人工智能技术
Artificial Intelligence

——人工智能: 经典智能+计算智能+机器学习
AI: Classical Intelligence + Computing Intelligence + Machine Learning

王鸿鹏、杜月、王润花、许丽

南开大学人工智能学院

第一部分 符号智能（经典人工智能）

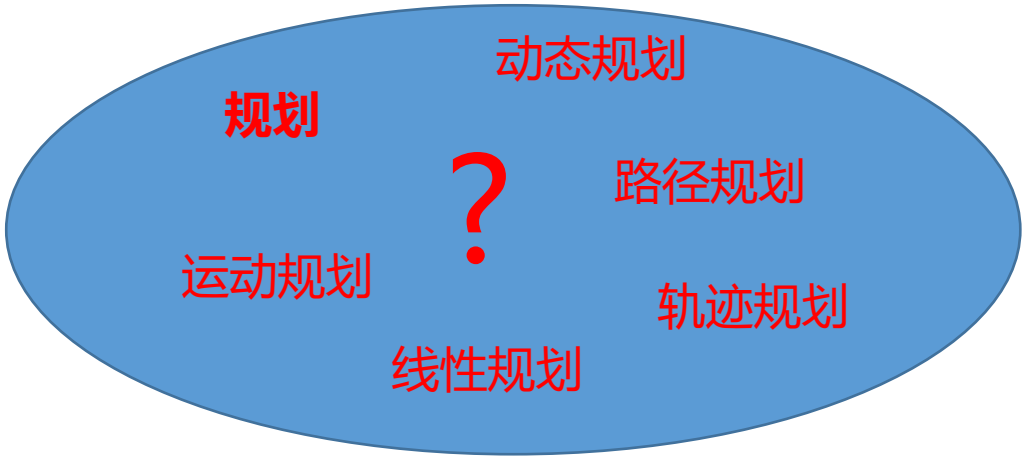
Symbolic Intelligence (Classical AI)

——第六章：行为规划

Chapter 6: Action Planning

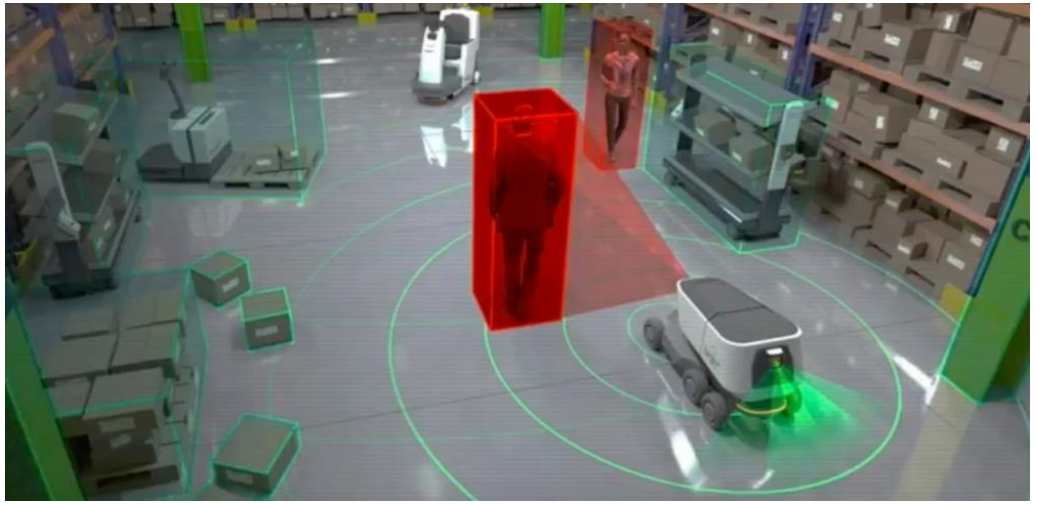
规划(program、plan、design、draft、sketch out)

- 规划，是融合多要素多人士看法的某一特定领域的发展愿景，意即进行比较全面的长远的发展计划，是对未来整体性、长期性、基本性问题的思考、考量和设计未来整套行动的方案。。
- 规划的基本意义由“规（法则、章程、标准、谋划，即战略层面）”和“划（合算、刻画，即战术层面）”两部分组成，“规”是起，“划”是落；从时间尺度来说侧重于长远，从内容角度来说侧重（规）战略层面，重指导性或原则性；在人力资源管理领域，一般用作名词，英文一般为program或planning。
- 规划具有综合性、系统性、时间性、强制性等特点。



自动规划(Automatic/Automated Planning)

自动规划是一种重要的问题求解技术，与一般问题求解相比，自动规划更注重于问题的求解过程，而不是求解结果。此外，规划要解决的问题，如机器人世界的问题，往往是真实世界的问题，而不是比较抽象的数学模型问题。



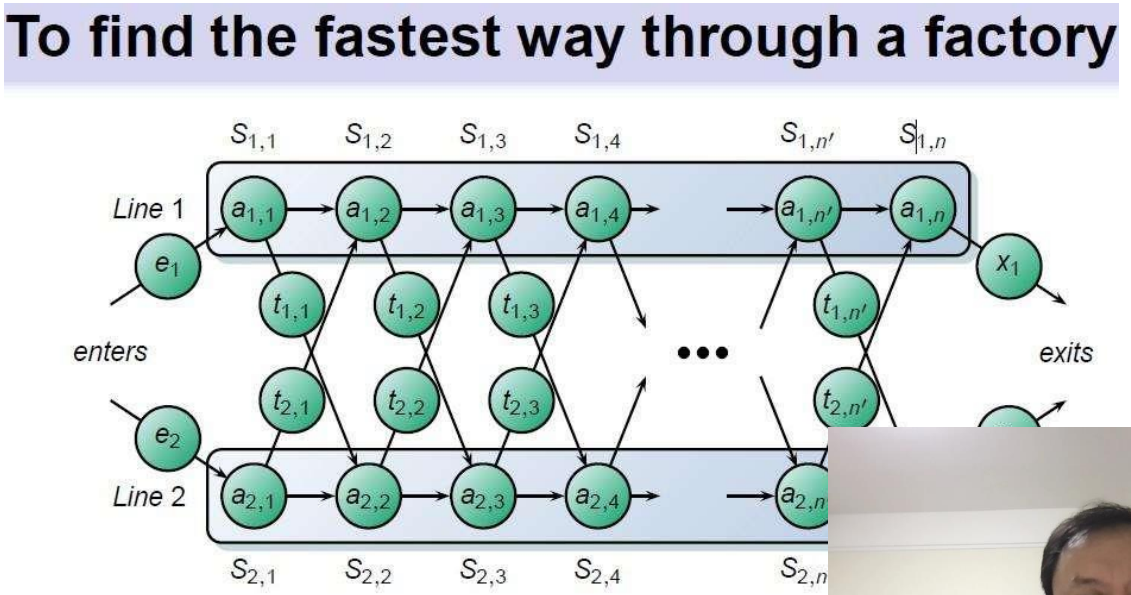
- 在研究自动规划时，往往以机器人规划和问题求解作为典型例子加以讨论。这不仅是因为机器人规划是自动规划最主要的研究对象之一，更因为机器人规划能够得到形象的和直觉的检验。有鉴于此，常常把自动规划称为机器人规划(Robot Planning)
- 机器人规划的原理、方法和技术，可以推广应用至其他规划对象或系统。自动规划是继专家系统和机器学习之后人工智能的一个重要应用领域，也是机器人学的一个重要研究领域，是人工智能与机人感兴趣的结合点。有些研究者把自动规划叫做智能规划(Intelligent Planning)

