2 组合优化问题和启发式算法综述

2.1 组合优化问题的定义

一般来讲，最优化问题有三个基本要素：变量、约束和目标函数[1,2]。在求解过程中选定的基本参数称为变量，对变量取值的各种限制称为约束，表示可行方案衡量标准的函数称为目标函数。最优化问题按照其变量的属性可以大致分为两大类：第一类是关于连续变量的优化问题，称为连续优化。第二类是关于离散变量的优化问题，称为离散优化或组合优化。在连续优化问题中，通常的目标是求解一个函数的最大值或最小值。在组合优化问题中，其目标是寻找离散事件的最优编排、分组、次序或筛选等。它需要从一个有限集合中寻找一个目标对象，通常是一个整数、集合的子集、排列或者图结构。本文为了深入研究组合优化问题，给出了组合优化问题的相关概念和形式化定义。

定义 2.1： 组合优化问题$P = (X, D, G, f)$可以定义为：

- 变量的集合$X = \{x_1, \ldots, x_n\}$；
- 变量的定义域集合$D = \{d_1, \ldots, d_n\}$；
- 变量的约束集合G；
- 最小化①的目标函数f，其中$f : d_1 \times \ldots \times d_n \to \mathbb{R}$；

变量的所有可能取值集合为：

$S = \{s = \{x_1, \ldots, x_n\} | x_i \in d_i, i = 1, \ldots, n; s$满足$G$中所有约束$\}$.

其中，n表示变量的个数，d_i是对应的变量x_i的定义域。集合S通常被称为问题的解空间，S中的每一个元素均是问题的合法解。组合优化问题的目标是在解空间中找到一个解$s^* \in S$，使其目标函数值f 最小，即$f(s^*) \leqslant f(s), \forall s \in S$，则$s^*$称为优化问题$P$的全局最优解。所有全局最优解构成的集合$S^* = \{s^* | s^* \in S, \forall s \in S, f(s^*) \leqslant f(s)\}$ 称为全局最优解集合。

① 最大化问题的目标函数f等价于最小化目标函数$-f$。为了不失一般性，在本文中，我们将组合优化问题统一定义为最小化问题。

求解组合优化问题的方法大致分为两大类：完备算法和近似算法。完备算法能在有限的时间内找到有限规模组合优化问题的最优解[1,2]。由于大多数组合优化问题都是NP难度的[3]，因此，学术界至今也没有找到能够有效求解组合优化问题的多项式时间的算法。完备算法在最坏情况下，例如求解大规模问题时，所需时间是指数级别的，在实际求解过程中不具备可操作性。于是，近似算法的研究受到了越来越多的重视。近似算法是以牺牲解的优度为代价，力求在合理的计算时间内求出最优解或近似最优解。

启发式算法是根据直观感觉或经验进行构造或依据特定规则进行局部搜索的算法[4]。它能在可以接受的时间和空间范围内求出问题的可行解，但又不保证求得最优解，即不保证解的最优性，无法确定可行解与最优解的偏离程度。因此，启发式算法是一种近似算法。局部搜索算法是从一个初始解出发，在当前解的邻域结构中找到一个更好的解代替当前解，并反复不断迭代此过程，直到找到局部最优解或全局最优解。这里出现了邻域结构、局部最优解和全局最优解的概念，分别定义如下：

定义 2.2： **邻域结构**是一个函数映射，$N: S \rightarrow 2^S$，其中2^S表示S的所有子集的集合。邻域结构是将S中的每一个解s映射到解空间S的一个子集$N(s) \subseteq S$ 上，$N(s)$表示解s的邻域结构。

定义 2.3： **局部最优解**是与邻域结构相关的概念。如果一个解s'满足$\forall s \in N(s'), f(s') \leqslant f(s)$，则$s'$是关于邻域结构$N(s')$中的局部最优解。

定义 2.4： **全局最优解**是与问题解空间相关的概念。如果一个解s^*满足$\forall s \in S, f(s^*) \leqslant f(s)$，则$s^*$是解空间$S$中的全局最优解。

由上面的定义可知，如果在一个解的邻域结构中找不到更好的解，这个解就是局部最优解；而全局最优解是解空间中最好的解。由于局部最优解是与邻域结构相关的，因此，不同的邻域结构对应的局部最优解也不同。一般而言，局部最优解不一定是全局最优解，而全局最优解一定是关于某一个邻域结构的局部最优解。换言之，全局最优解是最好的局部最优解。

