2 组合优化和启发式算法基础

2.1 组合优化

组合优化 (Combinatorial Optimization)[1]是属于计算机科学和离散数学的交叉学科，与算法理论、计算复杂性理论、运筹学密切相关。通常，在组合优化所研究的问题中，给定一个有限的解集合和目标函数，必须在解集合中找到最大化或最小化目标函数的解。下面给出严格定义。

定义 2.1： 组合优化问题的实例 (Instance) 是这样一个二元组 (S, f)，其中解空间 (Solution Space) S 是一个有限的解集合，目标函数 f 是一个解空间到实数集的映射 $f : S \to R$，即给解空间的中每一个解 $s \in S$ 赋予一个实数数值 $f(s)$。

定义 2.2： 组合优化问题由一组问题实例构成，并且是一个最小化 (Minimization) 或最大化 (Maximization) 问题。即对于给定的一个实例 (S, f)，找到实例的全局最优解 (Global Optimum) $s^* \in S$。对于最小化问题，全局最优意味着对于所有 $s \in S$，$f(s^*) \leq f(s)$ 成立；对于最大化问题，全局最优意味着对于所有 $s \in S$，$f(s^*) \geq f(s)$ 成立。在不造成歧义的情况下，全局最优解也称为最优解。

在接下来的介绍中，为了描述方便，在没有特别指出的情况下，所述组合优化问题都是最小化问题。以旅行商问题 (Traveling Salesman Problem, TSP)[25–27]为例，旅行商问题是研究历史最悠久的组合优化问题之一，它描述了给定 n 个城市，一个推销员需要访问 n 个城市各一次，并最终回到出发的城市。任意两个城市之间的距离已知，要求找到总距离最短的访问顺序，即找到一个访问 n 个城市的最短简单回路。旅行商问题显然属于组合优化问题，它对应着从所有访问 n 个城市的简单回路中找出距离最短的一个回路。对于该问题的一个实例 (S, f)，S 包含所有访问给定数目城市的简单回路，f 定义了每一条简单回路的长度。该问题实例的解可以用 n 个城市全排列 $\tau = (\tau(1), \tau(2), ..., \tau(n))$ 表示，其中 $\tau(i)$ 表示第 i 个访问的城市。可以看出，解空间的数量是城市数目的阶乘数量级。当有 100 个城市时，解空间的规模超过了 10^{150}，出现了组合爆炸，单纯依靠穷举搜索寻找最优解显然不切实际。通过旅

行商问题可以看出组合优化问题的典型特征，即组合优化问题可以依靠一些离散结构形式化表达出来，如图、顺序、排列、划分等。另外，组合优化问题的实例通常不是由直接给出解空间的每一个解以及对应的目标函数值来定义，而是通过一些间接的更加紧凑的数据形式来表达。如旅行商问题的实例可以通过给定任意一对城市之间的距离来定义，而不必直接给出每一个简单回路以及对应的长度。这样简洁的表达方式大大节约了问题实例需要占用的存储空间资源。

在组合优化问题中通常存在需要满足的约束，这些约束可以分成两类——硬约束 (Hard Constraint) 和软约束 (Soft Constraint)[28]。硬约束是指必须满足的约束，如在旅行商问题中，每个城市只能访问一次。满足所有硬约束的解称为合法解 (Feasible Solution)，否则称为非法解 (Infeasible Solution)。问题实例的解空间包含全部合法解。软约束是指尽量满足但不必一定满足的约束。例如在排课表问题中，为某门课安排上课时间槽，这门课的代课老师不希望安排在上午的最后一节时间槽。若最终的安排没有满足这一需求，得到的解仍是合法解，只是没有满足了这一需求的解更优。在许多问题中，存在多种软约束，并且不同的软约束之间存在冲突的可能性，即满足某些软约束必然会导致违反另外一些软约束。通常在实际情况中，目标函数定义为软约束的违反情况，即违反某一个软约束会引起一定量的惩罚值，不同种类的软约束按重要性设置不同的权重，目标函数等于所有软约束的违反惩罚加权之和。这种加权和的方式既可以体现相互冲突的软约束之间的平衡与折衷，也能够区分出不同软约束之间的重要等级。

在组合优化研究的对象中，存在着一类没有软约束的问题，这类问题本身没有目标函数，只需要找到满足所有硬约束的合法解。如在 k 图着色问题中，需要找到这样使用 k 种颜色为给定图的每个顶点着色的解，使得相邻顶点的颜色不同；在布尔可满足性问题 (Propositional Satisfiability Problem, SAT) 中找到使命题逻辑表达式为真的真值指派，这一类问题统称为判定型问题[29]。在设计算法求解判定型问题时，通常会将硬约束当成软约束处理，这样就可以定义具体的目标函数，使得找到合法解转换为找到一个目标函数值最小的解，即软约束违反最少的解。转换后，若找到的最优解的目标函数值等于 0，则表明找到了合法解，否则表明不存在合法解。如在处理 k 图着色问题时，解空间可定义为全部 k 着色方案集合，包括合法解和非

法解。对于某个解，若一对相邻顶点的颜色相同，则称这两个顶点为冲突顶点，连接这两个顶点的边为冲突边。目标函数定义为解的冲突顶点数或冲突边数。显然，若算法找到了一个目标函数值为 0 的解，则表明存在合法着色方案。

与判定型问题相对的是优化型问题[29]，这类问题本身存在目标函数来评估不同解的质量。如在图着色问题中，给定一个图，求最少需要多少种颜色为其着色，使得相邻顶点的颜色不同，该问题的解可由对应的着色数目评估。由此可以看出图着色问题与 k 图着色问题之间的区别和联系。k 图着色问题中颜色数目固定，需要找到一个合法着色方案，而在图着色问题中需要求出一个最小颜色数。对于每一个优化型问题，可以定义其相应的判定型问题：给定一个问题实例和一个数值 a，找到一个目标函数值小于等于 a 的解。通常，许多求解判定性问题的算法可以应用于求解相关的优化型问题。以求解图着色问题为例，图着色问题可以看做一系列的 k 图着色问题。当固定 k 的值，求解对应的 k 图着色问题时，若找到合法解，说明原问题至少可以用 k 种颜色着色。此时将 k 的值减 1 ，求解 $k-1$ 图着色问题。如此迭代下去直到对于 k' 图着色问题找不到合法解，则说明该算法找到的最少着色数是 $k'+1$。

2.2 计算复杂性理论

当设计求解组合优化问题的算法时，需要引入标准来评估什么样的求解算法是有效的算法。另一方面，学术研究和生产实践中存在各种各样的组合优化问题，是否所有组合优化问题都存在有效的求解算法，不同问题的求解困难程度是否相同，什么样的问题被认为更难求解，这些问题都属于计算复杂性理论的研究范围。

2.2.1 算法时间复杂度

给定一个组合优化问题的实例，一种自然直观的方法是搜索这个实例的解空间，找到最优化目标函数的解。因此，组合优化问题有时也被称作搜索问题。然而，解空间的规模可能非常大，如达到问题实例规模的指数数量级。这种情况下，当实例规模较大时，解空间中解的数目将达到天文数字级别。以 SAT 问题为例，当实例有 100 个布尔变量时，由于每个变量可以取值 True 或 False，那么一共就有 2^{100} 种真值

指派。当存在这种组合爆炸现象时，如何高效地搜索问题实例的解空间成为求解该问题的核心问题。

衡量一个算法的关键是看这个算法找到解需要消耗多少时间，在存储硬件成本不断降低的今天，运行时间成为评价算法广泛认可的标准之一。然而由于不同计算机的速度不同，采用自然时间会出现很大的差别。为了统一标准，采用算法需要执行的简单运算的总步数来作为算法的时间复杂度[1]。通常一个算法的时间复杂度与所求解问题的规模有关，即是问题规模的函数。在研究算法时间复杂度时，一般只关心当问题规模比较大时执行算法需要消耗的时间。这是因为当问题规模较小时，解空间中的解的数量少，不同时间复杂度的算法的运行时间差别不明显，一个很差的算法也可能在允许的时间范围内找到解。换句话说，只有当问题规模足够大时，才能看出运行时间的差别，体现不同算法之间的优劣。因此算法复杂性研究主要关心的是时间复杂度的数量级，即它的增长速度，相同数量级的时间复杂度的增长速度相同。

