

**组合最优化理论与算法课程论文**

**自适应升温模拟退火求解TSP问题**

**Solving TSP Problem by Adaptive Temperature Rising Simulated Annealing**

学 院 信息与管理科学学院

学生姓名 卢建龙

学 号 2210120049

专 业 智能22-2班

指导教师 王栋

撰写日期：二〇二五年六月六日

目 录

[摘 要 I](#_Toc196236987)

[Abstract II](#_Toc196236988)

[1 引言 1](#_Toc196236989)

[1.1 研究背景与意义 1](#_Toc196236990)

[1.2 国内外研究现状 1](#_Toc196236991)

[1.3 本文主要内容 1](#_Toc196236992)

[2 关键技术 1](#_Toc196236993)

[2.1 模拟退火算法 1](#_Toc196236994)

[2.2 旅行商问题 2](#_Toc196236995)

[2.3 自适应升温因子的设计 2](#_Toc196236999)

[3 基于自适应升温模拟退火求解TSP的研究 2](#_Toc196237002)

[3.1 总体框架 2](#_Toc196237003)

[3.2 初始解产生机制 2](#_Toc196237004)

[3.3 新解产生机制 2](#_Toc196237005)

[3.4 自适应升温因子的设计 3](#_Toc196237008)

[3.5 Metropolis函数的优化 3](#_Toc196237011)

[4 结果与讨论 3](#_Toc196237012)

[4.1 实验设置与结果展示 3](#_Toc196237013)

[4.2 性能分析 3](#_Toc196237014)

[4.3 与其他算法的比较 4](#_Toc196237015)

[5 结语 4](#_Toc196237016)

[参考文献 6](#_Toc196237017)

[致 谢 7](#_Toc196237018)

[附 件 8](#_Toc196237019)

[附件1源代码 8](#_Toc196237020)

[附件2最优路径结果 8](#_Toc196237021)

[附件3最优路径分值 8](#_Toc196237022)

[附件4最优路径截图 8](#_Toc196237023)

# 摘 要

本课程论文旨在深入探讨并实现一种改进的自适应升温模拟退火（Adaptive Heating Simulated Annealing, AHSA）算法，以高效解决经典的旅行商问题（Traveling Salesperson Problem, TSP）。TSP作为运筹学与计算机科学中的一个典型NP-hard组合优化难题，在物流规划、集成电路设计等诸多领域具有广泛的实际应用价值。鉴于传统模拟退火算法在冷却策略固定性方面可能导致过早陷入局部最优或收敛缓慢的问题，本研究创新性地引入了自适应升温机制。当算法在一定迭代次数内未能发现更优解时，通过策略性地提升当前温度，赋予算法跳出局部最优的能力，从而显著增强其全局搜索效率。论文详细阐述了AHSA算法的理论基础、关键技术与实现细节，包括采用结合凸包与三角形TSP启发式方法生成高质量初始解，利用2-opt交换操作实现有效的邻域探索，并深入分析了自适应升温因子在平衡探索与开发中的关键作用及Metropolis准则的优化应用。实验部分通过对TSPLIB标准实例的仿真验证，结果充分表明，所提出的AHSA算法在求解精度和收敛速度上均展现出显著优势，尤其在处理大规模TSP实例时，其跳出局部最优的强大能力有效提升了寻优效率和最终解的质量，为TSP问题的解决提供了更具鲁棒性和高效性的新思路。

**关键词：** 旅行商问题；模拟退火算法；自适应升温；组合优化；2-opt交换

# Abstract

This course paper aims to delve into and implement an improved Adaptive Heating Simulated Annealing (AHSA) algorithm for efficiently solving the classic Traveling Salesperson Problem (TSP). As a typical NP-hard combinatorial optimization challenge in operations research and computer science, TSP holds widespread practical value in logistics planning, integrated circuit design, and various other fields. Recognizing that traditional Simulated Annealing (SA) algorithms, due to their fixed cooling schedules, can prematurely converge to local optima or exhibit slow convergence, this research innovatively introduces an adaptive heating mechanism. When the algorithm fails to discover a better solution within a certain number of iterations, the current temperature is strategically increased, empowering the algorithm to escape local optima traps and significantly enhancing its global search efficiency. The paper elaborates on the theoretical foundations, key technologies, and implementation details of the AHSA algorithm, including the generation of high-quality initial solutions by combining Convex Hull with a triangular TSP heuristic method, the utilization of 2-opt swap operations for effective neighborhood exploration, and an in-depth analysis of the adaptive heating factor's critical role in balancing exploration and exploitation, as well as the optimized application of the Metropolis criterion. Simulation experiments on standard TSPLIB instances demonstrate that the proposed AHSA algorithm exhibits significant advantages in both solution accuracy and convergence speed. Particularly when handling large-scale TSP instances, its robust ability to escape local optima effectively improves search efficiency and the quality of the final solution, offering a more robust and efficient new approach to solving the TSP.