启发式算法在寻求最优解的过程中能够根据个体或者全局的经验来改变其搜索路径，当寻求问题的最优解变得不可能或者很难完成时，启发式算法就是一种高效的获得可行解的办法。根据自然界的某些现象或者物理过程，例如生物界中的遗传进化现象，物理中固体物质的退火过程等，将启发式算法加入到一些先进的框架中，得到搜索能力更强的算法，这种算法一般称为元启发式算法[5,6]。为了便于描述，在后面的章节中，我们将启发式算法和元启发式算法统一称为启发式算法。接下来我们将重点讨论启发式算法的分类和各类常见的启发式算法。

2.2 启发式算法的分类

启发式算法可以依据选取的特征和研究的角度的不同进行不同的分类。本文主要从是否从自然界获取灵感、解的数量、目标函数是否动态变化、邻域结构的数量以及是否重点依赖历史信息的角度对启发式算法进行分类：

- 是否从自然界获取灵感　这是依据启发式算法的设计初衷来进行分类的标准。学术界提出的很多算法都是受自然界中的生物现象，群体过程，物理现象等启发而来的。例如遗传算法[7]和模拟退火算法[8]等。而其他一类算法，例如禁忌搜索算法和迭代局部搜索算法等，从狭隘的角度来讲，不是从自然界获取灵感得来的。这有两方面的原因：首先，许多混合型的算法，由于具有本身固有的理论之外，还模拟了众多的生物现象或物理过程等，因此并不单纯属于某一类。其次，某些算法使用的策略，很难判断是否从自然界获取灵感。例如，禁忌搜索算法[9]中的禁忌表可以认为是模拟人脑的短期记忆功能，但禁忌搜索算法的原理却通常被认为不是源于自然界的灵感。
- 解的数量　从搜索的对象即解的数量出发，可以将启发式算法分为基于单一解的启发式算法和基于种群的启发式算法。在搜索过程中，如果算法始终只作用在单一解上，则称这种算法为基于单一解的启发式算法，或者称为轨道算法（Trajectory method）；如果算法以多个个体的集合为起点，作用在一个种群上，则称这种算法为基于种群的启发式算法。例如禁忌搜索算法、模拟退火算法和变邻域搜索算法等，都属于基于单一解的启发式算法；而进化算法[10]，粒子群算法[11]等，则属于基于种群的启发式算法。

- 目标函数是否动态变化　启发式算法可以根据目标函数的特征进行分类。大多数启发式算法的目标函数就是问题本身的目标函数，而其他的一些算法，例如引导式局部搜索算法[12]，在搜索过程中动态的改变目标函数。这种方法的原理是通过改变算法的搜索路径来跳出局部最优陷阱。
- 邻域结构的数量　对于同一个问题，不同的邻域结构对应着不同的适应度地形图[13]（Fitness landscape）。大多数启发式算法只作用在一个邻域结构上，因此，在搜索过程中，其适应度地形图保持不变；而其他一些算法，例如变邻域搜索算法[14]，使用多个邻域结构，并在搜索过程中不断改变邻域结构来增加算法的疏散性。
- 是否重点依赖于历史信息　是否重点依赖历史信息也是启发式算法的一个重要特征。不依赖或少依赖历史信息的算法，其每一步的动作是通过当前状态决定的。历史信息通常保存在内存中，重点依赖历史信息的算法也可以分为长期的内存使用和短期的内存使用。前者通常是记录最近做过的动作，访问过的解等。后者则主要是记录和统计算法在整个搜索过程中的一些特征。

2.3　各类常见的启发式算法

把启发式算法按照解的数量来分类，可以对算法的结构做深入的分析和研究。不仅如此，当前学术界研究的趋势之一是将单一解和基于种群的启发式算法结合起来，设计性能更加优越的混合型启发式算法。因此，在本章节中，我们按照从基于单一解和种群这两类算法出发，对各种常见的启发式算法进行详细介绍。

2.3.1　局部搜索算法

局部搜索算法(Local Search, LS)是研究最早的一类启发式算法。该算法从初始解出发，在当前解的邻域中，只要发现了比当前解更好的解，就用该解替换掉当前解，并反复迭代，直到在当前解的邻域内找不到更好的解，即当前解为局部最优解，算法停止运行。基于这样的特点，基本的局部搜索算法又称为迭代改进算法。算法1给出了局部搜索算法的伪代码描述。