时间复杂度 $O(n^k)$ 的算法称为多项式时间复杂度算法，其中 n 是算法求解问题实例的规模，k 是一个常数。通常，当 $k=0$ 时，$O(1)$ 的算法称为常数时间复杂度算法；当 $k=1$ 时，$O(n)$ 的算法称为线性时间复杂度算法。时间复杂度大于多项式的算法称为指数时间复杂度算法，如 $O(2^n)$、$O(n!)$。显然，指数时间复杂度算法的时间增长速度大于多项式时间复杂度算法。当问题实例规模增大时，任意一个多项式时间复杂度的算法将比指数时间复杂度的算法更加有效。因此，Cobham[30]与 Edmonds[31]提出了多项式时间复杂度算法有效的概念。

2.2.2 P 与 NP

目前，多项式时间复杂度的算法被广泛认为是有效的。如果一个组合优化问题存在多项式时间复杂度的求解算法，则认为该问题是容易解决的，反之则被认为不易解决。对于许多问题，多项式时间复杂度的算法是已知的，如求最短路径的 Dijkstra 算法[32]。但对于有些问题目前没有找到多项式时间复杂度的算法，是否存在多项式时间复杂度的算法未知，如旅行商问题、图着色问题等。计算复杂性理论[33]即从计算复杂类别对不同的问题进行了分类研究。

值得注意的是计算复杂性理论主要是对判定型问题划分复杂类别，这是由于如

之前所述，对于每一个优化型问题，都可以定义相应的判定型问题。以旅行商问题为例，给定 n 个城市以及每一对城市之间的距离，要求找到一条经过所有城市的距离最短的简单回路。其对应的判定型问题是给定原问题中的 n 个城市和每一对城市之间的距离，以及常数 a，问是否存在总长度不超过 a 的经过所有城市的简单回路。显然，依靠一个求解旅行商问题的算法容易回答其对应的判定型问题，假设算法求得的最优解的长度为 k，若 $k \leqslant a$，则判定型问题的答案是存在，否则答案是不存在。反过来，判定型问题不比其对应的优化型问题求解更困难，如果一个判定型问题是不容易解决的，则对应的优化型问题也是不容易解决的。在此背景下，可以只讨论判定型问题的计算复杂类别。目前最常用的两种复杂类是 P 问题和 NP 问题[33]。下面给出具体定义。

定义 2.3： 若一个问题 X 的实例能够在多项式时间界限的确定型图灵机 (Deterministic Turning Machine, DTM) 上求解，则称该问题属于 P 问题，$X \in P$。

定义 2.4： 若一个问题 X 的实例能够在多项式时间界限的确定型图灵机上验证它的解是否合法，则称该问题属于 NP 问题，$X \in NP$。

可见，P 问题是指存在多项式时间复杂度算法的一类问题，而 NP 问题是指可以在多项式时间内验证解的问题。以旅行商问题为例，给定任意一条简单回路，能够在多项式时间内求得这条回路的总长度，并与常数 a 比较。因此旅行商问题属于 NP 问题。但目前仍未找到该问题的多项式时间复杂度求解算法，也没能证明它在多项式时间内无法解决。

由定义可得出 $P \subseteq NP$，但 $P \subset NP$ 是否成立目前未知，即 $P = NP$？直觉上看，寻找一个解要比验证一个解困难，但理论上尚未证明。目前较多学者倾向于 $P \neq NP$，即 NP 问题中包含一些难以求解的问题，不存在多项式时间复杂度的算法，或者可以说 NP 中最难的问题不属于 P。反过来看，如果 NP 中最难的问题属于 P，则 NP 中的所有问题属于 P，那么 $P = NP$ 成立。

计算复杂性理论表明多项式时间归约可以反映问题的难易程度。若问题 X 可以多项式时间归约到问题 Y，那么问题 Y 的求解难度不低于问题 X。更准确地说，若问题 X 可以多项式时间归约到问题 Y，而问题 $Y \in P$，则 $X \in P$。即如果问题 Y 有

多项式时间复杂度算法，则问题 X 也有多项式时间复杂度算法。由此可以定义 NP 问题中最难的问题——NP 完全 (NP-Complete) 问题[33]。

定义 2.5： 如果任意一个 NP 问题都可以多项式时间归约到某问题 X，则称问题 X 属于 NP 难度 (NP-Hard) 问题。

定义 2.6： 如果问题 X 是一个 NP 难度问题并且 $X \in NP$，则称问题 X 属于 NP 完全问题。

由于 NP 完全问题是 NP 问题中最难的一类问题，若存在某个 NP 完全问题 $X \in P$，则 $P = NP$ 成立。Cook[34]于 1971 年给出了第一个 NP 完全问题——布尔可满足性问题 (SAT)，即著名的库克定理 (Cook's Theorem)。接下来，一些其它问题也被证明是 NP 完全问题[35,36]，如图着色问题、顶点覆盖问题、独立集问题、哈密顿回路问题、整数线性规划问题等。

尽管目前 $P = NP$？问题没有得到解决，但如上所述研究人员普遍倾向于相信 $P \neq NP$，因此通常又把 NP 难度作为一个问题难解的依据，即若某个问题属于 NP 难度问题，则认为该问题难以求解。

2.3 启发式算法

计算复杂性理论表明许多问题属于 NP 难度问题，对于这类问题目前无法在多项式时间内找到最优解。如果采用精确算法来求解 NP 难度问题，需要指数时间才能找到最优解。当问题实例的规模增大时，算法执行的时间大大增长，在有限的计算资源条件下，无法得到最终的结果。因此，精确算法通常适用于求解规模较小或者结构特殊的问题实例。

基于 NP 难度问题求解困难的特性，研究人员发展出了一类快速高效的方法——启发式算法[28]。与精确算法追求最优解不同，启发式算法希望在有限的时间内得到高质量的解，对最优性不提供保证。换句话说，启发式算法找到的解不一定是最优解，但是是在可接受的时间范围内获得的。可见，启发式算法寻求求解优度和计算时间之间的平衡。

启发式算法多受到了生物行为、人类思维、自然现象的启迪，从这些外界经验中总结出抽象的规律帮助设计算法。大致来看，启发式算法可以分成三类：构

造式算法 (Constructive Heuristics)、基于解重组的算法 (Solution Recombination Based Heuristics) 以及基于局部搜索的算法 (Local Search Based Heuristics)[37]。除此之外，现在还存在将两种启发式算法结合起来的混合算法，如将基于解重组的算法和基于局部搜索的算法结合起来的混合遗传算法。从操作解的数量来看，启发式算法又可以分成基于单个解的算法和基于群体的算法。一般来说，构造式算法和基于局部搜索的算法属于基于单个解的算法，而基于解重组的算法属于基于群体的算法。

构造式算法通过迭代的过程逐步构造问题的解[37]。通常，组合优化问题的解可以表示成一些离散元素的顺序、排列或集合。构造式算法从空开始构造，在每一迭代步中，算法选取还未添加的某一元素并确定该元素的取值，通过循环迭代不断扩展不完整的解直到得到最终的完整解。以图着色问题为例，一种简单的构造式算法是每次选取一个还没有着色的顶点，并给这个顶点着上某种颜色，使得该顶点与其相邻的已经有颜色的顶点的颜色不同。从中可以看出，构造顺序对算法结果的影响是十分关键的，常见的两种确定顺序的方法是随机方式和打分标准。随机方式是指完全随机指定每次添加的解元素；打分标准是指设置某种与问题结构特性相关的评价原则给元素打分。按打分标准排序可以分成两种：一种是在构造之前确定好所有元素的添加顺序，这是一种静态的方式；另外一种是在构造过程中确定顺序，即每次添加了新元素后需要根据打分标准确定下一个待添加的元素，新元素的选择取决于当前的不完整解，这是一种动态的方式。以图着色问题为例，一种打分标准是顶点的度，根据顶点的度对所有顶点进行降序排序，每次从还未着色的顶点集合中选中度最大的顶点着色。Brélaz[38]提出了一个更加复杂的算法——Dsatur，算法框架如算法 1 所示。Dsatur 中引入了饱和度的打分标准。对于每个顶点，它的饱和度定义为与其相邻的已经着色的顶点包含的颜色种类数。显然，一个顶点的饱和度不是一成不变的，它与这个顶点的相邻顶点的着色情况有关，因此 Dsatur 是动态的构造式算法。构造过程中，每次挑选出饱和度最大的顶点为其着色。

