中国科学院大学课程讲义

高级算法设计与分析

中国科学院大学 计算机与控制学院

软件与理论教研室

蔡少伟

2019 年 11 月 20 日

目录

第一章 组合优化问题与建模

组合数学（Combinatorics）是一个数学分支，主要研究有限的或者可数的离散结构，包括这些离散结构的存在性、计数和构造等问题。组合优化一般是指在有限个可能解的集合中找出最优解的一类优化问题。组合优化（或称为离散优化）是一门古老而又年轻的研究领域，著名数学家 Fermat，Euler 等都为其形成和发展作出过重要贡献。伴随着工业科技革命和现代管理科学的发展，特别是计算机技术的突飞猛进和在各行业的广泛应用，组合优化已壮大成为独立的研究分支，也是运筹学和计算机科学的交叉学科。

组合优化问题在各个领域都广泛存在。典型的组合优化问题包括旅行商问题、调度问题、背包问题、装箱问题、最大团问题、聚类问题、图着色问题等。这些问题描述非常简单，并且有很强的工程代表性。组合优化问题也广泛存在于人们的现实生活中，比如交通调度，路径规划，课程表安排，会议安排等。本章介绍组合优化问题的基本概念和主要的模型，并对求解方法做一个概述。

1.1 组合优化问题

我们经常会遇到需要寻找一个最优方案的问题，也即最优化问题。我们首先对实例和问题做一个区分。在本课程中，不失一般性地，当我们在做一般性讨论的时候，都假定所讨论的最优化问题是最小化问题。

定义 1.1-1 （最优化问题的实例） 一个最优化问题的一个实例是一个二元组(F,c)，其中 F 是一个集合，c 是从 F 到实数集合的一个映射，称为代价函数。问题要求找出一个$f \in F$，使得对于所有$f \in F$，都有$c(f) \leq c(y)$。这样的 f 称为全局最优解，或者简称最优解。

定义 1.1-2 （最优化问题） 一个最优化问题包含该问题的所有实例。

可以这么说，一个实例是给定了输入数据的，而一个问题则是实例的集合。所以当我们在面对生活中一个具体问题的时候，我们一般是面对一个问题的实例。

最优化问题可以自然地分成两类：一类是连续变量的问题，另一类是离散变量的问题。具有离散变量的问题，我们称它为组合的。在连续变量的问题里，一般地是求一组实数，或者一个函数；在组合问题里，是从一个无限集或者可数无限集里寻找一个对象——典型地是一个整数，一个集合，一个排列，或者一个图。一般地，这两类问题有相当不同的特色，并且求解它们的方法也是很不同的。对于具有离散变量的问题，从有限个解中寻找最优解的问题就是组合优化问题。

定义 1.1-3 （组合优化问题）组合优化问题研究从一个包含有限对象的集合中，找出一个最优化的对象。

例 1. 最基础的旅行商问题（Traveling Salesman Problem，TSP 问题）是指给一个带权无向图 G=（V，E），在 G 种找出一条权值最小的 Hamilton 路。Hamilton 回路是从图中某个点出发经过图中所有节点并回到出发点的路，限制条件为图上所有的顶点必须且最多经过一次。

TSP 问题是一个典型的组合优化问题，图 1.1-1 表示了旅行商问题的输入和输出。其输入是一个带权的无向图，其中的任意一个 Hamilton 回路均是该问题的一个研究对象，并且能保证有有限个数的研究对象，最优化的对象是其中边权值最小的那一个对象。其输出就是用某种方式表述的最优对象。

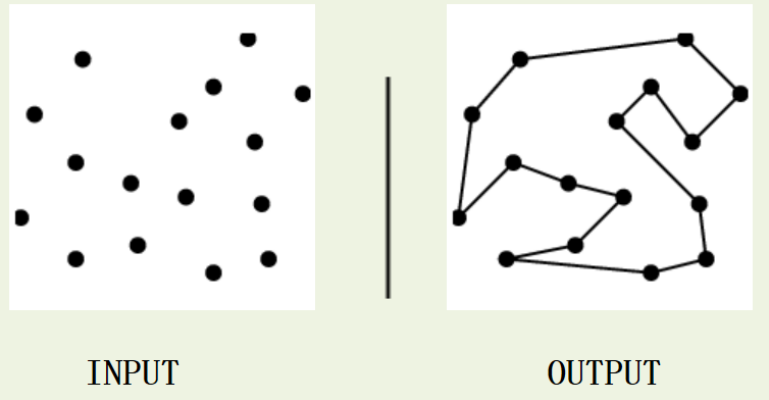

图 1.1-1 旅行商问题

TSP 问题又译为旅行推销员问题、货郎担问题，是数学领域中著名问题之一。它的应用场景为，一个商人想要去 n 个城市推销商品，如何走最短的路去周游各个城市，最后回到出发点。除此之外，旅行商问题在现实生活中的货物装配、路径规划、订单派送等问题上又很广泛的应用。

例 2. 最大团问题（Maximum Clique problem）也是一个经典的组合优化问题。若图 G=（V，E）的一个顶点集合 C 中任意顶点与 C 中其他顶点都相邻，则称 C 是 G 的一个团。最大团问题即找 G 中点数最多的团 C，使得 G 中任意 C’，均有|C|≥|C’|。

最大团问题中，图 G 的任意一个团为该问题的一个研究对象，由于 G 中的顶点集合是有限的，因此最大团问题的研究对象是有限的，最优化的对象是点数最多的团。最大团问题可以应用在社交网络，蛋白质结构分析等。图 1.1-2 表示了一个社交网中团的一个应用，若一家广告公司想要找一某博主打广告，那么找几个极大团的交点投放往往可以在节省成本的同时覆盖到更多的人。

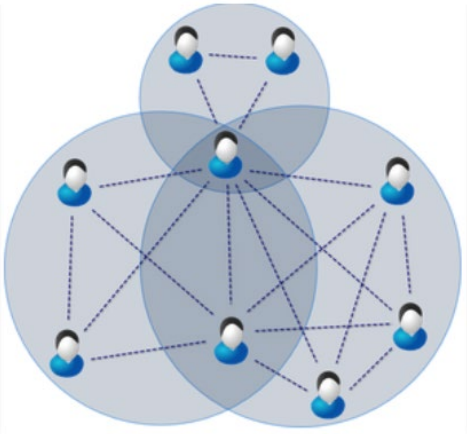

图 1.1-2 社交网

例 3. 顶点覆盖问题(Vertex Cover Problem）是在图 G=（V，E）中找一个顶点的集合 L 使得图中任意一边 e=uv 中至少一个顶点在 L 中。图 1.1-3 中左右两图均为一个顶点覆盖。最小顶点覆盖问题是在 G 中找一顶点集合 L 使得|L|最小。图 1.1-3 中左图和右图中标红点对应的集合皆为一个最小顶点覆盖。

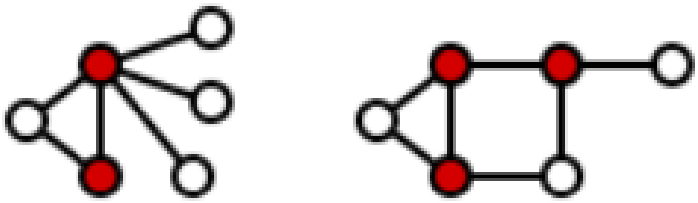

图 1.1-3 顶点覆盖问题

最小顶点覆盖问题也是一个经典的组合优化问题，图 G 的任何一个顶点覆盖 L 均为组合优化问题的研究对象，是有限的，最优化对象为顶点数最少的 L。

最小顶点覆盖问题也有很广泛的应用，例如道路监控，网络连接失败的监控分析等。其中连接失败的网络监控问题可以通过计算最小顶点覆盖集合，来判断如果摆放最少的网络监控器来监控网络中所有边的连接状况。

例 4.命题逻辑的可满足性问题（Propositional Satisfiability，SAT）是经典的组合判定问题，给定一个命题逻辑公式，SAT 问题要求判断是否存在一组变量的指派，使得公式为真。变量的任何一组指派都是该问题的一个对象，因为变量个数有限所以对象也是有限的，使得命题逻辑得到满足的一组变量指派则是问题的一个最优化对象。

SAT 问题在电路设计、时刻表的设计等实际问题上有很广泛的应用，有关 SAT 问题的详细内容将在后面的章节进行介绍。

通过以上 4 个比较经典的例子，有助于帮助我们理解什么是组合优化问题。除此之外，组合优化问题还有顶点着色问题、路径规划问题、时间表排布、调度问题、资源分配等组合优化问题。但是一般组合优化问题都是复杂的、难求解的，下一节中讲详细探讨这个问题。

1.2 NP 难解问题

为了刻画一个问题求解的难度，可以用求解该问题算法的时间复杂度和空间复杂度来描述。时间复杂度是用来描述一个算法运行时间的量，一般一个算法的求解时间有跟数据的大小有关，常见的时间复杂度可以分为阶乘级、指数级、对数级、多项式时间、线性等，而若与数据的大小无关的话则为常熟级别的时间复杂度。自然地，人们会想到一个问题：会不会所有的问题都可以找到复杂度为多项式级的算法呢？很遗憾，答案是否定的。有些问题甚至根本不可能找到一个正确的算法，这称之为“不可判定问题”(Undecidable Decision Problem)。人们为了刻画复杂问题的时间复杂度，引入了 NP 问题和 P 问题的概念。

定义 1.2-1 （时间复杂度） 设 A 是求解问题Π的算法，在用 A 求解Π的实例 I 时，首先要把 I 编码为二进制串作为 A 的输入，称 I 的二进制编码长度为 I 的规模。如果存在函数 f: N → N使得，对任意规模为 n 的实例 I，A 对 I 的运算在 f(n)步内停止，则称算法 A 的时间复杂度为 f(n)。

在以上定义中，问题的规模是以二进制编码的长度定义的。然而，在实际讨论中，我们往往会使用计算对象的某些自然的参数，比如图的顶点数，问题的变量个数等。这么做是安全的，因为这些实例的二进制编码的长度与这些自然参数都是多项式相关的，从而也可以直接用这些参数作为实例的规模。

关于以上定义中的计算步，也即操作指令，有两点需要说明。其一，执行不同指令所用的时间是不同的，但在这里把执行任何一条指令作为一步。这就要求操作指令集合中的每一条指令都是“合理的”指令，也就是每一条指令的执行时间是固定的常数。其二，算法的执行步数和所采用的指令集有关。但实际上对于任何两个合理的操作指令集，其中一个指令集的每一条指令都可以用另一个指令集的指令模拟，且模拟所用的指令条数不超过某个固定的常数，从而同一个算法在任何两个合理的操作指令集上的运算步数至多相差常数倍。

接下来我们会从时间复杂度来讨论问题的不同难易性。由于技术上的原因，在定义复杂性类时限制在判定问题上。判定问题对应一个语言，所有“yes”的实例构成了这个语言。而一个复杂性类实际上是一个语言集合，该集合中的语言有着相同的属性，就某一性能而言（时间，空间），它们都能够在特定范围内被判定。

定义 1.2-2（判定问题）判定问题是指答案只有两种---“Yes”和“No”---的问题。

以多项式为时间复杂度的算法称为多项式时间算法。一般认为，可以在多项式时间求解的问题是易解的，这类问题叫做 P-问题，也即多项式问题。复杂度类 P 的严格定义如下。

定义 1.2-3（P-类）所有多项式时间可解的判定问题组成的问题类称作 P 类。

为了引出 NP-问题，我们定义了非确定机和非确定性算法。非确定性机相对于确定性图灵机，确定性图灵机对于某个状态会执行特定的指令，而非确定性性则会有多条可以执行的

指令。非确定图灵机是一个不真实的计算模型，它可以被其他模型在指数时间损失下模拟（Christos H. Papadimitriou, 1994）。非确定性算法将问题分解成猜测和验证两个阶段。算法的猜测阶段是非确定性的，算法的验证阶段是确定性的，它在验证猜测阶段给出解的正确性。

定义 1.2-4（NP-类）由所有非确定图灵机多项式时间可解的判定问题组成的问题类称为 NP 类。

我们可以理解 NP 问题为可以在多项式时间内判定给定的解是否为问题答案的一类问题。显然所有的 P 问题都是 NP 问题，那么反之是否有 NP 问题都是 P 问题呢?即 P 是否等于 NP。目前这个问题没有得到一个肯定的解，也是“千禧难题”之首。

为了方便讨论，我们需要使一个问题至少与另一个问题一样难的含义称为精确的概念，这就是规约的概念。**一个问题 A 至少和问题 B 一样难，如果 B 规约到 A。**

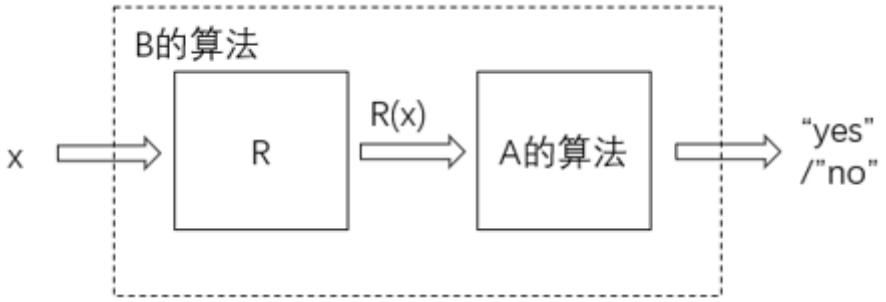

图 1.2-1 规约

什么叫规约呢？我们说 B 规约到 A，如果存在一个转换 R，它对每个 B 的输入 x，产生一个 A 的等价输入 R(x)。这里“等价”指的是将 R(x)作为 A 的输入，得到的回答”yes”或”no”，也是 B 的输入 x 的正确回答。下面给出规约的定义，我们将采用多项式时间规约作为“有效规约”的概念。

定义 1.2-5（多项式时间归约）我们说 L1 规约到 L2，如果有一个从串到串的确定性图灵机在多项式时间可计算的函数 R，它对于所有输入 x，$x \in L1$当且仅当$R(x) \in L2$。

有了规约的概念，我们就可以比较不同问题的难度了。特别的，在 NP 类中，有一类问题是最难的，称为 NP 完全问题，其严格定义如下。

定义 1.2-6（NP-complete 问题，NPC 问题）如果任何一个 NP 问题都可以在多项式的时间内归约到该问题，那么则称其为 NPC 问题。

因为 NP 完全的概念是针对判定问题而言的。对于组合优化问题，我们一般采用另一个更一般的概念，即 NP 难问题。

定义 1.2-7（NP-hard 问题）求解难度至少和 NPC 问题一样困难的问题。

由以上的定义，显然可以得到这些问题之间的关系如图 1.2-1 所示，两侧图分别表示当 P 不等于和等于 NP 的不同情况下 P、NP、NPC、NP-hard 四类问题的包含和被包含关系。此外，由以上的定义可知，如果一个 NPC 问题可以被证明可以找到一个多项式可求解的确定性算法，那么则可以证明 P=NP，反之，如果证明一个 NPC 问题不存在一个多项式可求

解的确定性算法，则可以证明 P≠NP。

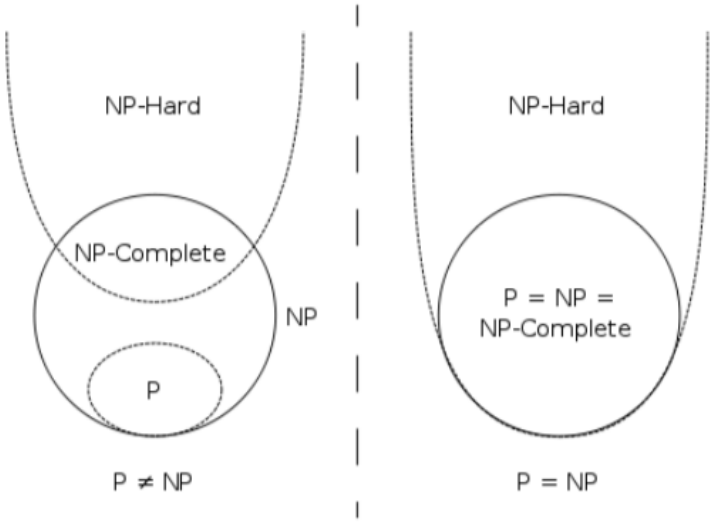

图 1.2-2 P、NP、NPC、NP-hard 关系图

显然，所有 NP 完全可以在多项式时间相互转换。如图 1.2-2 所示，它给出了三个 NPC 问题直接的转换关系，它们分别是顶点覆盖问题，独立集和团，红色的点为一个满足条件的集合。前面我们已经介绍了顶点覆盖和团的定义，而独立集是一个顶点集合，该集合上任意两点都不相邻。这三个问题的判定版本可以在线性时间内相关转化。

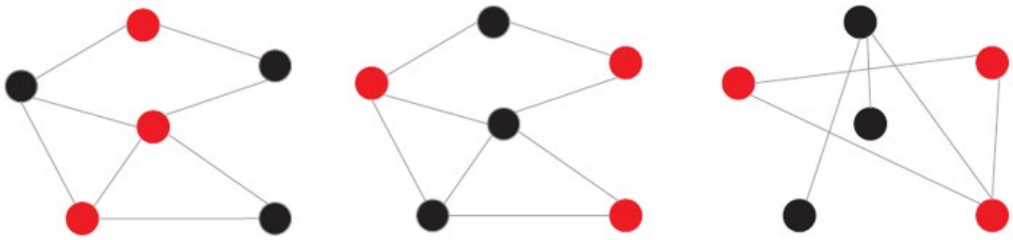

图 1.2-3 三个 NPC 问题顶点覆盖、独立集和团的关系

SAT 问题是第一个被证明的 NPC 问题，是计算机领域中的一颗掌上明珠。为了精确的描述 SAT 问题，我们将精确的定义变量、文字、子句、CNF 范式的概念。

定义 10（SAT 问题）给定一组可以被赋值为真（true）或者假（false）的 n 个变量（variable）$X=\{x_1, x_2, \cdots, x_n\}$，定义文字（literal）为变量或者变量的非，如 $l_1 = x_1, l_2 = \neg x_3$。文字的析取组成子句（clause），如 $c_1 = x_1 \vee x_3 \vee \neg x_5$。子句的合取组成合取范式（CNF），如 $F = c_1 \wedge c_2 \wedge c_3$。SAT 问题就是在给定一个 CNF 公式之后，判断是否存在一组变量的赋值，使得 CNF 公式为真，如果为真则返回“SAT”和一组指派，如果不存在则返回“UNSAT”。

