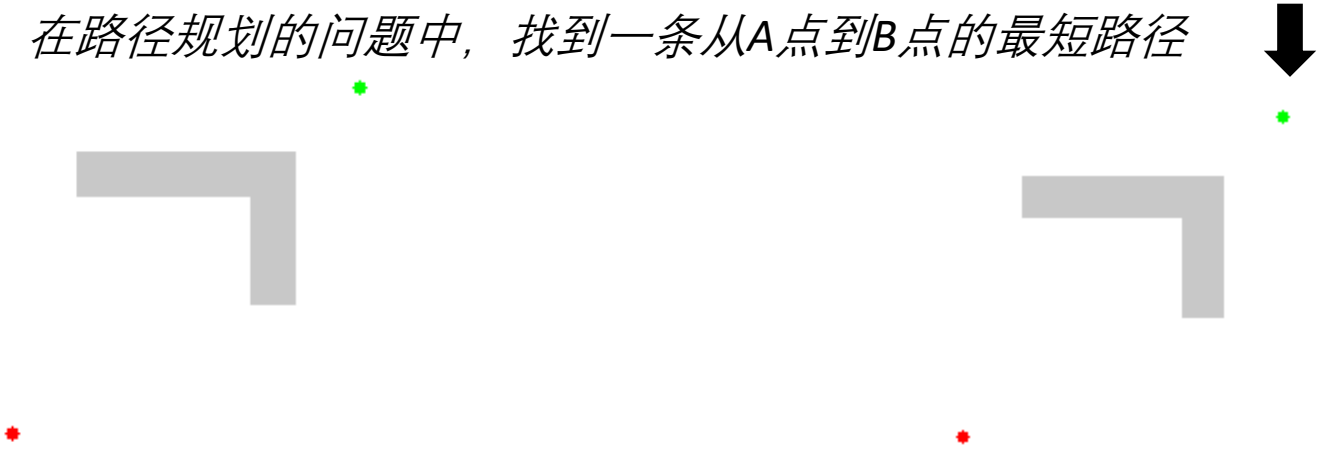
第七章 其他元启发式算法

启发式算法

- Heuristic: to find, discover（探索）
- **启发式算法（Heuristic Algorithm）**：是一种基于直觉或已有经验的算法。能够在可接受的计算成本（计算时间、占用空间等）内去搜寻最好的解，但是不保证能够找到最优解。
- 相对于一般的算法把各种**可能性都一一进行尝试**，最终能找到问题的答案，启发式算法则是在**有限的搜索空间**内，大大减少尝试的数量，能迅速地达到问题的解决。

在路径规划的问题中，找到一条从A点到B点的最短路径



一般算法（遍历问题空间中所有的解）

启发式搜索（利用启发式信息，减小问题空间）

启发式算法的发展

- 40年代：因为对求解速度的需求，提出了启发式算法。
- 50年代：逐步繁荣，其中**贪心算法**和**局部搜索**等到人们的关注。
- 60年代: 发现**贪心算法**和**局部搜索**速度很快，但是**解得质量不能保证**，而且对大规模的问题**收敛速度慢。**主要原因在于他们**只是在局部的区域内找解**，等到的解没有全局最优性。
- 70年代：引入**新的搜索机制和策略，**进化为**元启发式算法**（Meta-heuristic Algorithm）。比如：Holland的**遗传算法**在这个阶段出现（Genetic Algorithm）。
- 80年代以后：**模拟退火算法**（Simulated Annealing Algorithm），**人工神经网络**（Artificial Neural Network），**禁忌搜索**(Tabu Search)相继出现。

- **贪心算法**是在算法的每一步选择中都采取在当前状态下最好或最优（即最有利）的选择，从而希望导致结果是最好或最优的算法。
- **局部搜索**从一个初始解出发，然后搜索解的邻域，如有更优的解则移动至该解并继续执行搜索，否则返回当前解。

启发式的贪心算法

- 贪心算法通常可以分为两个步骤（1）定义问题的子问题，（2）然后使用贪心策略来解决子问题。**贪心策略是指每一步都选择当前状态下的最优解，而不考虑未来可能发生的情况。**（贪心算法并不能保证一定得到全局最优解）

例：一个简单的例子是找零钱问题。假设你需要找零36元，而你手上只有面值为1元，5元，10元，20元，50元的纸币。那么如何找零使得使用的纸币数量最少呢?

一种贪心的策略是每一次尽可能使用面值最大的纸币。首先可以使用一个面值为20元的纸币，然后使用两个面值为10元的纸币，最后使用一个面值为5元的纸币，共使用4张纸币。这是最少的纸币数量，而使用其他方式则需要更多的纸币。

元启发式与启发式算法

- **Meta（元）**: in an upper level ; **Heuristic**: to discover
- 元启发式算法和启发式算法都是**解决优化问题的算法**，但它们之间存在一些区别:
  - **启发式算法**是一种根据经验或启示性规则生成解决方案的算法，它通常不提供全局最优解的保证，但可以在较短的时间内找到接近最优的解。启发式算法的典型例子是**贪心算法**，它通过**每次选择局部最优解来逐步构建全局解**。
  - **元启发式算法**则是一种更高级的启发式算法，它可以在解决优化问题时进行“全局搜索”，而不是局部搜索。元启发式算法通常通过模拟自然现象（如遗传进化、蚁群行为、鸟群飞行等）或者人工设计的策略（如模拟退火、禁忌搜索、粒子群优化等）来生成解决方案。与启发式算法相比，元启发式算法通常**具有更好的全局搜索能力**，能跳出局部最优，有一定几率找到全局最优。

