# 概述

参考：<https://zhuanlan.zhihu.com/p/674095744>

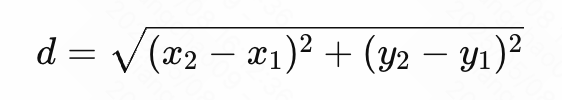
# 路径

参考：<https://www.cnblogs.com/AlvinSui/p/8931074.html>

## 欧氏距离(Euclidean Distance)

1、公式

欧几里得距离是最常见的直线距离计算方法，适用于平面上的两点之间。其计算公式为：



2、适用场景：

城市之间的平面距离估算。

无需考虑地球曲率的近似计算。

3、优缺点：

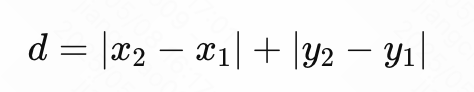
优点：计算简单，速度快。

缺点：不适用于大范围地理计算，忽略地球曲率。

## 曼哈顿距离(Manhattan Distance)

1. 公式

曼哈顿距离计算两点之间沿坐标轴的总距离，适用于网格状城市布局。其计算公式为：



2、适用场景：

城市道路呈网格状布局，如纽约等城市。

机器人路径规划中的网格环境。

3、优缺点：

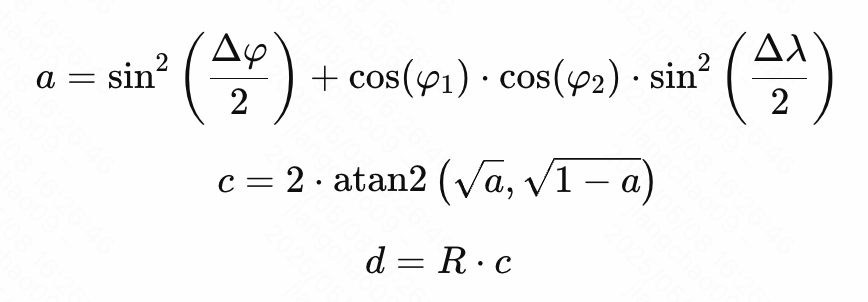
优点：适用于网格状环境，计算简单。

缺点：不适用于自然地形复杂的区域。

## 哈弗辛距离(Haversine Distance)

1、公式

哈弗辛公式用于计算球面上两点之间的最短距离，考虑地球曲率。其计算公式为：



其中，𝑅为地球半径，𝜑1,𝜑2为两点的纬度，𝜆1,𝜆2为经度。

2、适用场景：

城市间的地理距离计算。

导航系统中的路径规划。

3、优缺点：

优点：准确计算球面距离，适用于大范围地理计算。

缺点：计算复杂度较高。

## 大圆距离(Great Circle Distance)

1、原理：

大圆距离是球面上两点之间的最短路径，考虑地球曲率。其计算方法与哈弗辛距离类似，但使用不同的公式。

2、适用场景：

飞行航线规划。

航空公司调度系统。

3、优缺点：

优点：准确计算球面最短路径。

缺点：计算复杂度较高。

## 加权距离(Weighted Distance)

1、原理：

在计算距离时，考虑不同区域的权重，如交通状况、地形等因素。其计算公式为：

𝑑weighted = 𝑑 ⋅ 𝑤

其中，𝑤为权重因子。

2、适用场景：

城市交通网络中的路径规划。

考虑地形、交通等因素的路径优化。

3、优缺点：

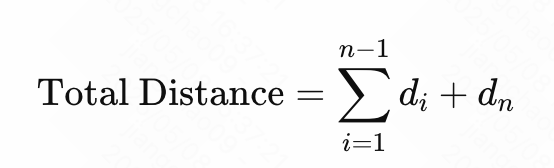
优点：更贴近实际情况，考虑更多因素。

缺点：需要额外的数据支持，计算复杂度增加。

## 旅行商问题（TSP）距离

1、原理：

旅行商问题是寻找一条最短路径，经过每个城市一次且仅一次，最终返回起点。其计算方式为：



其中，𝑑𝑖为城市间的距离，𝑛为城市数量。

2、适用场景：

快递配送路径优化。

销售路线规划。

3、优缺点：

优点：提供最优路径。

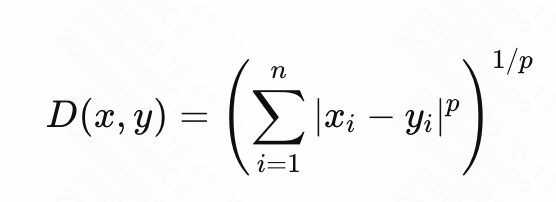
缺点：计算复杂度高，适用于城市数量较少的情况。

## 切比雪夫距离(Chebyshev Distance)

闵可夫斯基距离（Minkowski Distance）是一种度量两个点之间距离的通用方法，是欧几里得距离和曼哈顿距离的推广形式。它在机器学习、聚类、推荐系统等领域中广泛使用。