SAT 问题是一个概念上易懂，描述简单的问题，应用极为广泛，可以被应用在任何可以转化到命题逻辑进行求解的科研理论和实际问题上，如硬件验证、模型检测等。大部分组合优化问题都可以在多项式时间内转化到 SAT 问题。可满足性问题是一个被计算机、数学方面的学者和工业界广泛关注和研究的问题，并且有很多相关的研究理论。如图 1.2-3 为《The Art of Computer Programming》系列的一卷书就专门介绍了可满足性问题的算法。

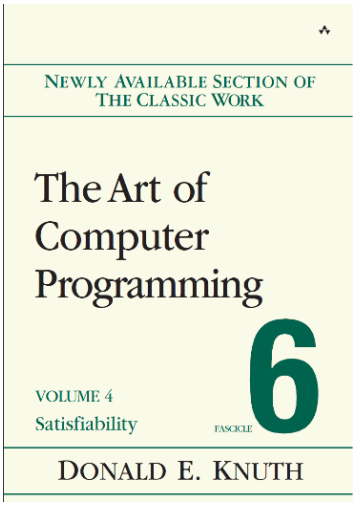

图 1.2-4 Knuth 写的可满足性问题专著

绝大多数组合优化问题都是难求解的，这是由于该类问题的最坏时间复杂度决定的。当我们面对这样的问题时，一般有以下一些途径来进行求解，分别对应不同的研究方向。

1）某些该类问题的子集有可能很简单就可以被求解。

2）采用近似算法（Approximation algorithms）可以多项式时间求得近似解，保证近似比。

3）采用随机\概率算法（Randomized \probabilistic algorithms）进行求解，以一定的概率可以求得问题的最优解。

4）采用参数化算法（Parameterized algorithms），对问题中的某些参数看为常数，也即固定参数，由此得到多项式的时间函数。

5) 采用启发式算法(Heuristic algorithms)，这类算法没有理论保证，但是经验上往往可以高效求得很好的解甚至最优解，在工程上有广泛应用。

1.3 常用模型与通用求解器

对于组合优化问题，设计一个高效算法常常是一件不容易的任务。幸运的是，我们还可以通过把这些问题转换为一些目前已经有成熟求解器的问题，从而可以用较低的开发成本使得问题得到解决。本节就介绍常见的用来建模组合优化问题的模型，每个模型是一个经典问题，而这些问题都有了成熟的求解算法。

（1）组合优化问题常用模型

可满足性问题(Satisfiability, SAT)与最大可满足性问题（MaxSAT）

从上一节的知识，我们了解到 SAT 问题是一个 NPC 问题，并且很多 NP 难的组合优化问题都有很简单的规约到 SAT 的算法， 如果有一个高效的求解器去求解该问题，那么由 SAT 问题表述的问题便可以高效的求解。其中，用 SAT 问题求解另一问题的工作，一般称作 SAT 编码问题。一般编码问题要求在多项式时间内解决，例 5 给出了由 k-图着色的判定问题到 SAT 问题的一种编码方式。

例 1.3-1. k-图着色问题（k-Coloring Problem）是判定一个无向图 G=（V，E）是否可以由 k 种不同的颜色对 V 中所有的顶点着色，其中要求 E 中任意边上两个顶点均着不同的颜色。

1) 首先，我们定义变量 $x_{i,j}$ 表示对 G 中的第 i 个顶点着第 j 种颜色。总共由 k|V|个变量。

2) 为了保证图中每个顶点都着色，我们定义子句,

$$c_i = \bigvee_{1 \le j \le k} x_{i,j}$$

这样的子句总共有|V|条。

3) 为了保证图中任意两条边都着不同的颜色，我们定义如下的子句，

$$c'_{i,j,k} = \neg x_{i,k} \vee \neg x_{j,k}$$

其中 i，j 表示 E 中的边，k 表示着某种色。该类子句总共加起来有 k|E|条。

这样就形成了一个 k-图着色问题可行的 SAT 编码。

上例中给出了一种简单、基础的编码方法，并不是最好的编码方法，此外也有一些对数编码变量的方法，打破对称性的研究等高效的方案。但是通过这个例子，我们能得到一个编码 SAT 问题的思路。

1）考虑问题可以转化到的约束变量个数和如何定义。

2）考虑问题本身的特性可以用什么子句表述。

3）考虑问题需要满足的约束和限制条件。

4）考虑问题种的对称性。

用 SAT 编码的方法求解一个问题，都需要考虑编码转化的时间复杂度、编码规模和 SAT 求解器求解该类 CNF 的难度。任何一个 SAT 编码方法，都可以在上面的四个角度来考虑如何降低空间复杂度和时间复杂度。除此之外，还可以针对该类问题调试参数设计该类 CNF 公式的专项高效求解器。

SAT 求解器有很多，类如 MiniSAT、Glucose、Maple、ReasonLS 等求解器，其中 MiniSAT 是最主流的完备算法求解器，可以从其主页获得。每年学者们都会组织一场 SAT 的求解比赛，涉及到 SAT 的完备算法和不完备算法，所有参与比赛的求解器均可以从 SAT 比赛的官网下载。现在的 SAT 求解器的求解效率已经有了很高的提升，在解决绝大多数实例上都有不错的表现。

既然 SAT 求解器求解效率那么高，而且又有这么好的表述能力，那为什么不把所有的

问题编码到 SAT 问题进行求解呢？因为对于大部分问题来说，专用求解算法的求解器要效率更高。类如图问题中的最短路问题、最小生成树问题，优化问题中的最大流问题、线性优化问题，他们自身设计的求解器效率就已经很高，没有必要通过多项式时间转化到 SAT 问题进行求解。

但是我们不能因噎废食，虽然因为 SAT 问题本身的难求解性、SAT 求解器不能在所有问题上都表现突出，但是由于 SAT 问题在案例分析上有它的独当一面的地位，因此我们不能完全摆脱 SAT 求解器。

SAT 问题有很多变形问题，其中比较出名的问题有 MaxSAT 和 Partial MaxSAT。

定义 1.3-1（MaxSAT）给定一个 CNF 公式，找到一个完备赋值，满足给定的 CNF 公式 F 中最多的子句。

显然，当 F 中所有的子句都可以被满足的时候，MaxSAT 问题就退化到了 SAT 问题。

例 1.3-2. 在图 1.3-1 中，给出了一组子句 F，如果我们给其中涉及的五个变量均赋值为 false，那么其中只有一条子句不能被满足。图中被满足的变量我们标为绿色，不能被满足的子句标红。这个时候我们找到了该 MaxSAT 问题的一个最优解。

$$
\begin{aligned}
F := \quad & (\neg x_1) \\
& \wedge (\neg x_2 \vee x_1) \\
& \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\
& \wedge (x_1 \vee x_2) \\
& \wedge (\neg x_4 \vee x_3) \\
& \wedge (\neg x_5 \vee x_3)
\end{aligned}
$$

图 1.3-1 MaxSat 问题

定义 1.3-2（partial MaxSAT）假如将子句分为硬字句和软子句两部分，partial MaxSAT 问题需要尽可能的满足更多的软子句并满足所有的硬字句。

SAT 问题的变形问题此外还有一些带权重的问题，在这里就不一一介绍。这些问题都有很强的表述能力，例 7 讲述了最大团问题如何转化到 partial MaxSAT 问题。

例 1.3-3. 在一个图 G=（V，E）中，如果找其中的最大团，可以按照如下的策略将其编码成 partial MaxSAT 问题。

1）对于每一个 G 中的点定义一个变量 x_i，表示对应的顶点是否在团中。总共定义了有|V|个变量。

2）对于图中任意一对不相邻的顶点 i，j，定义一条硬子句，

$$c_{i,j} = \neg x_i \vee \neg x_j$$

表示任何两个不相邻的点不在同一个团中。

3）对于图中的所有顶点，定一一个单元子句作为软子句，

$$c_i' = x_i$$

用来表示，一个团中应该有尽可能多的顶点。

这时候用一个通用 partial MaxSAT 求解器进行求解，如果得到一个最优解，那么就找到了一个最大的团，并且取值为 1 的变量表示所求得的团。

定义 1.3-3（weighted partial MaxSAT）如果 partial MaxSAT 问题的每一条软子句都有一个权值，求解的目标相应的转化为满足所有的硬字句并且使满足的软子句的权值和尽可能的大，该问题即为一个 weighted partial MaxSAT 问题。

显然，partial MaxSAT 是特殊的一个 weighted partial MaxSAT 问题，即所有软子句权值相等的 weighted partial MaxSAT 问题。而 SAT 和 MaxSAT 分别是 partial MaxSAT 问题的特特例，他们分别表示软字句个数为 0 和硬字句数目为 0 的 partial MaxSAT 问题。除此之外，SAT 问题也可以用来求解其衍生的变形问题，但是表述问题要更为复杂，并且求解效率低于专用求解器。

经过以上的描述和例子，不难发现 SAT 问题和其变形问题都可以作为一个通用的求解器。除了 SAT 问题之外，还有一些其它问题的求解器可以作为通用求解器，一般这类问题都要求有很强的描述能力。SAT 的表述能力限制于命题逻辑，另外，还有表述能力更强的 CSP 和 SMT 问题和相应的求解器，分别可以用来求解约束满足问题和一阶逻辑相关的问题。

约束满足问题(CSP)

定义 1.3-4（Constraint Satisfaction Problem，CSP）约束满足问题定义为一组对象，而这些对象需要满足一定的条件约束，CSP 问题一边表述为三元对（X, D, C），包含一个变量集合 $X=\{X_1, X_2, \cdots, X_n\}$、变量的值域 $D=\{D_1, D_2, \cdots, D_n\}$和一个约束集合 $C=\{C_1, C_2, \cdots, C_n\}$。其中每个变量 X_i 可以在非空的定义域 D_i 中取出。每个限制条件 $C_j \in C$ 依序对应一对$<t_j, R_j>$,其中 $t_j \subset X$ 是 n-tuple 的变量，R_j 则是在定义域 Di 中对应到子集合上得到的 n-ary 维的关系。变量有对应的评估函数 f，它是由变量到值域的映射。如果 $f(x_1), \cdots, f(x_n) \in R_j$，那么 f 满足$<(t_1, \cdots, t_n), R_j>$的条件限制。对于 CSP 问题，我们称赋值是无矛盾的当且仅当一个评估不违反任何的条件限制。称赋值是完备的当且仅当一个评估包含了所有变量。如果一个赋值是无矛盾且完备的，那么这个评估就是 CSP 问题的一个解。

CSP 问题有不错的表述能力，在数据库检索、集成电路涉及、计算机视觉、机器学习、计算机视觉、规划等领域都有很好的应用，因为它在表述一个问题的时候十分自然，因此用 CSP 求解器来求解问题的时候要比直接利用搜索等方法求解要更简单、清晰。例 8 给出了用 CSP 问题表述 4 皇后问题的一种描述方法。

例 1.3-4. 我们尝试用 CSP 问题来描述 4 皇后问题。4 皇后问题为判定是否可以将皇后互无冲突的放在在 4*4 的一个棋盘上。 无冲突指任何两个皇后不能在同一条对角线或者行列上。图 1. 3-2 表示的是一个可行的皇后安排情况。

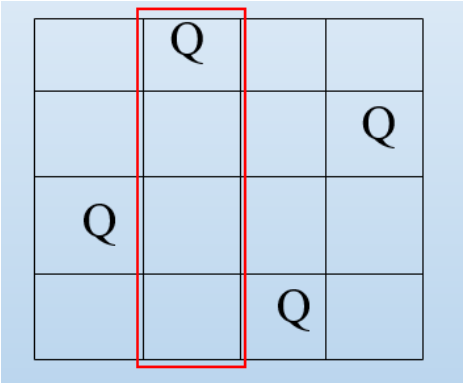

图 1.3-2

为了描述 CSP 问题，我们可以采用类似 SAT 问题的过程，首先确定变量个数和变量的取值范围，再确定变量之间的约束关系。

1）变量集合 X 中有四个变量，x_1, x_2, x_3, x_4，其中 x_i 用来表示第 i 列的皇后所在的行。

2）参考图 1.3-2 中标出的一列 x_2，由格子自身的特性，易知变量 x_i 的取值范围为 $D_i=\{1,2,3,4\}$，组成了值域集合 D。

3）由问题中任意两个皇后之不在同一行列和同一对角线上。得到约束条件如下所示。

$$x_i \neq x_j, (0<i<j\leq n) ;$$

$$|x_i - x_j| \neq |i - j|, (0<i<j\leq n)$$

其中每一条约束都可以写成二元对的形式，例如 x1 和 x2 的约束可以表示为下面的标准形式，

$$<(x1,x2),\{(1,3),(1,4),(2,4),(3,1),(4,1)\}>$$

如果以上的 4 皇后问题转为 SAT 问题，则需要 16 个变量才能表达。可以看到，CSP 可以比 SAT 更加紧凑地表达多取值的约束问题。

目前商业 CP 求解器中做的最好的是 IBM 公司的 ILOG CP-solver 它可以和 CPLEX 对接使用 主要应用在港口、机场等领域。开源的 CP 求解器学术界主要使用 gecode， 由 C++ 编写，效率很高，并且实现了主要的全局约束功能。chocoo 是一款使用 java 语言编写的 CP 求解器 代码可读性和可扩展性比较好， 但效率差了一些。

多理论下的可满足性问题（SMT）

除了约束满足问题之外，线性规划问题和可满足性模理论（Satisfiability Modulo Theories，SMT）的求解器也常作为一个通用求解器。本文接下来将介绍一种表述能力比 SAT 更强的模

型——SMT。

定义 1.3-5（SMT）是判定一阶逻辑公式在组合背景理论下的可满足性问题。

SMT 问题主要目标是解决一阶逻辑问题，一阶逻辑可以强力的描述现实世界中的绝大多数问题，求解能力要强于命题逻辑，即 SAT 问题。但是由于一阶逻辑的复杂性，给定的一个一阶逻辑问题通常是不可判定的。并且由于背景理论的引入，SMT 有比 SAT 更加灵活的描述能力，可以更加方便的表示一些学术和工业上的问题。

SMT 的背景理论使其能很好地描述实际领域中的各种问题,结合高效的可满足性判定算法,SMT 在测试用例自动生成、程序缺陷检测、RTL(register transfer level)验证、程序分析与验证、线性逻辑约束公式优化问题求解等一些最新研究领域中有着突出的优势。

一般 SMT 求解器能够处理的理论主要分为一些数学理论、数据结构理论和未解释函数。并可以继续细分为如下的几点。

1）未解释函数（Uninterpreted Function，UF）主要包括一些没有经过解释的函数符号和它们的参数。如果带有符号则称其为 EUF。例 9 表示了一个 EUF。

例 1.3-5. $\mathrm{a} = \mathrm{b} \ \wedge \ \mathrm{b} = \mathrm{f(c)} \wedge \neg(\mathrm{g(a)} = \mathrm{g(f(c))})$ 表示了一个由带符号的未解释函数组成的公式。

2）线性实数演算（liner real arithmetic，LRA）和线性整数演算（linear integer arithmetic，LIA）可以表示为$a_1x_1 + \cdots + a_nx_n \bowtie c$，其中⋈可以表示=、≠、≤、≥等符号，c 是一个常数，a_1，…，a_n分别在 LRA 和 LIA 中表示实数和整数。

3）非线性实数演算（non-liner real arithmetic，NRA）和非线性整数演算（non-linear integer arithmetic，NIA）的公式则可以表示任意的数学表达式。

4）实数差分逻辑（difference logic over the reals, RDL）和整数差分逻辑（difference logic over the integers, IDL）一般可以表示成$\mathrm{x} - \mathrm{y} \bowtie c$的形式，其中⋈和 c 的含义与 2）中一致，x，y 分别在 RDL 和 IDL 中表示实数和整数。

例 1.3-6. $\mathrm{x} - 2\mathrm{y} \geq 5 \ \wedge \ \neg(\mathrm{x} - \mathrm{z} < 5) \ \wedge \neg(x^2 = \mathrm{y})$ 表示的是由 2）、3）、4）组成的一组公式。易得这个例子是可满足的。

5）数组（arrays）和位向量（bit vector，BV）。这两个背景理论则用于处理计算机中的数据结构，用于处理数组和位向量的操作。

例 1.3-7. $\mathrm{w}[31:0] \gg 16! = 016:\mathrm{w}[31:16]$ 表示的是一个 BV 的公式，表示的是一个 32 位的位向量向右移动 16 位之后和其高 16 位不同，显然这个公式是错的，不可满足。

采用 SMT 可以用来求解很多实际问题，类如验证一阶逻辑的可靠性（validity），如图 1.3-3

所示，为了验证上面公式推导的可靠性，可以通过 SMT 来验证图下公式的不可满足性来检验。

Prove that

$$x \geq 1, y \geq 1 \Rightarrow x + y \geq 2$$

Is

$$x \geq 1, y \geq 1, not(x + y \geq 2)$$

UNSAT?