总之，元启发式算法是启发式算法的一种扩展形式，它可以在解决更复杂的优化问题时提供更好的性能和结果。

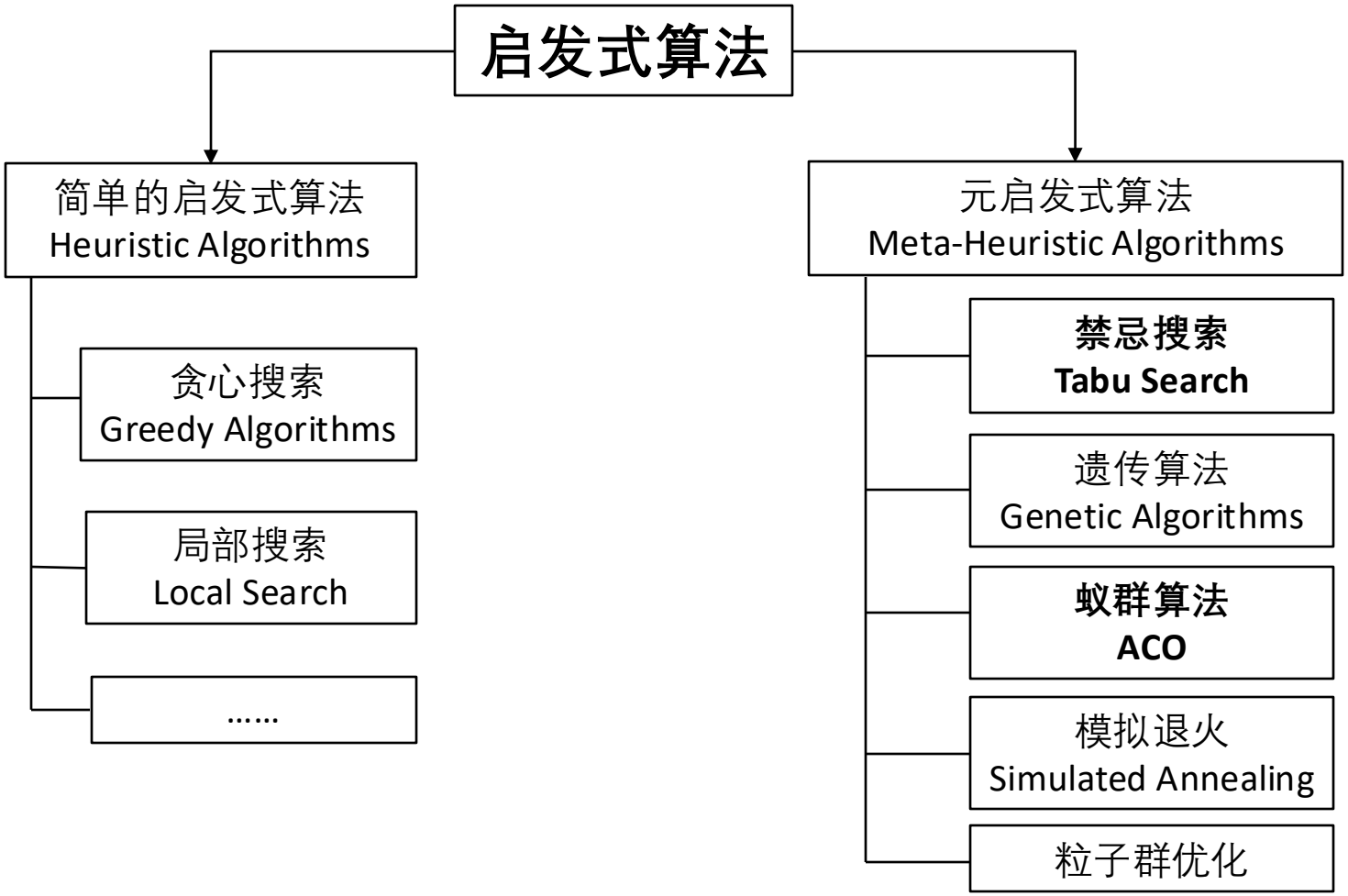
元启发式与启发式算法

- 元启发式算法的核心是Exploration（**探索**）和Exploitation（**充分利用**）。Exploration即尽量探索整个搜索空间，由于最优解可能存在整个搜索空间的任何位置。而Exploitation是尽可能地利用启发式信息。
- 相对于一般的启发式算法，元启发式算法能够在一定程度上**在全局进行搜索，找到最优解的近似解**。

启发式	*versus*	**元启发式**
搜索范围局部为主		搜索范围更偏向全局
固定规则		可自适应/进化
很少或没有使用随机		广泛使用随机

启发式算法的家族

- 启发式算法的“家族”很庞大



这些算法各自具有独特的策略和原理，广泛应用于组合优化领域中旅行商问题、调度问题、路径规划等。

组合优化领域的各种问题

旅行商问题（Travelling Salesman Problem，简称TSP）。这个问题可以描述为：假设有一个旅行商需要访问一系列城市，每个城市只能访问一次，最后返回出发城市。旅行商的任务是找到一条最短的路径，使得他能够完成访问所有城市的任务，同时总行程路径最短。

车辆路径问题（Vehicle Routing Problem，简称VRP）。与旅行商问题（TSP）有密切关系。VRP的目标是为一群车辆设计最佳路线，以便在满足各种约束条件的前提下，为一组客户提供货物或服务。与TSP不同，VRP需要考虑多辆车辆，而且车辆可能有不同的载重和行驶能力。

调度问题（Scheduling Problem）主要涉及分配有限的资源来执行一系列任务的最优安排。在调度问题中，任务通常有特定的执行时间、优先级和截止日期等约束条件，而资源也可能有其可用性和能力限制。调度问题的目标是在满足所有约束条件的前提下，找到一种资源分配和任务执行顺序的安排，使得某种性能指标最优化，如完成任务的总时间最短、等待时间最小或满足尽可能多的截止日期等。

禁忌搜索

- **禁忌搜索**（Tabu Search）是一种用于解决组合优化问题的元启发式算法。它由Fred Glover于1986年首次提出。禁忌搜索的主要目标是在问题空间中寻找最优解决方案，同时**避免陷入局部最优解**。

- 禁忌搜索的优点在于其**能够避免陷入局部最优解，并在问题空间中寻找更好的全局最优解。**它通过引入一种称为**禁忌表**的机制来避免算法反复探索已经访问过的解，从而跳出局部最优。

- **禁忌表**是禁忌搜索算法的核心。其功能同人类的短时记忆功能相似。它用来记录某些已经访问过的解或不允许的操作，从而避免在搜索过程中重复走回头路或陷入局部循环。