高层规划(High-level Planning) vs 低层规划(Low-level Planning)

动态规划(DP, Dynamic programming)

动态规划(dynamic programming)是运筹学的一个分支，是求解决策过程(decision process)最优化的数学方法。20世纪50年代初美国数学家R.E.Bellman等人在研究多阶段决策过程(multistep decision process)的优化问题时，提出了著名的最优化原理(principle of optimality)，把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，创立了解决这类过程优化问题的新方法——动态规划。1957年出版了他的名著《Dynamic Programming》

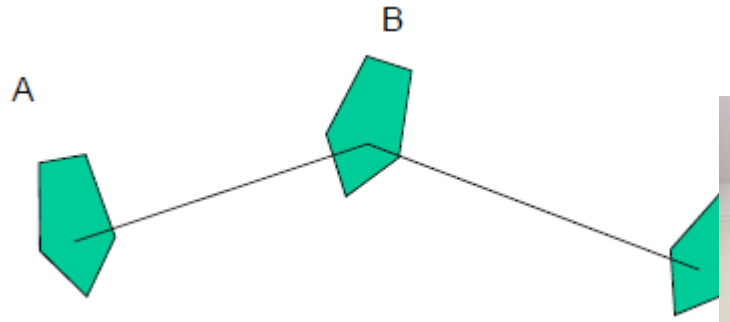
- 动态规划可将控制过程划分为若干个子过程，在每个子过程中，根据系统所处的状态和控制约束，确定该阶段的控制策略，整个控制过程的控制策略由各子过程的控制策略组合构成，总代价（性能指标）是各子过程的代价之和。这样便将最优控制问题转化为多阶段决策问题。动态规划是处理多阶段决策问题的有效方法。
- 动态规划一般可分为线性动规，区域动规，树形动规，背包动规四类。



运动规划(Motion Planning)

- 任务规划(Task Planning)：问题求解是一个寻求某个动作序列以达到目标的过程，机器人问题求解即寻求某个机器人的动作序列(可能包括路径等)，这个序列能够使该机器人达到预期的工作目标，完成规定的工作任务。
- 运动规划(Motion Planning)：在给定的位置A 与位置 B 之间为机器人找到一条符合约束条件的路径。这个约束可以是无碰撞、路径最短、机械功最小等。是机器人学的一个重要研究领域。
- 路径规划(Path Planning)：是指在起始位置和目标位置之间获得一条最优路径以完成某项任务，在获得路径的过程中需要经过一些必须经过的点，且不能触碰到障碍物。
- 轨迹规划(Trajectory Planning)：机器人末端执行器在参考坐标系空间由初始点运动到终止点产生的空间曲线称为机器人的轨迹。轨迹规划指操作机初始位置和目标位置之间用多项式函数来内插或逼近给定路径，并沿时间轴产生控制一系列设定点，供操作机控制用。

路径是Agent位姿的一个特定序列，而不考虑时间因素；
轨迹与何时到达路径中的每个点有关，依赖速度和加速度，强调了实践性



导航(Navigation)

导航问题的信息论公式化描述：假设机器人R在i时刻有一张地图M_i和一个初始的信任度状态b_i。机器人的目标是到达一个位置p，同时应满足某些时间约束： $loc_g(R) = p; (g \leq n)$。因此机器人必须在时间步n或之前到达目标位置p。

- 导航技术的移动机器人技术的核心和关键技术。自主移动机器人的导航就是让机器人可以自主按照内部预定的信息，或者依据传感器获取外部环境进行相应的引导，从而规划出一条适合机器人在环境中行走的路径。定位，就是机器人通过已经观测到的环境信息，结合自身已知的状态进行准确的极端出自身的位姿信息。
- 导航过程首先要获得相关的地图信息，然后进行路径规划，最后发送数据给机器人，并且通过相应的决策层，使其可以实现自主导航的功能。

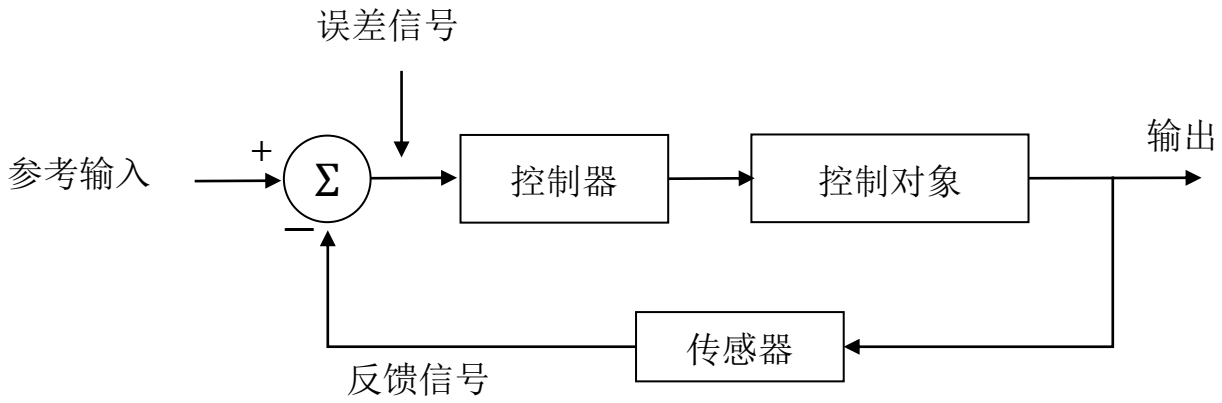


图 控制系统的基本组成

任务规划

NP-Hard问题与旅行商问题

NP-Hard问题(Non-deterministic Polynomial-Hard)

- 多项式时间
- 确定性算法与非确定性算法
- 规约/约化
- P类问题、NP类问题、NPC问题、NP难问题
- P=NP?