图 1.3-3 SMT 用来验证 validity

一般 SMT 的求解器分为 SAT 求解器和理论求解器两部分。对于目前主流的 SAT 求解器和理论求解器来说，他们都是高效的。分别来说，SAT 求解器是连续的（Incremental），可以支持向其中不断的增加子句而不改变更改后问题的可满足性；理论求解器有构建生成子句块的能力，并且可以生成理论的引理（theory lemma）。SMT-LIB（www.smtlib.org）是一个开源的 SMT 问题的库，定义了 SMT 的语言和涉及的背景理论，SMT 的语言是由类似于 Lisp 语言的符号组成的。

线性规划与整数线性规划

除了以上介绍的内容之外，还有一种通用模型叫做线性规划（linear programming，LP），它是运筹学中研究较早、发展比较快且应用很广的一类算法。它的一般形式表示为：

$$\min c'x$$

$$\text{s. t. } \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

其中 c 为一个权向量，x 表示的是变量组成的向量。A 表示的是一个矩阵，b 是一个列向量。线性规划的目标是求得满足所有约束条件下，目标最小的一组变量赋值。

一般常见的线性函数为 $a'x \infty b$ 的形式， 其中∞ 可以为=、≥、≤。

如果是 $a'x = b$ 的情况，那么即为标准形式。

如果是 $a'x \geq b$ 的情况，可以通过添加一个大于零的变量 x_s，变为标准形式 $a'x - x_s = b$。

如果是 $a'x \leq b$的情况的话，那么同理也可以通过添加变量的方法，变为标准形式 $a'x + x_s = b$。

如果对于变量的约束为$x_i \leq 0$的话，那么则可以通过等式两边同时乘以负号的形式引入变量 $y_i = -x_i$，来替换约束中的所有变量。

如果对于变量x_i没有约束的话，那么则可以引入两个大于零的新变量z_1, z_2，使$x_i = z_1 - z_2$,并且替换约束中的所有变量。

如果目标函数为 max 的话，则可以通过添加一个负号的形式更改为标准形式。

例 1.3-8. 假如商人想制作售卖价格为 100 和 200 的两个不同的商品 A，B，且单位的商品 A、B 分别需要原料 m 30 和 40 单位，需要原料 n 40 和 60 单位。如果现在仓库里有 m，n 的原料分别为 500 和 700 单位，那么如何分配原料生产能够使得总商品的价值最大。

根据题意,假如商品是食品，是可以称重售卖的，则有，

$$\max 100x_1 + 200x_2$$

$$\text{s. t. } \begin{cases} 30x_1 + 40x_2 \leq 500 \\ 40x_1 + 60x_2 \leq 700 \\ x_1, x_2 \geq 0 \end{cases}$$

化为标准形式，则有，

$$\min -(100x_1 + 200x_2)$$

$$\text{s. t. } \begin{cases} 30x_1 + 40x_2 + c_1 = 500 \\ 40x_1 + 60x_2 + c_2 = 700 \\ x_1, x_2, c_1, c_2 \geq 0 \end{cases}$$

化为标准形式之后，则可以采用通用的求解器进行求解。其中主要的方法有单纯形法、对偶单纯形法等。

但是一般来说，真实环境中不一定有对于每一个种类的个数都可以取做任意数字，于是在上述例子的情况下又有了一些限制，比如例 12 中的商品变化为椅子和桌子，那么则要求生产产品的种类必须为整数，这就出现了整数线性规划（Integer Linear Programming）的类型，我们在正常的生产和生活中，也可以通过将这类问题转化到整数线性规划的标准形式后，调用专用求解器进行求解。

此外，有时候对于问题的约束条件和目标函数不仅仅是线性的，也就相应的有了非线性规划（non-Linear Programming）的问题和整数规划（Integer Programming）的问题和相应的求解器。生活和科研中，人们经常会遇到可以用整数规划或者线性规划等模型进行求解的问题。它们是一种描述能力很强的模型。

（2）SAT 求解技术简介

为了求解 SAT 和 SMT 理论，人们设计了一些算法。对于 SAT 问题，主要分为完备性（complete）算法和不完备（incomplete）算法。完备算法对于给定的一组 CNF 公式一定能够证明其可满足性，一般采用了回溯加推理的方法。不完备算法则仅仅只可证明一组公式是可满足的，而不能证明一个公式是不可满足的，一般采用局部搜索的方法。

对于一个完备 SAT 算法来说，它的基本规则是单元子句传播（unit propogation），其中单元子句指的是仅仅包含一个文字的子句，则为了保证子句是可满足的，组成该文字的那个变量必有唯一确定的赋值，进而可以产生更多的单元子句，不断进行这个过程直到不能进行单元子句传播或者推出矛盾为止。

定义 1.3-6（Resolution rule，归结原理）如果一个 CNF 公式 F 包含两个子句$x \vee \alpha$和$\neg x \vee \beta$，

那么公式 F 可以推出α∨β，其中α，β均表示一组文字的析取。并且归结原理是一个充分（sound）且完备（complete）的规则。不难发现归结原理前后推导的子句之间是可以证明是正确的，即可以用于求解系统的证明。

例 1.3-9. 如图 1.3-4，给出一个 SAT 问题的 CNF 公式，我们来对这个公式不断的做归结原理，我们发现该问题的规模变小到了一个单元子句，显然该 CNF 公式是可满足的。

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \vdash$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \vdash$$

$$\cancel{(x_3 \vee \neg x_3)} \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \vdash$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \vdash$$

$$(x_3)$$

图 1.3-4 归结原理

完备的 SAT 求解算法的主要框架是回溯策略，其中比较有代表性的也是最基础的算法叫做 DPLL 算法，它是由四个人（Davis and Putnam 1960, Davis, Logemann and Loveland 1962）提出并以名字命名的。它采用了标准的回溯策略，详细的算法步骤如下表述为按部执行的算法，算法的每一个 Step 如下。

1）如果一个公式已经是可满足的，返回 SAT。

2）选择一个决策变量。

3）应用单元子句传播技术。

4）如果传播的过程中遇到了冲突，回溯，如果不能回溯则返回 UNSAT。

5）跳转到下一个 Step，继续从 1）开始执行。

DPLL 是一个简单的框架，但是他效率并不高，现在主流的求完备算法解器，如 MiniSat、Glucose 等均采用的是冲突驱动的字句学习（Conflict-Driven Clause Learning，CDCL）技术作为主要框架，其中涉及了冲突子句分析、子句学习、冲突子句最小化技术、子句库化简、快速重启等先进的求解技术。

对于 CDCL 框架，在回溯搜索的过程中，对于每一次单元子句传播过程中遇到的冲突，求解器都会从中学习一条新的子句辅助于接下来的搜索，它能够解释和预防遇到同样的冲突，从而实现跨层级的回退。下面将采用一个简单的例子来介绍 CDCL 技术的先进之处。

例 1.3-10. 加入给定的 CNF 公式如图 1.3-5 所示，并且当前做出的决策为 c=false 和 f=false，

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

图 1.3-5 CNF 公式和当前赋值

如果做出变量 a=0 的决策，那么通过使用单元子句传播技术，会推出矛盾，如图 1.3-6 所示。

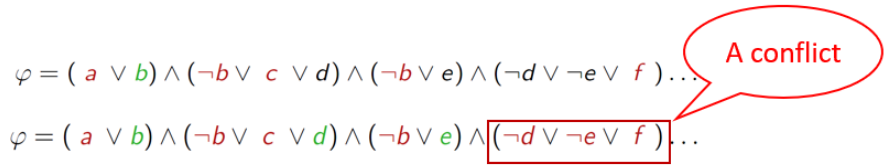

图 1.3-6 通过单元传播进行推理

如果 1.3-7，根据冲突，我们发现这个赋值是不合理的，因此我们通过一个新的子句来表示 a, c, f 三个变量不能按这个方式进行赋值，这个句子在今后的求解过程中会有一定的启发性，称作学习子句。

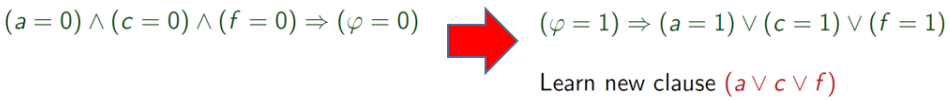

图 1.3-7 子句学习

根据产生方法，SAT 实例一般可以分为三种：

1） 随机组实例。专用于不完备算法设计的实例。例如 k-SAT 随机模型

2） 人工组实例。通过其他问题编码转化而来，不是直接来源于真实的世界。例如 Sudoku、Hidoku

3）应用组实例。由现实世界的真实问题直接编码得到，一般变量和子句的数目都比较大，比较稀疏，且有很多特别短的子句。

求解 SMT 的算法大致可以分为积极（eager）的算法和惰性（lazy）算法。

积极算法是将 SMT 公式转化为可满足性问题等价的 SAT 公式，然后求解该 SAT 公式。是早期的 SMT 求解器采用的算法，可以利用高效的 SAT 求解器，但是本身的效率极其以来 SAT 求解器的效率。

惰性算法是结合了 SAT 求解和理论求解，先得出一个对命题变量的赋值，然后再判断该赋值是否在理论上一致。是目前主流的方法，它先将 SMT 公式看作一个 SAT 公式求解，然后用理论求解器判定 SAT 公式的解表示的理论公式是否一致。它把 SAT 求解器的部分当作了一个黑盒。在求解的过程中可以不断的学习理论的引理。

SMT 有很广泛的应用，图 1.3-8 展示了部分 SMT 问题的应用场景。

图 1.3-8　SMT 在微软的应用

针对 SMT 问题的高效求解器有很多，其中比较出名的有微软开发的 Z3（http://research.microsoft.com/projects/z3），还有 Yices（http://yices.csl.sri.com），CVC 等，他们都可以在 Windows、OSX 和 linux 多平台上使用。

本章小结：

本章第一节首先介绍组合优化问题的定义，并给出经典的组合优化问题为例子；第二节介绍了时间复杂度的概念，以及 NP 完全和 NP 难的概念，许多组合优化问题都是 NP 难的，此课程主要也是讲授 NP 组合优化问题的求解算法；在第三节，我们介绍了组合优化问题的建模，常见的模型问题包括 SAT，SMT，CSP，整数规划，并且概述了 SAT 和 SMT 的主要求解方法。

对于本章内容，希望达到的目标是：能判断一个问题是否组合优化问题，懂得 NP 完全和 NP 难的概念，知道如何证明一个新的问题是否 NP 难，了解有本章介绍的通用组合优化问题模型，并对一些简单的问题掌握建模方法，了解 SAT 算法的主要技术。

本章参考文献：

1. Christos h. Papadimitriou: Computational Complexity, Pearson, 1994

2. Francesca Rossi, Peter van Beek, Toby Walsh: Handbook of Constraint Programming. Foundations of Artificial Intelligence 2, Elsevier 2006, ISBN 978-0-444-52726-4

3. 屈婉玲，刘田，张立昂，王捍贫，算法设计与分析，清华大学出版社，2011

第二章 局部搜索

2.1 局部搜索基础概念

局部搜索是解决组合优化问题时常见的一种策略，相对于系统搜索算法，它虽然不能保证一定能够得到一个最优解，但是可以高效的得到一个次优解，通常这个可行解就已经足够投入在实际应用之中。

局部搜索是一种基于邻域搜索的元启发式方法，是求解组合优化问题最常用的方法之一。启发式（heuristic）方法有各种不同的定义，目前很难给出一个确定的定义。不过启发式方法有一些特点是公认的：

- 依靠经验
- 依据有限的知识在短时间内找到问题解决方案
- 多求得的解与最优解的偏离程度一般不能被预计。

在求解组合优化问题的时候，一般不能保证设计一个高效的启发式算法求解出最优解。一个算法的启发式信息越充足，一般利用该启发式信息求解所需要的代价就会越大。因此在设计启发式的时候，我们需要权衡算法的效率和解的质量。

一般在使用启发式算法的时候，使用者只有这些现有的信息，并想采用这些信息对问题进行求解。并且启发式算法往往会有比较好的性能和效率，尽管它通常没有办法解释为什么会这样。鉴于当前数学上发展，大部分局部搜索算法都没有理论上的保证，实践上比较成果的局部搜索算法很难在理论上被分析并得到一个缜密的结果。

局部搜索算法是相对于系统搜索的算法而言的，一般系统搜索的算法是完备的、顺序性搜索的、确定性的，而局部搜索算法通常是不完备的、可并行的、随机的。如图 2.1-1，DPLL 算法是一个系统性的搜索算法。

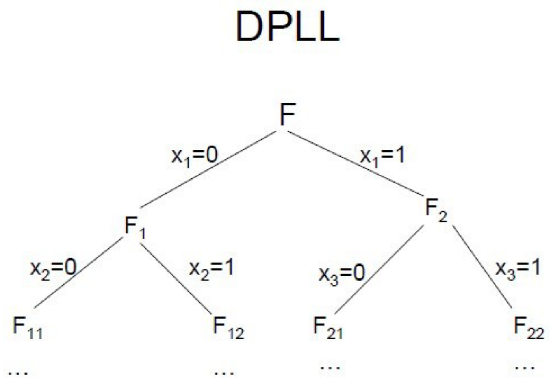

图 2.1-1 系统搜索算法

为了进一步细化局部搜索的内容，本文将给出一些基础的概念用于解释局部搜索相关的问题。

定义 2.1-1（搜索空间）从对问题对象的初始解开始，搜索理论上可以到达的所有候选

解叫做搜索空间。

定义 2.1-2（可行解）搜索空间中能够满足问题所有约束条件的候选解。

定义 2.1-3（邻域）搜索空间中相邻与某个候选解 s 相邻的所有候选解构成 s 的邻域。

定义 2.1-4（评价函数）针对问题本身的特点而设计，对候选解的质量进行定量描述的函数。

定义 2.1-5（最优解）使评价函数值达到最小的可行解。

如果将搜索空间中的候选解看为顶点，他们之间的邻域关系看作边，那么搜索空间则构成了一个网状的结构（无向图），这个网状的结构能够形象的描述搜索空间。从集合的角度来看，一个局部搜索算法，将会在这个无向图中随机的选取或者构造性的选择一个点作为初始点，然后按照启发式信息迭代的从当前位置出发，走到与其相连（邻域）的点，并且以评价函数作为启发式信息作为一个引导。

例 2.1-1. 在 SAT 问题中，搜索空间为所有可能的命题变量的指派组成的集合。可行解是可以满足所有子句的命题变量的指派，所有可行解组成了可行解空间。并且可以定义领域关系为指派之间有且只有一个变量的赋值不同，评价函数可以设为未满足子句的个数。

例 2.1-2. 在 MaxSAT 问题中，搜索空间、可行解和邻域关系时与 SAT 问题类似。其目标函数是在给定的指派下未满足子句得个数，评价函数设置为 g1+g2，其中 g1 为目标函数，g2 是在给定指派下，通过子句权重技术获得的未满足子句的权重之和。

例 2.1-3. 为了介绍随机搜索算法的大致过程，我们以 MaxSAT 问题的一个简单问题为例，下面是一个 MaxSAT 的实例，

$$F = \{x_1 \vee \neg x_2, x_1 \vee x_2, x_2, \neg x_1 \vee x_2 \vee \neg x_3\}$$

假设随机搜索的过程按照图 2.1-2 中的步骤进行，

	assignment	unsatisfied clauses
initial	000	$x_1 \vee x_2$, x_2
Step 1	100	x_2
Step 2	110	None （optimal）

图 2.1-2 MaxSAT 局部搜索

假设最初所有变量的赋值均为 false，此时有两条子句没有被满足，假设只考虑 g1 作为

评估函数，此时评估函数值为 2。第一步，在邻域中找一个评估函数比当前值小的解作为当前解，例如使变量 x_1 的值变为 true，此时评估函数为 1。第二步同理使 x_2 的值为 true，评估函数为 0。此时我们发现当前可行解的邻域中没有评估函数比 0 更小的值，因此现在的一个赋值即为最优解。

从以上的例子和介绍中不难发现，局部搜索算法中最重要的部分为邻域关系的设计和评价函数的设计。实际上，在设计局部搜索算法求解问题的时候，需要考虑的关键问题就包括，邻域关系，评估函数，以及算法策略。这也是局部搜索算法文献中最主要的内容。

局部搜索算法有它的优势和劣势。它一般是在概念上简单的、容易编码、容易并行化，通用性比较强且方便度量，绝大多数情况下它比系统搜索的方法效率更高。但是它比较明显的缺点，例如经常是不完备的（不能保证解的质量）、解有明显的随机性并且通常比较难以在理论上进行分析。

当我们遇到一个问题的时候，如果对问题的性质了解不全面、可以允许解不是那么精确、时间资源成本巨大或者实例本身特别庞大的时候，我们可以考虑采用局部搜索的思想设计算法并求解。

2.2 常见的局部搜索算法

对于局部搜索算法的研究最早可以追溯到上个世纪 50 年代，到现在已经能够从人工智能、运筹学、工业界广泛的找到其应用场景。一般来说，局部搜索都会从一个初始解开始搜索，然后在此附近邻域内进行搜索，直到找到最优解或者规定的停止条件位置。它通常在搜索的过程中采用一种类似于贪心的策略在邻域内进行搜索，算法的效率取决于启发式信息、搜索规则等，并且最主要的影响因素是算法的初始解。

接下来，本文将介绍几种常见的局部搜索算法。

1）迭代改进法

迭代改进（Iterative Improvement）又被叫做爬山法，是一种最基础的局部搜索策略，它本质上是一种贪心的策略。初始解是搜索空间中随机的一个点，在搜索的每一步，都会从当前节点的邻域中选择一个评估函数更优的点作为新的当前节点，直到邻域中不存在满足条件的解为止。伪代码如算法 1 所示。

算法 2.2-1：IterativeImprovement()

Input: 一个组合优化问题 P

Output: 问题的一个解 s

```
1    s = init(P) #初始化
2    while(true){
3        if p in neighbor(p) && evl(p)>evl(s) :
4            s = p          #邻域中有估值函数比 s 大的解
5        else :
6            return s
7    }
```