禁忌搜索的主要概念

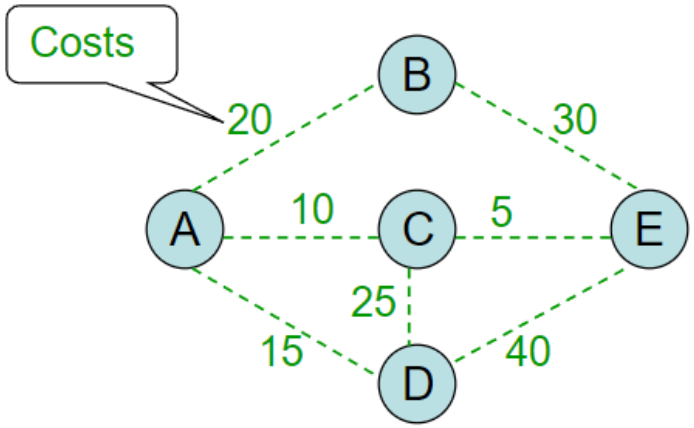
- **禁忌表（Tabu List，TL）**
  是用来存放（记忆）禁忌对象的表。它是禁忌搜索得以进行的基本前提。禁忌表本身是有容量限制的，它的大小决定了存放禁忌对象的个数。
- **禁忌对象（Tabu Object，TO）**
  是指禁忌表中被禁的那些元素。禁忌对象的选择可以根据具体问题而制定。例如在旅行商问题（Traveling Salesman Problem，TSP）中可以将交换的城市对作为禁忌对象，也可以将总路径长度作为禁忌对象。
- **禁忌期限（Tabu Tenure，TT）**
  也叫禁忌长度，指的是禁忌对象不能被选取的周期。禁忌期限过短容易出现循环，跳不出局部最优，长度过长会造成计算时间过长。
- **特赦规则（Aspiration Criteria，AC）**
  当所有的对象都被禁忌之后，可以让其中性能最好的被禁忌对象解禁，或者当某个对象解禁会带来目标值的很大改进时，也可以使用特赦规则。

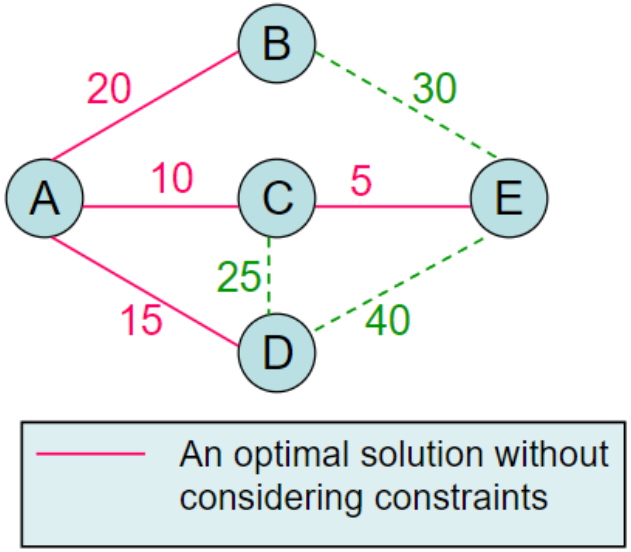
禁忌搜索算法

1. **初始化：**选择一个初始解决方案，并将其设为当前解决方案。同时，初始化一个禁忌表，用于存储禁止访问的解决方案。
2. **邻域搜索：**从当前解决方案出发，搜索其邻域内的其他解决方案。邻域是指当前解决方案附近的一组解决方案，通常通过对当前解决方案进行某种局部修改来生成。
3. **评估和更新：**评估邻域内的解决方案，并选择一个最优的解决方案。如果这个解决方案优于当前解决方案且不在禁忌表中，则将其设为新的当前解决方案。同时，将已访问过的解决方案添加到禁忌表中，以避免重复访问。
4. **禁忌表更新：**更新禁忌表，移除过期的解决方案。禁忌表的长度（即禁止访问的解决方案数量）可以是固定的或动态变化的，根据问题的特点和求解过程来设定。
5. **终止条件：**当达到预设的迭代次数或满足其他停止准则时，算法终止。返回找到的最优解决方案。

禁忌搜索例子

- Minimum spanning tree problem with constraints （带约束的最小生成树问题）
- 目标：以最小的代价连接所有的节点（图中有5个节点）





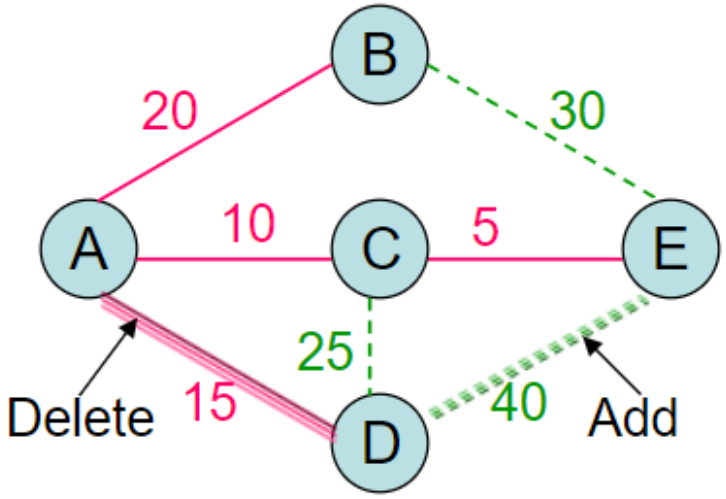
约束1：只有DE连在一起的情况下，A节点和D节点才能连接在一起（否则惩罚cost为100）

约束2：AD，CD以及AB三者之间最多只能有一个连接（有两个连接惩罚100，有三个连接惩罚200）

禁忌搜索例子

第一次迭代:

Cost = 50 + 100 + 100 (约束带来的惩罚)
历史最优解: 250

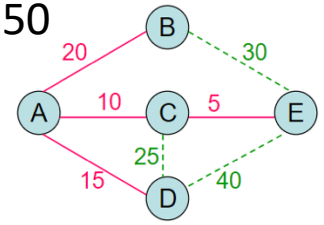


邻域

Add	Delete	Cost
BE	CE	75+200=275
BE	AC	70+200=270
BE	AB	60+100=160
CD	AD	60+100=160
CD	AC	65+300=365
DE	CE	85+100=185
DE	AC	80+100=180
DE	AD	**75+0=75**