构造式算法通常只能执行一次，没有迭代改进的过程，并且结果非常依赖于构造的顺序，因此所得解的质量往往不能达到满意的效果。目前，构造式算法主要用于求某个问题的初始解，或者作为子模块嵌入到其它启发式算法当中，如在贪心随机自适应搜索算法 (Greedy Randomized Adaptive Search Procedures, GRASP)[39–41]每次

Algorithm 1 求解图着色问题的 Dsatur 算法

1: **Input**: 图 $G = (V, E)$
2: **Output**: 颜色数
3: 给度最大的顶点着标号为 1 的颜色
4: **while** 未着色的顶点集合不为空 **do**
5: 从未着色的顶点集合中挑选出饱和度最大的顶点，若出现多个饱和度相等的顶点，从中挑选出在未着色顶点构成的剩余图中度最大的顶点
6: 使用标号尽可能小的颜色为挑选出的顶点着色，使得相邻顶点颜色不同
7: **end while**

迭代过程中先执行构造式算法，再利用局部搜索算法继续改进。

基于解重组的算法通过多个解来生成新的解[37]。这类算法属于基于群体的算法，算法中需要维护若干个解组成的群体，并不断更新群体。重组 (Recombination) 又称为交叉算符 (Crossover)，是指通过识别和选取两个或多个解的属性或元素，将这些不同解的部分属性或元素结合起来得到新的解。重组的原理在于将两个或多个解中优质的属性或元素结合起来可以得到新的有前途的解。遗传算法[42–45]是最经典的基于解重组的算法，其基本框架如算法框架 2 所示。

Algorithm 2 遗传算法基本框架

1: Initialization: 初始化群体
2: **while** 算法停止条件不满足 **do**
3: Selection: 选择父代解
4: Crossover: 对父代解进行交叉得到子代解
5: Mutation: 变异子代解
6: Replacement: 更新群体
7: **end while**

遗传算法受到了进化理论的影响，因此在算法中单个解又被称为染色体，解的属性或元素称为基因。用于重组的多个解称为父母或者父代解，生成的新的解称为子代。算法流程模拟了生物群体演化进程。初始化群体后，每次从群体中挑选出两个解作为重组的父母双亲，目前常见的挑选方法有随机方式、概率选择、排序机制、

锦标赛选择法 (Tournament Selection) 等[37]。为了增加生成优质子代的可能，也可以增加父代解的数量，不限于只选出两个。选出父代解之后，对其实行重组操作，得到子代解。遗传算法受到自然界物种进化过程中基因突变的启发，为提高解的多样性，也提供了对子代解实施变异操作的选择，通常只是对解做微小的改变。最后，用得到的子代解来更新群体。一种最简单的更新方法是用子代解替代群体中最差的解。除此之外，还有其它更加复杂的方法，这些方法在面对群体中被替代解的选择以及是否替代等问题时，都考虑了两方面的因素——群体的优度和多样性。优度方面，更新机制希望达到优胜劣汰的目的，将群体中质量较差的解替换掉，逐渐提高群体的整体优度；多样性方面，更新机制希望达到保持群体丰富性的目的。这是因为当群体中的解都趋于一致时，很难通过重组得到结构不同的子代解。

基于解重组的算法中最重要的是交叉算符。交叉算符截取每个父代解的部分属性或元素结合组成新的子代。常见的交叉算符有单点交叉算符 (One Point Crossover)、两点交叉算符 (Two Points Crossover) 以及均匀交叉算符 (Uniform Crossover) 等[37]。其中单点交叉算符如图 2-1 所示。图中示例解以 01 二元编码的方式表示，单点交叉算符随机选择一个交叉位置 (Crossover Point)，交叉位置之前的部分属性取自一个父代解，之后的部分属性取自另一个父代解，组合之后就可以生成两个子代。

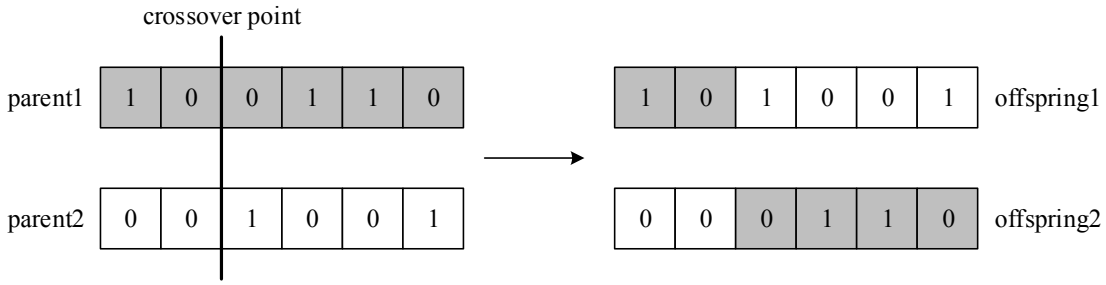

图 2-1 单点交叉算符示例

扩展单点交叉算符可得到 k 点交叉算符，$k=2$ 时两点交叉算符如图 2-2 所示。

与 k 点交叉算符中选取交叉位置来重组父代不同，在均匀交叉算符中，子代解的每个属性以相等的概率继承某个父代相应的取值。均匀交叉算符如图 2-3 所示。

目前，研究人员多将基于解重组的算法与其它启发式算法结合起来，形成一类新的混合算法。这是由于基于解重组的算法作为子模块可以提升整体搜索的多样性，

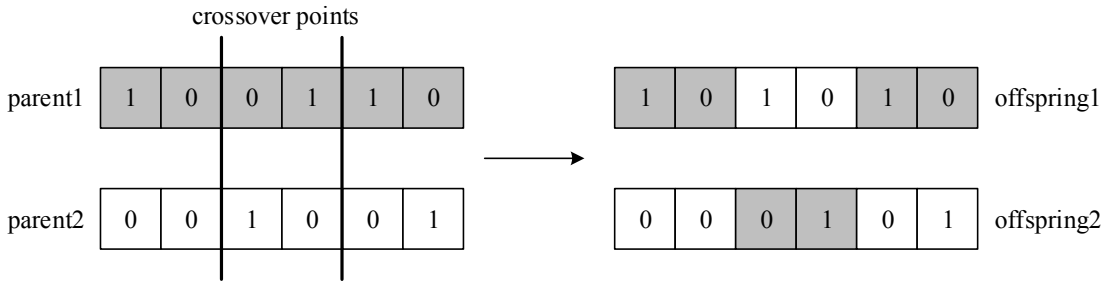

图 2-2 两点交叉算符示例

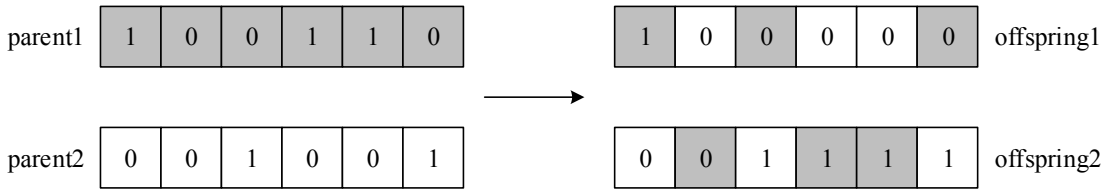

图 2-3 均匀交叉算符示例

使得算法可以探索更多的解空间区域。如 Xu 等人[46,47]提出了一种混合进化算法来求解单机调度问题，Peng 等人[48]提出了一种混合禁忌搜索路径重连算法来求解车间作业调度问题，Benlic 与 Hao[49]提出了一种多层次混合进化算法来求解图划分问题。