这种方法是一种典型的贪心算法，而且解的好坏完全取决于评价函数和初始解设置的好坏，容易陷入局部最优解。如图 2.2-1 所示，假如这个是按照某种评价标准描绘的评价函数和解的图像，全局最优解应该是处于心型的标识处，假如初始解在心形周围，那么必可以通过这种方法走到全局最优解处，并返回这个解。但是如果初始解处于最右侧笑脸的位置，那么经过几次迭代之后会在最后停在函数的一个极值点处，这个点的解叫做局部最优解，一般称这种现象叫做陷入局部最优，这也是局部搜索算法中经常遇到的一个问题。为了防止陷入局部最优，可以对初始解、估值函数做一定的优化，还可以对算法做一些调整。

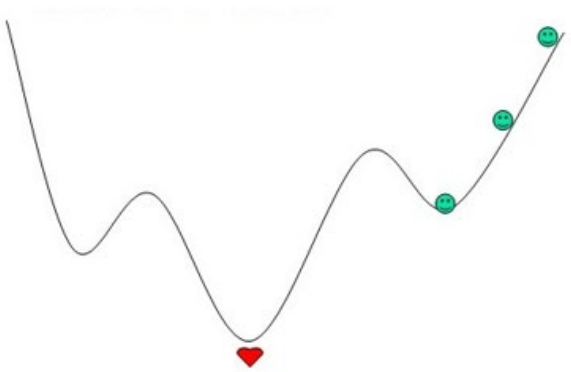

图 2.2-1 局部最优、全局最优

除此之外，局部搜索算法还需要考虑的一个问题是如何确定算法的停止。一般会有两种策略，一种是限制时间和可以走的 step 的步数，另一种是通过计算值的变化，当在一定时间段、步骤内或者其他条件下，最优评估函数值不会有变化或者变化很小，这时候就可以令算法结束搜索。在爬山法中，它停止搜索的条件取决于它的邻域关系，属于后一种停止条件。

例 2.2-1. 对于 MaxSAT 问题，如果采用迭代提升的方法，那么将会对所有变量 V 随机进行赋值作为一个初始解 s，它的邻域定义为一组与 s 只有一个变量的真值相反的指派，邻域大小为|V|，假如设置评价函数为当前指派下未满足子句的个数，那么在算法迭代的时候会不断在当前邻域中找可以使得使评价函数变小的值，直到找到一个局部最优，返回这个解。

对于迭代提升的方法，我们可以采用格局检测（configuration check，cc）的方法进行改进，cc 会对每个邻域中的解进行一定的评判，在随机的基础上有策略的选择一个合理的方向搜索。cc 的技术将在后面的章节更加仔细的介绍。我们将继续对算法做一个优化，并通过例 3 中的问题来说明改进的一种思路。

例 2.2-2. 如图 2.1-2 所示，假如当前给定了一个全为零的指派作为初始解，用<000>表示，其中分别用 0，1 表示变量的赋值为假或者真。它的邻域为 N(S)={S1,S2,S3}={<100>,<010>,<001>}。评价函数 g(S)=g(S3)=2, g(S1)=g(S2)=1。若采用朴素的迭代提升策略，则将会选择 S1，S2 中的一个解来更新当前解。

如果我们对每一个变量定义一个评价函数 score()=g(S)-g(S')，则 score(x1)=g(000) - g(100)=2-1=1， score(x2)=g(000) - g(010) = 2-1=1，score(x3)=g(000) - g(001) = 2-2=0。此时，算法选择 x1 或者 x2 中的一个变量做一次真假赋值的反转。

图 2.2-2 对比了这两种不同的算法。在实际算法的执行时，score 的值可以设置为变量反转前后由弄假子句转化为可满足子句的个数 q1 减去由可满足子句转化为弄假子句的个数 q2。在算法的实际执行过程中，也可以对 q1、q2 前面添加一些系数来平滑的调正评判标准，以达到更好的效果。在选择邻域中解的时候，可以随机的选取一个 score 值大于零的值，或者按照 score 的值进行排序，选则 score 值最大的那一个。

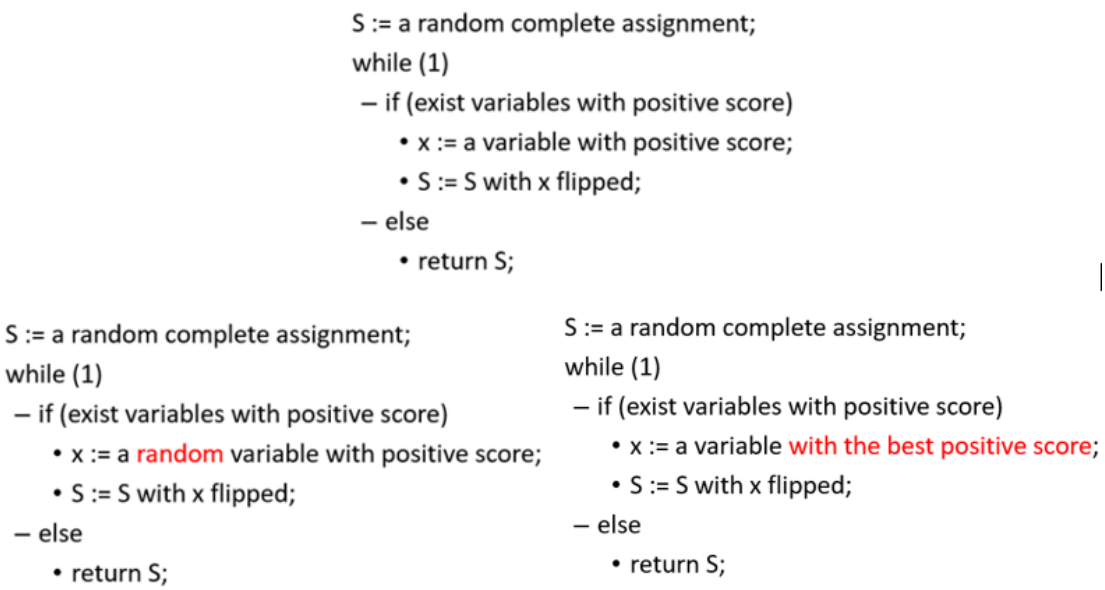

图 2.2-2 几种不同的迭代提升策略

2）跳出局部最优，模拟退火法

根据介绍的内容，执行迭代提升法，如果按照这种按照贪心的策略进行搜索，很容易陷入局部最优解。这时，影响搜索解质量的一个非常重要的因素就是算法的初始解的选取，人们很自然的就会想到一种优化的方法：重启。

通过重启的方法确实可以通过改变初始解的选择来避免陷入局部最优，但是这种方法有时就会显得特别鲁莽，那么有什么更好的策略么？答案肯定是肯定的。此时我们可以引入一个介于 0-1 之间的参数 ε，局部搜索算法以 1- ε 的概率在当前解邻域中选取一个解更新当前解，以 ε 的概率以另一种策略随机在搜索空间中选取一个解来更新当前解。其中最简单的策略就是随意在空间中选取一点更新当前解，这就类似于多次重启。我们称这种按一定的条件和概率跳出当前搜索过程的策略叫做 ε -Greedy 策略。

如算法 2 所示，为在基础的迭代提升策略上加入 ε -Greedy 策略的局部搜索算法。在这里展示其中的一种添加策略，但是并不是唯一的添加方法。

算法 2.2-2：IteImpWithE_G（）	
Input: 一个组合优化问题 P	
Output: 问题的一个解 s	
1	s = init(P) #初始化
2	while(true){
3	if p in neighbor(p) && evl(p)>evl(s) :
4	if(rand(0, 1)< ε) : # ε -Greedy 策略
5	s = randomSelect()
6	else:
7	s = p
8	else :
9	return s
10	}

除此之外，随机搜索算法在确定搜索结束的条件时也颇有讲究，为了充分的理解这个过程，本文将通过模拟退火法的例子来介绍这一过程。

模拟退火法（Simulate Anneal Arithmetic，SAA）是一种通用的概率演算算法，一般用来在一个大的搜索空间中找寻给定问题的最优解，它在解决旅行商等问题上由十分显著的提升效果。

SAA 算法灵感来源于冶金学，退火指的是将材料加热后再按照特定的速率冷却，目的是为了增大晶粒的体积，并且减少晶格中的缺陷。材料中的原子原来会停留在使内能有局部最小值的位置，加热使能量变大，原子会离开原来位置，而随机在其他位置中移动。退火冷却时速度较慢，使得原子有较多可能可以找到内能比原先更低的位置。

因此在模拟退火法的过程中引入了三个过程：

①加热的过程。目的是让粒子偏离平衡位置，增强粒子的热运动。温度是一个控制参数，确定了局部搜索的时间和随机性。

②等温过程。对于周围环境交换热量而温度不变的封闭系统，系统的状态总是自发的朝自由能减少的方向进行。当自由能达到最小的时候，系统达到平衡状态。一个状态代表了组合优化问题种的一个解。状态的能量实际上是解的评估函数。这一过程是局部搜索算法更新解的关键。

③冷却过程。使粒子的热运动减弱并逐渐有序，系统的能量逐渐下降，最后得到低能的晶体结构。这一过程实际上是在控制算法的收敛，能量最低状态其实就代表了局部搜索算法的一个最优解。

此外，算法有一个比较好的解的条件是需要保证初始温度比较高，每个温度下的状态交换必须充分，而且温度的下降必须足够缓慢。

有了以上的基础介绍和铺垫，下面将给出算法的伪代码，如算法 3 所示。

算法 2.2-3：SA	
Input: 一个组合优化问题 P	
Output: 问题的一个解 s	
1	s=random(P), k=0, t=T_MAX。
2	while(t>T_terminate){ #当没有满足终止条件
3	while(达到 t 的平衡条件){ #不能进行热交换
4	tmp_s = N(s) #s 邻域中选一个解
5	if(f(s)<f(tmp_s)):
6	s=tmp_s
7	else:
8	p = exp{- (f(tmp_s) - f(s)) / t}
9	if(p>rand(0,1)): #以一定的概率接受这个比较差的解
10	s = tmp_s
11	}
12	t=Drop(t,k),k=k+1 #以一定的规则降温
13	} return s

该算法有内外两重循环，其中内层的循环表示的是算法在降温后，物体内外进行热交换的过程，在这个温度下，如果有比较好的解将直接进行喜欢，否则将会以一定的概率接受比较差的解。外层的循环实际上是在控制系统的降温过程，收敛条件是降到一定的温度。

在实际使用这个算法的时候，其中的那些控制条件均可以通过参数的形式不断地调整和修改，以适应特定的问题。此外该类问题也有一定的衍生算法可以划分为一类，比如量子退火算法。

3）遗传算法

遗传算法（Genetic Algorithm，GA）也是一个优秀的局部搜索算法，它的灵感来源于种群的进化，优胜劣汰、适者生存。

生物的进化一般满足以下的几个条件。

①进化实质上是产生于染色体之上，而不是他们编码的生物体。

②自然选择使得染色体和它表现的结构紧密的结合在了一起，适应性好的个体一般会有更好的繁殖机会。

③进化发生在繁殖的那一刻。变异也是进化必要的一个部分。自由组合和交叉互换是基因重组的两个必要的部分。

④生物的进化没有记忆。每一个个体的进化不会受其父代的影响，即进化是环境选择的而不是一脉相传的。

根据这些规则，GA 在设计算法的时候，把个体当作问题的一个解，染色体是解的一个

编码。把环境当作适应函数，即评价函数，适应函数值最大的解被保留的概率最大。群体是被选定的一组以编码形式表示的解，种群是根据适应函数选择的一组以编码形式表示的解。交配的过程是以一定的方式由双亲产生后代的过程。编译是编码的某些分量发生变化的过程。

在 GA 中，选择、交配和变异是最主要的三个操作。根据适应值的大小，选择可以从规模为 M 的群体中选择若干染色体构成种群，种群可以与群体规模一致，也可以不一致，且适应性更好的染色体更有可能在种群中存在多个，而那些适应值比较小的染色体有可能就会被淘汰，后面均假设选出来的种群个数为 N。交配指的是从种群中随机的选取两个染色体，将染色体的某段基因互换，即基因充足中的自由组合和交叉互换的过程，图 2.2-3 代表的是两个染色体的交叉互换。假如当前有两个染色体 a、b，并且编码后为 01 的等长序列。在某个随机的位置发生了基因的重组，前后的变化即图所示。

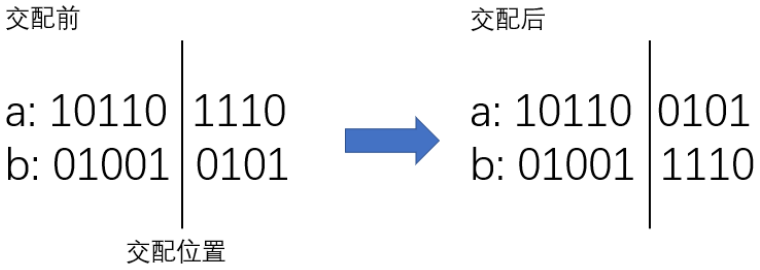

图 2.2-3 交配中的交叉互换

变异指的是染色体编码上的某个随机的位置发生了随机的变化。如图 2.2-4 所示。

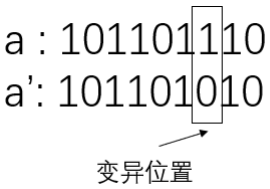

图 2.2-4 变异

在介绍了上述的问题之后，我们就能描述遗传算法的伪代码如算法 4。算法的结束条件是遗传的代数达到了限制的代数。在每一代，都会从群体中选择一个种群，如算法第 3 行表示从群体 Gm 中按照评价函数选择 N 个数据作为种群 Gn，其中主要使用的选择策略是轮盘赌策略。然后交配变异产生新的一代群体，如算法第 4 行所示表示从种群 Gn 中按照交配概率 p1 从中选择 M 对个体进行交配产生 M 个个体组成的群体 Gm，算法第 5 行表示从 Gm 中按照 p2 的变异概率从中选择个体执行变异的操作。最后当算法结束的时候返回当前代中按照评价函数 f 估值最大的那个个体。

算法 2.2-4：GA	
Input:	一个组合优化问题 P， 群体 Gm 规模 M，种群 Gn 规模 N，最多繁衍轮数 T_MAX, 交配概率 p1，变异概率 p2，
Output:	问题的一个解 s

```
1    t=0
2    while(t<T_MAX){ #有一定的交配轮数
3       Gn = choose(Gm, N, f)
4       Gm = getNextGeneration(Gn, p1, M)
5       Gm = Heteromorphosis(Gm, p2)
6       t = t+1
7    }
8    return maxUnit(Gm,f)
```

遗传算法是一个通用的框架结构，但是它并不是唯一的选择，而是作为一个模板供大家学习和参考，对算法 4 给出的伪代码有多可以进行调整的细节，比如如何对问题的解进行编码，如何选择种群，规模如何决定，如何选择交配和变异位置等等，这些都是值得考究的内容，并且不同的问题也会有不同的最优决策。

遗传算法属于群体智能算法一种，这类算法的特点是从现实生活或者自然界得到启发并按照这种自然的规律设计的。本文在本节只以遗传算法做例子，其他的群体智能算法包括蚁群算法、粒子群算法、人工鱼群算法、蜂群算法等。

2.3 局部搜索技术

在上一节，我们介绍了几种常见的局部搜索算法。为了描述的简单清晰，每种算法都只介绍最简单的算法流程，也就是算法框架。然而，在实际算法设计中，一个高效的算法往往会有各种算法策略。有些算法策略都是一般性的，并没有针对某种算法框架而设计；另外一些算法策略则可以是针对某种算法框架定制。本节主要介绍几个比较常见的一般性算法策略。

（1）Tie breaking 技术

在局部搜索中，算法会在每一步根据某个标准（或评分函数）挑选出将要执行的操作或变量。当存在多个变量或操作具备相等的评估值时，如果从这多个候选中选出一个来执行，这就是 tie breaking，可以翻译为“等链打破”。而这些变量或操作就构成了相对于该标准下的一个 tie。在 Tie breaking 方面，已有研究提出了不少方法，接下来我们介绍比较常用的 Tie breaking 方法。

- 字母序选择，也即按照变量或操作对应的字符串的先后顺序进行遍历，遇到第一个合适的（这里“合适的”是指算法可能对挑选的变量或操作还有其他要求需要满足）就返回；

- 随机选择，也即每个变量或操作以相同的概率被选择。
- 扩散规则，也即选择最近最少被访问的对象来执行，这是从搜索的扩散性考虑的，目的是尽量扩大搜索空间。
- 使用其他标准或评分函数进行选择，也即按照另一个标准（不同于导致 tie 的标准），从候选中选择评估值最优的变量或操作。

Tie breaking 技术对局部搜索算法的重要性取决于该算法出现 tie 的频率。研究发现，对于容易出现 tie 的局部搜索算法，Tie breaking 对算法的性能会产生显著影响。并且，有研究表明，对于同一个算法，在搜索的不同阶段使用不同的 Tie breaking 技术，可以比静态的使用一种 Tie breaking 技术取得更好的效果。不过，这些研究是还没有严格的理论证明，是经验性知识。

（2）基于冲突的随机游走

局部搜索算法往往会包含随机因素，而一个常见的体现就是算法中包含随机游走(random walk)步。有一种随机游走是被广泛使用且在多个问题上被表明是一个非常有效的技术，就是基于冲突的随机游走。这类技术一般在约束的语义下描述。

基于冲突的随机游走是指这样的过程：选择一个没被满足的约束，然后从该约束中随机选择变量进行操作。

对于约束满足问题，应用基于冲突的随机游走是很直接的。实际上，在约束满足问题中，比如 SAT 和 CSP，基于冲突的随机游走是被广泛使用的局部搜索技术，已经是这些问题的局部搜索算法研究中不可或缺的工具。

