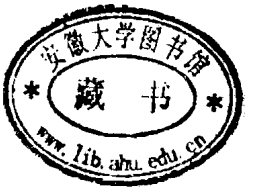
计算机难解问题的骨架理论与应用

江 贺　胡 燕　李明楚/著

科学出版社

计算机难解问题的骨架理论与应用

江 贺 胡 燕 李明楚 著



科学出版社

北 京

内 容 简 介

骨架理论是有效解决规模日益扩大的计算机难解问题的新途径，是当前智能计算领域的研究热点之一。

本书主要介绍面向计算机难解问题的骨架特征的挖掘及其算法设计。本书首先介绍了计算复杂性理论，并简要归纳了经典启发式算法及超启发式算法。在此基础上，本书重点阐述了骨架的概念，并归纳了骨架与计算复杂性理论的关系，深入介绍了如何分析骨架的计算复杂性。随后，本书介绍了获取骨架的有效方法，并系统地总结了现有的各种基于骨架的算法。为了便于运用本书阐述的算法，书后附有部分算法的源程序。

本书可供理工科大学计算机、软件工程和人工智能等专业的教师及研究生阅读，也可供自然科学和工程领域中的研究人员参考。

图书在版编目(CIP)数据

计算机难解问题的骨架理论与应用/江贺，胡燕，李明楚著. —北京：科学出版社，2013

ISBN 978-7-03-035846-2

Ⅰ.①计… Ⅱ.①江… ②胡… ③李… Ⅲ.①计算机算法-研究 Ⅳ.①TP301.6

中国版本图书馆 CIP 数据核字(2012)第 250233 号

责任编辑：刘宝莉 / 责任校对：郭瑞芝
责任印制：张 倩 / 封面设计：陈 敬

科 学 出 版 社 出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

骏 走 印 刷 厂 印刷

科学出版社发行 各地新华书店经销

*

2013 年 1 月第 一 版 开本：B5(720×1000)
2013 年 1 月第一次印刷 印张：13 1/2
字数：257 000

定价：60.00 元

(如有印装质量问题，我社负责调换)

前　言

在工业控制、网络规划和 VLSI 等应用领域存在大量 NP-难解问题。由于获取这类问题精确解的算法(最坏)时间复杂度为指数级,难以在实际应用中运用。因此,众多国内外学者研究重点在于如何在较短时间内获取高质量近似解,并由此提出了多种类型的启发式算法。面对应用领域的飞速发展(比如 VLSI 设计中出现的实例规模已经达到 10^9 量级),已有的启发式算法性能难以适应 NP-难解问题的规模扩张,成为该领域的发展瓶颈。研究人员逐渐认识到:解决这类组合爆炸问题的一条根本出路在于深入理解所求解问题的本身特征。骨架作为描述 NP-难解问题内在特征的新途径,开始被广泛应用于启发式算法设计,形成了当前智能计算领域的研究热点。

研究骨架对于启发式算法具有重要意义。首先,基于骨架的启发式算法容易实现。骨架提供了一种新的框架,能十分容易与现有启发式算法相互结合,从而给出新的启发式算法。其次,基于骨架的启发式算法性能提升显著。基于骨架的启发式算法可以对实例的搜索空间进行有效收缩,从而显著提高搜索的效率。例如,利用骨架可以将旅行商问题的实例转化成规模为原实例 30%～40%的新实例,所获得的多级归约算法比循环 LK 算法在求解质量(即解与全局最优解的差别)上提升了 30%～60%。

本书围绕着骨架在启发式算法设计中的功能,介绍了骨架所处理的 NP-难解问题的计算复杂性理论、常见的求解 NP-难解问题的启发式算法及超启发式算法、骨架与计算复杂性相关理论(如相变、后门等)的关系、完整或者部分骨架的计算复杂性分析方法、用于获取部分或近似骨架的有效方法、各种基于骨架的算法等。在本书的各个章节,注重实际 NP-难解问题的求解。本书的主要章节均以实际问题为例,给出相关理论或算法的具体应用技巧。

全书共七章,其中前两章为背景知识部分;第三章、第四章和第五章为本书的核心内容,介绍了骨架的计算复杂性分析方法、骨架的获取方法及基于骨架的算法设计方法;第六章以三个具体问题为例,给出了骨架在 NP-难解问题求解过程中的完整示例;第七章则介绍了与骨架相关的概念的研究概况。为了便于读者深入研究,每章均列出了相关历史文献。本书最后还给出了部分代码,供感兴趣的读者结合相关内容进行分析研究。