基于局部搜索的算法目前是运用最为广泛的一类启发式算法，这类算法通过迭代变换解来尝试改进解的优度，在每次迭代过程中，通常只是对解做局部的改变。除了基本的局部搜索算法以外，经典的基于局部搜索的算法包括禁忌搜索[50–52]、模拟退火[53,54]、迭代局部搜索[55,56]、变邻域搜索[57,58]、大邻域搜索[59,60]等。

2.3.1 局部搜索

局部搜索通过不断变换解来搜索解空间，这种变换是局部搜索算法的基本要素。邻域结构 (Neighborhood Structure) 或者邻域函数 (Neighborhood Function) 定义了这种变换，换句话说，对于某个解，邻域结构给出了解空间中哪些解在某种程度上是与其相邻的。通过这种变换，得到的解称为邻域解 (Neighboring Solution 或 Neighbor)。下面给出邻域结构相关定义[61]。

定义 2.7： 对于给定的组合优化问题实例 (S, f)，邻域结构是从解空间 S 到它的幂

集 2^S 的映射 $N : S \to 2^S$。对于解空间中的某个解 $s \in S$，邻域结构给出了这个解的邻域 $N(s) \subseteq S$，$N(s)$ 的规模 $|N(s)|$ 称为解 s 的邻域规模或邻域大小。如果解 $s' \in N(s)$，则称解 s' 为解 s 的邻域解。如果 $s' \in N(s)$ 当且仅当 $s \in N(s')$，则称邻域结构 N 是对称的。对某个解到其邻域解做的变换也称为邻域动作。

以旅行商问题为例，旅行商问题实例可以用一个带权有向完全图表示，其中顶点表示城市，弧的权值表示两个城市之间的距离。如果两个城市之间的距离是对称的，又称为对称旅行商问题，可以用带权无向完全图表示。以下为便于说明，以对称旅行商问题作为示例。对称旅行商问题的解可以表示为一条经过所有顶点的简单回路，这条回路包含顶点总数条边。通过删除这条回路中的两条边，再添加另外两条没有出现在回路中的边可以得到一条新的回路，即一个新的解，这种邻域结构称为 2-exchange。图 2-4 给出了 2-exchange 的一个例子，示例中删除了 (B, C) 和 (F, G) 两条边，添加上 (B, F) 和 (C, G) 两条边，得到新的回路 (A, B, F, E, D, C, G, H)。

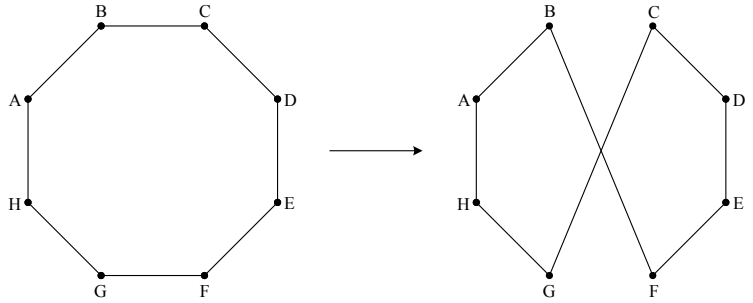

图 2-4 2-exchange 邻域结构示例

从图 2-4 可知，2-exchange 邻域结构是对称的，并且只删除一条边或删除两条相邻的边是不行的。因此，假设实例中有 n 个顶点，则对于任意一个解，它的 2-exchange 邻域规模是 $n(n-3)/2$。将 2-exchange 进一步扩展，可得到 k-exchange[62]。k-exchange 是指删除回路中 k' ($k' \leq k$) 条边并添加 k' 条新的边。图 2-5 给出了 3-exchange 的一个例子，注意示例中只给出了变换 3 条边的结果，省去了变换两条边的结果。

从图 2-5 中可以看出，变换 3 条边，可以得到 4 个邻域解，3-exchange 的邻域规模大于 2-exchange。并且根据 k-exchange 定义可知，对于任意一个解，它的 2-exchange 邻域解集合一定包含在 3-exchange 邻域中，是 3-exchange 邻域的子集，

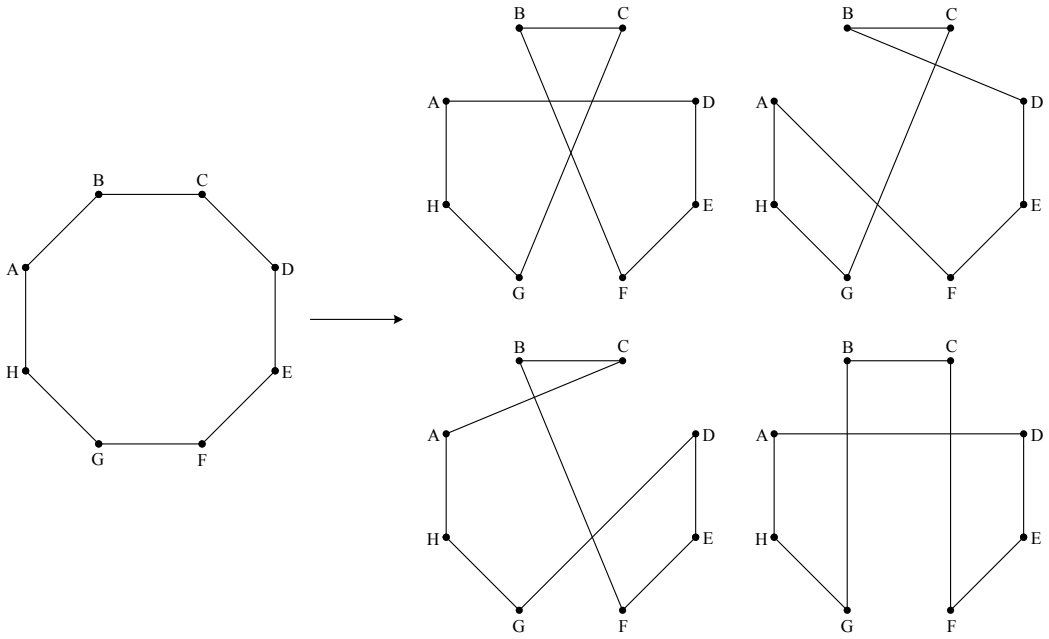

图 2-5 3-exchange 邻域结构示例

这种情况可称 3-exchange 包含 2-exchange。

定义 2.8： 给定一个组合优化问题实例 (S, f) 以及两种邻域结构 N_1 和 N_2，如果对于任意的解 $s \in S$ 都有 $N_1(s) \subseteq N_2(s)$，则称邻域结构 N_2 包含 N_1。

局部搜索算法从某个初始解开始，通过迭代不断将解变换为它的邻域解来搜索解空间，其基本流程如图 2-6 所示。

局部搜索持续用邻域解替代当前解的过程也可以模拟为搜索邻域图 (Neighborhood Graph)[61]。

定义 2.9： 给定一个组合优化问题的实例 (S, f) 以及相应的邻域结构 N，邻域图是一个点带权重的有向图 $G = (V, E)$。其中顶点集合 V 表示解空间 S，每个顶点表示一个解。对于任意一个顶点 i，它的权重即相应的目标函数值 $f(i)$。对于任意两个顶点 i 和 j，当且仅当 $j \in N(i)$ 时，在邻域图中存在一条由 i 到 j 的弧，即 $(i, j) \in E$。如果邻域结构 N 是对称的，那么可以用无向图来表示邻域图。

定义 2.10： 如果在邻域图 G 中存在一条从 i 到 j 的路径，则称解 j 对解 i 是可达的。也即存在一系列解 $s_1, s_2, ..., s_k$，$k > 1$，其中 $s_1 = i$，$s_k = j$，满足 $s_{l+1} \in N(s_l)$，$1 \leq l < k$。