例. 对于 SAT 问题，一个基于冲突的随机游走步可以这样描述：选择一个当前不满足的子句 c，然后从子句 c 中随机选择一个变量改变取值。

然而，对于子集合问题（问题的形式是需要在一个集合中找出一个优化的子集合），这个技术的应用就不是那么直接，一般需要做一些修改。比如，对于顶点覆盖问题，一个典型的局部搜索算法每一步交换两个点，从候选解移出一个点，然后从候选解外面移入一个点到候选解中。对于顶点覆盖问题，一个不满足的约束对应是一条没有被覆盖的边。然而，一个搜索步需要选择两个点进行操作，“移除”点是属于覆盖的边的，“加入”点则是从未覆盖边的点集合中选择（否则没有意义）。这样，基于冲突的随机游走只对“加入”点的选择可以应用：选择一条未覆盖的边，然后随机选择该边的一个点。

（3）禁忌机制（Tabu）

美国科罗拉多大学教授 Fred Glover 在 1986 年左右提出禁忌搜索（Tabu Search，TS，又称禁忌搜寻法），其核心就是禁忌机制。禁忌搜索法就是加入了禁忌机制的局部搜索算法。禁忌搜索最重要的思想是标记对应已搜索的候选解的一些对象，并在接下来若干次迭代搜索中尽量避开这些对象（而不是绝对禁止循环），从而保证对不同的有效搜索途径的探索。

禁忌机制可以这样描述：给每个解部件（也即解的基本元素，比如变量，顶点）关联一个属性表示是否被禁忌。当一个解部件被执行操作的时候，它的禁忌属性设为真，并且持续

若干个搜索步（这个周期长度称为禁忌周期或禁忌长度 T），而局部搜索过程中就会避免选择被禁忌的解部件来执行操作。

一种常见的实现方法是采用先进先出队列储存被禁忌的解部件，该队列的长度固定为禁忌周期的长度；当一个变量的禁忌属性从假变为真的时候，加入到禁忌队列尾端，当禁忌队列满的时候，从队头移除元素。在这个实现当中，在禁忌队列中的，就是被禁忌的元素，反之，所有不在队列中的元素都没有被禁忌。禁忌周期是一个很重要的关键参数，其大小直接进而影响整个算法的搜索进程和行为。如果该参数设置太大，则会因为禁止过多的对象而导致搜索过于局限，如果参数设置太小，则不能有效避免循环。禁忌长度的参数设置往往要通过实验方法进行确定，并没有有效的配置原理可以计算。这也使得禁忌机制在使用的时候不可避免地引入了额外的算法调试工作。

禁忌机制在应用于局部搜索时，经常会搭配藐视准则。若禁忌对象对应的适配值优于“ best so far”状态，则无视其禁忌属性而仍采纳其为当前选择，也就是通常所说的藐视准则（或称特赦准则）。同时，禁忌表中禁忌对象的替换是采用 FIFO 方式，当然也可以采用其他方式，甚至是动态自适应的方式。

禁忌搜索是以 Tabu 机制为主体的局部搜索算法，由美国科罗拉多大学系统科学家 Glover 教授于 1986 年在一篇论文中首次提出。之后不久，Glover 教授分别在 1986 年和 1990 年发表了两篇著名的标题为 Tabu search 的论文，提出了现在大家所熟知的禁忌搜索的大部分原理。因为其简单和容易实现，禁忌搜索已经被广泛用于求解各种组合优化问题。

禁忌搜索的大体过程可以由下面的步骤描述。对于禁忌搜索算法来说，最重要的部分为禁忌表的设置、藐视准则和终止条件。

①设置算法参数、初始解 s，设置禁忌表

②满足终止条件则输出当前最优解。

③从 s 的邻域中选出一部分解作为候选解。

④判断候选解是否满足藐视准则？如果满足则替换 s，并用该候选解来替换之前禁忌表中对应的解。如果不满足则继续。

⑤判断候选解对应的各对象的禁忌情况，选择候选解集中非禁忌对象对应的最佳状态为心得当前解，同时用与之对应得禁忌对象替换最早进入禁忌表的禁忌对象。转步骤②。

禁忌搜索一般通常和其他的一些随机搜索算法组合使用，往往能达到不错的效果。

（4）格局检测（Configuration Checking）

格局检测是最近（2011）才提出的局部搜索策略，该策略能很好地阻止局部搜索算法重新访问最近访问过的候选解。在格局检测策略中，每一个变量的格局（也被称作环境信息）将会被记录，格局检测会禁止变量重复进入刚离开的格局，从而避免局部循环。从直觉上来说，这样可以有效地避免重复搜索。从以上描述可以很容易看出，格局检测的核心关键是如何定义变量的格局。

下面我们通过格局检测在 SAT 局部搜索算法上的应用来阐述该方法。在一个典型的格局检测策略中，变量的格局定义为该变量的所有邻居变量的取值所构成的向量，这里邻居变

量是指共享至少一条约束的变量。

定义2.3-1 （变量的格局） 给定一个CNF形式的SAT实例F以及一个关于$V(F)$的完备赋值s，对于每一个变量$x \in V(F)$，变量x的格局$configuration(x)$是一个向量，该向量的每一个元素是变量x的每一个邻居变量的布尔真值（0或1）。

可以很容易得知，变量x的格局$configuration(x)$的长度和$|N(x)|$是相等的。对于一个变量x，$configuration(x)$上每一个元素的一次改变被认为是向量上的一个改变，也就是x的格局发生了改变。

一个典型的格局检测策略可以这样描述：对于那些自上次改变取值之后格局没有发生变化的变量，禁止把它的取值变回前一个取值。图2.3-1给出了这个策略的直观理解。我们考虑变量v，周围的4个点代表它的所有邻居变量，两种颜色代表两种不同取值（0或1）。图中给出了两种情况。在第一种情况，当v改变了取值之后，经过若干步，算法再次考虑选择v翻转取值，这时候，根据格局检测策略的定义，检查一下v的格局---也就是v的邻居变量取值---是否发生改变，检查发现v的格局确实发生了改变，所以v是允许变回原来的取值的。而对于第二种情况，在检查的时候发现，v的格局和上次v取值发生改变的时候是一样的，所以不运行变量v变回原来的取值。

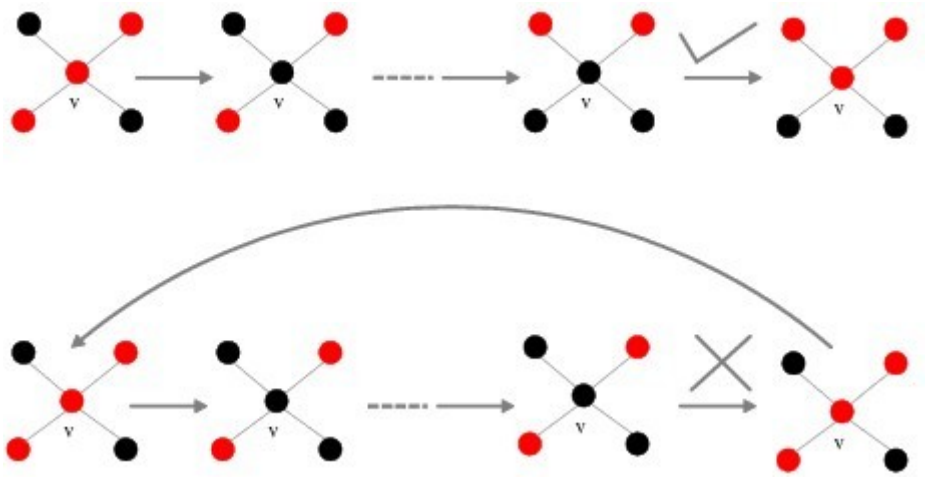

图 2.3-1 基于邻居变量的格局检测

格局检测和 Tabu 都属于局部搜索的禁止策略(forbidding strategy)。Tabu 方法侧重于从时间的维度考虑，在一定时间范围内禁止某些操作；而格局检测侧重于从空间，或者说问题的结构的问题都进行考虑，禁止会带来局部结构发生循环的操作。格局检测自提出以来，在许多组合优化问题上都得到了成功应用，说明了该方法的有效性。总的来说，格局检测有两个突出的优点：首先，该策略没有引入参数，不需要针对不同的实例来配置参数；其次，格局检测是灵活的，是一个一般性原则，具体的策略根据格局的定义和检测的方法有所不同，不同的策略对应的禁止(forbidding)力度不一样，算法设计者可以根据算法行为的观察调整格局检测的策略。

（5）约束加权

局部搜索算法的加权机制，是指通过对约束或变量引入新的权值，并通过实时更新这些权值，引导搜索算法避免局部最优，并增加搜索的扩散性。一般地，当约束被违反的时候，它的权值就会被增加，以此表示惩罚，从而加强算法满足该约束的倾向。

当使用约束加权的时候，局部搜索的评估函数也会相应地定义为加权的版本。比如，在SAT 问题中，如果采用子句加权技术，对应的评估函数则应该从不满足子句数修改为不满足子句的总权值。在使用约束加权技术的时候，一个重要的考虑是在什么情况下更新权值。对这个问题有两种常见的方案：

- 在搜索中的每一步执行之后，对每一条不满足的约束，增加其权值（一般加 1）
- 在遇到局部最优的时候，对每一条不满足的约束，增加其权值（一般加 1）

在 SAT 问题的局部搜索算法中，一个早期的子句加权技术 Breakout 就是在每次遇到局部最优的时候，对每一个不满足的子句权值加 1。这种朴素的加权技术会产生一个问题：允许约束的权值不限制地增加，从而导致“畸形”的评估函数。为了避免这种情况，后来的约束加权技术引入了“平滑”机制，使得约束的权值在一定条件下会被减少。下面介绍几种常见的平滑机制的启动时机：

- 在权值增加操作达到一定次数后，对子句权值进行平滑操作。
- 在每一次更新权值的时候，以一定的概率，对子句权值进行平滑。
- 在约束的评价权值达到一个阈值之后，对约束权值进行平滑操作。

而具体的平滑操作也有几种常见的设计：

- 对指定的约束（可以是全部约束或者一部分约束），减去一个常量，如果该操作使得某个约束的权值小于等于 0，那么对该权值赋值为 1.
- 对指定的约束（可以是全部约束或者一部分约束），使用平滑公式为$\mathrm{w(c)} = \rho \cdot \mathrm{w(c)} + (1-\rho)w_{avg}$进行平滑，该公式会把子句权值分布在一定程度上往平均值靠拢。
- 对指定的约束（可以是全部约束或者一部分约束），使用平滑公式为$\mathrm{w(c)} = \rho \cdot \mathrm{w(c)}$进行平滑，该公式对约束的权值按一定比例缩小。

2.4 局部搜索理论分析

局部搜索算法的理论分析是非常有挑战性的任务。目前已有的研究也只能对非常简单的局部搜索算法进行分析。实际上，作为最常见的启发式算法，在局部搜索算法中存在一个矛盾：实际中拥有好的求解性能的局部搜索算法往往加入了一些不好分析策略，使得理论分析难以进行；为了得到理论上的结果，往往算法需要保持简单并且满足某些规范，但这样的算法在实践中往往不具备好的性能。实际上，图灵奖获得者 Karp 在理论计算机会议 STOC 2009 上就曾经提出，分析启发式算法是理论计算机科学家面临的一个挑战。

2.4.1 马尔可夫链分析局部搜索

考虑在每个时间段有一个值的随机过程。令 X_n 表示它在时间段 n 的值，假设我们要对一系列相继的值 $X_0, X_1, X_2, \cdots$ 建立概率模型，最简单的模型可能就是假设 $X_n (n=0, 1, 2, \cdots)$是

独立的随机变量，但这个假设常常是不合理的。例如，从某个时刻开始 X_n 代表某种股票（例如 Google）在未来 n 个交易日末的价格。若假定在第 $n+1$ 个交易日末的价格与第 $n, n-1, n-2, \cdots, 0$ 日的价格独立，这显然是不合理的。然而，若假定第 $n+1$ 个交易日末的价格通过第 n 日末的价格依赖于以前的盘后价格，这可能是合理的。也就是说，给定以前的盘后价格 $X_n, X_{n-1}, \cdots, X_0$ 时，X_{n+1} 的条件分布只通过第 n 个交易日末的价格依赖于以前的这些盘后价格。这种假设就定义了一个马尔可夫链，这是本节将要研究的一种随机过程，现在我们正式地定义它。

令 $\{X_n, n=0,1,2,\cdots\}$ 是有限个值或者可数个可能值的随机过程。除非特别提醒，这个随机过程的可能值的集合都将记为非负整数的集合 $\{0,1,2,\cdots\}$。如果 $X_n=i$，那么称该过程在时刻 t 在状态 i，我们假设只要过程在状态 i，就有一个固定的概率 $P_{i,j}$ 使它在下一个时刻在状态 j。即我们假设对于一切状态 $i_0, i_1, \cdots, i_{n-1}, i, j$ 以及一切 $n \geqslant 0$，有

$$P\{X_{n+1}=j \mid X_n=i, X_{n-1}=i_{n-1}, \ldots, X_1=i_1, X_0=i_0\}=P_{ij} \qquad (2.4.1)$$

这样的随机过程称为马尔可夫链，方程 2.4.1 可以解释为，对于一个马尔可夫链，在给定过去状态 $\mathrm{X}_0, X_1, \ldots, X_{n-1}$ 和现在的状态 X_n 时，将来的状态 X_{n+1} 的条件分布独立于过去的状态，且只依赖于现在的状态。

$P_{i,j}$ 表示过程处在状态 i 时下一次转移到状态 j 的概率。由于概率都是非负的，又由于过程必须转移到某个状态，所以有

$$P_{i,j} \geq 0, \qquad i, j \geq 0; \qquad \sum_{j=0}^{\infty} P_{ij}=1, \quad i=0,1,\ldots$$

以 P 记一步转移概率 $P_{i,j}$ 的矩阵，所以

$$P=\begin{bmatrix} P_{00} & P_{01} & P_{02} \cdots \\ P_{10} & P_{11} & P_{12} \cdots \\ \vdots & \vdots & \vdots \\ P_{i0} & P_{i1} & P_{i2} \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

例 2.4.1(天气预报) 假设天气下雨的机会只依赖于前一天的天气条件，即今天是否下雨，不依赖于过去的天气条件。再假设如果今天下雨，那么明天下雨的概率为α；如果今天没有下雨，那么明天下雨的概率为β。

如果下雨，我们假定过程在状态 0；如果不下雨，我们假定过程在状态 1，那么，上面的内容是一个两个状态的马尔可夫链，其转移概率矩阵给定为

$$\begin{bmatrix} \alpha & 1-\alpha \\ \beta & 1-\beta \end{bmatrix}$$

用随机游动分析可满足性问题的概率算法

考察一个状态为 $0, 1, \cdots, n$ 的马尔可夫链，其

$$P_{0,1}=1,\quad P_{i,i+1}=p,\quad P_{i,i-1}=q=1-p,\quad 1\le i<n$$

并且假设我们想要研究这个链从状态 0 到状态 n 所花的时间。计算达到状态 n 的平均时间的一种方法是以 m_i 记从状态 i 走到状态 n 的平均时间，$i=0,\cdots,n-1$。如果我们取条件于初始转移，就得到下面的方程：

$$\begin{aligned}m_0 &= 1+m_1,\\ m_i &= E[\text{到达n的时间}|\text{下一个状态是i+1}]p\\ &\quad + E[\text{到达n的时间}|\text{下一个状态是i-1}]q\\ &= (1+m_{i+1})p+(1+m_{i-1})q\\ &= 1+pm_{i+1}+qm_{i-1},\quad i=1,\ldots,n-1\end{aligned}$$

尽管从上面的方程中可以解出 m_i, $i=0,\cdots,n-1$, 但是我们并不想要它们的解，而是想利用这个马尔可夫链的特殊结构得到一组更简单的方程。首先以 N_i 记这个链先进入状态 i 直至进入状态 $i+1$ 所用的附加转移次数。由马尔科夫性质推出这些随机变量 N_i $(i=0,\cdots,n-1)$是独立的。此外，我们可以将这个链从状态 0 进入到状态 n 所用的时间 $N_{0,n}$表示为

$$N_{0,n}=\sum_{i-1}^{(n-1)}N_i \qquad (2.4.2)$$

令 $\mu_i=E[N_i]$，对这个链进入状态 i 后的下次转移取条件，对于 $i=1,\cdots,n-1$ 有

$$\mu_i = 1+E[\text{到达i+1的附加转移次数}\,|\,\text{链到i-1}]q$$

现在，如果这个链下一次进入状态 $i-1$，那么为了到达 $i+1$，他必须先回到状态 i，然后必须走向状态 $i+1$。因此，由前面可得

$$\mu_i = 1+E[N_{i-1}^{*}+N_i^{*}]q$$

其中 N^{*}_{i-1}和 N^{*}_{i}分别是从状态 $i-1$ 回到 i 的附加转移次数和从 i 到达 $i+1$ 的次数。现在，由马尔可夫性质推出这些随机变量分别与 N_{i-1}和 N_i有相同的分布。此外，他们是独立的(虽然我们只用它计算 $N_{0,n}$的方差)。因此，我们有

$$\mu_i = 1+q(\mu_{i-1}+\mu_i)$$

从而

$$\mu_i=\frac{1}{p}+\frac{q}{p}\mu_{i-1},\quad i=1,\ldots,n-1$$

由 $\mu_0=1$，并令 $\alpha=q/p$,我们由上面的递推公式得到

$$\begin{aligned}\mu_1 &= 1/p+\alpha\\ \mu_2 &= 1/p+\alpha(1/p+\alpha)=1/p+\alpha/p+\alpha^2,\\ \mu_3 &= 1/p+\alpha(1/p+\alpha/p+\alpha^2)=1/p+\alpha/p+\alpha^2/p+\alpha^2\end{aligned}$$

一般地，我们有

$$\mu_i = \frac{1}{p}\sum_{j=0}^{i-1}\alpha^j + \alpha^i, \quad i = 1,...,n-1$$

现在我们用方程(2.4.2)得到

$$\mathrm{E}[N_{0,n}] = 1 + \frac{1}{p}\sum_{i=1}^{n-1}\sum_{j=0}^{i-1}\alpha^j + \sum_{i=1}^{n-1}\alpha^i$$