- 第一次迭代的最佳选择是增加DE而删掉AD。最优解从250减少到75。

250

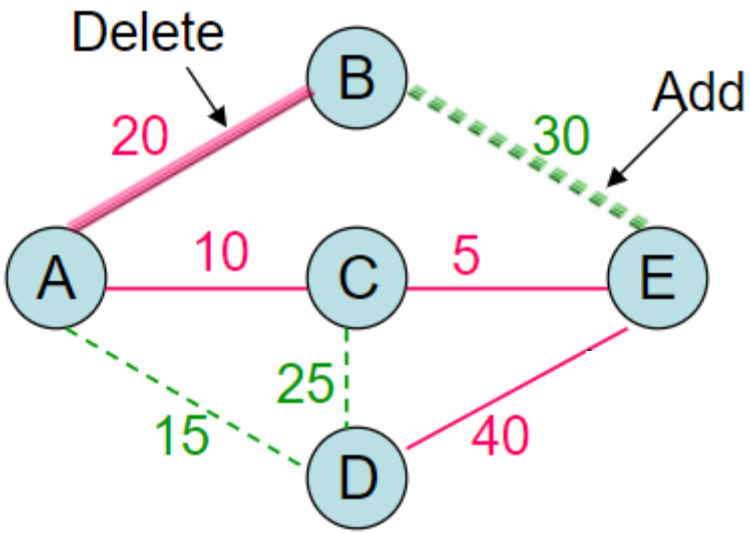


禁忌表:

最优解: 75

禁忌搜索例子

第二次迭代



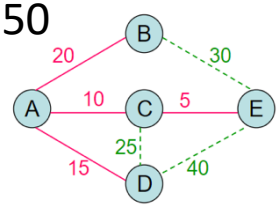
邻域

Add	Delete	Cost
AD	DE*	Tabu move
AD	CE	85+100=185
AD	AC	80+100=180
BE	CE	100+0=100
BE	AC	95+0=95
BE	AB	**85+0=85**
CD	DE*	60+100=160
CD	CE	95+100=195

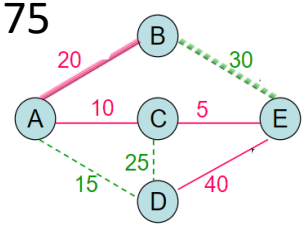
- 第二次迭代的最佳选择是增加BE而删掉AB。历史最优解不变，还是75。（但当前cost已经由75到85（禁忌搜索接受劣解）

禁忌表：

250



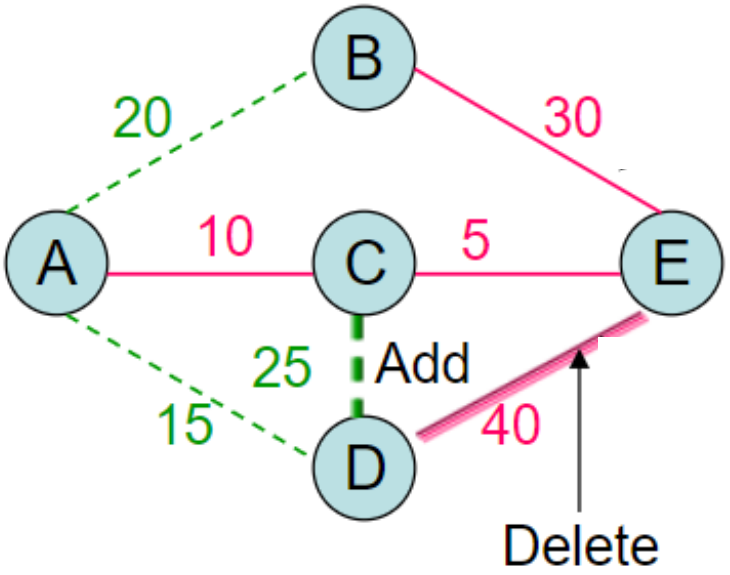
75



最优解： 75

禁忌搜索例子

第三次迭代

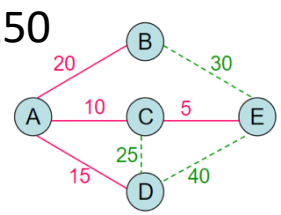


邻域

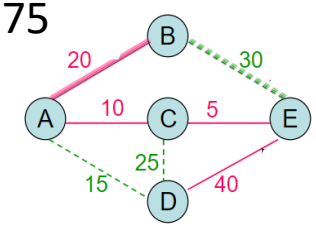
Add	Delete	Cost
AB	BE*	Tabu move
AB	CE	100+0=100
AB	AC	95+0=95
AD	DE*	60+100=160
AD	CE	95+0=95
AD	AC	90+0=90
CD	DE*	**70+0=70**
CD	CE	105+0=105

禁忌表：

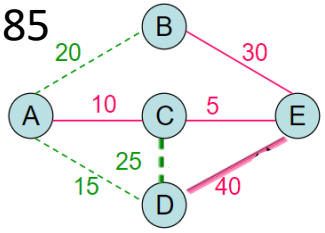
250



75



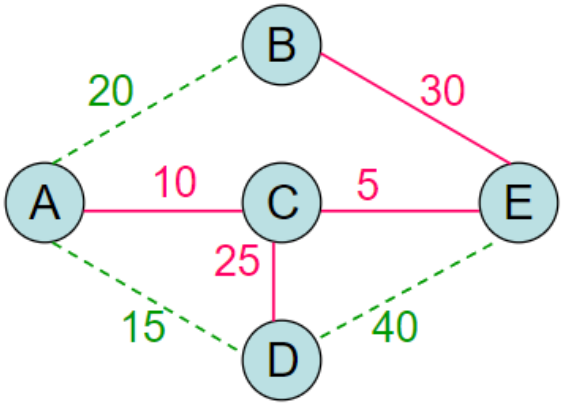
85



最优解： 70

禁忌搜索例子

第三次迭代



Optimal Solution

Cost = 70

Additional iterations only find inferior solutions

约束1：只有DE连在一起的情况下，A节点和D节点才能连接在一起（否则惩罚cost为100）

约束2：AD，CD以及AB三者之间最多只能有一个连接（有两个连接惩罚100，有三个连接惩罚200）

群体智能-蚁群优化

群居行为——超过任何一个单独的个体能够做的事情。

蚂蚁的行为方式

Franks等人观察到蚂蚁群有如下行为:

- 利用自身的身体搭一个桥梁;
- 合作搬运大件物品。
- 寻找从巢穴到食物来源的最短路线。

“活”的桥梁

合作搬运

寻找最佳路线

蚂蚁的行为方式

Franks等人观察到蚂蚁群有如下行为:

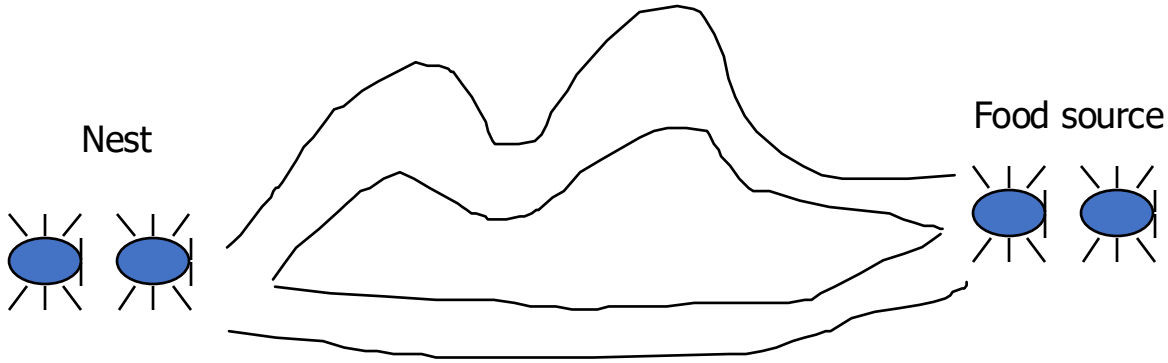
- 利用自身的身体搭一个桥梁;
- 合作搬运大件物品。
- 寻找从巢穴到食物来源的最短路线。

寻找最佳路线

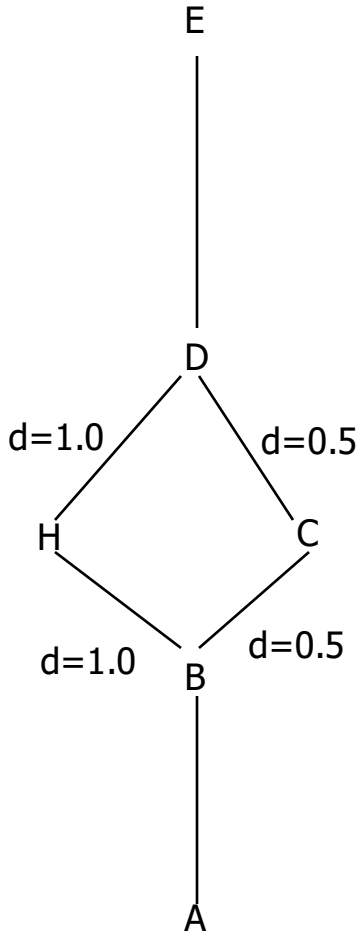
蚁群优化算法（ Ant Colony Optimisation， ACO）就受此启发

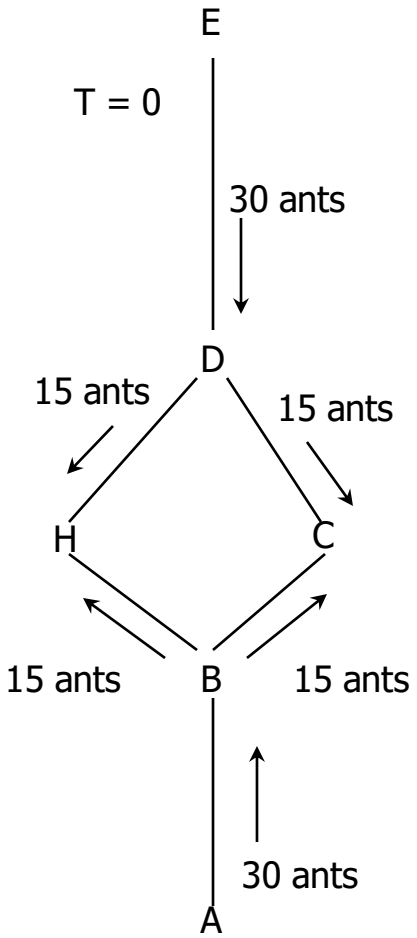
蚂蚁如何寻找最短路径

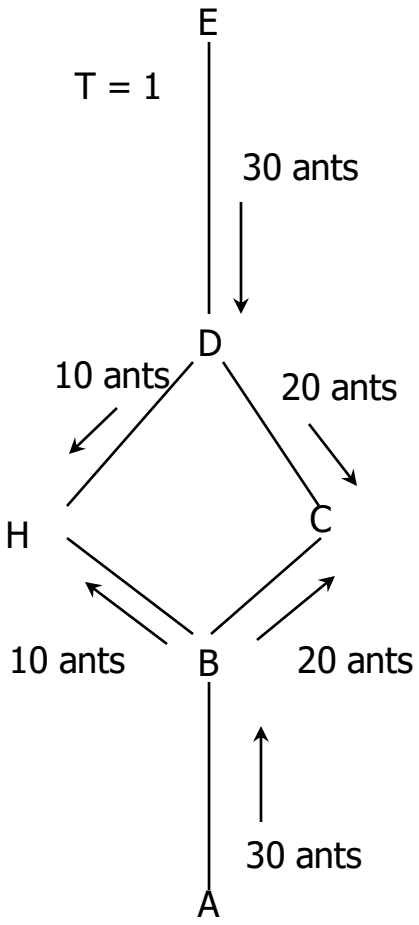
- 蚂蚁开发了一种能够使得个体之间能够通信的物质：信息素，蚂蚁释放信息素，能够让其他蚂蚁感知并跟随。
- 蚂蚁在去往食物的路上，回到巢穴的路上都会释放信息素。
- 信息素的强度会随着蚂蚁使用路径次数的增多而积累。
- 距离较短的路径因为单位时间内往返蚂蚁的个数比较多，信息素就比较多。
- 蚂蚁倾向于走信息素较多的路径，即距离较短的那个。