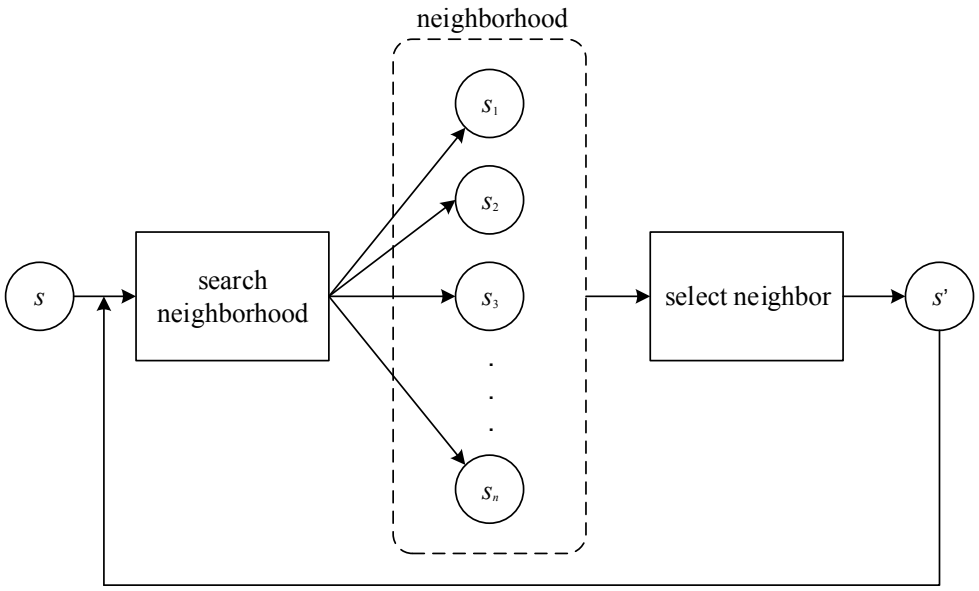

图 2-6 局部搜索基本框架

定义 2.11： 如果对于任意一对解 i 和 j，j 对 i 是可达的，则称相应的邻域图是强连通 (Strongly Connected) 的。如果对于任意一个解 i，存在一条从 i 到最优解的路径，则称相应的邻域图是弱最优连通的 (Weakly Optimally Connected)。显然，一个强连通邻域图肯定是弱最优连通的。

表 2.1 给出了一个具体的问题实例以及相应的邻域结构。从表中可知，该问题实例的解空间包含 8 个解，有两个全局最优解，分别是 s_3 和 s_6。另外，所给邻域结构是对称的，因此相应的邻域图可以用无向图表示，如图 2-7 所示。从图中可以看出，该邻域图被分成了两个分别由顶点集 $\{s_1, s_2, s_3, s_4\}$ 和 $\{s_5, s_6, s_7, s_8\}$ 构成的互不连通的子图，因此该邻域图不是强连通的，一个子图中的任一顶点对另一个子图中的任一顶点是不可达的。但由于两个子图中都包含了一个全局最优解，因此该邻域图是弱最优连通的。

表 2.1 问题实例及邻域结构示例

解	$f(s)$	$N(s)$	解	$f(s)$	$N(s)$	解	$f(s)$	$N(s)$	解	$f(s)$	$N(s)$
s_1	2	$\{s_2\}$	s_2	3	$\{s_1, s_3\}$	s_3	1	$\{s_2, s_4\}$	s_4	4	$\{s_3\}$
s_5	2	$\{s_6\}$	s_6	1	$\{s_5, s_7\}$	s_7	3	$\{s_6, s_8\}$	s_8	2	$\{s_7\}$

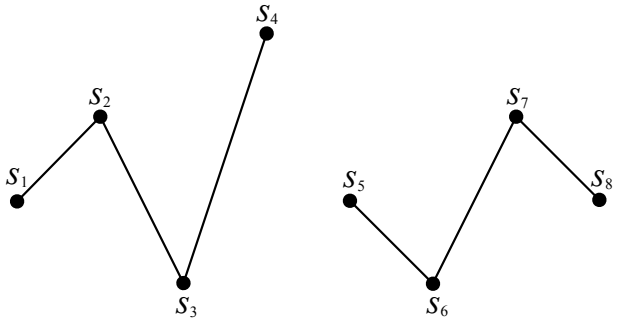

图 2-7 邻域图示例

游历邻域图有许多不同的方式，其中一种最简单的方式是迭代改进算法 (Iterative Improvement Algorithm)，也称为爬山法 (Hill Climbing Algorithm)[28]。在迭代改进算法的每一迭代步中，搜索当前解的邻域，寻找一个目标函数值更优的邻域解。如果存在这样的邻域解，则用它替换当前解；如果不存在，则称当前解为局部最优 (Locally Optimal)[61]，算法停止并返回局部最优解。

定义 2.12： 如果对于解 s' 的任意一个邻域解 $s \in N(s')$ 都有 $f(s') \leq f(s)$，则称 s' 是邻域结构 N 下的局部最优，在不引起歧义的情况下，也可简称为局部最优。

如在表 2.1 所给例子中，有 4 个局部最优解，分别是 s_1、s_3、s_6 和 s_8。局部最优是与具体的邻域结构相关的，一个局部最优解不一定是全局最优，如表 2.1 中的 s_1 和 s_8，而一个全局最优解一定是任意一种邻域结构下的局部最优。另外，邻域结构 N_1 下的局部最优不一定是邻域结构 N_2 下的局部最优，但如果 N_1 包含 N_2，则 N_1 下的局部最优一定是 N_2 下的局部最优。

相对于一般的局部搜索算法可以看做游历邻域图，迭代改进算法的执行过程可以看做游历转移图 (Transition Graph)[61]。

定义 2.13： 给定一个组合优化问题的实例 (S, f) 以及相应的邻域结构 N，转移图 $T = (V', E')$ 是邻域图 $G = (V, E)$ 的子图。其中 $V' = V$，$E' = \{(i, j) \mid \forall\, (i, j) \in E, f(j) < f(i)\}$，即转移图是删除了邻域图中两个权重相等的顶点之间的弧以及从权重较小的顶点到权重较大的顶点的弧。显然，转移图是一个无回路的有向图。

表 2.1 所给例子的转移图如图 2-8 所示。从图中可以看出，s_1、s_3、s_6 和 s_8 的出度为 0，s_4 和 s_5 的出度为 1，s_2 和 s_7 的出度为 2。根据转移图的定义可知，若一

个解是局部最优解，则它在转移图中出度为 0。如果某个顶点的出度等于 1，表示这个解的邻域中只有一个解比它更优，当访问到这个解时，下一步只能变换为唯一能够改进的邻域解。如果某个顶点的出度大于 1，表示这个解的邻域中有多个比它更优的解，需要从中选择一个邻域解来做变换。最常见的两种选择策略是最优改进 (Best Improvement) 和首次改进 (First Improvement)。

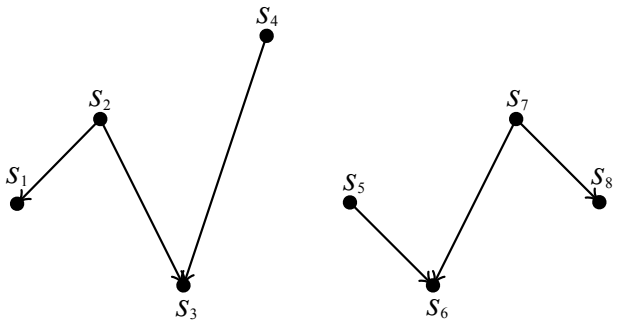

图 2-8 转移图示例

最优改进[37]是指每次搜索当前解的邻域，评估邻域中每个解，从中找出目标函数值最优的解。如果这个解优于当前解，则用它替换当前解，否则表示当前解是局部最优，算法停止并返回局部最优。最优改进算法框架如算法 3 所示。

Algorithm 3 最优改进局部搜索算法

Input: 问题实例 (S, f)
Output: 局部最优解 s
$s \leftarrow$ 生成初始解
while 还未找到局部最优解 **do**
    搜索 s 的邻域 $N(s)$，找到 $N(s)$ 中目标函数值最优的解 s'
    if $f(s') < f(s)$ **then**
        $s \leftarrow s'$
    end if
end while

首次改进[37]是指按某种顺序搜索评估当前解的邻域，当找到第一个能够改进当前解的邻域解时，搜索即停止，并用这个邻域解替换掉当前解。若搜索完整个邻域

都没有找到一个目标函数值优于当前解的邻域解，表示当前解是局部最优，算法停止并返回局部最优。首次改进算法框架如算法 4 所示。在首次改进算法中，搜索邻域的顺序既可以是完全随机，也可以是按照某种规则设定的先后次序。