当 *p=1/2* 时有：*α=1*，从上面我们看到

$$\mathrm{E}[N_{0,n}] = 1 + (n-1)n + n - 1 = n^2$$

当 *p≠1/2* 时有，我们得到

$$\begin{aligned}\mathrm{E}[N_{0,n}] &= 1 + \frac{1}{p(1-\alpha)}\sum_{i=1}^{n-1}(1-\alpha^i) + \frac{\alpha-\alpha^n}{1-\alpha} \\ &= 1 + \frac{1+\alpha}{1-\alpha}[n-1-\frac{(\alpha-\alpha^n)}{1-\alpha}] + \frac{\alpha-\alpha^n}{1-\alpha} \\ &= 1 + \frac{2\alpha^{n+1} - (n+1)\alpha^2 + n - 1}{(1-\alpha)^2}\end{aligned}$$

其中第二个等式用了 *p=1/(1+α)*。所以，我们看到当 *α>1* 时，或者等价地，当 *p<1/2* 时，到达 *n* 的转移次数的期望是对 *n* 指数地递增的函数。另一方面，当 *p=1/2* 时，$\mathrm{E}[N_{0,n}] = n^2$，而当 *p>1/2* 时，对很大的 *n*，$\mathrm{E}[N_{0,n}]$实质上对 *n* 是线性的。

下面我们用以上马尔可夫链演化模型分析一个可满足性问题的局部搜索算法。考虑下面的求解 2-SAT 问题（即每个子句都刚好包含 2 个文字的 SAT 问题）的算法。

Focus Local Search (FLS)

Pick an assignment $S \in \{0.1\}^n$;

For $i = 1$ *to* $2n^2$

if (S is a solution) return S;

c ← a (random) unsatisfied clause;

v ← a random variable in c;

S ← S with v flipped;

算法从任意一个完全赋值出发，在算法的每一步考察一个不可满足的子句，于是推出这个子句的两个变量中至少有一个值与 φ 中的中的对应值不一致。结果，当我们在此子句中随机选取一个变量时，则至少以概率 1/2 有 $Y_{j+1} = Y_j + 1$，而至多以概率 1/2 有 $Y_{j+1} = Y_j$-1，即独立于此算法中以前所已知的情况，每一步设置与 φ 中的值相一致的个数将增加或减少 1，而增加 1 的概率至少是 1/2(若两个变量的值都与 φ 中的值不一致，则概率是 1)。从而，即

使过程 $Y_j (j \geq 0)$本身不是马尔可夫链，直观上显然地，为得到 φ 的值所需要的算法步数的期望将少于或等于以上考虑的马尔可夫链从状态 0 到状态 n 的转移次数的期望， 也即n^2。所以，算法运行$2n^2$步，根据马尔可夫不等式，对于一个可满足的 2-SAT 公式找不到解的概率小于 1/2. 我们可以通过运行此算法多次（记为 m），那么最终找到解的概率至少为$1-\left(\frac{1}{2}\right)^{\mathrm{m}}$.

2.4.2 更复杂的例子

除了马尔可夫链演化模型，其他概率论的方法对局部搜索算法的分析都是非常重要的。本节通过条件概率的方法分析一个求解 3-SAT 问题的局部搜索算法。对于局部搜索算法，比较典型的理论分析，是计算该算法找到最优解所需要的期望步数。

下面的这个例子，是对 SAT 问题的一个经典局部搜索算法进行期望步数的分析。整体的分析思路为：对初始解的可能情况进行分类，对每一类情况进行条件概率计算，最后通过全概率公式获得最终结果。

Focus Local Search (FLS)

Pick an assignment $S \in \{0.1\}^n$ at random; // 记为S_0

For $i = 1 \rightarrow n$

if (S is a solution) return S;

c ← a (random) unsatisfied clause;

v ← a random variable in c;

S ← S with v flipped;

1) 对可满足的 SAT 公式，计算 FLS 找到一个特定解S^*需要的期望运行次数$\mathrm{d}(S_0, S^*)$: S_0与S^*的 Hamming distance，即取值不同变量个数。

$$\Pr(find\ S^*\ in\ n\ steps) = \sum_{t=0}^{n} \Pr(\mathrm{d}(S_0, S^*) = t) \cdot \Pr(find\ S^*\ in\ n\ steps \mid \mathrm{d}(S_0, S^*) = t)$$

$$\geq \sum \Pr(\mathrm{d}(S_0, S^*) = t) \cdot \Pr(find\ S^*\ in\ t\ steps \mid \mathrm{d}(S_0, S^*) = t)$$

其中，$\Pr(\mathrm{d}(S_0, S^*) = t) = \binom{n}{t} \cdot 2^{-n}$

我们知道，每个 unsat 子句中，必存在至少一个变量，它当前赋值与S^*中不同。所以，每一步选择的变量可以使S与S^*距离少 1 的概率$\geq \frac{1}{3}$ 。

∴ 条件概率$\geq \left(\frac{1}{3}\right)^t$

所以，

$$\Pr(find\ S^*\ in\ n\ step) \geq \sum_{t=0}^{n} \binom{n}{t} \cdot 2^{-n} \cdot \left(\frac{1}{3}\right)^t = 2^{-n} \sum_{t=0}^{n} \binom{n}{t} \cdot 3^{-t} = 2^{-n} \left(1 + \frac{1}{3}\right)^n = \left(\frac{2}{3}\right)^n$$

所以，算法的每次执行找到一个解的概率是$\left(\frac{2}{3}\right)^n$，在找到一个解之前，算法的期望执行次数为$1.5^n$次，每次运行 n 步。

2) 以上分析是假定算法的每一步都正确，放松一下，若允许算法有些步犯错，当“好”步法比“坏”步法多 t 步时，可找到S^*。

特别，考虑算法的 for 循环有 3n 步，

$\Pr\ (find\ S^*\ in\ 3n\ steps)$

$$\geq \sum_{t=0}^{n} \Pr(\mathrm{d}(S_0, S^*) = t) \cdot \Pr(find\ S^*\ in\ 3t\ steps \mid \mathrm{d}(S_0, S^*) = t)$$

条件概率为$\geq \binom{3t}{t}\left(\frac{1}{3}\right)^{2t}\left(\frac{2}{3}\right)^{t}$，取 2t 为好步，t 为坏步。

由 Stirling 公式 $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ 得$\binom{3t}{t} \geq \frac{1}{\sqrt{5t}} \cdot \frac{3^{3t}}{2^{2t}}$

所以，$\Pr\ (find\ S^*\ in\ 3n\ steps)$

$$\geq \sum_{t=0}^{n} \binom{n}{t} \cdot 2^{-n} \cdot \frac{1}{\sqrt{5t}} \cdot \frac{3^{3t}}{2^{2t}} \cdot \left(\frac{1}{3}\right)^{2t} \cdot \left(\frac{2}{3}\right)^{t}$$

$$= \sum_{t=0}^{n} \binom{n}{t} \cdot 2^{-n} \cdot \frac{1}{\sqrt{5t}} \cdot \left(\frac{1}{2}\right)^{t}$$

$$\geq 2^{-n} \cdot \frac{1}{\sqrt{5n}} \cdot \sum_{t=0}^{n} \binom{n}{t} \cdot \left(\frac{1}{2}\right)^{t}$$

$$= 2^{-n} \cdot \frac{1}{\sqrt{5n}} \cdot \left(1 + \frac{1}{2}\right)^{n} = \frac{1}{\sqrt{5n}} \cdot \left(\frac{3}{4}\right)^{n}$$

所以，在找到一个解之前，需要执行该算法的期望次数为$\sqrt{5n} \cdot \left(\frac{4}{3}\right)^n$。

本章小结：

本章介绍了局部搜索算法，从算法的相关定义，基本算法，算法策略和算法分析几个方面进行了介绍。介绍了几种基本算法，包括迭代改进，带随机的迭代改进，模拟退火，遗传算法。介绍了几种一般性局部搜索策略，包括 Tie breaking，基于冲突的随机游走，Tabu，格局检测，以及约束加权。这些方法可用于各种局部搜索算法中，通用性强。最后，我们通过对一个 SAT 问题的典型局部搜索算法进行期望步数的理论分析，说了最基本的分析步骤和工具。

对于本章的内容，希望达到的教学目标是：理解局部搜索算法的原理和过程；会对一个组合优化问题设计一个局部搜索算法，掌握两到三个局部搜索策略；了解局部搜索算法的理论分析的现状以及一般采用的数学工具。

本章参考文献：

1. Holger H. Hoos, Thomas Stützle: Stochastic Local Search: Foundations & Applications. Elsevier / Morgan Kaufmann 2004, ISBN 1-55860-872-9
2. Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh: Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications 185, IOS Press 2009, ISBN 978-1-58603-929-5
3. Rafael Martí, Panos M. Pardalos, Mauricio G. C. Resende: Handbook of Heuristics. Springer 2018, ISBN 978-3-319-07123-7
4. Shaowei Cai, Chuan Luo, Kaile Su: Improving WalkSAT By Effective Tie-Breaking and Efficient Implementation. Comput. J. 58(11): 2864-2875 (2015)
5. Darrell Whitley, Adele E. Howe, Doug Hains: Greedy or Not? Best Improving versus First Improving Stochastic Local Search for MAXSAT. AAAI 2013
6. Fred Glover: Tabu Search - Part I. INFORMS Journal on Computing 1(3): 190-206 (1989)
7. Fred Glover: Tabu Search - Part II. INFORMS Journal on Computing 2(1): 4-32 (1990)
8. Uwe Schöning: A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems. FOCS 1999: 410-414
9. Shaowei Cai, Kaile Su: Local search for Boolean Satisfiability with configuration checking and subscore. Artif. Intell. 204: 75-98 (2013)

第三章 局部搜索案例学习---图着色问题

图着色问题(Graph Coloring Problem, 简称 GCP)是图论和组合优化领域中最古老的问题之一。它是重要的 NP 难组合优化问题，同时也是美国离散与应用数学学会于 1992 年提出三大难解问题之一（其余两个是最大团问题和命题可满足性问题）。许多实际生活中的问题都可以直接转化为图着色问题来求解，包括调度问题、时间表问题、 铁路客运优化、 频率分配、寄存器分配、通讯网络等等。 正因为该问题所具有的重要理论意义和求解的困难程度，以及在工业邻域中的诸多具体应用，其在学术界得到了广泛的关注。

3.1 图着色问题

定义 3.1-1（简单无向图） 一个简单无向图$G=(V,E)$由点集合$V=\{v_1,v_2,v_3\ldots\ldots,v_n\}$以及边集合$E\in V\times V$组成。

当两个顶点u和v中有一条边连接时，我们称u和v是邻接的，也称u和v相为邻居。

所有与顶点v邻接的顶点集合记为$N_G(v)$，若 G 在上下文中是明确的，则简写为$\mathrm{N}(v)$。

图 G 的补图$\bar{G}$有着和 G 相同的顶点集合，且在图$\bar{G}$中两个顶点是邻接的，当且仅当在图 G 中这两个顶点是不邻接的。

定义 3.1-2（顶点着色） 给定一个简单无向图$G=(V,E)$，顶点着色（或简称为着色）是一个从顶点集合V到颜色集合$Color$的函数$\alpha:V\to Color$，使得V中的任意两个顶点u和v，若$(u,v)\in E$，则$\alpha(u)\neq\alpha(v)$。

一般情况下，$Color$是一组自然数的集合，其中每一个数代表一种颜色。图 G 的一个 k 着色是指只使用了最多 k 种颜色的着色。另一方面，如果图 G 可以使用 k 种颜色对其着色，则我们称图 G 为 k 可着色的。

定义 3.1-3（图着色问题） 给定一个简单无向图$G=(V,E)$，图着色问题是寻找图 G 的一个着色，使得所使用的颜色数量最少。

图 G 的色数$\chi(G)$是使得图 G 的 k 着色存在的最小 k 值，当上下文意义明确的时候，$\chi(G)$简写为χ。由于图着色问题的难解特性，一般情况往往很难确定最优解的大小，因此在实际求解时，我们的目标是使得所使用的颜色数尽可能的小。

与图着色相关的一个概念是独立集，其定义如下。

定义 3.1-4（独立集） 给定一个简单无向图$G=(V,E)$，一个独立集是顶点集合V的一个子集，其中的任意两个点都互不邻接。

由于相同颜色的顶点互不邻接，因此它们构成了一个独立集，从而图的着色可以视为顶点集合V的一种划分，即根据颜色将顶点划分为不同的独立集。我们将每种颜色所对应的独

立集称为一个颜色顶点集。

另一个相关概念是团，其定义如下。

定义 3.1-5（团） 给定一个简单无向图$G=(V,E)$，一个团是顶点集合V的一个子集，其中任意两个顶点都彼此邻接。

图 G 中最大的团的顶点个数记为$\omega(G)$，若 G 在上下文中是明确的，我们简记为ω。值得注意的是，一个图的独立集是其补图的团，反之亦然。

在一个团中，因为每个顶点之间都彼此邻接，所以它们所使用的颜色必然彼此不同，换而言之，团的顶点个数与对其着色所需的颜色数相等。因此，团的大小可以作为着色所需颜色数的一个有效下界。如果一个图中存在一个大小为 k 的团，则可以知道对这个图着色至少需要 k 种颜色。

另外，给定图的一个着色，因为颜色相同的顶点之间彼此互不邻接，所以对于图中的任意一个团， 同一种颜色中的顶点至多只能有一个出现在团中。 因此，如果能够使用 k 种颜色对图进行着色，则在该图中团的大小最多为 k，即图的着色可以作为团的一个上界。

由于图的着色和团的相关性，许多最大团问题的求解算法往往会使用图着色来优化性能，反之，许多图着色问题的求解算法也利用了团的信息。

例 3.1-1 图 1.1-1 给出了一个由 6 个点，8 条边构成的简单无向图以及它的着色方案。为每个点选择{1,2,3}中的一个颜色进行着色(1,2,3 分别用不同深度的灰色来表示)。用$\mathrm{a}(v_i)$来表示点v_i所着的颜色，那么在这个例子中，$\mathrm{a}(v_1)\coloneqq\mathrm{a}(v_6)\coloneqq 1$(浅灰色)，$\mathrm{a}(v_2)\coloneqq\mathrm{a}(v_5)\coloneqq 2$(深灰色)，$\mathrm{a}(v_3)\coloneqq\mathrm{a}(v_4)\coloneqq 1$(白色)。该图的色数为 3。事实上通过进一步观察可以发现，该图中最左(右)边的三个点构成了一个团，这也为该图的色数提供了一个下界，因为要对由 k 个点组成的团进行合法着色，至少需要 k 种颜色。

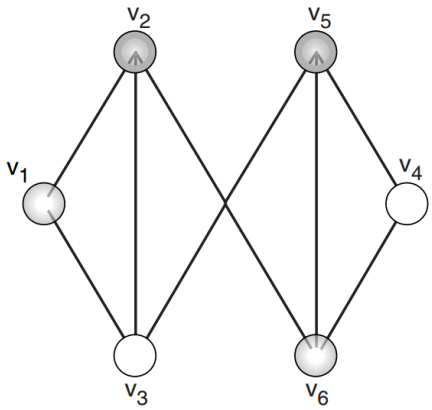

图 3.1-1 一个由 6 个点组成的图着色实例及其最优着色方案

3.2 图着色问题的局部搜索算法

图着色问题的求解算法主要可以分为三大类，分别是精确算法、贪心着色算法(如著名的 DSATUR 算法)、启发式搜索算法。由于着色问题本身的难解特性，精确算法目前只能稳定求解顶点数较小的实例，因此，实际应用过程通常不会采用。在本章中，我们将主要介绍两种常见的局部搜索算法。

局部搜索算法的求解过程是，从一个初始解出发，不断地进行迭代改进，直到找到了满意的解，或者耗尽给定的计算资源为止。在局部搜索中，算法不断地从当前解移动到下一个解中。使用局部搜索算法，需要定义如下三个要素：

1. **候选解集合：**该集合定义了局部搜索的搜索空间。在图着色问题中，一个候选解可以是一个完整着色，也可以是一个部分着色。
2. **候选解之间的邻域关系：**局部搜索的每一步操作是从一个候选解移动到下一个候选解， 对于一个候选解，它的邻域为从该候选解移动一步可以到达的所有候选解，邻域中的每个元素称为该候选解的一个邻居。
3. **候选解的评估函数：**用于衡量该候选解的质量。

当一个着色方案中不存在两个相邻的点颜色相同时，则称该方案为合法的。当一个着色方案中所有的点都有对应的着色时，则称该方案为完整的。由此，在图着色问题中，按照候选解的特点主要可以将局部搜索算法分为以下三类：

1. 颜色数k固定的情况下、在不一定合法的完整着色方案中进行搜索，搜索目标为合法(冲突数为 0)的着色方案。
2. 颜色数k固定的情况下、在合法的不完整着色方案中进行搜索，搜索目标是完整(未着色点的数目为 0)的着色方案。
3. 颜色数k可变的情况下、在完整的合法着色方案中进行搜索，搜索目标为满足条件的最小 k 值。

其中当所使用的颜色数固定时，算法解决的是图着色问题的判定版本，当颜色数可变时，算法解决的是图着色问题的优化版本。特别地，对于第三种类别，算法的候选解一般是一个合法的完整着色，而在每一步迭代中，利用 Kempe Chain 的概念改变一些顶点的颜色，同时保持着色的完整性和合法性，但这种框架的算法进展比较缓慢。

（1） 基础算法 TABUCOL

TABUCOL 是最早的求解图着色问题的启发式搜索算法之一，该算法的候选解是颜色数固定的完整着色。TABUCOL 作为一个非常著名的算法，已经被成功地改进和内嵌到了更复杂的算法当中。

TABUCOL 属于图着色算法分类中的第一类。对应局部搜索算法中的相关要素定义如下：

候选解集合：图 G 的一个完整的k着色，但着色允许是非法的，即允许存在一些冲突边，这些边的两个的顶点被赋予了同样一种颜色。

图 2.1-1

如图 2.1-1 中，对于一个由四个点和四条边构成的图，当 k = 3 时，左图(合法)与右图(不合法)都可以作为 TABUCOL 的候选解。

候选解之间的邻域关系：算法的每一步移动修改当前候选解中一个顶点的颜色。在每一步移动中，TABUCOL 只选择那些关键的顶点，即冲突边中的顶点。

候选解的评估函数：在当前着色方案中存在的冲突边的数量。同样在图 2.1-1 中，左图的评估函数值为 0(合法的)，右图的评估函数值为 1。

算法 3.2-1. (TABUCOL)

Input： $G = (V, E)$

$k =$ number of colors

$|T| =$ size of tabu list

$rep =$ number of neighbors in sample

$nbmax =$ max number of iterations

Output： valid k-coloring of G or $\emptyset$

Generate a random solution $s = \{V_1, V_2, V_3 \dots \dots, V_k\}$;

$nbiter := 0$;

Choose an arbitratry tabu list T;

While $f(s) > 0$ **and** $nbiter < nbmax$ **do**

Generate rep neighbors s_i of s with move $s \to s_i \notin T$ or $f(s_i) \leq Best(f(s))$

// as soon as we get an $f(s_i) \leq Best(f(s))$ with we stop the generation.