1、原理

闵可夫斯基距离的公式如下：



其中：

x = (x1, x2, ..., xn)、y = (y1, y2, ..., yn)：表示两个n维空间的点；

p：参数，决定了具体的距离类型。

常见的特例：

当 p = 1：曼哈顿距离（Manhattan Distance）

当 p = 2：欧几里得距离（Euclidean Distance）

当 p -> ∞：切比雪夫距离（Chebyshev Distance）

2、适用场景

1）聚类分析（如K-Means）：

可根据不同的 p 值选择合适的距离度量。

2）推荐系统：

用于度量用户/物品之间的相似度。

3）分类算法（如K近邻KNN）：

KNN可以灵活选择不同的 p 值来优化分类效果。

4）图像处理：

不同距离度量影响图像相似度判断。

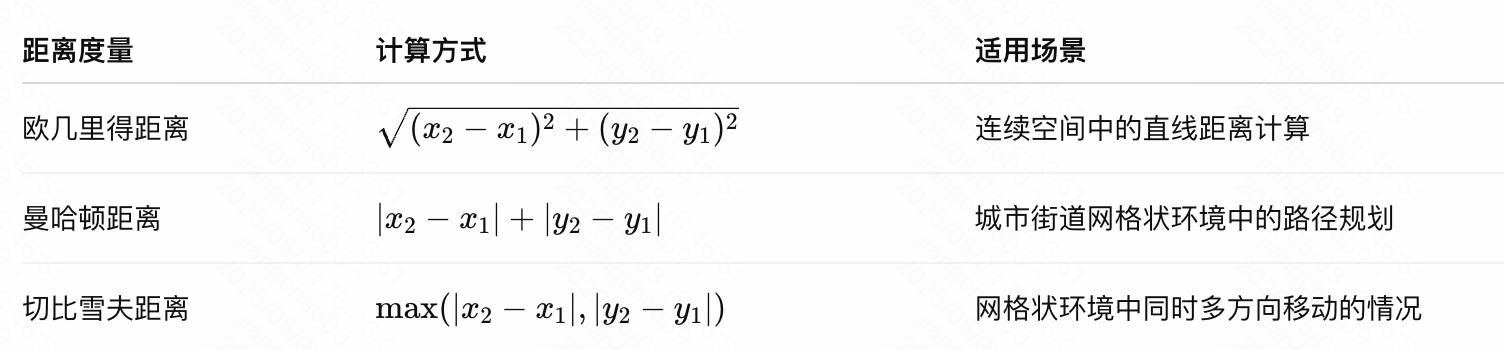
3、优点



4、缺点



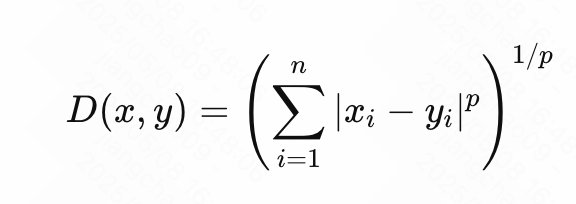
5、对比：



## 闵可夫斯基距离(Minkowski Distance)

闵可夫斯基距离是一种 度量两个向量之间差异的指标，常用于机器学习和数据分析领域，尤其适合在向量空间中衡量相似度或距离。

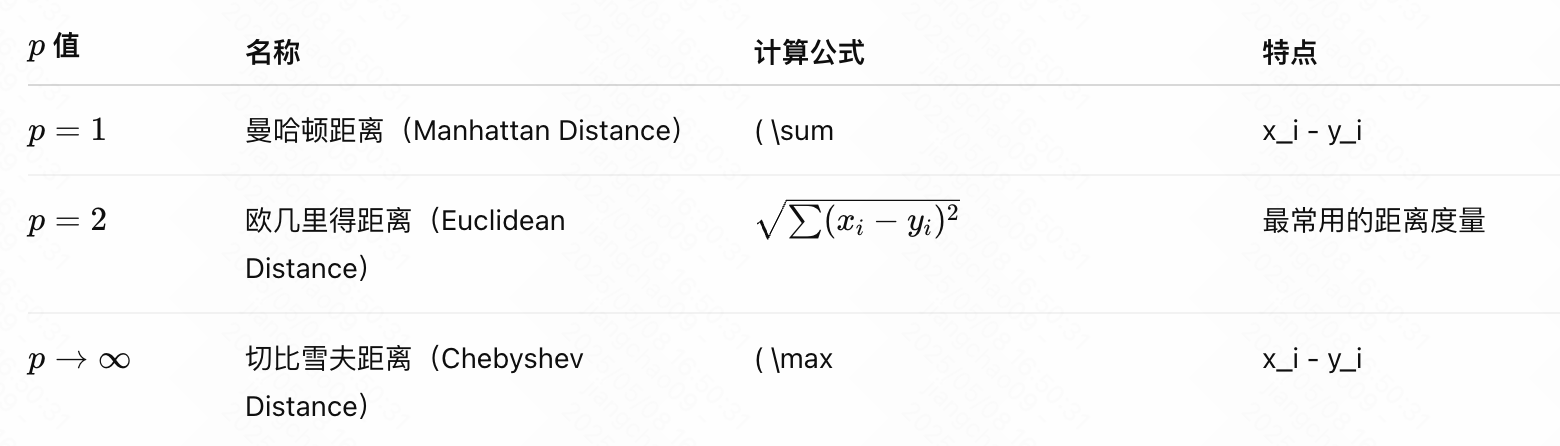
1、公式：



其中：𝑥=(𝑥1,𝑥2,...,𝑥𝑛)、𝑦=(𝑦1,𝑦2,...,𝑦𝑛)：两个点或向量；

𝑝：是一个可变的参数，用来控制距离的“形状”。

2、特殊情况（常见的 𝑝值）：



3、适用场景

K近邻算法（KNN）：自定义 𝑝值可以调节相似度判断方式。

聚类分析（如 K-Means、DBSCAN）：根据问题选取合适的距离度量。

推荐系统：衡量用户/物品的向量相似性。

异常检测：判定数据点与中心的距离。

图像识别：衡量像素或特征向量间的距离。

4、优点

通用性强：可以统一各种距离计算方式。

可调节性：通过 𝑝调整度量“灵敏度”。

适配多种算法：灵活集成于各种算法框架中。

5、缺点

对数据尺度敏感：数据未标准化时，某些特征影响过大。

维度灾难问题：在高维空间中效果可能下降。

参数调节难：选择合适的 𝑝需要经验或验证。