Algorithm 4 首次改进局部搜索算法

1: **Input**: 问题实例 (S, f)
2: **Output**: 局部最优解 s
3: $s \leftarrow$ 生成初始解
4: **while** 还未找到局部最优解 **do**
5: 按某种顺序搜索 s 的邻域 $N(s)$，第一次遇到目标函数值优于 s 的解 s' 或搜索完 $N(s)$ 即停止
6: **if** 存在更优的邻域解 s' **then**
7: $s \leftarrow s'$
8: **end if**
9: **end while**

从最优改进和首次改进的算法描述中可以看出两者之间的区别。最优改进每次迭代需要搜索整个邻域，而首次改进大多数情况下只用搜索部分邻域，特别是当邻域中有多个可以改进当前解的邻域解时，首次改进往往只需搜索整个邻域的一小部分，这种情况常见于局部搜索算法的初始阶段，因此首次改进每一步所花费的时间通常要小于最优改进。然而，由于最优改进是挑选出整个邻域中目标函数值最优的解，因此每一步的改进幅度通常大于首次改进，到达局部最优解的迭代步数通常也少于首次改进执行的步数。采用不同的搜索策略会导致算法每一迭代步需要搜索评估的邻域规模以及到达局部最优执行的迭代步数有所差别，这两个因素都对局部搜索算法的运行时间有影响。深入来看，评估单个邻域解是局部搜索算法最基本的操作，迭代步数和每一步评估的邻域规模限定了算法评估单个邻域解的总次数。因此，提升评估单个邻域解的效率对任何局部搜索算法都是关键且有效的。

不管是最优改进还是首次改进，迭代改进算法找到局部最优解时即终止。而局部最优解不一定是全局最优，也可能优度较差，迭代改进算法的主要缺陷即在于对于返回的局部最优解的优度没有保障。为了克服这种缺陷，需要引入其它策略帮助

局部搜索算法逃脱局部最优陷阱，使得算法不会在找到一个较差的局部最优时即终止。一些常见的策略包括：(1) 设计多种邻域结构，如之前所述，局部最优是与具体的邻域结构相关的，一种邻域结构下的局部最优不一定在其它邻域结构下也是局部最优，通过多种邻域结构来搜索解空间可以探寻更广的区域；(2) 执行多轮局部搜索，迭代改进算法找到的局部最优与初始解有关，变换一个初始解，可能得到不同的局部最优解；(3) 放松接受邻域解的标准，在迭代改进算法中，不管是最优改进还是首次改进都只接受能够改进当前解的邻域解，如果遇到局部最优解时，可以考虑接受优度等于或差于当前解的邻域解，使得局部搜索算法可以继续迭代下去以搜索其它解空间区域。

2.3.2 迭代局部搜索

在迭代改进算法中，每次搜索限制于当前解的邻域范围，并以改进目标函数为导向，当到达局部最优解时即停止。如之前所述，邻域动作通常只是对解做局部的改变，所得到的解称为邻域解。一种简单逃脱局部最优的思路是对局部最优解做一个较大的改变，或称为扰动 (Perturbation)。这种扰动摆脱了邻域的限制，扰动之后，得到一个新的解，可在此之上继续进行局部搜索。迭代局部搜索[55]即是基于这种思路设计的启发式算法，其框架如算法 5 所示。

Algorithm 5 迭代局部搜索算法框架

1: **Input**: 问题实例 (S, f)
2: **Output**: 算法找到的最好解
3: $s_0 \leftarrow$ GenerateInitialSolution()　// 生成初始解
4: $s^* \leftarrow$ LocalSearch(s_0)　// 局部搜索
5: **while** 算法停止条件未满足 **do**
6: 　$s_0 \leftarrow$ Perturbation($s^*, history$)　// 扰动
7: 　$s' \leftarrow$ LocalSearch(s_0)
8: 　$s^* \leftarrow$ AcceptanceCriterion($s^*, s', history$)　// 接受标准
9: **end while**

从算法 5 可以看出，通过扰动可以获得新的初始解 (第 6 行)，给定一个初始解

s_0，子模块局部搜索 (第 7 行) 可以得到一个局部最优解，并判断是否接受这个局部最优解。作为子模块的局部搜索，既可以是简单的迭代改进算法，如最优改进局部搜索算法，也可以是其它更加高级的启发式算法，如禁忌搜索、模拟退火等。迭代局部搜索可以看做一种更高层次的局部搜索算法，它对局部最优解的集合进行搜索。如果扰动某个局部最优解，生成的初始解经过局部搜索后得到另一个局部最优解，这两个局部最优解可以看做是“相邻的”。

对于生成迭代局部搜索算法的首个初始解模块 (GenerateInitialSolution)，通常在设计算法时有两种实现方式。一种是构造式启发式算法，一种是随机生成。构造式启发式算法如之前所述，通常按照某种与问题相关的次序逐步添加解的元素，最终得到一个完整的解，如求解图着色问题的 Dsatur 算法[38]。而随机生成中没有引入与问题相关的信息，按照随机无偏向的方式构造一个解，以图着色问题为例，为每一个顶点以均等概率着上某一种颜色可以快速得到一个随机初始解。相对于随机生成方式，构造式算法得到的初始解的优度更高，接下来的局部搜索到达局部最优解所需的时间更短。因此，设计启发式算法时，在计算时间较短的条件下，通常采用构造式算法来生成初始解。如果可以保证计算时间，多数实践证明两种方法对最终结果优度的影响没有明显差异。

在迭代局部搜索算法中，相对于子模块局部搜索中的邻域动作，扰动对解做了幅度更大的改变，改动幅度或扰动次数称为扰动强度 (Perturbation Strength)[55]。除了幅度更大以外，执行扰动操作时通常也不考虑目标函数。一种最简单的方式是随机执行一个对解的结构改动更大的邻域动作，以旅行商问题为例，如果在子模块局部搜索中采用了 2-exchange 邻域结构，那么扰动时可以考虑变换更多边的 k-exchange ($k > 2$)。扰动的目的是为了得到一个新的初始解，逃脱局部最优解。理想的结果是扰动后的解处于新的有前途的解空间区域，经过局部搜索之后找到更优的局部最优解。如果扰动强度太小，对解的改动太小，新一轮的局部搜索可能回到原先的局部最优。如图 2-9 所示，当对 s^* 扰动得到 s_1 时，经过局部搜索仍将回到 s^*，而如果扰动到 s_0，则经过局部搜索将得到一个目标函数更优的局部最优 s' 。如果扰动强度太大，完全改变解，那么算法变成了随机重启 (Random Restart)，失去了迭代局部搜索对局部最优解的集合搜索的优点，最终所得解的质量降低，并且扰动

强度过大得到的解的优度很差，以此作为初始解，局部搜索需要更长时间得到局部最优解。通常，扰动强度设置为动态自适应的取值，而不是固定不变。例如将扰动强度设定成与问题实例规模成正比，当处理较大规模的实例时，扰动强度也适当增大；或者随着算法的运行动态调整扰动强度。扰动机制的精细程度取决于引入其中的问题相关的信息量，当考虑更多与问题结构特性相关的信息时，设计的扰动更加有针对性和引导作用。如 Lourenço[63]在设计求解车间作业调度问题算法的扰动机制时，固定解的一部分属性不变，求解所得子问题，这样做既能达到生成新的初始解的目的，同时也能保证扰动后的解的优度不会变差太多。Qin 等人[64]在设计求解单机调度问题的迭代局部搜索算法时，通过概率选择的方式将三种扰动策略结合起来使用，发挥不同策略扰动力度不同的特点。