本书第一章简要介绍计算机难解问题的基本概念及一些经典的难解问题。首先以一个简单的周游全国城市的例子,给出计算机难解问题的具体应用。其次,介

绍计算复杂性理论的一些基本概念，包括问题与实例的概念、多项式时间与指数时间算法、P 类与 NP 类问题等。最后，较为详细地介绍本书中频繁使用到的经典的难解问题，包括旅行商问题、二次分配问题和 p-中位问题。

第二章介绍目前求解 NP-难解问题的主流启发式算法和超启发式算法。启发式算法关注的是如何在较短的时间（通常是多项式时间复杂度）内，获得与全局最优解的目标函数值相近的高质量解。超启发式算法提供了一种高层框架，通过调用底层启发式算子来实现问题的求解。超启发式算法侧重于如何将应用领域知识屏蔽在高层框架以外。

第三章主要介绍骨架的计算复杂性理论。该章首先介绍了骨架与复杂系统的相变之间的关联；其次，分析了骨架与 NP-难解问题的后门之间的关系；最后，给出了分析骨架的计算复杂性的一般性方法。作为示例，分别给出了图的二划分问题、p-中位问题和加权 Max-SAT 问题的骨架分析过程。

第四章归纳了现有的几种骨架获取方法。其中，限界交叉方法是一种精确获取骨架的方法，可以在理论上保证所获得的骨架的纯度。限界交叉方法所获得的骨架数量较少，所需运行时间也较多。针对这种困难，加速的限界交叉方法采取了一种折中的方法，考虑如何更快获得骨架。该章也重点介绍基于地貌分析的局部最优解近似方法，该方法可以在较短的时间内快速获得大量的近似骨架。

第五章主要介绍如何利用骨架进行高效启发式算法设计。通过固定骨架或者近似骨架，可以对待求解问题的搜索空间进行限制，从而在更小的搜索空间中进行高效求解。根据骨架在限制搜索空间中所起的作用不同，基于骨架的启发式算法可以进一步分为基于实例归约的骨架算法和基于初始解构造的骨架算法两类。

第六章以几个典型 NP-难解问题为例，给出如何利用骨架进行问题求解的完整示例。该章选用了两个经典的组合优化问题，即二次分配问题和图的划分问题，以及软件工程领域的应用问题，即软件的下一版本发布问题。

第七章介绍与骨架相对应的脂肪和肌肉的概念。脂肪指一个 NP-难解问题的实例中不在任何全局最优解出现的成分，其研究重点在于如何剔除实例中的脂肪，以缩小实例的搜索空间。肌肉则定义为一个 NP-难解问题的实例中所有全局最优解的并集，其研究重点在于如何获取近似肌肉，并在近似肌肉中进行高效搜索。

本书以作者多年来从事智能计算研究工作为基础，在这个过程中得到了很多人的帮助，在此向大家致以衷心的感谢！首先感谢众多师长和合作者，是他们的长期支持和不断鼓励，使得作者能持续在本领域开展研究；其次，感谢大连理工大学软件学院的领导和同事们提供了一个宽松的工作环境，并在书稿撰写过程中提出了很多建设性意见；同时，作者感谢玄跻峰、任志磊、聂黎明、屈世超、王润青、潘兴亮、蔡景媛等同学在本书成稿过程中完成的计算机编辑和绘图任务。最后，作者感谢家人在本书撰写过程中的支持，没有他们的付出，本书的完成是不可能的。本书

的出版获得了国家自然科学基金项目(60805024、61033012、61175062)和中央高校基本科研业务费专项资金的资助,在此一并表示感谢!