多项式时间(Polynomial Time)

时间复杂度

时间复杂度并不是表示一个程序解决问题需要花多少时间，而是当程序所处理的问题规模扩大后，程序需要的时间长度对应增长得有多快。也就是说，对于某一个程序，其处理某一个特定数据的效率不能衡量该程序的好坏，而应该看当这个数据的规模变大到数百倍后，程序运行时间是否还是一样，或者也跟着慢了数百倍，或者变慢了数万倍。

不管数据有多大，程序处理所花的时间始终是那么多的，我们就说这个程序很好，具有$O(1)$的时间复杂度，也称常数级复杂度；数据规模变得有多大，花的时间也跟着变得有多长，比如找n个数中的最大值这个程序的时间复杂度就是$O(n)$，为线性级复杂度，而像冒泡排序、插入排序等，数据扩大2倍，时间变慢4倍的，时间复杂度是$O(n^2)$ ，为平方级复杂度。还有一些穷举类的算法，所需时间长度成几何阶数上涨，这就是$O(a^n)$ 的指数级复杂度，甚至$O(n!)$ 的阶乘级复杂度。

多项式时间

时间复杂度

不会存在$O(2*n^2)$的复杂度，因为前面的那个"2"是系数，根本不会影响到整个程序的时间增长。同样地，$O(n^3+n^2)$的复杂度也就是$O(n^3)$的复杂度。因此，我们会说，一个$O(0.01*n^3)$的程序的效率比$O(100*n^2)$的效率低，尽管在n很小的时候，前者优于后者，但后者时间随数据规模增长得慢，最终$O(n^3)$的复杂度将远远超过$O(n^2)$。我们也说，$O(n^{100})$的复杂度小于$O(1.01^n)$的复杂度。

容易看出，前面的几类复杂度被分为两种级别，其中后者的复杂度无论如何都远远大于前者。像$O(1)$,$O(\ln(n))$,$O(n^a)$等，我们把它叫做多项式级复杂度，因为它的规模n出现在底数的位置；另一种像是$O(a^n)$和$O(n!)$等，它是非多项式级的复杂度，其复杂度计算机往往不能承受。当我们在解决一个问题时，我们选择的算法通常都需要是多项式级的复杂度，非多项式级的复杂度需要的时间太多，往往会超时，除非是数据规模非常小。

确定性算法与非确定性算法

确定性算法

设A是求解问题B的一个解决算法，在算法的整个执行过程中，每一步都能得到一个确定的解，这样的算法就是确定性算法。

非确定性算法

设A是求解问题B的一个解决算法，它将问题分解成两部分，分别为猜测阶段和验证阶段，其中：

- 猜测阶段：在这个阶段，对问题的一个特定的输入实例x产生一个任意字符串y，在算法的每一次运行时，y的值可能不同，因此，猜测以一种非确定的形式工作。
- 验证阶段：在这个阶段，用一个确定性算法（有限时间内）验证。①检查在猜测阶段产生的y是否是合适的形式，如果不是，则算法停下来并得到no；② 如果y是合适的形式，则验证它是否是问题的解，如果是，则算法停下来并得到yes，否则算法停下来并得到no。它是验证所猜测的解的正确性。

规约/约化

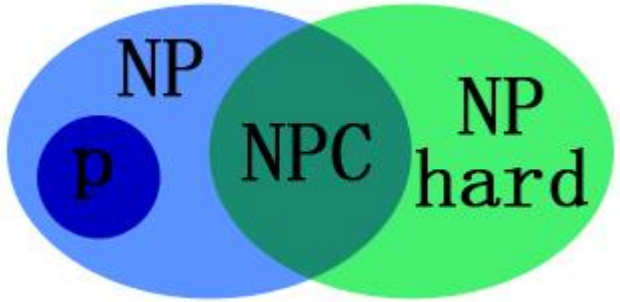
问题A可以约化为问题B，称为“问题A可规约为问题B”，可以理解为问题B的解一定就是问题A的解，因此解决A不会难于解决B。由此可知问题B的时间复杂度一定大于等于问题A。

《算法导论》中有一个例子：现在有两个问题：求解一个一元一次方程和求解一个一元二次方程。那么我们说，前者可以规约为后者，意即知道如何解一个一元二次方程那么一定能解出一元一次方程。我们可以写出两个程序分别对应两个问题，那么我们能找到一个“规则”，按照这个规则把解一元一次方程程序的输入数据变一下，用在解一元二次方程的程序上，两个程序总能得到一样的结果。这个规则即是：两个方程的对应项系数不变，一元二次方程的二次项系数为0。

从规约的定义中我们看到，一个问题规约为另一个问题，时间复杂度增加了，问题的应用范围也增大了。通过对某些问题的不断规约，我们能够不断寻找复杂度更高，但应用范围更广的算法来代替复杂度虽然低，但只能用于很小的一类问题的算法。存在这样一个NP问题，所有的NP问题都可以约化成它。换句话说，只要解决了这个问题，那么所有的NP问题都解决了。这种问题的存在难以置信，并且更加不可思议的是，这种问题不只一个，它有很多个，它是一类问题。这一类问题就是传说中的NPC问题，也就是NP-完全问题。

P类问题、NP类问题、NPC问题、NP难问题

- P类问题：能在多项式时间内可解的问题。
- NP类问题：在多项式时间内“可验证”的问题。也就是说，不能判定这个问题到底有没有解，而是猜出一个解来在多项式时间内证明这个解是否正确。即该问题的猜测过程是不确定的，而对其某一个解的验证则能够在多项式时间内完成。P类问题属于NP问题，但NP类问题不一定属于P类问题。
- NPC问题：存在这样一个NP问题，所有的NP问题都可以约化成它。换句话说，只要解决了这个问题，那么所有的NP问题都解决了。其定义要满足2个条件：
  - 它是一个NP问题；
  - 所有NP问题都能规约到它
- NP难问题：NP-Hard问题是这样一种问题，它满足NPC问题定义的第二条但不一定要满足第一条（就是说，NP-Hard问题要比 NPC问题的范围广，NP-Hard问题没有限定属于NP），即所有的NP问题都能约化到它，但是他不一定是一个NP问题。NP-Hard问题同样难以找到多项式的算法，但它不列入我们的研究范围，因为它不一定是NP问题。即使NPC问题发现了多项式级的算法，NP-Hard问题有可能仍然无法得到多项式级的算法。事实上，由于NP-Hard放宽了限定条件，它将有可能比所有的NPC问题的时间复杂度更高从而更难以解决。



P=NP?