信息素







A：Nest
E：Food source

单位时间内，从A经C到E以及从E经C到A往返的蚂蚁总个数是最多的，所以该路径有最多的信息素

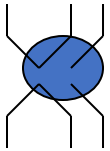
蚁群优化算法：基本想法

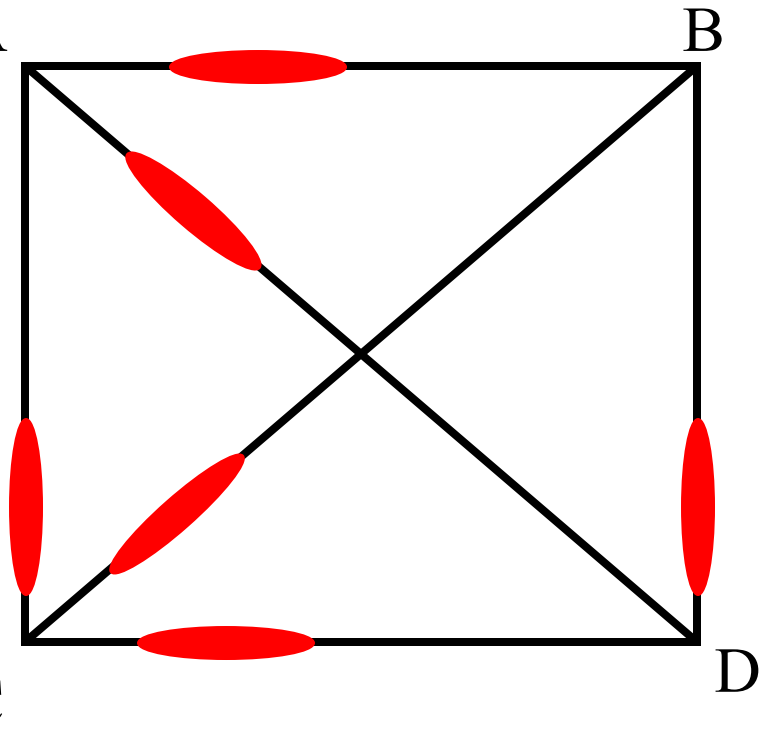
- 蚂蚁（Agents）在图（Graph）的节点间移动。
- 根据信息素来选择朝向哪个节点移动
- 蚂蚁行走的某一路径代表一个候选解。
- 如果蚂蚁走完了一个路径，即得到一个候选解，蚂蚁会根据该候选解的优劣在该路径上释放信息素。（信息素的释放是延后的）