由于作者水平有限,书中难免存在不足之处,恳请读者不吝批评指正。所有关于本书的意见,请发送电子邮件到 jianghe@dlut. edu. cn,我们相信在和读者交流的过程中能有所裨益。

目　录

第一章　计算机难解问题与计算复杂性理论

本章简要介绍计算机难解问题的基本概念及一些经典的难解问题。首先，以一个简单的周游全国大城市的例子，给出计算机难解问题的一个具体应用。其次，介绍计算复杂性理论（computational complexity theory）的一些基本概念，包括问题与实例的概念、多项式时间与指数时间算法、P 类与 NP 类问题等。最后，较为详细地介绍本书中频繁使用到的几个经典的难解问题。

1.1　现实世界中的难解问题

在算法设计或者数据结构的很多知识体系中，存在很多有趣的问题，比如串的匹配问题、二叉树的查找问题等。这些问题均可以在较短的时间（多项式时间）内快速求解。然而，在现实世界中，还存在一大类问题，它们的结构或简单或复杂，但要获得这些问题的精确解却是极其困难的。

一个简单的例子是：一位大学生规划在毕业后周游全国主要的大城市，比如北京、上海、天津、重庆、南京、长沙、合肥、大连。他计划从大连出发，经过上述每个城市一次，最后返回大连。现在的问题是：如何找到一条最合理的旅游路线，使得这位大学生的旅费最少？在这里，假设该大学生经过前期的调研，已经把各个城市之间直达的旅费调研清楚了。这个问题是一个非常有趣的问题，也非常简单易懂，在 5min 内就可以向任何一位小学生讲清楚。但是，如何获得该问题的精确解，却不像该问题的定义那么简单。可以用一个排列来表示解，比如大连→北京→天津→重庆→长沙→合肥→上海→南京→大连。一种直观的精确求解方法是穷举法，把所有的可能性都试一次，然后挑出最好的解。如果用这种求解方法，则共有 7！种可能性（注意到是固定地从大连出发，又回到大连）。对于少量的城市数而言，这种穷举法是可行的，但是当城市数增加为 100 甚至更多时，需要尝试的解的个数则迅速增加而难以逐一枚举。另一种直观方法是采用贪心法，即从大连出发，每次挑选距离当前城市最近且没有访问过的城市作为下一个访问的城市，直到回到大连为止。这种方法显然速度比穷举法快得多，然而该方法是一种“鼠目寸光”的方法，并不能确保获得的是精确解，通过反例法很容易证实。

实际上，上述例子就是经典的组合优化问题——旅行商问题（traveling salesman problem，TSP）的具体应用。在上述例子中，两个城市之间的旅费，也可以换成是城市间的距离、旅行时间等。而 TSP 问题的应用背景实际是十分广泛的，在

交通调度、线路板设计、工业控制等诸多领域都能找到它的应用例子。以线路板设计为例，如何在线路板上放置若干个元器件，使得它们之间存在一条最短环路，以节省印制电路的成本？将元器件映射为城市，则该线路板设计问题就转化为 TSP 问题。需要注意的是，这个应用中城市的数量（通常也称为问题的规模）是成千上万的，采用穷举法，即便在目前的天河、蓝光等超级计算机上的运行时间也是难以接受的。

有趣的是，TSP 问题经过一些变换后，依然是很难求解的。比如，假设有些城市之间距离为无穷大（即两个城市间没有通车），那能否找到一条通过每个城市一次且回到出发地的环路？该问题实际上是另外一个难解问题，即哈密尔顿环路问题（Hamiltonian cycle problem）。另外一种针对 TSP 问题的变换是引入额外的目标函数，比如除了要求最少的旅费之外，还希望旅游线路的旅行时间最少。这实际上又将 TSP 问题转化为双目标优化问题。此时，精确解不再是单个解，而是一个解集。同时，由于两个目标之间可能相互冲突，因此这个解集通常被称为 Pareto 解集，即该解集中的任何一个解都不会比集合中其他解在两个目标上均更优。这种双目标优化问题通常比单个目标函数的问题更加困难。

以上讨论的种种问题，都具有一些共同的特征，比如在多项式时间内难以精确求解，而若设计一些多项式时间复杂度的算法，通常只能给出一些近似的解。由于这类问题在实际应用中数量极其庞大，有必要在计算机科学及人工智能领域展开专门研究。

1.2 P 与 NP

本节将讨论 P 和 NP 的概念，它们是计算复杂性理论的基础定义，而计算复杂性理论则是整个计算机科学的基础。为了介绍 P 和 NP 的含义，本节将首先给出问题和实例的定义，然后再定义多项式时间和指数时间算法的概念。