- “P=NP?” 通常被认为是计算机科学最重要的问题。在很早的时候，就有个数学家毫不客气的指出，P=NP? 是个愚蠢的问题，并且为了嘲笑它，专门在4月1号写了一篇“论文”，称自己证明了 P=NP。
- 首先，我们要搞清楚什么是“P=NP?” 为此，我们必须先了解一下什么是“算法复杂度”。为此，我们又必须先了解什么是“算法”。我们可以简单的把“算法”想象成一台机器，就跟绞肉机似的。我们给它一些“输入”，它就给我们一些“输出”。比如，绞肉机的输入是肉末，输出是肉渣。牛的输入是草，输出是奶。“加法器”的输入是两个整数，输出是这两个整数的和。“算法理论”所讨论的问题，就是如何设计这些机器，让它们更加有效的工作。就像是说如何培育出优质的奶牛，吃进相同数量的草，更快的产出更多的奶。
- 世界上的计算问题，都需要“算法”经过一定时间的工作（也叫“计算”），才能得到结果。计算所需要的时间，往往跟“输入”的大小有关系。你的牛吃越是多的草，它就需要越是长时间才能把它们都变成奶。这种草和奶的转换速度，通常被叫做“算法复杂度”。算法复杂度通常被表示为一个函数f(n)，其中n是输入的大小。比如，如果我们的算法复杂度为n^2，那么当输入10个东西的时候，它需要100个单元的时间才能完成计算。当输入100 个东西的时候，它需要10000个单元的时间才能完成。当输入1000个数据的时候，它需要1000000个单元的时间。所谓的“P时间”，多项式时间，就是说这个复杂度函数f(n)是一个多项式。

P=NP?

- “P=NP?”中的“P”，就是指所有这些复杂度为多项式的算法的“集合”，也就是“所有”的复杂度为多项式的算法。为了简要的描述以下的内容，定义一些术语：

“f(n)时间算法”=“能够在f(n)时间之内，解决某个问题的算法”

- 当f(n)是个多项式（比如）的时候，这就是“多项式时间算法”（P时间算法）。当f(n)是个指数函数（比如2^n）的时候，这就是“指数时间算法”（EXPTIME算法）。很多人认为NP问题就是需要指数时间的问题，而NP跟EXPTIME，其实是风马牛不相及的。很显然，P不等于EXPTIME，但是P是否等于NP，却没有一个结论。

NP

- 现在我来解释一下什么是NP。通常的计算机，都是确定性（deterministic）的。它们在同一个时刻，只有一种行为。如果用程序来表示，那么它们遇到一个条件判断（分支）的时候，只能一次探索其中一条路径。比如：

```
if (x == 0) {
    one();
}
else {
    two();
}
```

- 在这里，根据x的值是否为零，one()和two()这两个操作，只有一个会发生。然而，有人幻想出来一种机器，叫做“非确定性计算机”（nondeterministic computer），它可以同时运行这程序的两个分支，one()和two()。这有什么用处呢？它的用处就在于，当你不知道x的大小的时候，根据one()和two()是否“运行成功”，你可以推断出x是否为零。这种方式可以同时探索多种可能性。这不是普通的“并行计算”，因为每当遇到一个分支点，非确定性计算机就会产生新的计算单元，用以同时探索这些路径。这机器就像有“分身术”一样。当这种分支点存在于循环（或者递归）里面的时候，它就会反复的产生新的计算单元，新的计算单元又产生更多的计算单元，就跟细胞分裂一样。一般的计算机都没有 这种“超能力”，它们只有固定数目的计算单元。所以他只能先探索一条路径，失败之后，再回过头来探索另外一条。所以，它们似乎要多花一些时间才能得到结果。

P=NP?

- 到这里，基本的概念都有了定义，于是我们可以圆满的给出P和NP的定义。P和NP是这样两个“问题的集合”：

P = “确定性计算机”能够在“多项式时间”解决的所有问题
NP = “非确定性计算机”能够在“多项式时间”解决的所有问题
（注意它们的区别，仅在于“确定性”或者是“非确定性”。）

“P=NP?”问题的目标，就是想要知道P和NP这两个集合是否相等。为了证明两个集合（A和 B）相等，一般都要证明两个方向：

A 包含 B；
B 包含 A。

上一个标题中我们已经说过NP包含了P。因为任何一个非确定性机器，都能被当成一个确定性的机器来用。你只要不使用它的“超能力”，在每个分支点只探索一条路径就行。所以“P=NP?”问题的关键，就在于P是否也包含了NP。也就是说，如果只使用确定性计算机，能否在多项式时间之内，解决所有非确定性计算机能在多项式时间内解决的问题。

P=NP？

- 我们来细看一下什么是多项式时间（Polynomial time）。我们都知道，n^2是多项式，$n^{1000000}$也是多项式。多项式与多项式之间，却有天壤之别。把解决问题所需要的时间，用“多项式”这么笼统的概念来描述，其实是非常不准确的做法。在实际的大规模应用中，n^2的算法都嫌慢。能找到“多项式时间”的算法，根本不能说明任何问题。对此，理论家们喜欢说，就算再大的多项式(比如 $n^{1000000}$，也不能和再小的指数函数（比如 1.0001^n）相比。因为总是“存在”一个M，当n>M的时候，1.0001^n会超过$n^{1000000}$。可是问题的关键，却不在于M的“存在”，而在于它的“大小”。如果你的输入必须达到天文数字才能让指数函数超过多项式的话，那么还不如就用指数复杂度的算法。所以，“P=NP?”这问题的错误就在于，它并没有针对我们的实际需要，而是首先假设了我们有“无穷大”的输入，有“无穷多”的时间和耐心，可以让多项式时间的算法“最终”得到优势。

旅行商问题(TSP, Traveling Salesman Problem)