蚁群优化算法可以用来解决TSP问题

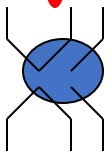
例：A 4-city TSP

- 在初始阶段，每条路径随机产生信息素强度。





信息素

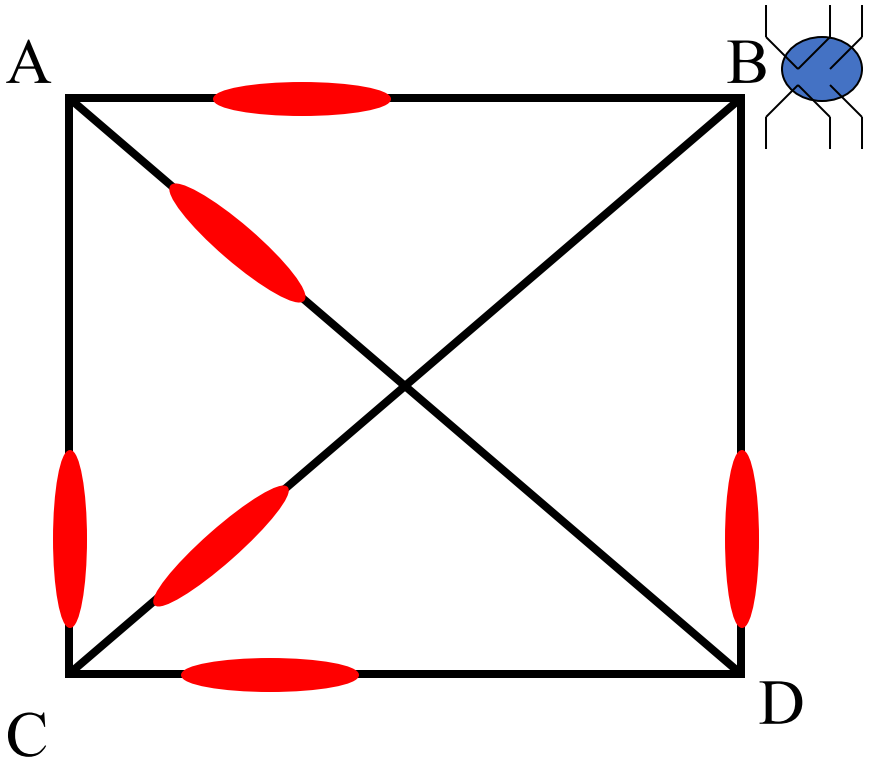


蚂蚁

AB: 10,　AC: 10,　AD, 30,　BC, 40,　CD 20

例：A 4-city TSP

- 一只蚂蚁被随机的安排到某个节点B。



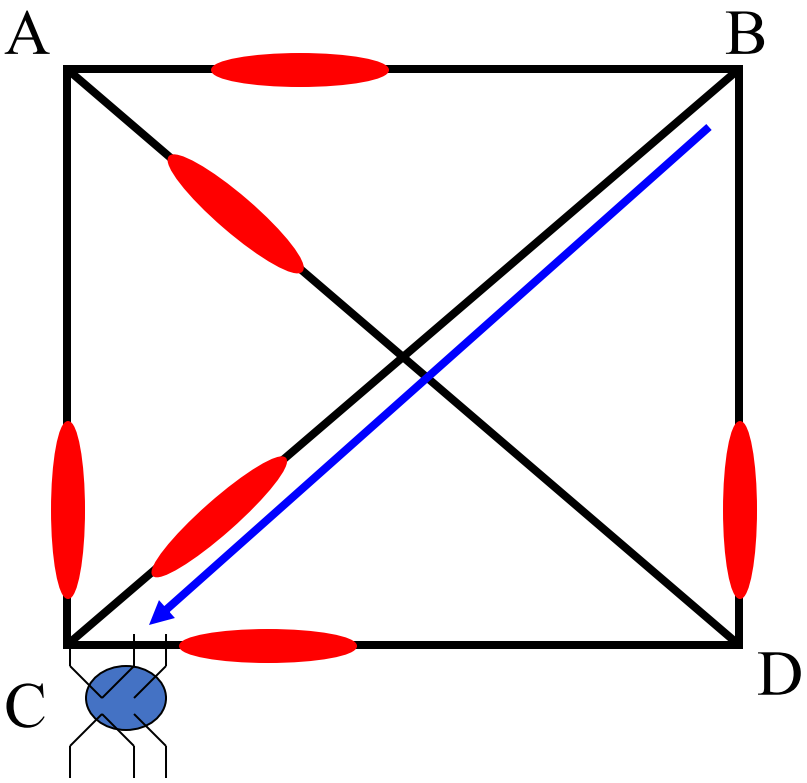
信息素

蚂蚁

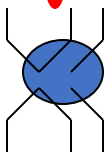
AB: 10, AC: 10, AD, 30, BC, 40, CD 20

例：A 4-city TSP

- 蚂蚁会根据**信息素的强度**以及**距离**来计算概率决定到底走向哪个方向。**（问题1）**
- 假设现在蚂蚁走了路径BC



信息素

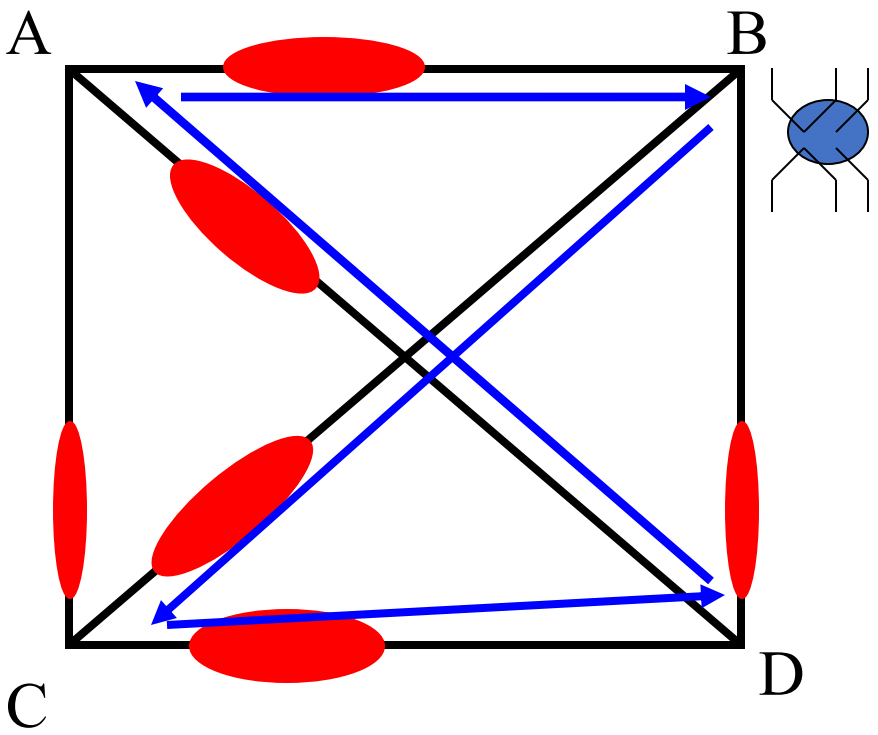


蚂蚁

AB: 10,　AC: 10,　AD, 30,　BC, 40,　CD 20

例：A 4-city TSP

- 之后，蚂蚁又从节点C，经D，A回到最初的节点B以完成任务。
- 完成任务后，蚂蚁会对本次路线进行评价并根据评价结果释放信息素。（问题2）



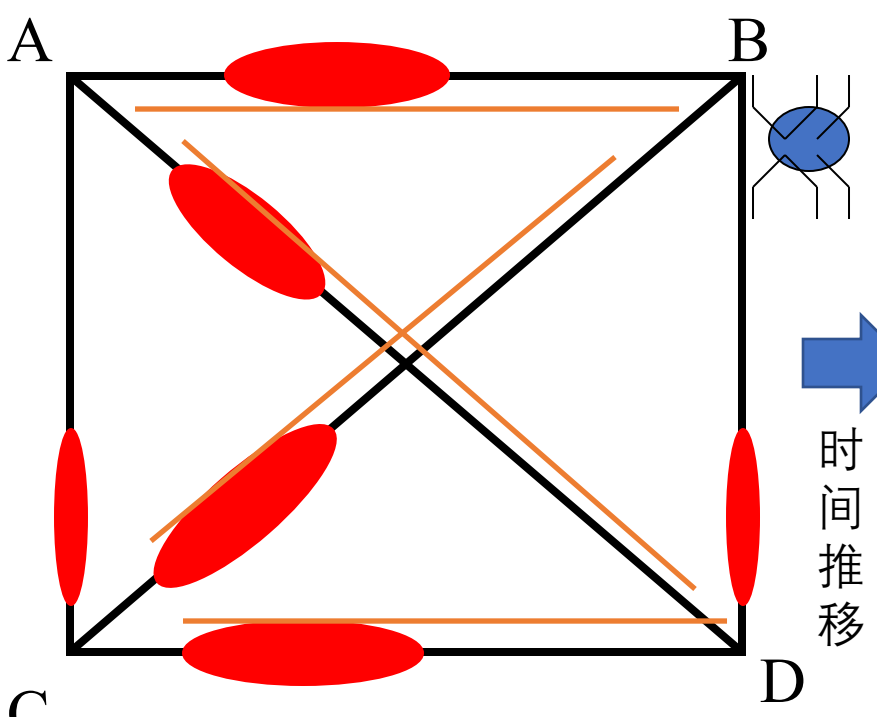
信息素

蚂蚁

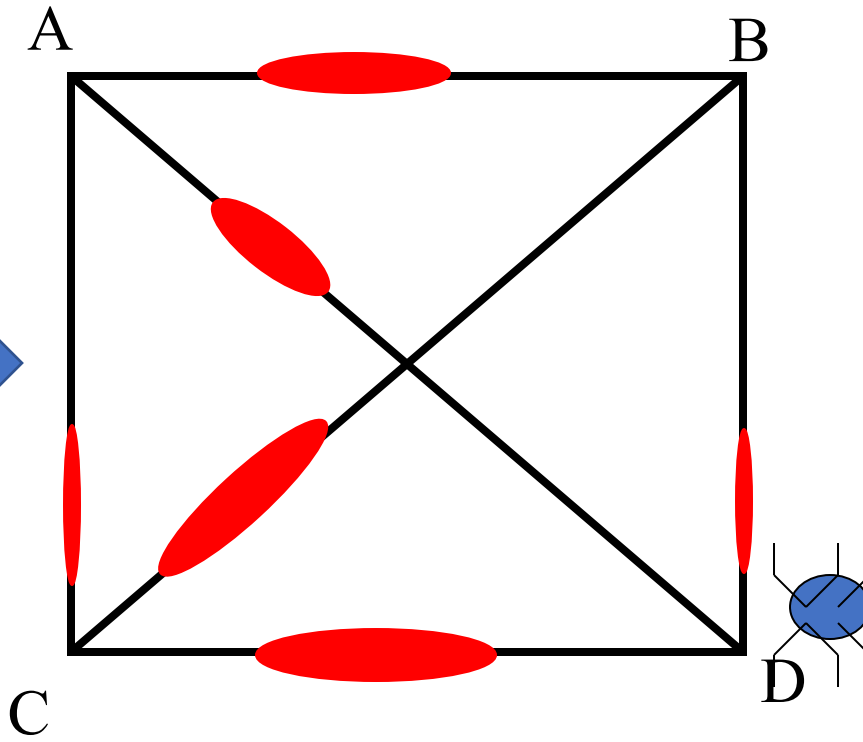
AB: 10, AC: 10, AD, 30, BC, 40, CD 20

例：A 4-city TSP

- 接下来，各路径的信息素会随着时间推移稍微减弱，以模拟信息素强度随着时间的推移而衰减的特性。 **（问题3）**



时间推移



信息素

蚂蚁

AB: 10, AC: 10, AD, 30, BC, 40, CD 20

TSP问题的ACO算法

我们有一个TSP问题，该问题有n个城市：

1. 我们在每个城市放置一些蚂蚁，然后每只蚂蚁做这样的事情：