1.2.1 问题与实例

所谓问题，是需要解答的一般性的疑问，它通常包含一系列的参数（或者自由变量），并且这些参数的值没有固定。通常来说，一个问题的定义应该是给定：①各种参数的定义及约束；②解应该满足的条件。当一个问题的具体参数都指定后，就可以获得一个实例。在很多情况下所说的问题求解，实际上是给出某种算法，以求解该问题的各种实例。

下面，将以 TSP 问题为例，给出其定义及实例。TSP 问题的严格定义如下：给定加权图 $G=(V,E,w)$，V 为顶点集，$|V|=n$，E 为边集，$w:E\to R^+$ 为边权函数，G 中一条环路就是一条不重复地访问 V 中所有顶点的哈密尔顿环路，记为 $s=\langle v_1,$

$v_2,\cdots,v_n\rangle$，其中 $v_i\neq v_j(\forall 1\leqslant i,j\leqslant n,i\neq j)$ 且 $\langle v_i,v_{(i \bmod n)+1}\rangle\in E(\forall 1\leqslant i\leqslant n)$，记 $S(G)$ 为 G 中所有环路的集合。定义 $w(s)=w(\langle v_1,v_n\rangle)+\sum\limits_{1\leqslant i\leqslant n-1}w(\langle v_i,v_{i+1}\rangle)$，要求环路 s^*，使得 $w(s^*)=\min\limits_{s\in S(G)}\{w(s)\}$。

在上述定义下，本章开始提到的周游全国大城市的计划就是 TSP 问题的一个具体实例。其中，顶点个数 n 为 8，其对应的图 $G=(V,E,w)$ 如图 1.1 所示。

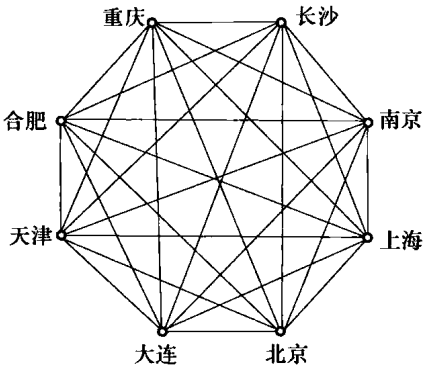


图 1.1　周游全国大城市形成的 TSP 问题实例

为了阅读方便，在图 1.1 中未在边上标注权值（即城市间的旅费）。各个城市之间的旅费信息可以单独表示为一个二维表（见表 1.1）。在该表中，相同的城市之间的旅费均设置为 0。值得注意的是，在现实生活中，两地之间的往返旅费不一定是完全相同的，本实例中为了简化起见，假定任意两个城市之间的往返旅费完全相同，比如北京→上海的旅费是 500，而上海→北京的旅费也是 500。

表 1.1　城市间的旅费（权重）

城　市	北　京	上　海	天　津	重　庆	南　京	长　沙	合　肥	大　连
北京	0	500	200	800	600	500	500	300
上海	500	0	550	600	100	300	280	350
天津	200	550	0	780	680	600	320	260
重庆	800	600	780	0	700	620	720	820
南京	600	100	680	700	0	270	120	380
长沙	500	300	600	620	270	0	230	450
合肥	500	280	320	720	120	230	0	390
大连	300	350	260	820	380	450	390	0

1.2.2　多项式时间算法与指数时间算法

算法是指求解问题所采用的步骤。更加具体地说，算法可以视为利用某种编程语言形成的计算机程序代码。若一个算法能够解决某个问题的所有实例，则说

算法能解决该问题。为了衡量算法的性能，通常可以从空间复杂度和时间复杂度两个方面进行评价。空间复杂度是为了度量算法在运行时所需要的内存消耗，而时间复杂度是为了度量算法的运行时间。对于这两种指标而言，又可以分别计算指标的平均情况、最好情况、最坏情况等。由于近年来计算机的内存容量迅速增加，因此在很多情况下，研究人员不再特别关注算法的空间复杂度。在本书中，主要关注算法的时间复杂度。在不引起歧义的上下文环境下，有时候也将时间复杂度简称为复杂度。

一般情况下，一个算法的时间复杂度表示为待求解问题的规模的函数。因此，有必要对问题的规模进行定义。对于给定的问题，可以考虑用描述问题实例的所有输入数据总量作为问题规模，但是在实际分析中，往往采用非形式化的方式来度量问题的规模。比如，在 TSP 问题中，城市数量是输入的一部分，而城市间的距离或者旅费也是输入的一部分，并且两者是关联的。正如在将来的研究中所看到的，选择城市数量或者城市间距离数作为问题规模，对于分析算法的时间复杂度并没有实质性差别。因此，在 TSP 问题上，研究者大多采用了城市数量作为问题规模。