在算法1中，函数*Improve(N(s))*的功能是从邻域结构$N(s)$中选取一个邻域解来替

Algorithm 1 局部搜索算法的伪代码描述

1: $s \leftarrow$ Generate_Initial_Solution()
2: **while** improvement is still possible **do**
3: $s \leftarrow$ Improve($N(s)$)
4: **end while**

换当前解。选取策略可以是首次改进、最好改进或介于两者之间的规则。首次改进策略是按一定的顺序依次遍历邻域结构，一旦发现某个解比当前解好，就选择这个解。最好改进策略是遍历完整个邻域结构，从中选取一个目标函数值最小的解。每次迭代中，都从邻域结构中选取一个解替换当前解，重复此过程直到遇到局部最优解。由此可见，局部搜索算法的性能很大程度上依赖于邻域结构的定义和邻域解的选取策略。一般来讲，局部搜索结束后，算法陷入了局部最优的陷阱，因此，单纯的局部搜索算法求解组合优化问题的效果并不好，需要结合其他一些策略来使算法跳出局部最优陷阱。

2.3.2 模拟退火算法

模拟退火算法（Simulated Annealing, SA）源于物理现象中固体退火的原理[15,16]。它是模拟将固体充分加温，再让其逐渐冷却的过程。加温时，固体内部粒子随温升变为无序状，内能增大，而逐渐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。用固体退火模拟组合优化问题，将内能模拟为目标函数值，温度转化成控制参数，即得到求解组合优化问题的模拟退火算法[15,16]。模拟退火算法的基本原理是以一定的概率接受质量较差的解，并且这种概率在搜索过程中有规律的逐步减小。算法2给出了模拟退火算法的伪代码描述。

算法2从一个初始解s和初始温度T_0开始进行搜索的。接下来在每一次迭代过程中，从邻域结构中随机选取一个解$s' \in N(s)$，并根据$f(s)$，$f(s')$和当前温度T来决定是否被接受。当$f(s') < f(s)$时，将s替换为s'。否则，当$f(s') \geqslant f(s)$时，以一定的概率将s替换为s'。这一概率由当前温度T和$f(s') - f(s)$来决定，通常用玻尔兹曼分布$exp(-\frac{f(s')-f(s)}{T})$计算而来。

在搜索过程中，当前温度是逐渐降低的。因此，在搜索的前期阶段，算法以较

Algorithm 2 模拟退火算法的伪代码描述

```
s ← Generate_Initial_Solution()
T ← T_0
while termination conditions not met do
    s ←Improve(N(s))
    s' ← Pick_At_Random(N(s))
    if f(s') < f(s) then
        s ← s'
    else
        Accept s' as new solution with probability p(T, s, s')
    end if
    Update(T)
end while
```

大的概率接受随机解，而在后期阶段，算法逐渐收敛为迭代改进算法。这一过程就是模拟金属的退火冷却过程。退火过程由冷却进度表控制，包括控制参数的初值T_0，每个T值时的迭代次数和停止条件。冷却进度表的控制是影响模拟退火算法性能的决定性因素。一些较好的控制冷却进度表的方法在文献[17,18]中有详细描述，在此不展开讨论。算法停止条件的设置比较灵活，方法多种多样。常见的停止条件包括最大的运行时间、最大的迭代次数、目标函数达到了预先设置的阈值或者最大连续没有改进解的迭代次数等。本章中其他启发式算法的停止条件与此相同。学术界的研究表明，模拟退火算法在求解排课表问题[19]、车辆路由问题[20]、并行机调度[21]等经典组合优化问题上具有突出的性能。

2.3.3 禁忌搜索算法

禁忌搜索算法(Tabu Search, TS)是求解组合优化问题中使用最多的启发式算法之一。在1986年由Fred Glover[5,9,22]提出后，禁忌搜索算法便被广大学者运用于求解各类组合优化问题，并取得了十分显著的优异成果。禁忌搜索算法是借助搜索过程中保留的历史信息来跳出局部最优陷阱并进行集中性搜索的。禁忌搜索算法采用禁忌

表来保存已经做过的动作或访问的解，并禁忌这些动作，使它们在接下来的一段时间或迭代次数内不重复出现。因此，禁忌搜索算法是通过禁忌掉之前已经遍历过的解来避免迂回搜索，从而跳出局部最优解的陷阱，实现全局寻优能力。算法3给出了基本的禁忌搜索算法的伪代码描述。

Algorithm 3 禁忌搜索算法的伪代码描述

1: $s \leftarrow$ Generate_Initial_Solution()
2: $tabu\_list \leftarrow \emptyset$
3: **while** termination conditions not met **do**
4: $s^{'} \leftarrow$ Choose_Best($N(s) \setminus tabu\_list$)
5: Update($tabu\_list$)
6: **end while**