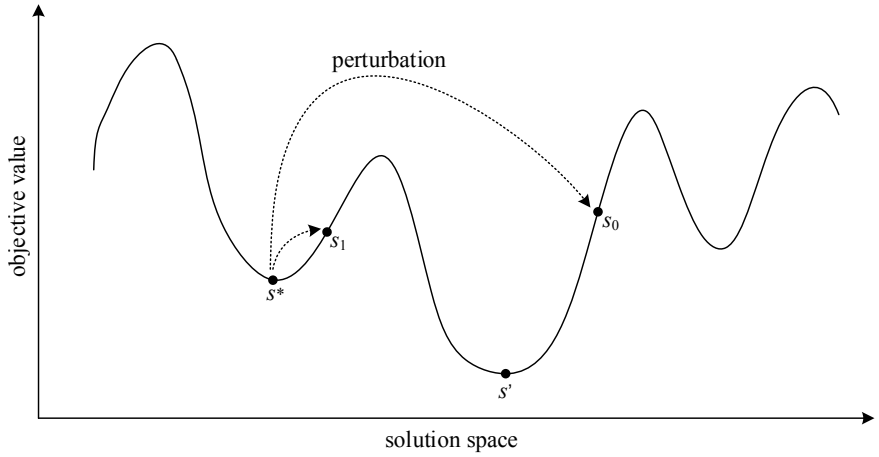

图 2-9 迭代局部搜索算法扰动示例

迭代局部搜索算法可看做是对局部最优解的搜索，扰动以及之后的局部搜索联合一起对当前的局部最优解做变换得到新的局部最优解，类似于迭代改进算法，一种最简单的接受标准是当新的局部最优解更优时，用它替换当前的局部最优解。为了增加算法的多样性，可以考虑接受较差的局部最优解，如模拟退火的概率接受方式。而当不考虑新的局部最优解的优度，总是用它替换当前局部最优解时，迭代局部搜索变成了在局部最优解组成的集合上的随机游走 (Random Walk)。实际处理时，需依据不同问题的结构特性，选择不同的接受标准。

迭代局部搜索在许多组合优化问题的求解上表现出色，如二次分配问题[65]、

车辆路由问题[66–68]、单机调度问题[64,69]、流水车间调度问题[70]、取送货旅行商问题[71]等。

2.3.3 禁忌搜索

当迭代改进算法访问到局部最优解时，由于没有目标函数值更优的邻域解，可以考虑接受较差的邻域解。选择某个较差的解替换局部最优解变成当前解后，若邻域结构是对称的，则之前的局部最优解也一定在当前解的邻域里，由于目标函数值优于当前解，那么很有可能又重新回到这个局部最优解，出现循环重复搜索的现象。为了解决重复访问的问题，需要引入某种记忆机制辨别已经访问过的解以避免走回头路。基于这种思路，Glover[50]于 1986 年首次提出了禁忌搜索，并在之后继续完善扩展了禁忌搜索的原理和策略[51,52]，其基本框架如算法 6 所示。

Algorithm 6 禁忌搜索算法框架

1: **Input**: 问题实例 (S, f)
2: **Output**: 算法找到的最好解
3: $s \leftarrow$ GenerateInitialSolution()　// 生成初始解
4: InitializeTabuList(T)　// 初始化禁忌表
5: **while** 算法停止条件未满足 **do**
6: 　搜索当前解 s 的邻域 $N(s)$，令 $\tilde{N}(s)$ 表示其中非禁忌的邻域解和满足特赦准则的邻域解组成的集合，找到 $\tilde{N}(s)$ 中目标函数值最优的解 s'，即 $s' = \arg\min_{x \in \tilde{N}(s)} f(x)$
7: 　$s \leftarrow s'$
8: 　UpdateTabuList(T)　// 更新禁忌表
9: **end while**

从算法 6 可以看出，禁忌搜索每一步迭代从当前解邻域的子集中挑选出一个最好解来替换当前解，这个子集包括没有禁忌的邻域解和满足特赦准则的邻域解。禁忌搜索不限制挑选出的邻域解一定要优于当前解，可以接受优度没有改进的解。算法的基本思想在于通过允许接受非改进解使得遇到局部最优解后搜索可以继续进行，通过设置禁忌表 (Tabu List) 这种记忆结构使得搜索避免回到之前已经访问过的解。

相对于其它基于局部搜索的启发式算法，禁忌是禁忌搜索算法最显著的特性。当搜索从某个局部最优解迭代到非改进的邻域解后，算法保留下的一些信息使得接下来的搜索禁止选择存储在“记忆”中的解。以 SAT 问题为例，当某一迭代步翻转一个布尔变量后，即将这个变量的取值由 True (False) 变为 False (True)，那么下一步不允许再翻转这个布尔变量，因为这样会抵消上一步翻转的效果，使得搜索返回到之前的状态。除了避免返回到局部最优解，禁忌还可以引导搜索离开已经访问过的解空间区域，探索其它未访问的区域。

在设置记忆的禁忌对象时，通常有多种选择。一种直观的选择是记下访问过的解，但在实际算法实现中，往往不采用这种方式。因为随着搜索的进行，保存下完整的解需要消耗大量的存储资源，并且每次迭代需要花费很大代价来检查一个邻域解是否之前访问过。因此，通常选择邻域动作或解的属性作为禁忌对象，保存下最近执行过的邻域动作或者出现过的属性[37]。以图着色问题为例，假定邻域动作是变换某个顶点的颜色，这样的一个邻域动作由三个基本元素构成——顶点、顶点当前颜色以及变换的颜色。面对这个问题时，有多种设置禁忌对象的方式。如当某一迭代步顶点 v 的颜色从红色变为蓝色，可以将这样一个三元组 (v, 蓝色, 红色) 设为禁忌对象，表明接下来禁止将顶点 v 的颜色从蓝色变为红色。可以看出这种方式的禁忌力度不是很强，因为它只是不允许顶点 v 的颜色从蓝色变为红色，但是从其它颜色变为红色是可以的。顶点 v 可以先变成黄色，再由黄色变为红色，可以看做绕了一条更远的路，结果顶点 v 再次着上红色，出现了循环的现象。为了加强禁忌的力度，可以将二元组 (v, 红色) 设为禁忌对象，它表明不论顶点 v 着何种颜色，禁止将其变为红色，这种方式避免了通过其它中间颜色回到红色的重复现象。除此之外，一种更强的方式是只记下顶点 v ，将 (v) 设为禁忌对象，接下来禁止对其做任何邻域动作。这三种方式禁忌的力度逐渐增强，在面对具体问题时，需要分析问题的特点来设计相应的禁忌对象。另外，若算法中有多种邻域结构时，也可能需要采用多种禁忌对象。

禁忌对象存储在禁忌表中，因此禁忌表可看做算法的记忆结构。禁忌搜索利用这种记忆结构，通过禁止选择最近执行的邻域动作的逆动作或者禁止选择最近出现过一些解属性，避免搜索在局部最优解和它的邻域解之间循环往复或走出一条由若

干个中间解构成的回路。当搜索执行了一定步数后，解已经发生了足够的改变，选择一些“记忆久远”的禁忌对象不会使得搜索返回到之前访问的区域，这时再继续保持这些对象的禁忌状态没有意义，甚至可能产生负作用。因此，经过一段时间后，需要解除禁忌状态，这种只记忆近期搜索信息的策略称为短期记忆。禁忌表中对象保持禁忌状态的周期长度或迭代步数称为禁忌周期 (Tabu Tenure)[37]，禁忌周期的设置对于一个纯禁忌搜索算法具有重要的意义。当禁忌周期太短时，同一时间禁忌对象的数量较少，搜索可能绕路通过一系列中间解重新回到之间访问过的解，探索新的解空间区域的可能性降低。适当增长禁忌周期，可以避免搜索限制于一小片解空间区域，增强算法的疏散性，提高找到优质解的概率。而当禁忌周期过长时，同一时间禁忌对象的数量太大，可选择的邻域动作变少，这时某种程度上少数可选的邻域动作变成搜索的引导力量，而不是目标函数，算法的集中性减弱，找到一片区域中的局部最优解的可能性降低。