- 旅行商问题(Traveling Salesman Problem, TSP)指的是：一个售货员从某城市出发，访问n个城市(售货)各一次且仅一次，然后回到原地，他走什么样的路线才能使走过的路程最短(或旅行费用最低)。
- 这个问题就是寻求总权最小的哈密尔顿(Hamilton)回路问题。
- 到目前为止，一般TSP问题还没有多项式算法，对于较大的问题(例如n大于40)就需要使用启发式方法求解。

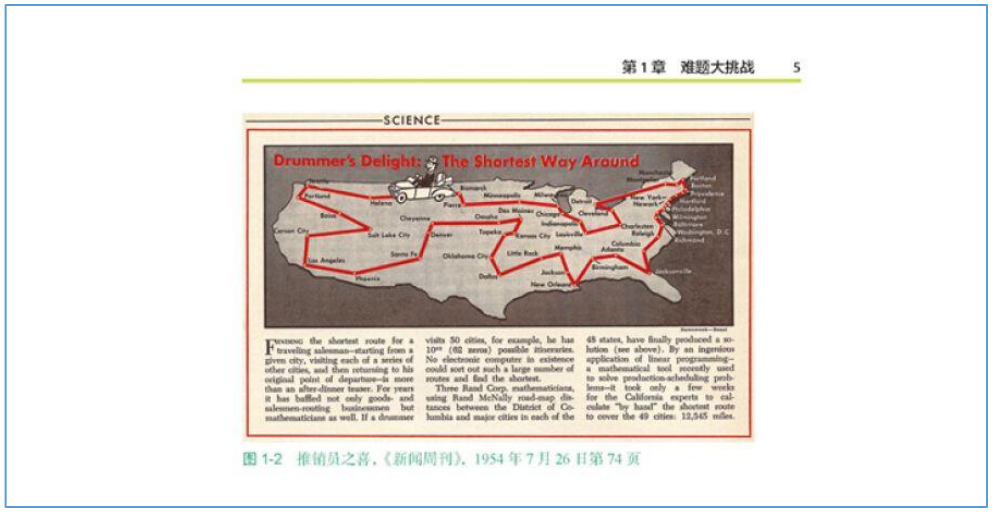
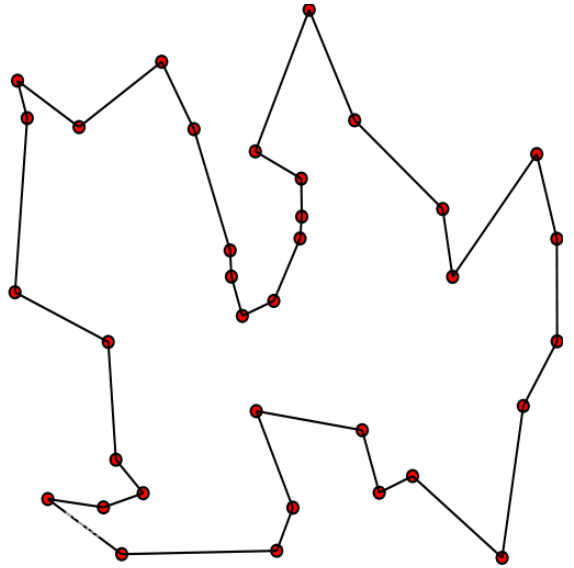
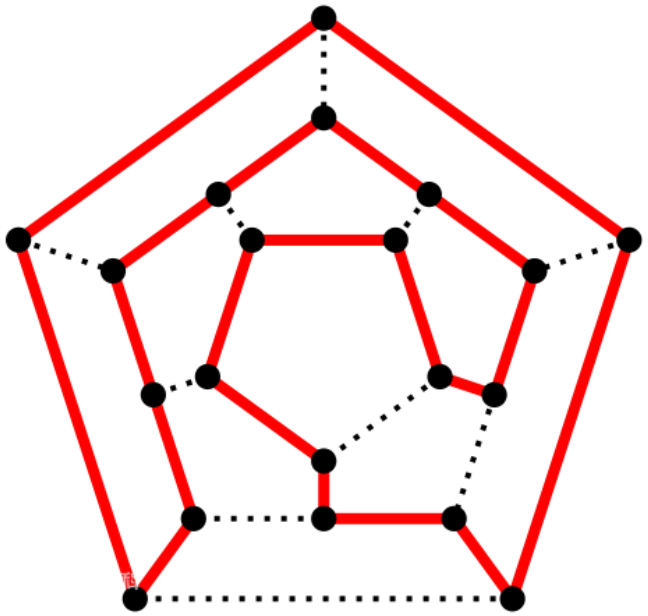


图 1-2 推销员之喜，《新闻周刊》，1954 年 7 月 26 日第 74 页



哈密尔顿回路(Hamilton Cycle)

- 哈密顿图(哈密尔顿图)(Hamiltonian graph，或Traceable graph)是一个无向图，由天文学家哈密顿提出，由指定的起点前往指定的终点，途中经过所有其他节点且只经过一次。在图论中是指含有哈密顿回路的图，闭合的哈密顿路径称作哈密顿回路（Hamiltonian cycle），含有图中所有顶点的路径称作哈密顿路径（Hamiltonian path）。



- 这个问题和著名的七桥问题的不同之处在于，过桥只需要确定起点，而不用确定终点。哈密顿问题寻找一条从给定的起点到给定的终点沿 途恰好经过所有其他城市一次的路径。
- 哈密顿路径问题在上世纪七十年代初被证明是“NP完全”的。据说具有这样性质的问题，难于找到一个有效的算法。

见示例c程序代码

启发式算法1：C-W节约算法

- 假定有n个访问地(例如城市)，把每个访问地看成一个点，并取其中的一个点为基点(起点)，例如以1点为基点。首先将每个点与该点相连接，构成线路$1 \to j \to 1(j = 2,3,\cdots,n)$，这样就得到了一个具有$n-1$条路线的图(当然，这时尚未形成Hamilton回路)。旅行者按此路线访问这n个点所走的路程总和为

$$z = 2\sum_{j=2}^{n} c_{1j}$$

其中c_{1j}为由点1到点j $(j = 2,3,\cdots,n)$的路段长度，注意此处假定$c_{1j} = c_{j1}$(对所有j)

若连接点i和点$j(i, j \neq 1)$，即使旅行者走弧(i,j)时(当然这时就不再经过弧$(i,1)$和$(1,j)$)，所引起的路程节约值$s(i,j)$可计算如下：

$$\begin{aligned} s(i,j) &= 2c_{1i} + 2c_{1j} - (c_{1i} + c_{1j} + c_{ij}) \\ &= c_{1i} + c_{1j} - c_{ij} \end{aligned}$$