## 标准化欧氏距离(Standardized Euclidean Distance)

## 马氏距离(Mahalanobis Distance)

## 余弦距离(Cosine Distance)

## 汉明距离(Hamming Distance)

## 杰卡德距离(Jaccard Distance)

## 相关距离(Correlation distance)

## 信息熵(Information Entropy)

# 分类

路径规划智能算法是用于解决从起点到终点寻找最佳路径问题的算法，通常根据问题的性质、算法的原理以及应用场景来进行分类。具体的分类方法可以从以下几个维度进行分析：

## 根据问题的空间类型分类

### 离散空间路径规划算法

离散空间路径规划算法：用于处理离散的图或网格环境，在这种情况下，路径是由一系列离散的节点和边组成的。

#### 图搜索算法

图搜索算法：基于图的表示方法来进行路径搜索。例如：

##### A\*算法（A-Star）

A\*算法 (A-Star)：最常用的启发式搜索算法，结合了最短路径算法和启发式策略。

原理：Dijkstra + 启发式函数（如曼哈顿距离）

优点：

- 比Dijkstra更快（利用启发式剪枝）

- 仍保证最优解（启发函数可采纳时）

缺点：

- 启发函数设计影响性能

- 高维空间可能内存爆炸

适用场景：

- 游戏AI寻路

- 机器人栅格地图导航

##### Dijkstra算法

Dijkstra算法：一种经典的最短路径算法，适用于加权图中所有边权值非负的情况。

原理：广度优先搜索 + 贪心策略，逐步扩展最短路径树

优点：

- 保证找到全局最优解

- 适用于带权有向/无向图

缺点：

- 时间复杂度高（O(V²)）

- 无法处理负权边

适用场景：

- 城市道路导航（无负权）

- 网络路由规划

扩展：

Dijkstra算法无法处理负权边，这主要是由于其算法机制和贪心策略所决定的。以下是详细解释：

1、Dijkstra算法的基本原理

Dijkstra算法是一种用于解决带权有向图中单源最短路径问题的算法。它的核心思想是贪心策略，即每次从未找到最短路径的顶点中选择一个距离最短的顶点，然后更新其相邻顶点的最短路径。算法通过不断迭代，逐步确定从起始节点到各个节点的最短路径。