在算法研究过程中，研究人员比较关注的是算法的时间复杂度的量级，最常见的量级是多项式时间算法和指数时间算法。为了定义多项式时间算法和指数时间算法，首先给出一些定义。

记问题的规模为 n，若存在某个常数 c，使得对于所有的 $n\geqslant 0$，均有 $|f(n)|\leqslant c|h(n)|$，则称该函数 $f(n)$ 是 $O(h(n))$ 的。多项式时间算法是指时间复杂度函数是 $O(q(n))$ 的算法，其中 $q(n)$ 是某个多项式函数。时间复杂度函数不满足以上约束的算法，通常被称为指数时间算法。需要注意的是，有些非多项式时间复杂度函数（如 $n^{\log n}$），其对应的算法也通常被称为指数时间算法。

将算法划分为多项式时间算法和指数时间算法的意义在于，随着待求解问题的实例规模的增加，可以发现两类算法所需的计算时间增长速度显然不同。多项式时间算法所需的计算时间，远远小于指数时间算法。表 1.2 给出了在目标机器速度加速后，一些拥有典型的多项式时间函数和指数时间函数的算法所能解决问题规模的变化。表 1.2 的第一列是算法的时间复杂度函数；表 1.2 的第 2 列给出了对于当前目标机器的情况下，算法所能解决问题的规模（分别用记号 M_1，M_2，…，M_6 表示）；表 1.2 的第 3 列、第 4 列则分别给出了目标机器加速 100 倍、10 000 倍后各算法所能解决的问题规模的变化。从中可以发现，随着计算机运算速度的增加，在相同时间内不同类型算法所能求解问题规模的增加是不同的。从表 1.2 中可以发现，对于时间复杂度函数为 n 的算法，当目标机器速度提高 10 000 倍时，所能解决的问题规模也相应地提高 10 000 倍。而对于时间复杂度函数为 n^4 的多项式时间算法，当目标机器速度提升 10 000 倍时，所能解决问题的规模提高 10 倍。对于指数时间的两个算法而言，当目标机器速度提升 10 000 倍时，所能解决

问题的规模增加就更加有限了。比如，对于时间复杂度函数为 4^n 的算法而言，当目标机器速度提升 100 倍时，所能解决问题的规模仅仅增加 3.32，甚至当目标机器速度提升 10 000 倍时，所能求解问题的规模也仅增加了 6.64。通过以上分析可以知道，算法的时间复杂度函数显著决定了算法所能解决问题的规模。在求解问题时，应该尽力避免指数时间算法。

表 1.2　目标机器速度提升对于多项式和指数时间复杂度算法的影响

时间复杂度函数	目标机器	速度快 100 倍的目标机器	速度快 10 000 倍的目标机器
n	M_1	$100M_1$	$10\,000M_1$
n^2	M_2	$10M_2$	$100M_2$
n^3	M_3	$4.64M_3$	$21.54M_3$
n^4	M_4	$3.16M_4$	$10M_4$
2^n	M_5	$M_5+6.64$	$M_5+13.28$
4^n	M_6	$M_6+3.32$	$M_6+6.64$

1.3　P 类与 NP 类问题

在引入了时间复杂度的概念后，可以定义所谓的 P(polynomial)类问题和 NP(non-deterministic polynomial)类问题。P 类问题是指所有可以由一个确定型图灵机在多项式时间内解决的问题；而 NP 类问题是指那些解可以在多项式时间内验证的问题。通俗地说，P 类问题是指存在多项式时间算法的问题，而 NP 类问题则是指给定它的一个候选解，可以在多项式时间内验证其是否为该问题的一个真实解。

虽然 P 类问题是在多项式时间内能够解决的，但是并不意味着 P 类问题是简单的，原因在于：

(1) P 类问题的定义忽略了常数因子。比如，一个需要 $10^{500}n$ 时间的问题是线性时间的，故此也是属于 P 类的。可是，由于常数值 10^{500} 太大，事实上无法在计算机上进行处理。反之，一个需要 $10^{-500}2^n$ 时间的问题是指数时间的，不属于 P 类，但是 n 取值较小时却是可以在计算机内处理的。