对不同的点对(i,j)，$s(i,j)$越大，旅行者通过弧(i,j)时所节约的路程越多，因而应优先将其安排到旅行路线中去，使旅行者旅行时通过这一条弧。

启发式算法1：C-W节约算法

在具体应用该方法时，可按照以下步骤进行：

① 选取基点，例如选取点1为基点。将基点与其他各点连接，得到$n-1$条路线$1 \to j \to 1 (j = 2,3,\cdots,n)$。

② 对不违背限制条件的所有可连接点对(i,j)，如下计算其节约值(i,j不为基点)：

$$s(i,j) = c_{1i} + c_{1j} - c_{ij}$$

③ 将所有$s(i,j)$按其值大小由大到小排列。

④ 按$s(i,j)$的上述顺序，逐个考察其断点i和j，若满足以下条件，就将弧(i,j)插入到线路中。其条件是：

- 点i和点j不在一条路线上，而且
- 点i和点j均与基点相邻。

⑤ 返回步骤④，直至考查完所有可插入弧(i,j)为止。

通过以上各迭代步骤，使问题的解逐步得到改善，最后达到满意解（也有可能达到最优解）。

路径规划

Path Planning

路径规划(Path Planning)

- 智能机器人(智能体)是一类能够通过传感器感知环境和自身状态，实现在有障碍物的环境中面向目标的自主运动，从而完成一定作业功能的机器人(智能)系统。
- 导航技术是移动机器人技术的核心，而路径规划是导航研究的一个重要环节和课题。
- 路径规划(Path Planning)是指移动机器人按照某一性能指标(如距离、时间、能量等)搜索一条从起始状态到达目标状态的最优或次优路径。
- 路径规划主要涉及的问题包括：利用获得的移动机器人环境信息建立较为合理的模型，再用某种算法寻找一条从起始状态到达目标状态的最优或近似最优的无碰撞路径；能够处理环境模型中的不确定因素和路径跟踪中出现的误差，使外界物体对机器人的影响降到最小；利用已知的所有信息来引导机器人的动作，从而得到相对更优的行为决策。
- 移动机器人路径规划的主要方法包括：基于事例的学习规划方法、基于环境模型的规划方法、基于行为的路径规划方法等。
- 新的发展趋向于将多种方法相结合：
  ① 基于反应式行为规划与基于慎思行为规划的结合；
  ② 全局路径规划与局部路径规划的结合；
  ③ 传统规划方法与新的智能方法的结合（如人工势场与神经网络、模糊控制的结合等）。

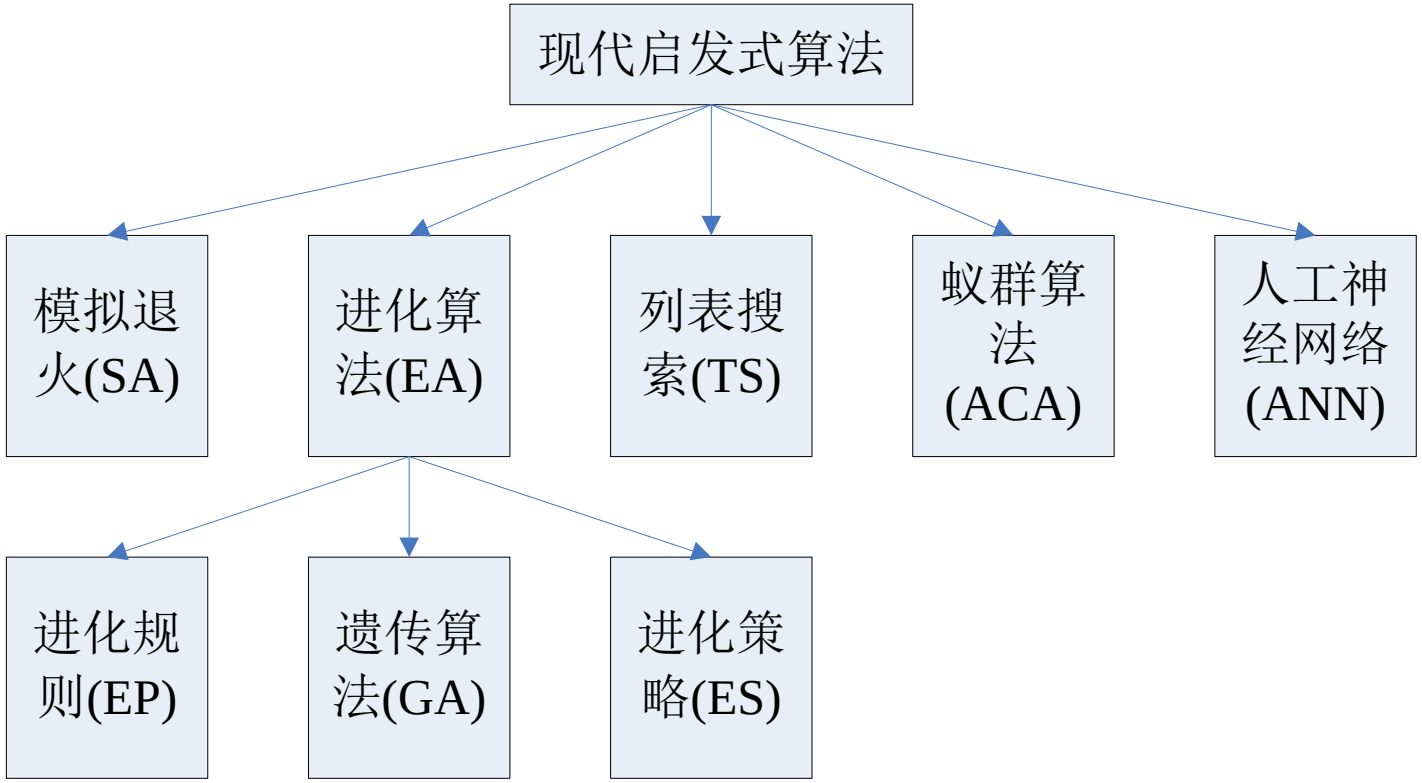
路径规划(Path Planning)