2、负权边对Dijkstra算法的影响

贪心策略失效：

Dijkstra算法基于贪心策略，每次选择当前距离起始节点最近的节点进行扩展。然而，当图中存在负权边时，这种贪心策略可能失效。因为负权边可能导致路径的代价不断降低，使得算法在选择节点时可能跳过某些当前代价较高但最终能够获得更短路径的节点。

陷入无限循环：

如果图中存在负权边，Dijkstra算法可能会陷入无限循环。因为每次选择最小代价的节点时，如果存在负权边，算法可能会不断降低路径的代价，导致无法终止。

无法找到最短路径：

负权边的存在可能导致Dijkstra算法找到的路径不是真正的最短路径。因为算法在选择节点时可能受到负权边的影响，从而跳过某些关键节点，导致路径不是最优的。

3、实例说明

假设有一个简单的有向图，其中包含一个源顶点A，以及三个其他顶点B、C和D。边的权重如下：

A到B：1

A到C：3

B到C：-2

B到D：4

如果使用Dijkstra算法来寻找从A到D的最短路径，算法可能会陷入无限循环或得出错误的结果。具体来说，算法可能无法正确识别通过B到C的负权边来缩短路径的可能性，从而错过真正的最短路径。

4、解决方案或替代算法

由于Dijkstra算法无法处理负权边，因此在处理包含负权边的图时，可以考虑使用以下算法：

Bellman-Ford算法：

可以处理含有负权边的图，并且能够检测出负权环，从而避免无限循环的问题。但其时间复杂度较高，为O(V\*E)，其中V为顶点数，E为边数。

Floyd-Warshall算法：

同样可以处理负权边和负权环，并且能够计算出图中所有顶点对之间的最短路径。但其时间复杂度为O(V^3)，适用于小型至中型网络的路径计算。

概括起来，Dijkstra算法在处理包含负权边的图时存在局限性，可能导致不正确的结果或无限循环。因此，在实际应用中，应根据具体需求和图的特性选择合适的算法。

##### 广度优先搜索（BFS）

广度优先搜索（BFS）：用于无权图中寻找最短路径。

##### 深度优先搜索（DFS）

深度优先搜索（DFS）：在某些特定问题中使用，适用于寻找路径，但不一定是最短路径。

### 连续空间路径规划算法

连续空间路径规划算法：用于处理连续空间中的路径规划问题（例如机器人在二维或三维空间中的运动）。

#### 快速扩展随机树 (RRT)

快速扩展随机树 (RRT)：主要用于高维空间的路径规划，适合动态和不规则环境。

原理：随机采样构建空间搜索树

优点：

- 高维空间高效

- 避障能力强

缺点：

- 路径不一定最优

- 可能产生锯齿路径

适用场景：

- 无人机三维避障

- 机械臂运动规划

#### PRM (概率路线图)

PRM (概率路线图)：用于高维空间的路径规划，尤其适合大规模机器人路径规划问题。

原理：先建图后搜索

优点：

- 可预处理静态环境

- 查询阶段高效

缺点：

- 建图质量影响结果

- 动态环境需重建

适用场景：

- 仓库AGV调度

- 静态环境机器人导航

#### 人工势场法

人工势场法：通过模拟目标和障碍物的引力与斥力来生成路径。

原理：模拟目标引力与障碍物斥力，通过合力控制运动方向。

优点：计算简单，实时性好，路径平滑。

缺点：易陷入局部最优，对复杂障碍物敏感。

适用场景：实时避障（如机器人局部导航）。

## 根据路径规划算法的优化策略分类

### 确定性算法

确定性算法：这些算法可以保证找到一个最佳解或最优解，通常在静态环境中使用。