算法3中采用了最好改进的局部搜索策略来进行深入搜索，并使用一部分特定的内存区域来帮助算法跳出局部最优陷阱，避免绕圈或迂回搜索。算法所需的内存区域是以禁忌表是形式使用的，它用来保存最近访问过的一部分解或最近做过的一部分动作，并标记为禁忌状态，避免算法在接下来的一段时间或迭代次数内重新访问这些被禁忌的解或执行被禁忌的动作。因此，当前解的邻域结构便限制为非禁忌的解，这部分解称为合法候选邻域。每一次迭代中算法均从合法候选邻域中选取最好的解替换为当前解，并将这个解或动作加入到禁忌表中。此外，将禁忌表中满足条件的一个解移出来，通常根据先进先出的原则，将保存时间最久的解移出来。由于这种邻域解的动态限制机制，禁忌搜索算法可以看成是一种动态邻域搜索算法[23]。

禁忌表是用来阻止算法在一段时间内重复访问的已访问过的解或执行过的动作，其长度称为禁忌步长。禁忌步长决定了搜索过程中记录信息量的大小。禁忌步长越大，禁忌的解越多，合法候选邻域就越小，算法搜索的空间就相对较小。相反，禁忌步长越小，算法搜索的空间就越大。为了增加算法的鲁棒性，通常使禁忌步长在一定范围内随机或者按照特定的策略变化。文献[24–26] 均对禁忌步长变化的时机和策略做了深入研究，文献[22]也介绍了一些智能的动态控制禁忌步长的方法。其中一个较为常见且易于实现的策略是，当发现解重复出现的概率较大，即需要引入一定的疏散性时，增加禁忌步长；而当发现较长时间没法改进解，这时需要增强算法的集

中性，便减小禁忌步长。

实际编程过程中，用禁忌表来保存算法访问过的完整的解需要消耗大量的计算资源，因此，这种方法的可操作性不强，是不可取的。通常算法禁忌的是解的一些属性而非解本身。这些属性可以是构成解的关键元素、邻域结构的动作等。相比整个解而言，这些属性更为简单，在保存、查找和对比的过程中需要的时间和空间都更小，效率更高。然而，这种方法的缺点是丢失了解的一些信息，因为禁忌一个属性意味着包含这个属性的所有解都被禁忌掉了，而这些解中不乏有一些质量较高的解，它们就被排除在合法候选邻域之外。解禁策略[22]能够克服这一缺点，它将符合条件的处于禁忌状态的解或动作移出禁忌表，提前结束其禁忌状态。通常使用的解禁条件是某个解比当前邻域中所有的解都好，而且还比搜寻到的历史最优解更好。

如果存在满足解禁策略的解，则算法优先从中选择一个最好的解。如果不存在，则从处于非禁忌状态的解中选择一个最好的解。值得注意的是，禁忌搜索算法和局部搜索算法最大差别在于：局部搜索算法总是接受比当前解更优的解，而禁忌搜索允许接受比当前解更差的解，因为禁忌搜索算法只在合法候选邻域里选择解，这个邻域是整个邻域结构的一个子集。正是由于禁忌搜索算法具有此特性，才使得它具备跳出局部最优解陷阱的能力，拥有比局部搜索算法更强的全局搜索能力。

禁忌搜索的应用十分广泛，它能有效的求解各类组合优化问题，最新的应用包括车间作业调度问题[27,28]、最大覆盖问题[29]、车辆路由问题[30]。并且，禁忌搜索的应用已经扩展到机器学习和神经网络[31]等人工智能领域。

2.3.4 进化算法

进化算法（Evolutionary Computation, EC）是一种基于种群的启发式优化算法[10,32,33]。进化算法产生的灵感借鉴了大自然中生物的进化过程，一般包括基因编码、种群初始化、进化过程中的选择、交叉和变异算符，种群更新和保留机制等基本操作过程[34,35]。由于具备一定的鲁棒性和自适应性，进化算法被用来求解各种复杂的组合优化问题，尤其在求解多目标优化问题上具有优越的性能[36,37]。种群初始化之后，算法便开始模拟生物的进化过程。在进化过程的每一代，算法对种群中的个体进行一系列的进化操作后，产生新的子代个体。通常由两个或多个个体产生新个体的操作称为交叉算符，而从单个解进行自适应的进化操作称为变异或突变算符。