Let s' be the best neighbor generated;

Update tabu list T;

// introduce move $s \to s'$ and remove oldest tabu move

$s := s'$

$nbiter := nbiter + 1$;

if $f(s) = 0$ **then**

return *A coloring of G with k colors:* $\{V_1, V_2, V_3 \dots \dots, V_k\}$*are color sets.*

else

return *no coloring has been found with k colors.*

另外，TABUCOL 采用禁忌搜索策略来避免搜索陷入局部最优。即在求解过程中维护一个禁忌列表（tabu list）,当顶点v的颜色值从c_1改变为c_2后，将(v, c_1)添加到禁忌列表中，不允许在后续的若干步数(称为禁忌长度，tabu tenure) 之内又重新将顶点v的颜色赋值为c_1。

在 TABUCOL 算法中，首先给图 G 中所有的顶点随机赋一个颜色值，然后执行一个不断改进候选解的循环。在循环的每一步迭代中，首先生成当前候选解的rep个邻居，然后从其中选择代价函数最小的一个，作为新的候选解。其中，这些邻居都是从当前候选解中随机选择一个冲突的顶点，并为其赋予一种随机颜色值而得到的（要求该操作不违反禁忌条件）。需要特别注意的是，在生成邻居时，TABUCOL 使用了 Aspiration 策略，即如果修改一个顶点的颜色后，所得到的代价函数比目前所记录的最好的代价函数还要更好，则忽略禁忌策略，并选择该候选解。最后，当退出循环之后，如果代价函数为 0，则返回图 G 的一个合法k着色，否则对图 G 的k着色失败。

（2）演化算法

演化算法是求解 NP 难组合优化问题的通用算法框架，它也是一种特殊的局部搜索算法。演化算法在求解图着色问题上也取得了非常好的效果。本小节以 1999 年提出的 MA-GH 算法为例介绍图着色问题的演化算法。该算法使用嵌入了 TABUCOL 的重组和改进操作替换了演化算法中的随机突变操作。

算法 3.2-2. (MA-GH)

Input： $G = (V, E)$

Output： valid k-coloring of G or $\emptyset$

$sp \coloneqq init(G)$;

for each $s \in sp$ **do**

    $s \coloneqq Tabucol(G, s)$;

end

While not $terminate(G, sp)$ **do**

    $(s_1, s_2) \coloneqq selectParents(G, sp)$;

    $s \coloneqq GPXrecombination(G, s_1, s_2)$;

    $s' := Tabucol(G, s)$;

    $sp \coloneqq updatePopulation(G, \mathrm{sp}, \mathrm{s}')$;

end

if $g\big(best(G, sp)\big) = 0$ **then**

    return $best(G, sp)$

else

    return $\emptyset$

算法的初始化过程采用了 k-固定的贪婪构造方法。该构造方法首先建立 k 个空的颜色集，每步选择一个未着色的点，将其放入具有最小索引的合理颜色集中。对于一个特定的点 v，颜色集 c 是合理的当且仅当将 v 放入 c 中不会产生新的冲突边。如果不存在合理的 c，则 v 保持未着色的状态不变。显然，这样的操作无法对全部的点进行着色，因此遍历完所有

的点后，需要对所有未着色的点进行独立的随机染色。

自然地，演化算法中的 TABUCOL 与 2.1 中采用相同的候选解集合，邻域关系和评估函数。不同的是，MA-GH 规定禁忌长度(tabu tenure)的设置应与当前的评价函数值有关。

正如前文所提到的，MA-GH 在生成子代时，并没有采用突变算子。当种群初始化结束后，对于种群中的每一个候选解，都会执行禁忌搜索的过程，并将搜索得到的解集作为最初的父代候选解种群，进入循环。在循环的每一步中，随机选择的两个父代候选解将通过特定的重组算子 GPX(Greedy Partition Crossover) 生成子代候选解，并再次通过 TABUCOL 进行改进。改进后的子代候选解将替换父代中较差的一个。当找到满足条件的解(冲突数为 0)，或运行时间到达上限，或种群的多样性低于设定的阈值时,算法终止。

MA-GH 算法的一项重要创新在于 GPX 算子的使用。GPX 算子的具体工作过程如下：给定两个候选解作为父代，它们都是对图 G 的颜色数为 k 的着色方案（注意：这里的候选解可以是不合法的）。交叉的过程以颜色集为单位进行，在该过程中交替地选中这两个父代候选解中的一个，并从其中选择一个顶点个数最多的颜色，将其传递到子代中，然后从父代的两个候选解中删除相应的顶点。在第 k 步执行结束之后，子代候选解中已经有了 k 种颜色，此时如果仍有一部分顶点未被着色，则分别为它们随机选择 k 个颜色中的一个。GPX 算子的优势在于，它能够将上一代的结构特征（即独立集）传递到下一代中，从而比以往的重组算子更加有效。

本章小结：

本章以图着色问题对局部搜索算法进行案例学习。第一节介绍了图着色问题的相关定义和背景知识，在第二节介绍求解图着色问题的两个局部搜索算法。第二节先介绍了图着色问题设计局部搜索算法需要考虑的关键要素，给出相关定义，接着描述了两个图着色问题的经典局部搜索方法，TABUCOL 和 MA-GH。其中第一个算法属于 Tabu Search 方法，第二个算法属于演化算法，分别代表了两种典型的局部搜索方法。

通过本章的学习，希望达到的教学目标是：能够熟练描述图着色问题的禁忌搜索和演化算法。

本章参考文献：

1. 林锦坤.大规模图着色问题的高效算法[D].北京:北京大学,2013
2. Hoos,Stützle. Stochastic Local Search: Foundations and Applications [M].2005: 468-476.
3. Rafael Martí, Panos M. Pardalos, Mauricio G. C. Resende: Handbook of Heuristics. Springer 2018, ISBN 978-3-319-07123-7
4. Sheldon M. ross. Introduction to Probability Models (11th Edition). Elsevier 2014.

第四章 图论问题的归约规则

图论（Graph theory）是组合数学的一个重要分支，它主要的研究对象为图。一个图$G(V,E)$是一个二元组，其中V是由 n 个顶点组成的集合$\{v_1,v_2,\dots,v_n\}$，称为顶点集，E是V中元素构成的无序二元组，称为边集。图通常被用于描述事物之间的二元关系，其中，顶点用于表示事物，而连接两个顶点之间的边则用于表示两个事物之间是否具有这种二元关系。

图论问题在实际生活中有着广泛的应用，其中最小顶点覆盖问题、最大团问题、图着色问题是经典的NP难组合优化问题，由于其重要性，这三个问题常出现在算法的教科书当中。另外，美国离散与应用数学会于 1992 年提出了三大难解问题，其中就有两个属于图论问题，即最大团问题和图着色问题。

近几十年来，这些图论问题由于重要的理论意义以及诸多的具体应用，吸引了许多科研工作者为其设计高效的求解算法，并取得了许多突破性的进展。这些算法主要是针对传统的基准测试图进行设计的，这些图的顶点个数往往小于五千。然而，随着互联网的快速发展，许多现实应用的图迅速增加到了百万甚至千万个顶点，这些图是传统的算法难以处理的。

为了高效地求解在大规模图上的图论问题，一种可行的方案是对图进行归约，本章介绍针对顶点覆盖问题、最大团问题以及图着色问题上的归约规则。

4.1 最小顶点覆盖问题的归约规则

我们先给出顶点覆盖问题的形式化定义。

定义 4.1-1（顶点覆盖）给定一个图$G=(V,E)$，顶点覆盖是图G顶点集的一个子集$C\in V$，使得图G的每一条边都至少有一个顶点属于C。

定义 4.1-2（最小顶点覆盖问题）给定一个图$G=(V,E)$，最小顶点覆盖问题要求找出一个具有最小基数的顶点覆盖。

如图 4.1-1 所示的两个图中，红色顶点所构成的集合是一个顶点覆盖，因为图中每一条边都至少有一个顶点是红色。同时，红色顶点所构成的集合是一个最小顶点覆盖，因为在图中找不到顶点数量更少的顶点覆盖。

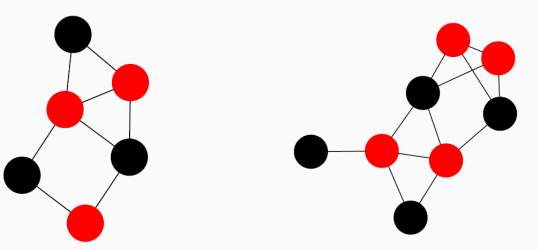

图 4.1-1 最小顶点覆盖问题

如果在一个图G中存在孤立点v，即不存在与顶点v关联的边，则任意一个最小顶点覆盖都不可能包含顶点v。因为对于任意一个包含v的顶点覆盖，将v中移除可以得到一个更小的

顶点覆盖。因此我们有下面这条归约规则。

归约规则 1（Degree-0）将图G中所有的孤立点删除。

如果一个顶点u是叶子（度数为 1），假设顶点v是它的唯一邻居。为了覆盖边$e(u,v)$，这两个顶点中的一个必须被选中。由于选中u没有任何的附加收益，相反，选择v可能覆盖更多的边。因此，选择顶点v显然更好。

归约规则 2（Degree-1）如果图G存在叶子顶点u，则将其邻居顶点v固定为最优解的一部分，并从图G中删除u和v。

证明：令集合C为图G的一个最小顶点覆盖，则顶点u和v至少有一个属于C，否则边$e(u,v)$将不能被覆盖。下面分情况讨论，证明必然存在一个最小顶点覆盖包含v但不包含u。

如果$u \in C$且$v \notin C$，则除了$e(u,v)$之外，所有边都被$C \setminus \{u\}$被覆盖，因此$C \setminus \{u\} \cup \{v\}$仍然是一个最小顶点覆盖。

如果$u \notin C$且$v \in C$，则C满足条件。

如果$u \in C$且$v \in C$，则$C \setminus \{u\}$仍然是一个顶点覆盖，与C是最小顶点覆盖矛盾。□

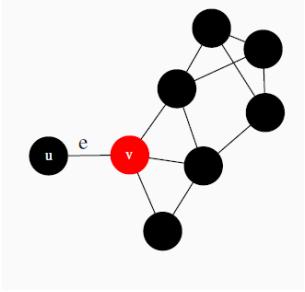

图 4.1-2 归约规则 Degree-1

对于一个度数为 2 的顶点u，其邻居为v和z，如果v和z相邻，则这三个顶点构成一个三角形，因此对于任意一个顶点覆盖，至少包含三个顶点中的两个。由于选择顶点u没有任何附加收益，因此，选择顶点v和z显然更好。

归约规则 3（Degree-2）如果图G存在一个度数为 2 的顶点u，且其邻居顶点v和z相邻，则固定v和z为最优解的一部分，并从图 G中删除u、v和z。

证明：由于最小顶点覆盖必然包含顶点u、v、z中的两个，且不可能同时包含这三个顶点（否则可以将u移除而得到一个更小的顶点覆盖）。假设存在一个包含顶点u和v，但不包含z的最小顶点覆盖C，则$C / \{u\}$覆盖除了边$e(u,z)$之外所有的边，因此$C/\{u\} \cup \{z\}$是一个最小顶点覆盖。 □

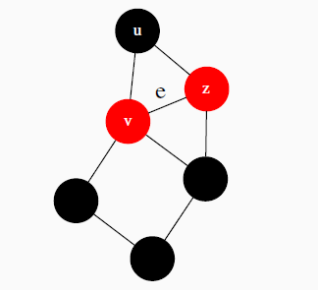

图 4.1-3 归约规则 Degree-2

归约规则 4（Domination）给定两个顶点u和v，如果$N[u] \subseteq N[v]$，则将v固定为最优解的一部分，并删除顶点v。

证明：为了覆盖边$e(u,v)$，任意一个顶点覆盖必然包含u或v，假设最小顶点覆盖C包含顶点 u 但不包含 v。因为$N[u] \subseteq N[v]$，所以从C中移除u并添加v之后，仍是一个顶点覆盖。因此，必然存在一个最小顶点覆盖包含顶点v。 □

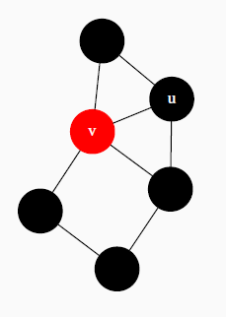

图 4.1-4 归约规则 Domination

归约规则 5（High degree）若最小顶点覆盖C的顶点数$|C| \leq k$，则固定顶点度数大于k的顶点，并从图中删除这些顶点。

证明：假设存在一个最小顶点覆盖$|C|$，不包含度数大于k的顶点u。则$\forall v \in N(u)$，必有$v \in C$，否则边$e(u,v)$将不能被覆盖。由于$|N(v)| > k$，则有$|C| > k$，与前提$|C| > k$矛盾。因此，任意顶点度数大于k的顶点必然在最小顶点覆盖中。 □

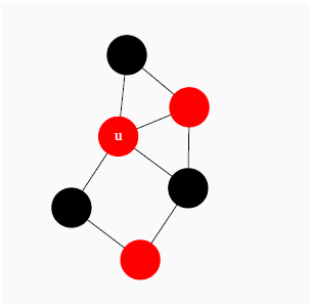

图 4.1-5 归约规则 High degree，$C \leq 3$

如图 4.1-5 所示，红色顶点构成了一个顶点覆盖，因此可以知道最小顶点覆盖的顶点数必然小于或等于 3，由于顶点u的度数为$4 > 3$，因此任意一个最小顶点覆盖必然包含顶点u，可以将其固定为解的一部分，并将其从图中删除。

为了描述最后一个归约规则，我们先定义一种成为 Crown 的图结构。

定义 4.1-3（Crown）一个 crown 是图G的一对有序顶点集(I,H)，且满足：

1) $I \neq \emptyset$是 G 的一个独立集，即I中的顶点互不邻接。
2) $H = N(I)$。
3) 在连接I和H的边中，存在一个匹配M，使得$\forall v \in H$被匹配。

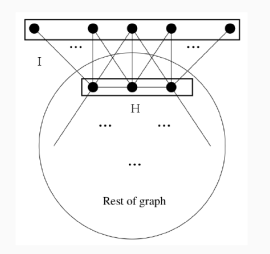

图 4.1-6 Crown

归约规则 6（Crown）给定一个 crown (I,H)，则固定 H 为最优解的一部分，删除(I,H)可以简化图 G。

证明：我们证明必然存在一个最小顶点覆盖，包含所有在H中的顶点，且不包含I中的任何顶点。为了覆盖M，任何一个顶点覆盖都必包含M中每条边的至少一个顶点，因此至少需要H个顶点。选择H中所有的顶点，并且不选择I中的任何一个顶点，可以达到这个最小值。同时由于I和H中，只有H的顶点有外部连接，这样的选择能够覆盖最多I和H外部的边。□

4.2 最大加权团问题的归约规则

我们先给出最大加权问题以及相关概念的形式化定义。

定义 4.2-1（团）给定一个图$G=(V,E)$，团是指图 G 顶点集的一个子集$C\in V$，其中的任意两个顶点都相互邻接。

如图 4.2.1 所示，在左边的图中，三个红色顶点构成图的一个团，其中每个顶点都相互邻接。根据团和顶点覆盖的定义可知，一个图G的团，是其补图$\bar{G}$的顶点覆盖。图 3.2.1 的右图为左图的补图，而红色顶点集恰好是一个顶点覆盖。

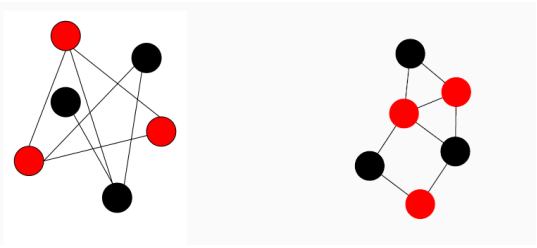

图 4.2-1 团和顶点覆盖

定义 4.2-2（团的权值）给定一个图$G=(V,E,w)$，其中w是顶点V的权值函数，使得每一个顶点v的权值为一个正整数$w(v)$。一个团C的权值定义为$w(C)=\Sigma_{v\in C}w(v)$。

定义 4.2-3（最大加权团问题）给定一个图$G=(V,E,w)$，最大加权团问题要求找出一个具有最大权值的团。

定义 4.2-4（$UB(v)$）给定一个图$G=(V,E,w)$，对于一个顶点 v，所有包含 v 的团的权值上界记作$UB(v)$，即

$$UB(v)\geq max\{w(C)|C\ is\ a\ clqiue\ of\ G,v\in C\}$$

对于一个图$G(V,E,w)$，假设我们已经找到一个加权团C，如果对于顶点v，有$UB(v)\leq w(C)$，则可以知道任意一个包含顶点v的团，都不会比C的权值大，因此我们有如下归约规