#### A\*算法

A\*算法：通过结合启发式函数和实际路径代价来进行搜索，保证找到最短路径。

#### Dijkstra算法

Dijkstra算法：一种最短路径算法，适用于边权非负的图，保证找到全局最短路径。

### 启发式算法

启发式算法：依赖于启发式函数来引导搜索，能在较短时间内找到近似最优解。

#### 遗传算法 (GA)

遗传算法 (GA)：模拟自然选择和遗传进化的过程，通过遗传操作（交叉、变异等）来找到路径。

原理：模拟生物进化（选择/交叉/变异）

优点：

- 适合高维非线性问题（复杂的组合优化问题）

- 可处理多目标优化（能够处理大规模问题，并且可以避开局部最优）

缺点：

- 参数敏感（种群大小、变异率等）

- 早熟收敛风险

- 计算量大，收敛速度慢，有时难以确保找到全局最优解

适用场景：

- 物流配送路径优化

- 超大规模TSP问题

- 组合优化问题：例如旅行商问题（TSP）、背包问题、调度问题等。

- 大规模复杂问题：在搜索空间巨大且复杂度高时，GA能够有效避免陷入局部最优。

- 多目标优化：GA能够同时处理多个目标，通过适应度函数进行权衡。

应用领域：

工业设计、机器学习模型调参、资源分配、进化计算等。

#### 粒子群优化算法 (PSO)

粒子群优化算法 (PSO)：模拟鸟群觅食行为的优化算法，寻找全局最优路径。

原理：粒子群优化模拟鸟群觅食的过程。每个粒子代表问题的一个潜在解，粒子根据自己的历史位置和全局最优解来调整自己的位置。

- 局部最优：每个粒子根据自己的最佳位置和群体的最佳位置来更新位置。

优点：

- 简单、易实现，适用于连续优化问题

缺点：

- 对于复杂的离散优化问题性能较差，容易陷入局部最优

场景：

- 连续优化问题：例如函数优化问题、机器学习模型的参数调优、神经网络训练等。

- 高维空间优化：PSO适用于高维度问题，能快速收敛到最优解。

- 非线性优化问题：适合处理复杂的非线性、多峰函数问题。

应用领域：

控制系统优化、机器学习、图像处理、信号处理等。

**扩展：**

粒子群算法PSO 在处理连续优化问题时表现良好，但在复杂的离散优化问题中（如路径规划、组合优化等），容易遇到以下两个主要问题：

1、PSO本质是为连续空间设计的

2.1）粒子的位置和速度是连续值：

在PSO中，每个粒子的位置和速度都是实数向量（连续值），适用于连续函数的优化。

而在离散优化问题中（例如：旅行商问题中的路径是城市排列的序列，或背包问题中的二进制选择），解决方案是离散的、组合的。

将PSO直接用于这些离散空间时，就必须进行离散化处理（比如二进制PSO或位置映射），而这种“改造”往往会破坏PSO算法的原始性能优势。

**2、易陷入局部最优的根本原因**

2.1）更新机制缺乏多样性：

PSO 中的每个粒子都会根据：

自己的历史最优位置（个体经验）

群体中找到的全局最优位置（群体经验）来更新自身的位置。

这种更新策略强调“跟随”，缺乏“探索”，导致粒子很快聚集到一个区域，如果这个区域只是局部最优，则整个群体容易陷入困境，无法跳出。

尤其是在目标函数呈现多个局部极值点的复杂问题中，PSO 的粒子更新机制很容易让所有粒子“盲从”全局最优点，失去对其他区域的探索能力。

2.2）缺乏变异机制（对比遗传算法）：

遗传算法通过“变异”操作，可以强制引入多样性，从而跳出局部最优。

而标准的PSO没有类似变异机制，除非专门加以设计（例如加入扰动、混沌机制等），否则容易“早熟收敛”。

