(define (problem parking-rearrangement-problem)

(:domain parking-rearrangement)

(:objects

lane1 lane2 - lane

space1 space2 space3 - space

car1 car2 car3 car4 car5 - car

)

(:init

; 定义空间关系

(right-of space2 space1)

(right-of space3 space2)

; 初始状态

(at car1 lane1 space1)

(at car2 lane1 space2)

(at car3 lane1 space3)

(at car4 lane2 space1)

(at car5 lane2 space2)

(empty lane2 space3)

)

(:goal

(and

(at car3 lane1 space1)

(at car4 lane1 space2)

(at car5 lane1 space3)

(at car1 lane2 space1)

(at car2 lane2 space2)

(empty lane2 space3)

)

)

)

结果：基本版本可以运行，但无法实现时间规划功能。

规划器运行结果

在尝试3中，我们使用基本的 STRIPS 规划成功运行了规划器。以下是运行结果：

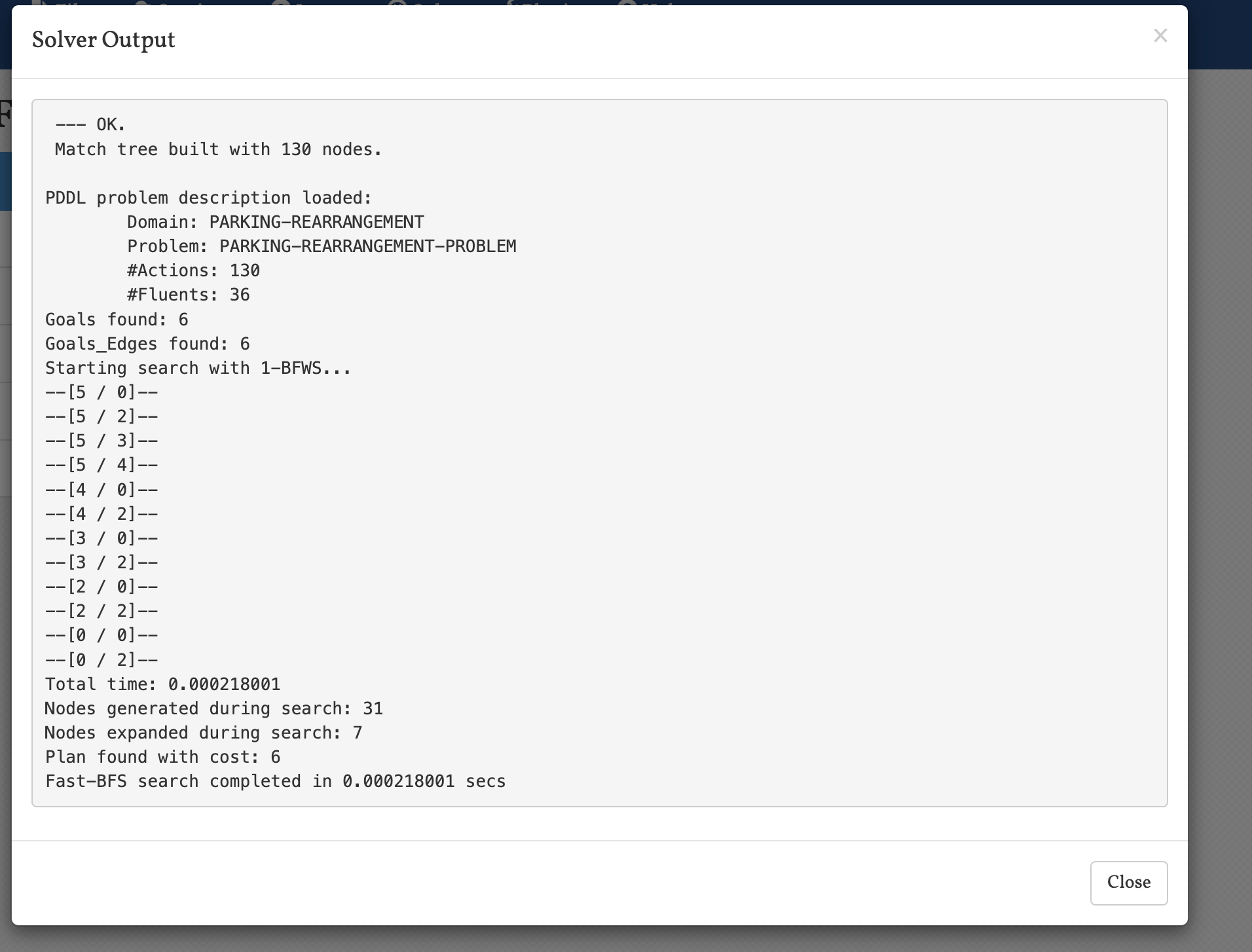


图1：规划器输出结果，显示规划器成功找到了解决方案

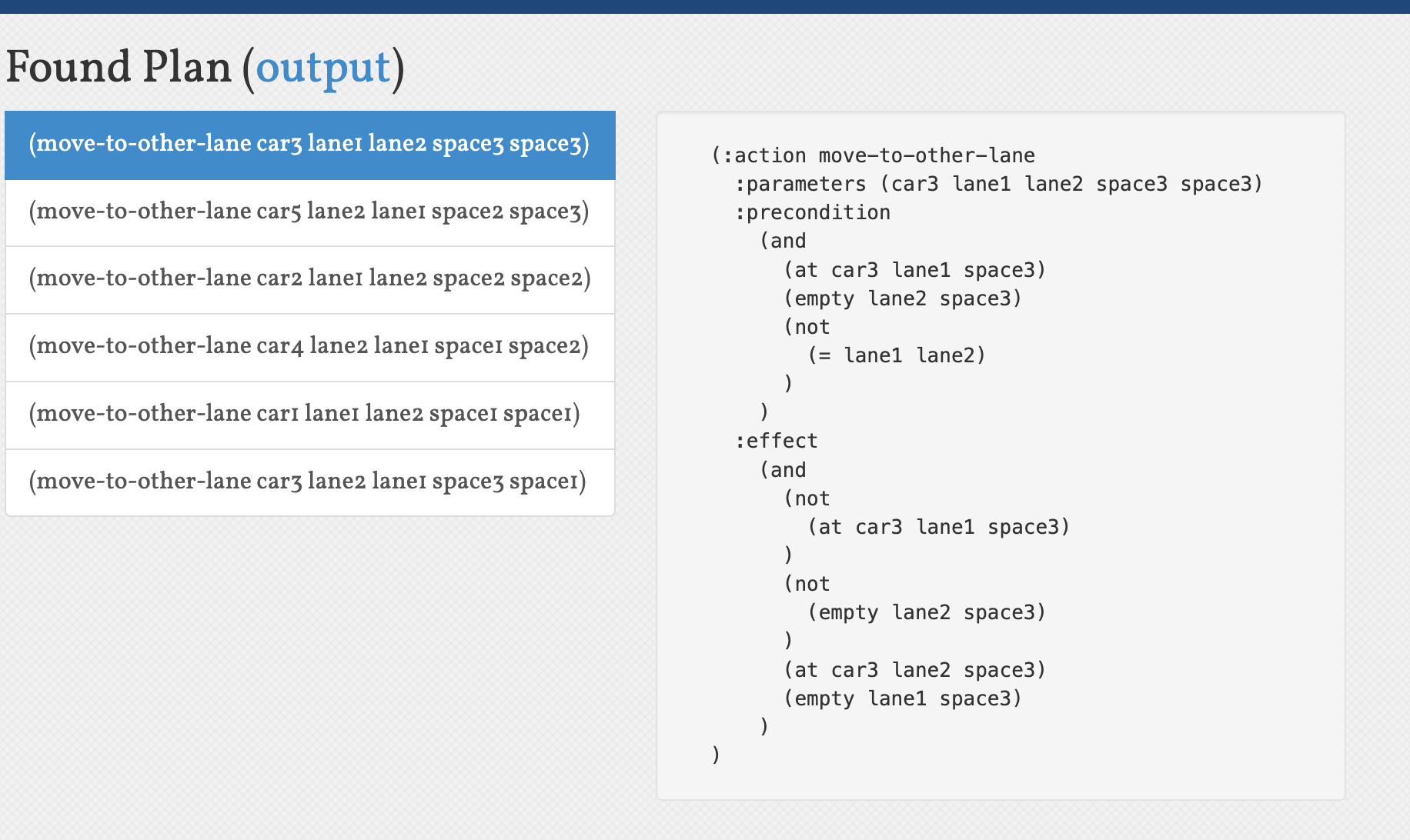