则：

归约规则 给定一个简单无向图$G(V,E,w)$，以及图G的一个团C，对于一个顶点v，如果有$UB(v) \leq w(C)$，则将顶点v从G中移除。

上述归约规则依赖于顶点v的上界$UB(v)$的计算，该上界的计算越精确，图G所归约到的图也就越小。由于包含顶点v的团，只可能是$N[v]$中的顶点，因此，一种平凡的上界计算如下：

上界$UB_0(v)$ 顶点v及其邻居的权值之和，可以作为所有包含v的团的权值上界。

$$UB_0(v) = w(v) + w(N(v))$$

为了进一步提高上界，我们从$N(v)$中选出权值最大的顶点n^*，则包含顶点v的团可以分为两种情况：不包含n^*；包含n^*。如果团不包含n^*，则其权值上界为$UB_{1a}(v) = w(v) + w(N(v) \setminus \{n^*\})$。如果团包含$n^*$，则其权值上界为$UB_{1b}(v) = w(v) + w(n^*) + w(N(v)) \cap N(n^*))$。因此，我们可以得到新的上界：

上界$UB_1(v)$ 对于包含顶点v的团，其权值上界为$UB_1(v) = \max\{UB_{1a}, Ub_{1b}\}$，即

$$UB_1(v) = w(v) + \max\{w(N(v) \setminus \{n^*\}), w(n^*) + w\big(N(v)\big) \cap N(n^*)\}$$

在先进的最大团求解算法中，图着色常被用于求解过程中。我们这里利用图着色来进一步获得更精确的上界。给定一个图，一个合法的着色是指对于图中每一个顶点都赋予一种颜色，使得任意两个相邻顶点的颜色都不相同。从而，我们有如下命题成立：

命题 4.2-1 给定一个图$G(V,E)$，一个合法着色是对V的一个划分$\{A_1, A_2, \ldots, A_k\}$，其中对于每一个$i < k$，$A_i$是一个独立集，即$A_i$中的顶点互不邻接。

由于A_i是独立集合，所以对于任意一个团，最多只能包含A_i中的一个顶点。因此，我们有如下命题成立：

命题 4.2-2 给定一个图$G(V,E,w)$，它的团权值上界为$UB_c(V) = \Sigma_{i=1}^{k} \max_{v \in A_i}\{w(v)\}$。

结合UB_1中所提到的分支，我们可以得到包含顶点v但不包含n^*的团权值上界为$UB_{2a}(v) = w(v) + UB_c(N(v) \setminus \{n^*\})$，包含$v$且包含$n^*$的团权值上界为$UB_{2b}(v) = w(v) + w(n^*) + UB_c(N(v) \cap N(n^*))$。因此，我们得到新的上界如下：

上界$UB_2(v)$ 对于包含顶点v的团，其权值上界为$UB_2(v) = max\{UB_{2a}, UB_{2b}\}$，即

$$UB_2(v) = w(v) + max\{UB_c(N(v) \setminus \{n^*\}), w(n^*) + UB_c(N(v) \cap N(n^*)\}$$

4.3 图着色问题的归约规则

在本节中，我们将处理图着色问题的归约。这里首先给出一些必要的定义。

定义 4.3-1（着色）给定一个无向图$G = (V,E)$，着色是对顶点V的颜色赋值，使得任意相邻的两个顶点的颜色不同。其中，对图G着色所需的最少颜色数，称为色数$\chi(G)$。

定义 4.3-2（图着色问题）给定一个无向图$G = (V,E)$，图着色问题要求找到所使用的颜色数最少的一个着色。

如图 4.3-1 所示，可以使用三种颜色对该图进行着色，可以看到任意相邻的两个顶点之

间的颜色都不相同。

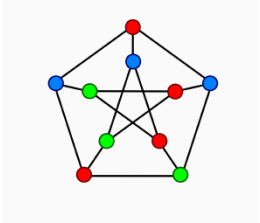

图 4.3-1 图着色

定义 4.3-3（ℓ-bound独立集）给定一个图$G=(V,E)$，独立集S是顶点V的一个子集，其中的元素互不邻接。如果$\forall v \in S$，其度数$d(v)<\ell$，则S称为图G的一个ℓ-bound独立集。

下面我们证明如下命题成立：

命题 4.3-1 给定一个图$G(V,E)$，图G的色数下界ℓ，以及图G的一个ℓ-bound独立集S，令$G'=G[V\backslash S]$，则

1） 如果$\chi(G')<\ell$，则$\chi(G)=\ell$。

2） 如果$\chi(G')\geq\ell$，则$\chi(G)=\chi(G')$。

证明：假设我们有了G'的最优着色，其所使用的颜色数为$\chi(G')$。

1） 当$\chi(G')<\ell$时，因为S是独立集，所以可以将S中的所有顶点都着色为$\chi(G')+1$，得到G的着色。因此，有$\chi(G)\leq\chi(G')+1<\ell+1$。由于$\chi(G)\geq\ell$，则$\chi(G)=\ell$。

2） 当$\chi(G')\geq\ell$时，对于一个顶点$v\in S$，有$d(v)<\ell$，所以$N(v)$所使用的颜色数$|C_{N(v)}|<\ell\leq\chi(G')$，则$v$可以被赋值为颜色$c\notin C_{N(v)}$，且$c\leq\chi(G')$。由于$S$中的顶点互不邻接，所以它们之间的着色互不影响，因此，所有在S中的顶点都可以被着色为小于等于$\chi(G')$的颜色。这样，我们从图G'的最优着色得到了图 G 的一种着色，故$\chi(G)\leq\chi(G')$，又因为G'是G的子图，因此有$\chi(G)=\chi(G')$。 □

上述命题表明，当从图G中移除一个ℓ-bound独立集之后，如果我们求得了G'的最优着色，则总是可以在线性时间复杂度内（如上述证明过程所示）得到图G的最优解，因此可以设计图着色问题的归约规则如下：

归约规则 给定一个图$G(V,E)$，及其色数下界ℓ，从图G移除一个ℓ-bound独立集S。

以上归约规则依赖于图G的色数下界ℓ，注意到对于图G的一个团C，因为C中任意两个顶点之间都是邻接的，所以它们必然都属于不同的颜色。因此，可以通过求解图G的一个团（不要求是最大团）而得到色数下界，我们有如下命题成立：

命题 4.3-2 给定一个图$G(V,E)$，以及图G的一个团C，如果$|C|=\ell$，则图G的色数$\chi(G)\geq\ell$。

值得注意的是，以上所介绍的所有归约规则，都是可以迭代使用的，即在使用一次归约规则之后，所得的子图可以被视为一个新的图而再次使用相应的规则。由于实际应用所涉及的大规模图往往具有大量的顶点，且大多非常稀疏，使得这些规则在求解相应图论问题时，常常能够取得不错的效果。

4.4 顶点覆盖问题的 Depth-Bounded 搜索树分析

本节考虑以下顶点覆盖问题：给定一个图$G(V,E)$和非负整数k，要求找出一个顶点数不大于k的顶点覆盖。对于该问题的搜索树大小的改进，已经有很长历史，目前最好的结果是$O(1.28^k)$。通过折衷考虑描述的复杂度以及上界的好坏，本节我们给出一个大小为$O(1.33^k)$的搜索树的完整介绍。主要的分析策略是“度数分支”，即基于图中顶点的度数，我们分析相应的递归调用情况（从度数为 1 到度数至少为 5）。值得注意的是，在更复杂的情况下，通过分别考虑度数为 5 和 6 的顶点，能够得到更好的搜索树大小上界。在下文中，我们不妨假设图是连通的，因为对于非连通图，我们可以完全独立地处理各个连通分量。

该算法通过如下方式递归地找到最优地顶点覆盖：给定一个图G，我们选择若干子图$G_1,\dots,G_i$，并计算它们的最优顶点覆盖，从而得到G的最优顶点覆盖。 例如，假设x是图G的一个顶点，G_1是通过删除x和所有关联边而得到的子图。这样，G_1的顶点覆盖加上x，也是图G的顶点覆盖。如果G存在一个包含x的最优顶点覆盖，则通过G_1的最优顶点覆盖，我们可以构造出图G的最优顶点覆盖。否则，如果图G不存在包含x的最优顶点覆盖，则x的所有邻居$N(x)$必然包含于最优顶点覆盖。因此，令G_2为从G中删除$N(x)$后的图，则类似地，G的顶点覆盖可以通过G_2的顶点覆盖加上$N(x)$而得到。因为G的任意一个顶点覆盖，必定包含x或$N(x)$，因此如果通过G_1和G_2的最优顶点覆盖构造G的顶点覆盖，其中必有一个是G的最优顶点覆盖，我们称之为**基于x和$N(x)$分支**。在第一个分支中，x将成为顶点覆盖的一部分，而在第二个分支中，$N(x)$将成为顶点覆盖的一部分。由于所构造的顶点覆盖随着每一步而增大，而其大小不能超过k，因此该算法终将停止，从而隐式地构造了深度上限为k的搜索树。

原则上，这就是我们算法的工作方式，但是我们选择子图$G_1,\dots,G_i$的方式更为复杂，并根据更为复杂的方式进行分支。下文中，如果一个图的所有顶点都具有相同的度数，我们称该图是正则的。如果图是连通的，则选择分支的规则如下：

1. 如果存在度数为 1 的顶点x，则选择$N(x)$作为顶点覆盖的一部分，不需要分支，因为必然存在一个包含$N(x)$且不包含x的最优顶点覆盖。
2. 如果顶点的度数至少为 5，则基于x和$N(x)$分支。
3. 如果规则 1 和 2 不适用，且图是正则的，则选择任意顶点x，并基于x和$N(x)$分支。（这种情况在搜索树的每个路径中最多出现三次，并且树的大小最多增加一个小的常数因子。）
4. 如果不存在度数为 1 或至少为 5 的顶点，但存在度数为 2 的顶点，则按照下文方案进行处理。
5. 如果为规则 1 - 4 不适用，但存在度数为 3 的顶点，则按照下文方案进行处理。

在详细介绍度数为 2 和 3 的顶点的情况之前，我们先验证以上情况是完备的，即所有可能出现的情况都已经被覆盖。这里唯一的微妙之处在于为什么我们不必单独研究度数为 4 的顶点：在第 4 种情况之后，我们知道该图至少存在一个 3 度点和一个 4 度点。 因此，第 5 种情况总是适用，所以不需要额外考虑 4 度点的情况。下面，我们给出 2 度点和 3 度点两种情况的详细描述。

2 度点 令x是一个 2 度点，其邻居为a和b，则可以分为如下三种情况。

1. 如果a和b之间存在一条边，则将a和b都加入顶点覆盖中，不需要分支。因为x，a，b形成了一个三角形，所以这三个顶点中必有两个属于顶点覆盖，而a和b所覆盖的边是其他“三选二”组合所覆盖的边的超集。
2. 如果在a和b之间没有边，但a和b的度数都为 2，且拥有除x外的共同邻居c，则将x和c加入顶点覆盖，不需要分支。因为x和 c 所覆盖的边，是从集合$\{x，a，b，c\}$中选择任何其他一对顶点所覆盖的边的超集。
3. 否则，我们有$|N(a) \cup N(b)| \geq 3$，则可以基于$N(x)$和$N(a) \cup N(b)$分支。这两个分支中至少有一个可以得到最优解：第一个分支处理的情况是x不属于最优顶点覆盖，因此所有邻居都必须是属于最优顶点覆盖。第二个分支是x属于最优顶点覆盖，这种情况下我们可以有更多推断。关键点在于我们不必搜索包含x和a，或者x和b的最优顶点覆盖，因为此时用a和b替换它们可以得到同样大小的顶点覆盖。 因此，相比于选择a和b（第一个分支）所得到的顶点覆盖，通过选择x（第二个分支）得到更小顶点覆盖的唯一可能是，选择x且不选择a和b。这意味着我们在第二个分支中选择$N(a) \cup N(b)$。因为$N(a) \cup N(b) \geq 3$，所以相应的分支向量（branching vector）至少为（2,3）。 从而分支数（branch number）小于 1.33。

3 度点 令x是一个 3 度点，其邻居为a，b和c。我们分四种情况讨论（最后一种情况不可能发生）。

1. 如果x是三角形的一个点，例如，令$\{x，a，b\}$为三角形（可能会有更多的三角形）。那么我们可以基于$N(x)$和$N(c)$分支。如果x不属于顶点覆盖，则$N(x)$必然属于。如果x属于顶点覆盖，则为了覆盖三角形上的边，a或者b必须属于顶点覆盖，此时如果c也属于顶点覆盖，则不可能得到比基于$N(x)$分支小的顶点覆盖。因为这意味着x及 2 个邻居在顶点覆盖中，由于它们所覆盖的边是$N(x)$所覆盖边的子集，所以用$N(x)$代替它们（第一个分支）可以得到同样大小的顶点覆盖。因此，要获得比第一个分支更小的顶点覆盖，唯一的机会是不选择c而选择$N(c)$，其中x属于$N(c)$。所以分支向量至少为（3,3）,从而分支数小于 1.27。
2. 假设x是长度为 4 的回路的一部分，该回路由a，x，b和d组成，这里d是一个新顶点。那么我们可以基于$N(x)$和$\{x，d\}$分支。如果x不属于顶点覆盖，则$N(x)$必然属于。如果x属于顶点覆盖，此时如果不选择d，则意味我们不得不同时选择a和b，但是这不可能得到比选择$N(x)$更小的顶点覆盖。因此，我们可以将第二个分支限制为选择$\{x，d\}$。 从而分支向量至少为（3,2），分支数则小于 1.33。
3. 假设a,b和c之间没有边，不妨设$N(a) = \{x, a_1, a_2, a_3\}$,$|N(a)| = 4$。则可以基于$N(x)$，$N(a)$和$\{a\} \cup N(b) \cup N(c)$。 如果$x$不属于顶点覆盖，则$N(x)$必然属于。如果$x$属于顶点覆盖，则可以进一步分为两种情况。如果$a$不属于顶点覆盖，则$N(a)$必然属于。如果$a$属于顶点覆盖，此时如果选中$b$或者$c$，则与之前的情况一样，不可能得到比选择$N(x)$更好的顶点覆盖。因此，第三个分支限制为选择$\{a\} \cup N(b) \cup N(c)$。分支向量至少为（3,4,6），这是因为根据前面的情况可知$N(b) \cap N(c) = \{x\}$，所以$|N(b) \cup N(c)| \geq 5$，由于$a \notin N(b)$且$a \notin N(c)$，则$|\{a\} \cup N(b) \cup N(c)| \geq 6$，分支数

小于 1.31。

4. 剩下最后一种需要考虑的情况是，存在一个 3 度点，且其所有邻居的度数也为 3。然而，由于在搜索树算法过程中，每个顶点的度都是单调递减的，并且由于正则图已经在第三个规则下处理了，因此我们总能找到一个度数为 3，且其邻居有一个 4 度点，此时可以按照上一种情况进行处理。

至此，我们分情况处理了 3 度点，并得到了完整的搜索树算法。可以看到，上面的例子展示了可以通过讨论特定的情形来避免一些细分的情况——该例子中是对于正则图的观察。通常，尤其是对于顶点覆盖，可以通过分析特定分支后会发生的情况，而在一定程度上改善分支。因为可能会出现具有特别好的分支向量的有利情况，可以利用该分支向量来改善上限。主要的思路是将两个后续的递归调用合并为一个大调用。在同样的方向上，我们找到了“迭代分支”的思想，我们试图引导分支情况的选择，并以产生比最坏情况更好的新情况为目标。更多信息请参阅相关文献。

综上所述，通过分情况讨论，我们得到了分支数为 1.33、1.27 和 1.31。可以推断得到搜索树的最坏情况上界：

定理 4.4-1 对于顶点覆盖，存在一个大小为$O(1.33^k)$搜索树。

换言之，顶点覆盖问题可以通过如上所述的分支算法求解，其中递归调用分支的次数为$O(1.33^k)$。目前针对顶点覆盖问题最优的固定参数算法（基于深度限制的搜索树）的搜索树大小为$O(1.28^k)$。

本章小节：

本章介绍了几个图优化问题的规约技术，这个三个问题分别是最小顶点覆盖问题，最大团问题，图着色问题。图规约在求解大规模图的时候，图化简技术的使用能够显著地提升求解性能，尤其是对于大规模稀疏图的求解。本章介绍的技术是比较代表性的化简技术，也都是基于一些图的基本概念进行设计的，包括利用度数，独立集等概念，容易掌握和实现。另外，本章还介绍了参数算法的经典框架，深度有界搜索树，并以顶点覆盖为例子介绍了相关技术。深度有界搜索树是以分治算法为基础，以图规约技术化简分支情况的一种搜索算法，常常能得到更好的复杂度。

通过本章的学习，希望达到的教学目标是：对于本章的三个问题，能够掌握至少两种图规约技术；对于图优化问题，了解图规约技术的有效性，并在求解大规模图问题的时候，能有意识使用图规约技术。了解深度有界搜索算法的原理。

本章参考文献：

1. Shaowei Cai, Jinkun Lin, Chuan Luo: Finding A Small Vertex Cover in Massive Sparse Graphs: Construct, Local Search, and Preprocess. J. Artif. Intell. Res. 59: 463-494 (2017)
2. Abu-Khzam, F. N., Collins, R. L., Fellows, M. R., Langston, M. A., Suters, W. H., & Symons, C. T.(2004). Kernelization algorithms for the vertex cover problem: Theory and experiments.

In Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, pp. 62–69.

3. Fedor V. Fomin, Fabrizio Grandoni, Dieter Kratsch: A measure & conquer approach for the analysis of exact algorithms. J. ACM 56(5): 25:1-25:32 (2009)
4. Shaowei Cai, Jinkun Lin: Fast Solving Maximum Weight Clique Problem in Massive Graphs. IJCAI 2016: 568-574
5. Jinkun Lin, Shaowei Cai, Chuan Luo, Kaile Su: A Reduction based Method for Coloring Very Large Graphs. IJCAI 2017: 517-523