**3、搜索空间复杂性带来的挑战**

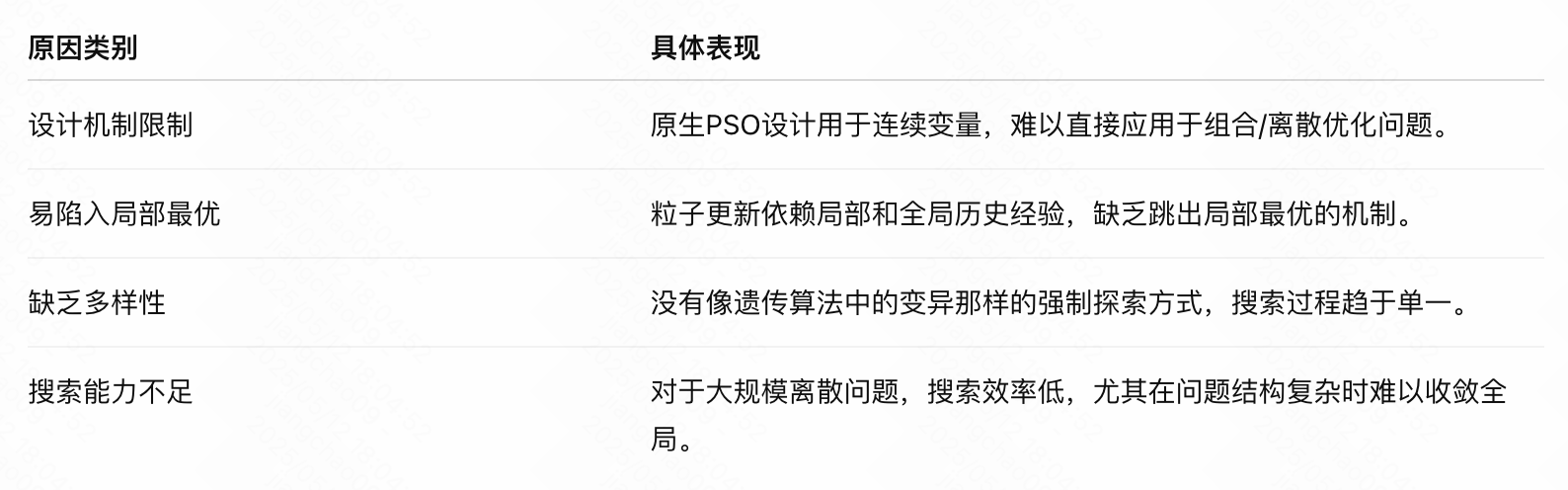
离散优化问题的搜索空间呈现组合爆炸，例如：

TSP（旅行商问题）有 n! 种可能路径。

背包问题有2^n 种物品组合。

在这样的搜索空间中，单靠PSO中每个粒子对自身经验的微调，很难在合理时间内探索到全局最优解，尤其当粒子已经陷入局部区域时，更难逃脱。

* **总结：PSO在离散优化问题中性能较差的原因**



* **改进方向**

为了解决这些问题，研究人员提出了以下几种改进型PSO：

离散粒子群优化（Discrete PSO）：使用编码方式将离散问题映射到粒子的位置表示中。

混合算法：将PSO与遗传算法、模拟退火等结合，引入变异操作。

引入局部扰动、混沌初始化：提升多样性和跳出局部最优的能力。

#### 蚁群算法 (ACO)

蚁群算法 (ACO)：模拟蚂蚁寻找最短路径的行为，通过信息素来引导搜索。

原理：模拟蚂蚁信息素追踪

优点：

- 自适应动态环境（特别适合解决路径优化问题，能处理动态变化的环境）

- 分布式计算友好

缺点：

- 收敛速度慢

- 需要调参（信息素挥发系数）

- 容易陷入局部最优，且在高维问题中表现较差

适用场景：

- 动态交通路径规划

- 电力网络布线

- 路径优化问题：例如旅行商问题、车辆路径规划、机器人路径规划等。

- 图着色问题、任务分配问题：ACO能通过信息素更新机制在图中寻找最优解。

- 动态环境下的优化：适用于动态变化的优化问题（如实时路径调整）。

应用领域：

网络路由、物流配送、机器人导航、自动化生产调度等。

#### 总结

##### 区别



##### 联系

群体智能：GA、ACO和PSO都属于群体智能算法，模拟生物体或群体的集体行为来解决优化问题。它们通过个体之间的合作和信息交流来共同逼近最优解。

全局优化：它们都能够进行全局搜索，在搜索空间中寻找最优解，而不仅仅是局部最优。

随机性：这三种算法都具有一定的随机性（GA的变异、ACO的蚂蚁行为、PSO的粒子更新），使得它们能够避免陷入局部最优解。

适应性：它们都具有较强的适应性，能够应对各种动态和复杂的优化问题。

##### 适用场景

遗传算法 (GA) 适用于组合优化问题，能够处理复杂的搜索空间，适合求解大规模问题。

蚁群算法 (ACO) 擅长路径优化和图优化问题，适用于动态和实时的路径规划任务。

粒子群优化 (PSO) 更适合连续优化问题，尤其是在需要快速收敛的情况下，表现出色。

### 随机化算法

随机化算法：依靠随机性来进行路径搜索，适用于动态变化或复杂的环境。

#### 快速随机树（RRT）

快速随机树 (RRT)：通过在空间中随机扩展树的方式来快速探索路径，适用于高维空间和动态环境。

#### PRM（概率路线图）

PRM (概率路线图)：通过随机采样空间来构建图，适合大规模问题。

## 根据路径规划问题的环境性质分类

### 静态环境路径规划算法

静态环境路径规划算法：适用于障碍物不变化的环境。常见算法包括：

#### A\*算法

#### Dijkstra算法

#### 遗传算法

#### 模拟退火算法（SA）

模拟退火算法（Simulated Annealing, SA）是一种概率性全局优化算法，受到固体物理中退火过程的启发，用于在庞大、复杂、可能存在多个局部最优的搜索空间中寻找全局最优解。它常用于组合优化问题，尤其在避免陷入局部最优方面表现突出。