(2) P 类问题的定义未考虑多项式中指数的大小。比如，一个 P 类的时间复杂度为 n^{500} 的问题，由于指数过大，在计算机中实际上难以处理。反之，一个 NP 类的时间复杂度为 $2^{n/500}$ 的问题，当问题 n 取值较小时却是可以在计算机内处理。

(3) P 类问题的定义仅仅描述了最坏情况的复杂度。在实际应用中，可能存在一类问题，它在多数时间内是线性时间复杂度的，只有极个别情况下需要指数时间(比如 2^n)。这个问题的平均时间复杂度可能是多项式时间的，但是根据 P 类问

题的定义，它不是 P 类的。

自从 P 和 NP 类问题的概念提出后，很多研究者在不断探索 P 与 NP 之间的关系，即 P 是否等于 NP。根据定义，P 类问题是 NP 类问题的一个子集。但是 NP 类问题是否是属于 P 类的，目前还没有明确的结果。美国 Clay 数学研究所提供了一百万美金作为悬赏，奖励给任何可以证明 P=NP 或者 P≠NP 的研究者。

在 NP 类问题中存在一类非常重要的问题，即 NP-完全(NP-complete)问题。NP-完全问题是指同时满足以下条件的问题：

(1) 它是一个 NP 类问题。

(2) 其他任何 NP 类问题可以在多项式时间内变换为它。

其中，一个问题 A 能在多项式时间变换为问题 B，是指 A 的任意一个实例 a 可以在多项式时间内转化成 B 的一个实例 b，并且给定 a 的一个解，该解可以经过多项式时间变换为 b 的解，反之亦然。

在计算复杂性理论中，还有一个常见的概念，即 NP-难解(NP-hard)问题。一个 NP-难解问题是指任意的 NP 类的问题均可以在多项式时间内转化为它。根据定义，NP-难解问题不一定是 NP 类问题，即有可能比 NP 类问题还要难。同时，NP-完全问题是属于 NP-难解问题的一个子集。

最早被证明是 NP-完全问题的是可满足性问题(satisfiability problem，SAT)，一个经典的组合优化问题。在 1971 年的第 3 届 STOC 会议上，Cook[1] 证明了 SAT 问题是 NP-完全的。由于该结论在计算复杂性理论中的里程碑式的贡献，很多研究者也把该结论称为 Cook 定理。

1.4 典型的 NP-难解问题

本节介绍一些最常见的 NP-难解问题，包括 TSP 问题、QAP 问题、p-中位问题等。

1.4.1 TSP 问题

由于 TSP 问题形式简单、易于描述，同时又属于典型的 NP-难解问题，因此其作为算法分析与验证的平台而被广泛研究。在 TSP 问题上取得的理论和实验成果，可以应用于其他的 NP 问题。实际上，求解 NP-难解问题的许多方法都源自于 TSP 问题的研究。例如，目前广泛使用的分支限界(branch and bound)方法，就是首先使用在 TSP 问题上的。同时值得一提的是，TSP 问题的研究也是推动 20 世纪 70 年代以来蓬勃发展的计算复杂性理论的重要力量。在本书中，将多次以 TSP 问题为例，介绍基于骨架(backbone)设计启发式算法(heuristic algorithm)的各种技巧。因此，本小节将对 TSP 问题进行更为深入的介绍。

1. TSP 问题的分类

在长期的研究过程中，人们按照不同划分标准，对 TSP 问题进行了分类：

1）对称性

根据 TSP 问题中顶点之间距离是否对称，可以将 TSP 问题分为对称旅行商问题（symmetric traveling salesman problem，STSP）和非对称旅行商问题（asymmetric traveling salesman problem，ATSP）两种：任意两个顶点 $v_i \neq v_j$，前者满足 $w(\langle v_i, v_j \rangle) = w(\langle v_j, v_i \rangle)$；对于后者，存在两个顶点 $v_i \neq v_j$，使得 $w(\langle v_i, v_j \rangle) \neq w(\langle v_j, v_i \rangle)$ 成立。