    - 这些蚂蚁会遍历所有城市，然后回到起始城市。这些蚂蚁会利用 某种**规则**（transition rule）去决定每一步的走向. 这个规则是利用**信息素浓度**以及其它**启发性信息**（比如距离）来选定蚂蚁的下一座城市。
2. 当所有的蚂蚁完成了它们的旅程，更新整个图的全局信息素。
    - 信息素由每只蚂蚁（延迟）投放，具体方法如下：它将**信息素投放**在其旅行的所有路径上，其强度取决于路径的好坏程度。（这里面的“好”可以是路径最短，可以是成本最小，也可以是其他的自己定义的度量方式）
    - 所有路径当前的**信息素都会减弱**

然后我们回到步骤1，多次重复整个过程，直到我们触发了预先设定的终止条件（比如迭代次数）。

问题1：判断规则

判断规则

蚂蚁k能够选择从城市r到城市s路径的概率

T(r, s)从城市r直接到城市s的路径上的信息素的量

H(r, s)是该路径的启发价值– 在经典的TSP问题中，H(r, s) 是1/distance(r, s)

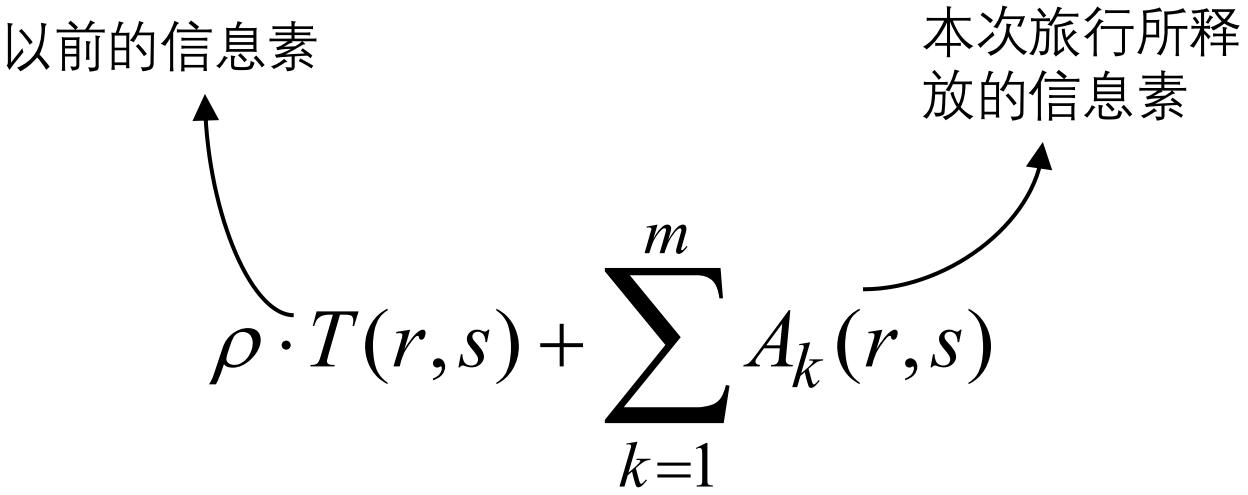
$$p_k(r,s) = \frac{T(r,s) \cdot H(r,s)^{\beta}}{\sum_{\text{unvisited cities } c} T(r,c) \cdot H(r,c)^{\beta}}$$

- 式中β是一个参数，我们可以称之为启发强度。

该式表达的是当蚂蚁在城市r，蚂蚁k能够选择从城市r到城市s的路径的概率。

问题2. 3：信息素更新

信息素更新公式



$T(r, s)$	是城市r与城市s之间旅行前的信息素
ρ	是衰减参数，用来描述信息素的衰减率
$A_k(r, s)$	是由蚂蚁k释放的城市r与城市s之间的信息素
m	是蚂蚁的数量

要点总结

1.每只蚂蚁对应一个计算智能体

2.蚂蚁根据概率选择位置进行移动

3.在经过的路径上留下“信息素”

4.“信息素”会随时间挥发

5.“信息素”浓度大的路径在后序选择中会以更高的概率被选中

蚁群优化算法（ACO）可以用来在图（Graph）中两顶点间寻找一条“好”的路径。（这里面的“好”可以是路径最短，可以是成本最小，也可以是其他的自己定义的度量方式）