1、模拟退火算法（SA）原理

物理背景：

退火过程：金属加热至高温后缓慢冷却，原子从无序（高能态）排列逐渐转变为有序（低能态）结构，从而形成稳定的晶体。

数学模拟：通过接受“较差解”的概率机制，跳出局部最优，从而逼近全局最优。

算法流程简述：

1）初始化温度 T 和初始解 S。

2）在当前解的邻域中随机生成新解 S'。

3）计算目标函数差值 ΔE = E(S') - E(S)。

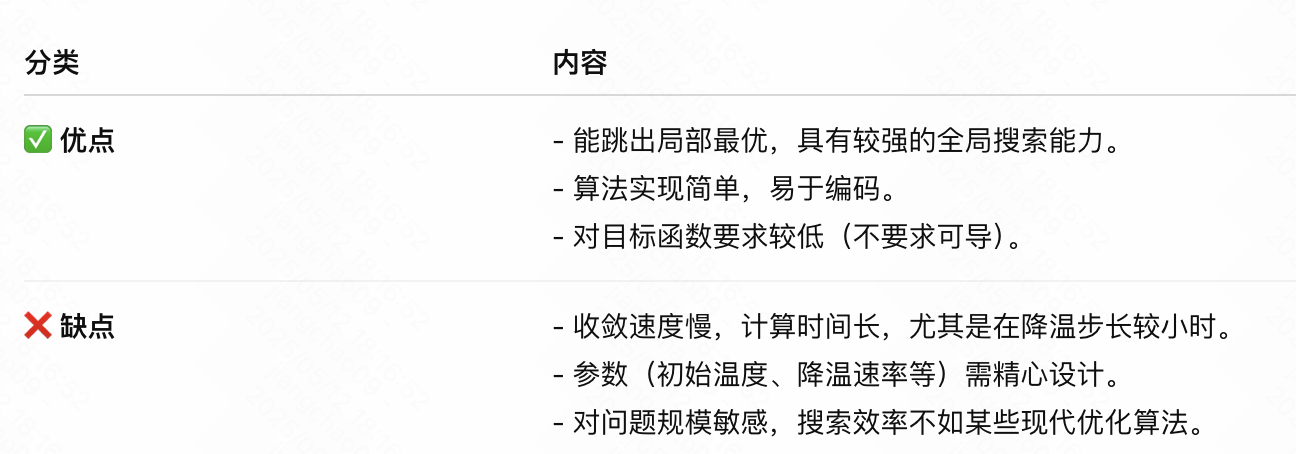
4）如果ΔE < 0（新解更优），接受新解；

如果ΔE > 0（新解更差），以概率 P = exp(-ΔE / T) 接受新解。

5）缓慢降低温度 T（即“降温”）。

6）重复以上步骤直到温度趋于0或达到终止条件。

2、优缺点总结



3、适用场景

模拟退火特别适用于：

组合优化问题（如旅行商问题、作业调度、背包问题等）

路径规划与布局问题（如电路布图、机器人路径规划）

图像处理（如图像分割、图像配准）

机器学习中的模型参数选择

函数优化（特别是非凸、多峰函数）

4、与 GA / ACO / PSO 的区别对比



5、总结归纳



### 动态环境路径规划算法

动态环境路径规划算法：适用于障碍物不断变化或有移动障碍物的环境。这类算法能在路径搜索过程中动态更新。

#### D算法（动态A）

D算法（动态A）：适用于动态环境中路径实时更新，能够在地图发生变化时重新计算路径。

原理：A\*的动态版本，反向搜索并重用路径信息，适用于动态环境。

优点：动态环境下效率高，路径重规划速度快。

缺点：实现复杂，极端动态场景性能可能下降。

适用场景：动态障碍物环境（如自动驾驶、无人机避障）。

#### 动态规划算法

动态规划算法：在环境发生变化时，通过更新已有的解来重新进行路径搜索。

#### RRT\*算法

RRT算法：是RRT的改进版，能够在动态环境中重新规划路径。

## 根据解的质量分类

### 全局最优解算法

全局最优解算法：能够保证找到从起点到终点的最佳路径，通常需要较高的计算量。

#### A\*算法

#### Dijkstra算法

#### 动态规划算法

### 近似最优解算法

近似最优解算法：这些算法并不保证全局最优解，但能在较短时间内找到一个较好的解。

#### 遗传算法

#### 粒子群优化算法

#### 蚁群算法

#### 模拟退火算法

## 根据智能优化策略分类

### 基于进化的智能算法

基于进化的智能算法：这类算法通过模拟生物的进化过程来优化路径。

#### 遗传算法（GA）

遗传算法 (GA)：模拟自然界的选择、交配和变异过程。

### 基于群体智能的优化算法

基于群体智能的优化算法：模拟群体行为来优化路径搜索过程。