2）三角不等式

针对 TSP 问题是否满足三角不等式（triangle inequality），TSP 问题可以被分为满足三角不等式的 TSP 问题和不满足三角不等式的 TSP 问题。前者有 $w(\langle v_i, v_k \rangle) \leqslant w(\langle v_i, v_j \rangle) + w(\langle v_j, v_k \rangle)$ 成立（$\forall 1 \leqslant i, j, k \leqslant n, i \neq j \neq k$），即两个顶点间的直线距离是最短的；后者则不满足这一点。

3）距离计算方法

针对 TSP 问题顶点之间距离计算方法的差别，也可以将 TSP 问题分为欧氏空间上的 TSP 问题（Euclidean TSP）和非欧氏空间上的 TSP 问题（non-Euclidean TSP）两种。欧氏空间上的 TSP 问题是指采用欧氏空间上的距离函数计算每对顶点之间的距离，这类实例的表示方法通常用坐标及距离矩阵。非欧氏空间上的 TSP 问题不能用欧氏空间上的距离函数计算每对顶点之间的距离，这类实例多用距离矩阵来表示。

2. TSP 问题的变种

由于 TSP 问题的研究很深入，很多应用问题可以转化为 TSP 问题或者它的变种形式。现有的 TSP 问题的变种形式包括黑白旅行商问题、多旅行商问题、带回程的旅行商问题和带时间窗的旅行商问题等。

1）黑白旅行商问题

在黑白旅行商问题（black and white traveling salesman problem，BWTSP）中，所有的顶点被划分为黑色和白色两种。该问题的目标是如何找到一条最短的环路，以使该环路上连续的任何两个黑色顶点之间的白色顶点数目以及路径长度不超过某个给定的阈值。黑白旅行商问题的应用包括使用 SONET 技术的光纤网络设计、飞机航班的调度等。

2）多旅行商问题

多旅行商问题（multiple traveling salesman problem，MTSP）的描述如下：给定 M 位旅行商，他们从一个固定的起点城市出发，希望访问 N 个城市后能回到

起点，如何求一组最短的环路，使得每个非起点的城市刚好被其中一位旅行商访问一次，并且每位旅行商最少访问了一个城市。该问题的主要应用是路由和调度领域。

3）带回程的旅行商问题

带回程的旅行商问题（traveling salesman problem with backhaul，TSPB）也是TSP问题的一个变种。它将顶点分成两大类：linehaul和backhaul，问题的要求是找到一条最短的环路，使得旅行商从起点开始，先不断访问linehaul的顶点，然后不断访问backhaul的顶点并回到出发点。TSPB问题的应用主要集中在物流、航运等领域。

4）带时间窗的旅行商问题

带时间窗的旅行商问题（traveling salesman problem with time window，TSPTW）是在TSP问题的基础上增加了新的约束，访问每个顶点的时间必须在给定的最早时刻和最晚时刻之间。TSPTW问题主要应用在银行、邮局、自动化生产等领域。

3. TSP问题的算法

目前求解TSP问题的算法可以分为精确算法和启发式算法两大类。

精确算法的特点是可以确保算法在有限的步数内获得全局最优解。截至目前，精确算法可以求解包含数百个顶点的TSP问题。目前，最有效的精确算法是切平面（cutting-plane）方法和分支限界算法。通常这类算法相当复杂，算法实现的代码行数多在一万行以上。此外，精确算法对计算机的计算速度要求很高。例如，对于2392个顶点的STSP问题，应用精确算法求解，在超级计算机上需要耗时长达27h。而若要求解更大规模（比如pla7397）的实例，对于一个大的SPARC2工作站机群，需要花费时间估计在3年左右。相比较而言，ATSP问题比STSP问题更易求解。目前，已经被解决的最大的STSP实例是pla7397，而有人已经宣布解决了高达50万个顶点的ATSP实例。

与精确算法相比较而言，启发式算法在较短的运算时间内能够得到（在解的性能上）可以接受的解。但是，启发式算法不能确保一定获得全局最优解。这类算法通常的特点是结构比较简单、容易实现及运行时间较短。在实际应用中，很多启发式算法均可以在较短的时间内获得与全局最优解相差不大（比如，目标函数值差别为几个百分点）的解。因此，在允许存在少量误差的情况下，启发式算法是求解TSP问题的较优选择。

现有的求解TSP问题的启发式算法可以分为环路构造算法（tour construction algorithm）和环路改进算法（tour improvement algorithm）两大类。