进化算法中进化的动力来自于基于个体适应度的选择策略，它优先选择适应度强的个体，这里的个体适应度可以是解的目标函数值或解的质量的某种度量。总体而言，其遵循的原则是优胜劣汰，将适应度强的个体选择出来进行各种进化操作，把这些解中的一些优良基因或品质遗传给下一代。算法4给出了进化算法的基本框架。

Algorithm 4 进化算法的伪代码描述

1: $P \leftarrow$ Generate_Initial_Population()
2: Evaluation(P)
3: **while** termination conditions not met **do**
4: $P' \leftarrow$ Recombination(P)
5: $P'' \leftarrow$ Mutation(P)
6: Evaluation(P'')
7: $P \leftarrow$ Selection($P \cup P''$)
8: **end while**

在算法4中，P代表由个体组成的种群。从初始种群出发，算法对初始种群进行评估。之后的每一代中，算法对种群中的个体进行交叉和变异操作，产生一个新的种群P''，并对这个新种群进行评估。新种群和上一代的老种群合并后，通过选择策略选出适应度强的个体组成新的当前种群。算法不断重复这一过程，直到满足停止条件。下面具体介绍进化算法中的几个重要特征：

- **个体的编码和解码：**进化算法是作用在由个体组成的种群上的。这些个体并不一定是问题的解本身，它需要符合生物进化过程中的规律，可以是解的集合或者是一种能够转化为解结构的表达式。因此，种群中的个体通常需要进行特定的描述和表达，这就是个体的编码与解码[38]。组合优化问题中使用最多最常见的个体编码方案是位串和自然数的排列。在进化算法中，对解的表达称为基因编码。种群中的每一个个体都有一个解与之对应。这样做的目的是为了区别解的表达和解本身，使进化算法更具备生物学上的意义。
- **进化过程：**每一次迭代中，当通过进化操作产生新种群后，算法通过选择策略来决定新种群中的哪些个体进入下一轮迭代的种群中。选择策略是进化算法的重要环节，它通常是对种群进行评估后，根据特定的规则选择适应度较

强的个体[39]。许多进化算法的种群大小保持不变，其目的是为了保证最好的个体解一直在当前种群中，并使得这些个体的特征稳定的继承下去。种群的大小也可以是变化的，在种群数量逐渐减小的进化算法中，当种群中仅剩下一个个体时，算法便停止运行。

- **新个体的生成：**进化算法中新个体主要是通过交叉和变异等进化操作产生的。最常见的交叉算符是作用在两个父代个体上，而变异算法作用在单一个体上。当然，不论是交叉还是变异算符，它们都可以作用在一个或多个特体。例如多父代个体的交叉算符[40]以及基于基因池的交叉算符[41]。
- **集中性机制：**在实际应用过程中，使用改进型算法对个体进行集中性搜索，提高种群中个体的适应度，能够大大提高进化算法的整体搜索效率。增强进化算法的集中性搜索能力的方法有很多，其中之一是在进化算法的每一代中，用局部搜索算法优化种群的每个个体，这种算法通常称为文法基因算法[42,43]。另一种方法是通过交叉算符将每个个体的好的基因结合起来，形成新的更好的个体，这种技术通常称为交叉学习策略[44–48]。
- **疏散性机制：**进化算法中的重点和难点是如何有效避免种群趋于早熟。种群趋于早熟是指种群中的每个个体均过早的收敛为质量较高的解，且个体之间的差异变得越来越小，种群的多样性变小。为了更好的模拟生物进化的原理，维持种群的多样性，学术界提出很多策略[33,40,49–51]。变异算符是避免种群过早收敛的重要方法。最简单的变异算符是对个体进行一定程度的扰动[52]。适应度共享[53]是另一种维持种群多样新的策略，它依据基因是相似度适量的减小种群中个体的适应度，使相似的个体进入下一次进化过程的概率变小，从而使个体之间保持一定的差异。

2.4 本章小结

本章给出了组合优化问题的描述以及相关概念的严格定义，并重点概述了启发式优化的基本理论，对启发式算法进行分类，并介绍了几种经典的启发式算法的核心思想和基本步骤。

第一节概述了组合优化问题，给出了组合优化问题的形式化定义。此外，概述

了求解组合优化问题的一般方法。

第二节从不同的角度和特征对各类启发式算法进行分类，并分析了它们的相关性。

第三节从解的数量的角度，对一些常见的启发式算法做了详细的介绍，描述了相关的基本概念、算法原理、算法框架和主要特征。