#### 蚁群算法（ACO）

蚁群算法 (ACO)：模拟蚂蚁通过信息素的更新来寻找最优路径。

#### 粒子群优化（PSO）

粒子群优化 (PSO)：模拟粒子在解空间中的飞行行为，通过个体和群体的学习来寻找最优解。

### 基于模拟退火的优化算法

基于模拟退火的优化算法：模拟物体在冷却过程中能量逐渐减少的过程来优化路径。

#### 模拟退火算法（SA）

模拟退火算法 (SA)：通过模拟物理系统的退火过程来避免局部最优解。

### 深度学习算法

#### 深度强化学习（DRL）

原理：Q-learning + 神经网络

优点：

- 可学习复杂策略

- 适应未知环境

缺点：

- 需要大量训练数据

- 黑箱决策难解释

适用场景：

- 自动驾驶决策

- 游戏NPC智能路径

#### GNN路径规划

原理：图神经网络处理拓扑关系

优点：

- 直接处理图结构数据

- 端到端训练

缺点：

- 泛化能力受限

- 计算资源消耗大

适用场景：

- 社交网络信息传播优化

- 电路板布线

## 根据应用领域分类

### 机器人路径规划

机器人路径规划：专门针对机器人在环境中进行导航的路径规划，常用算法有A\*、RRT、PRM等。

### 无人驾驶车辆路径规划

无人驾驶车辆路径规划：涉及到无人驾驶汽车、无人机等在复杂道路或飞行环境中的路径规划，常用算法有D\*、深度强化学习等。

### 物流与自动化路径规划

物流与自动化路径规划：例如仓储机器人、自动化生产线上的路径规划，常用算法有遗传算法、粒子群优化等。

总结：

路径规划智能算法可以从多种角度进行分类，最常见的分类方式是根据环境的类型（离散/连续空间）、算法的优化策略（确定性/启发式/随机）、环境性质（静态/动态）以及解的质量（全局最优/近似最优）等维度进行分类。选择适合的路径规划算法依赖于具体的应用场景、问题的复杂性以及所需的精度和计算效率。

## 对比



# 联系

- 递进关系：Dijkstra → A\* → D\*（增量式A\*）→ 强化学习

- 互补使用：常用组合如：

- PRM + A\*（先建图后搜索）

- RRT\* + 轨迹优化（粗规划+精细调整）

- 蚁群算法 + 局部Dijkstra（全局+局部优化）

实际应用中常采用混合策略，例如京东物流使用遗传算法规划大区域配送路线，结合A\*算法处理最后一公里路径。

# 选择建议

1. 精度优先：选A\*或Dijkstra（低维静态环境）
2. 动态环境：选D或RRT变种（如RRT），需实时性时用DWA。

2、实时性要求：RRT系列（机器人避障）

3、超大规模问题：遗传/蚁群算法（物流配送）

4、未知环境：DRL（自动驾驶）