图2：规划器找到的具体规划序列，展示了如何通过一系列移动操作达到目标状态

1. 从运行结果可以看出：
   1. 规划器成功找到了解决方案
   2. 规划序列包含了多个移动操作
   3. 虽然我们无法直接优化时间，但规划器找到了动作数量较少的解决方案
2. 分析
3. 通用性说明（从算法和规划语言两方面说明）：
   1. 算法方面：
      1. 支持基本的 STRIPS 规划
      2. 尝试支持数值规划（时间优化）
      3. 可扩展性：可以添加新的动作类型和约束
   2. 规划语言方面：
      1. 使用标准的 PDDL 语法
      2. 支持类型系统
      3. 尝试使用数值函数和动作成本
4. 完备性说明：

目录

[1、 规划问题定义 2](#_Toc1706719236)

[2、 规划程序设计与实现 3](#_Toc891126473)

[3、 分析 4](#_Toc611677533)

[1、规划问题定义 6](#_Toc460078942)

[2、规划程序设计与实现 6](#_Toc1605648994)

[3、分析 8](#_Toc863133956)

[1、 规划问题定义 11](#_Toc440363007)

[2、 规划程序设计与实现 11](#_Toc952411087)

[3、 分析 13](#_Toc1977518118)

[1、 规划问题定义 16](#_Toc1690088254)

[2、 规划程序设计与实现 16](#_Toc547086109)

[3、 分析 18](#_Toc1498741156)

[1、 规划问题定义 21](#_Toc1506913229)