- 基于环境模型的规划方法

  - 基于环境模型的规划方法首先需要建立一个关于机器人运动环境的环境模型。
  - 根据掌握环境信息的完整程度可以细分为环境信息完全已知的全局路径规划和环境信息完全未知或部分未知的局部路径规划。
  - 全局路径规划：由于环境模型是已知的，其设计标准是尽可能使规划的效果达到最优。在此领域已经有了许多成熟的方法，包括可视图法、切线图法、Voronoi图法、拓扑法、惩罚函数法、栅格法等。
  - 局部路径规划：
    - 对环境信息完全未知的情况，机器人没有任何先验信息，因此规划是以提高机器人的避障能力为主，而效果为其次。已经提出和应用的方法有增量式的D*Lite算法和基于滚动窗口的规划方法等。
    - 对环境信息部分未知的情况，规划方法主要有人工势场法、模糊逻辑算法、遗传算法、人工神经网络、模拟退火算法、蚁群优化算法、粒子群算法和启发式搜索方法等。启发式方法有A*算法、增量式图搜索算法(又称为Dynamic A*算法)、D*和Focus色的D*等。

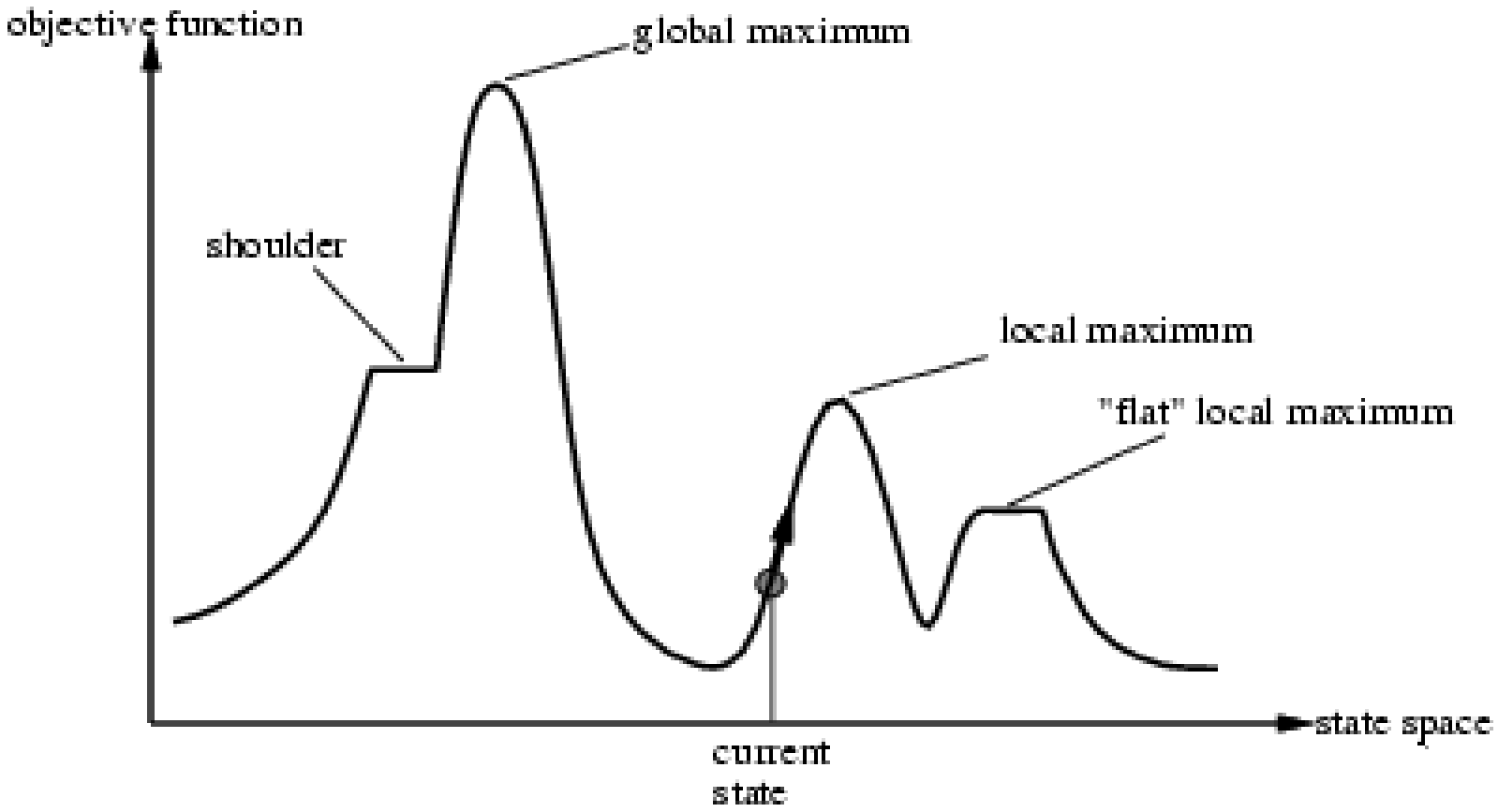
全局规划一般是建立在已知环境信息的基础上，适应范围相对有限；局部规划能适用于环境未知的情况，但有时反应速度不快，对规划系统品质的要求较高，因此如果把两者结合往往可以达到更好的规划效果

路径规划(Path Planning)



丛明煜, 王丽萍, 现代启发式算法理论研究, 高技术通讯, 2003,(05), p105-110

局部搜索与最优化



状态空间地形图

爬山法

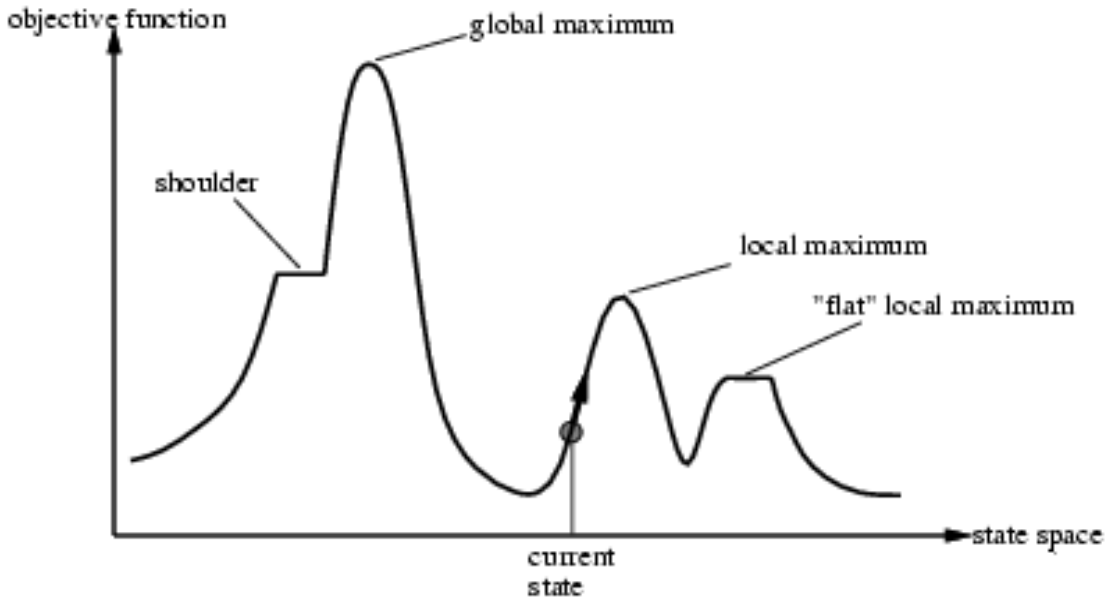
- 爬山法(Hill-climbing search)
  - 通过简单的循环，不断向值增加的方向持续移动
  - 节点只需要记录当前状态和目标值
  - 爬山法不会考虑与当前状态不相邻的状态。