随着搜索的进行，一些老的禁忌对象的禁忌状态结束，一些新的对象开始进入禁忌状态。禁忌表的更新逻辑上可以模拟为一个先进先出表的入队出队操作[37]，图 2-10、图 2-11 和 图 2-12 分别给出了不同情况下操作禁忌表的示例。当搜索还未开始时，禁忌表为空，此时没有禁忌对象，随着搜索迭代下去，开始有禁忌对象进入禁忌表。当新的禁忌对象从队尾入队时，禁忌表中其它禁忌对象往队首挪动。若禁忌表已满，则存储在队首的禁忌对象出队，表示该对象的禁忌状态结束。因此，先进先出表的长度决定了禁忌周期，以图 2-10、图 2-11 和 图 2-12 为例，一个对象进入禁忌表，经过 7 步迭代后即出队结束禁忌状态。禁忌周期的设置有两种方式：一种是静态固定值；另一种是动态自适应方式，如与邻域规模成正比，或在某个范围内随机取值，或随着搜索的进行动态改变等，这种动态改变的方式既可以是整体一致的，也可以是针对每一个禁忌对象的。相比于固定值的禁忌周期，动态自适应的禁忌周期更加灵活智能。

记忆禁忌对象的结果不仅是避免搜索回到之前访问过的解，其它任何可由禁忌动作或禁忌解属性带来的解也无法访问。禁忌周期越长，这些无意中被禁忌却实际没有访问过的解的数量越多，其中可能包括一些质量很好的解，这种效应不利于提升算法找到的解的优度。在这种情况下，应当忽视禁忌对象的禁忌状态，在算法中

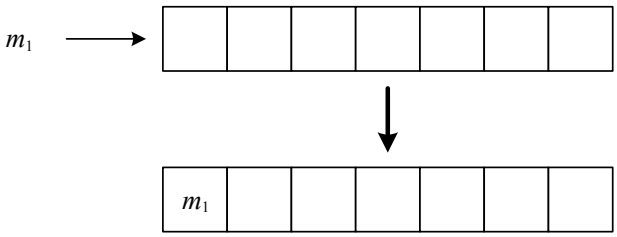

图 2-10 空禁忌表

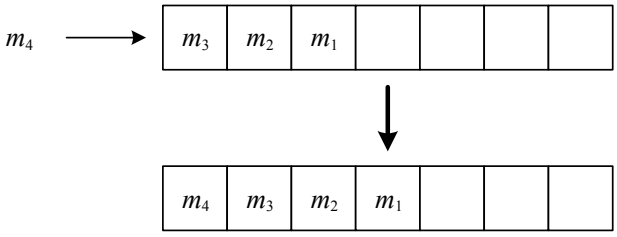

图 2-11 部分满禁忌表

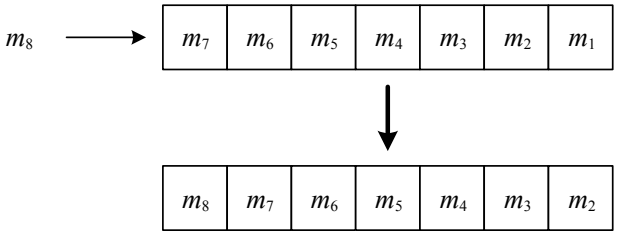

图 2-12 满禁忌表

称之为特赦准则 (Aspiration Criterion)[51,52]。特赦准则的原则在于当禁忌对象不会导致重复访问时，应当不考虑它的禁忌状态。目前禁忌搜索实现中，一种最常用的特赦准则是当选择某个禁忌对象可以得到一个比当前找到的最好解更优的解时，忽略其禁忌状态。显然，一个优于当前最好解的解肯定是从未访问过的。

禁忌搜索广泛应用于求解各种组合优化问题，如图着色问题[72]、车间作业调度问题[73]、路由与波长分配问题[74]、时刻表问题[75]、人员排班问题[76]等。

2.3.4 模拟退火

模拟退火是最著名的基于局部搜索的启发式算法之一，由 Kirkpatrick[53]和 Černý[54]于上世纪 80 年代分别独立提出。模拟退火受到了结晶固体加热然后逐渐降

温冷却以达到规律的晶格结构的启发，当结晶固体降温过程足够慢时，最终将得到结构高度完整的结晶固体，没有晶格缺陷。模拟退火求解的组合优化问题的目标函数相当于结晶固体的能量状态，固体退火时通过温度的调节进入稳定的低能量状态，模拟退火中引入温度参数辅助算法找到优质解，其基本框架如算法 7 所示。

Algorithm 7 模拟退火算法框架

1: **Input**: 问题实例 (S, f)
2: **Output**: 算法找到的最好解
3: $s \leftarrow$ GenerateInitialSolution()　// 生成初始解
4: $t \leftarrow$ InitializeTemperature()　// 初始化温度参数
5: **while** 算法停止条件未满足 **do**
6:　**for** $i : 1 \rightarrow n$ **do**
7:　　随机从当前解 s 的邻域 $N(s)$ 中选择一个解 s'
8:　　$\Delta f = f(s') - f(s)$　// 计算邻域解和当前解的目标函数差
9:　　**if** $\Delta f \leqslant 0$ **then**
10:　　$s \leftarrow s'$
11:　　**else**
12:　　以 $e^{-\Delta f/t}$ 概率接受邻域解 s'
13:　　**end if**
14:　**end for**
15:　UpdateTemperature(t)　// 降低温度参数
16: **end while**

从算法 7 可知，不同于禁忌搜索等算法搜索邻域的方式，模拟退火每次从当前解的邻域中随机挑选一个邻域解。如果这个邻域解的目标函数值不差于当前解，则用它替换当前解 (第 9-10 行)，否则按一定概率 $e^{-\Delta f/t}$ 接受 (第 12 行)，这种接受较差邻域解的可能性为搜索提供了逃脱局部最优解的机会。接受一个变差的邻域解的概率由温度参数 t 控制，当 t 值较大时，$e^{-\Delta f/t}$ 值较大，接受邻域解的可能性高；当 t 值变小时，$e^{-\Delta f/t}$ 值也随之变小，接受邻域解的可能性降低。t 值趋近于无穷大时，算法一定会接受变差的邻域解，搜索变成了随机游走；t 值趋近于 0 时，算法不会接

受任何变差的邻域解，搜索变成了迭代改进。由于按概率接受较差邻域解，因此当前解也有可能没有发生改变。可见，温度参数控制了模拟退火算法的搜索轨迹。

模拟退火算法在某一温度下，执行一定步数的迭代搜索。当算法刚开始运行时，为了提升算法逃脱局部最优解的能力和搜索的疏散性，使得尽可能多的解空间区域被探索到，初始温度应设为一个相对较大的值，使得较差的邻域解容易被接受。常见的做法是预先对解空间进行随机抽样，根据样本的标准方差来确定初始温度，并可通过后期计算实验继续调整至合适水平。随着算法的进行，逐步降低温度，增加算法的集中性，使得搜索可以聚焦于一片区域找到其中的优质解。常见的调节温度方式有两种：(1) 线性下降，即 $t = t - c$，$c > 0$，每次将当前温度减去一个降温常数得到下一阶段的温度；(2) 几何下降，即 $t = \alpha \cdot t$，$0 < \alpha < 1$，每次将当前温度乘以一个降温系数得到下一阶段的温度[37]。

为了增强算法的求解能力，可以进一步扩展基本的模拟退火算法。当温度下降到某一水平，搜索已经连续较长时间没有更新找到的最好解，并且当前解发生变换的比例很低时，可以认为此时搜索已“冻结”[37]，即算法很难找到更好的邻域解，并且由于接受差解的概率低，当前解也难以被替代，搜索进入停滞阶段。一旦出现“冻结”现象，继续降低温度意义不大，可以考虑适当地升温，提高接受差解的可能性，加强算法的疏散性，通过这种反复升温降温的操作可以改进模拟退火算法求解问题的能力。

模拟退火是较早提出的基于局部搜索的启发式算法，经过 30 年的发展，在许多组合优化问题的求解上展现出良好的效果，如超大规模集成电路设计[77,78]、设施配置问题[79]、团队定向问题[80]等。

2.4 本章小结

本章首先给出了组合优化的相关概念，描述了组合优化问题的特征及分类；然后介绍了计算复杂性理论，对组合优化问题的求解困难程度进行分类；最后阐述了启发式算法，介绍了启发式算法的分类和特点。

第 1 节组合优化：给出了组合优化问题的定义，描述了组合优化问题中的两种约束——硬约束和软约束，并将组合优化问题分成判定型问题和优化型问题两大类。