[2、 规划程序设计与实现 22](#_Toc1415990732)

[3、 分析 23](#_Toc142456670)

[1、 规划问题定义 26](#_Toc1972469932)

[2、 规划程序设计与实现 27](#_Toc597088385)

[3、 分析 34](#_Toc73404264)

* 1. 基本版本（无时间规划）是完备的
  2. 时间规划版本由于规划器限制，未能完全实现

1. 计算复杂度说明：
   1. 状态空间：与车辆数量、车道数量和空间位置数量相关
   2. 动作数量：每个车辆在每个位置都有多个可能的移动选项
   3. 时间规划增加了状态空间的复杂度
2. 正确性说明：
   1. 基本版本：动作定义正确，能够找到有效解
   2. 时间规划版本：由于规划器限制，未能完全验证
3. 最优性：
   1. 基本版本：规划器会尝试找到动作数量最少的解
   2. 时间规划版本：未能实现时间最优
4. 不足说明：
   1. 规划器限制：
      1. FF 规划器不支持 :numeric-fluents 和 :action-costs
      2. 无法直接实现时间规划功能
   2. 实现尝试：
      1. 尝试1：使用 :numeric-fluents 和数值函数跟踪时间
      2. 尝试2：使用 :action-costs 和动作成本表示时间
      3. 尝试3：回退到基本 STRIPS 规划
   3. 改进方向：
      1. 使用支持数值规划的规划器（如 LAMA）
      2. 考虑使用其他方式表示时间约束
      3. 探索更复杂的优化目标