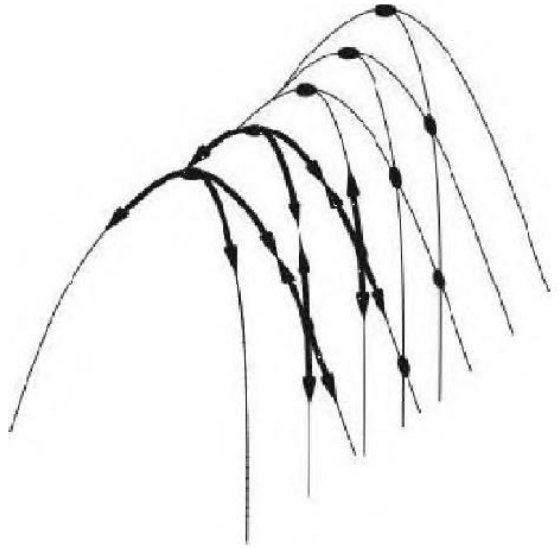
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

爬山法

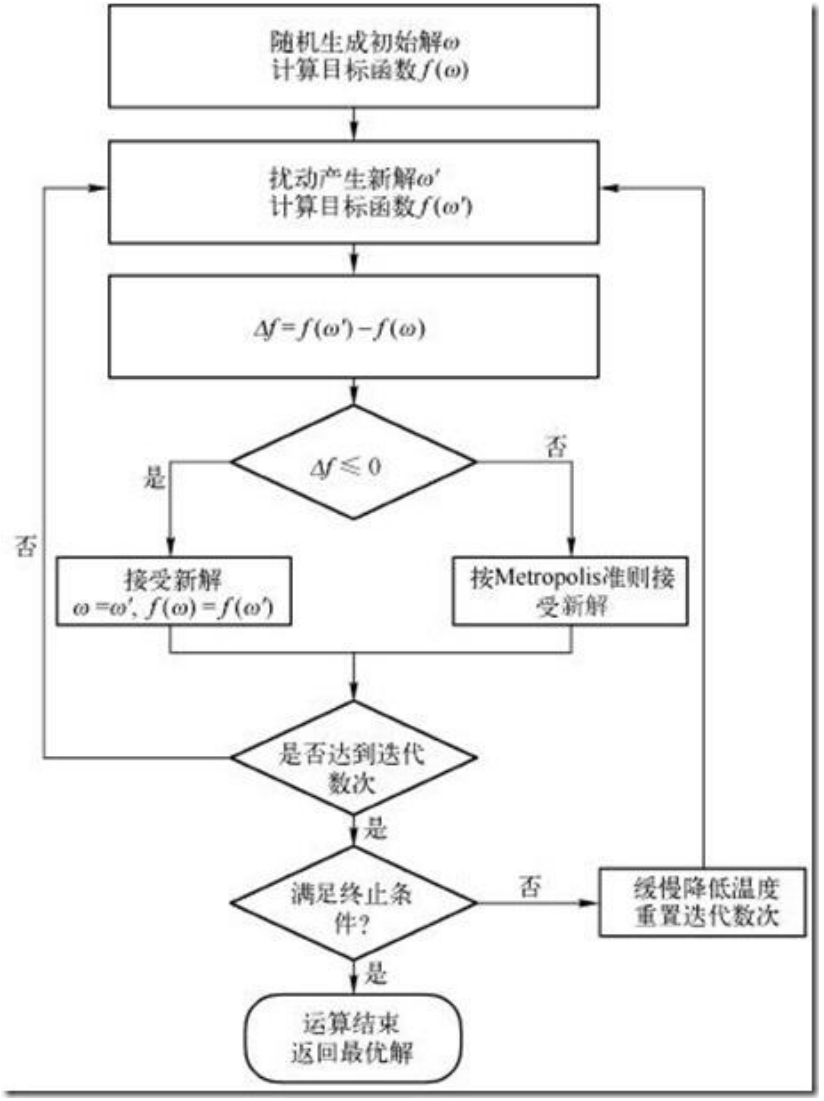
- 爬山法(Hill-climbing search)
  - 局部极大值：一个比它的每个邻接节点都高的顶峰，但是比全局最大值小。
  - 山脊：造成一系列局部极大值
  - 高原：在状态空间地形图上的一块平原区域。





模拟退火算法(Simulated Annealing, SA)

- 模拟退火算法是一种随机搜索算法，其原理是依据金属物质退火过程和优化问题之间的相似性。物质在加热的时候，粒子间的布朗运动增强，到达一定的强度后再进行退火，粒子热运动减弱，并逐渐趋于有序，最后达到稳定。
- 模拟退火算法过程是一个马尔可夫(Markov)决策过程，基于马尔可夫过程理论，可以证明模拟退火算法以概率1收敛于全局最优值。
- SA是一种解决组合优化问题的通用算法，只要优化问题能提供要给候选方案的适应性函数或费用函数，即可使用SA对它求解。
- SA算法通常应用于组合优化问题，典型的如TSP、大规模集成电路设计等。把机器人再未知环境下的随机漫游行为看作液体中粒子的布朗运动，则可以对其随机性的扰动应用SA方法来引导其向势能减小的方向上运动，从而实现未知环境下的在线动态规划。



Q&A

THANKS!