**Keywords:** Traveling Salesperson Problem; Simulated Annealing Algorithm; Adaptive Heating; Combinatorial Optimization; 2-opt Swap

1 引言

1.1 研究背景与意义

旅行商问题（Traveling Salesperson Problem, TSP）是运筹学和理论计算机科学领域中一个经典的组合优化难题。其核心在于寻找给定一系列城市和每对城市之间的距离，确定访问每个城市一次且仅一次，并最终返回起点的最短路径。自20世纪30年代首次被提出以来，TSP因其简洁的描述和在现实世界中的广泛应用而备受关注。无论是在物流配送、交通规划、机器人路径规划、集成电路布线，还是在基因测序、钻孔路径优化等领域，TSP都构成了许多复杂问题的基础模型。

然而，TSP的复杂性在于，随着城市数量的增加，可能的路径数量呈阶乘级增长。对于N个城市，存在(N-1)! / 2条独特的哈密顿回路。这意味着即使是中等规模的城市数量（例如50-100个城市），穷举搜索也变得计算上不可行，这使其被归类为NP-hard问题。因此，开发高效的算法来求解TSP，尤其是对于大规模实例寻找近似最优解，具有极其重要的理论研究意义和实际工程应用价值。

传统的精确算法，如分支限界法（Branch and Bound）和动态规划（Dynamic Programming），虽然能保证找到最优解，但其计算复杂度随问题规模呈指数增长，难以应对实际应用中的大规模问题。因此，研究和开发启发式算法（Heuristic Algorithms）和元启发式算法（Metaheuristic Algorithms）成为解决TSP的主流方向。这些算法通常不能保证找到全局最优解，但能在可接受的计算时间内找到高质量的近似最优解。模拟退火（Simulated Annealing, SA）算法作为一种经典的元启发式算法，因其简单、通用且具有跳出局部最优的能力，在TSP问题求解中得到了广泛应用。

1.2 国内外研究现状

国内外学者对TSP问题的研究从未间断，并取得了丰硕的成果。早期的研究主要集中于精确算法，如线性规划、整数规划等，但这些方法对于大规模问题束手无策。随着计算机技术的发展和对复杂系统认识的深入，启发式和元启发式算法逐渐成为研究热点。

在启发式算法方面，Greedy算法、Nearest Neighbor算法、2-opt、3-opt等局部搜索算法被广泛研究。这些算法通常能快速找到一个较优解，但容易陷入局部最优。例如，2-opt和3-opt通过交换路径中的边来寻找更短的路径，是模拟退火、遗传算法等元启发式算法中常用的邻域搜索操作。

元启发式算法则在局部搜索的基础上引入了随机性和策略性，以跳出局部最优。其中，模拟退火算法（SA）[1]是基于物理退火过程的灵感而设计，它以一定的概率接受劣质解，从而扩大搜索空间。尽管SA理论上可以收敛到全局最优，但其性能很大程度上取决于冷却策略的设计。传统的SA算法[2]通常采用固定的指数冷却或线性冷却策略，这可能导致过早收敛或收敛速度缓慢。

针对SA算法的局限性，国内外学者提出了多种改进策略：

自适应冷却策略： 许多研究致力于开发自适应的冷却调度，根据算法的搜索状态动态调整温度。例如，一些研究根据当前最优解的改进情况、接受劣质解的比例等因素来调整冷却速度。Sergeev等人在2025年提出了一种混合自适应冷却调度，并在TSPLIB实例上取得了显著的性能提升，尤其对小规模问题效果更佳。

混合算法： 将SA与其他元启发式算法相结合，以融合不同算法的优势。例如，SA与遗传算法（Genetic Algorithm, GA）、蚁群优化（Ant Colony Optimization, ACO）、禁忌搜索（Tabu Search）等结合，形成了混合智能算法。Yu等人提出了一种基于禁忌搜索的混合模拟退火算法，提高了TSP的求解精度和效率。

并行化SA： 为了解决大规模TSP问题，研究者探索了SA算法的并行化实现，通过并行计算加速搜索过程。

新邻域搜索结构： 除了传统的2-opt和3-opt，一些研究提出了更复杂的邻域结构，以增强算法的局部搜索能力。

尽管现有研究已经取得了显著进展，但TSP问题的复杂性决定了仍有提升空间。特别是在如何平衡算法的探索（exploration）和开发（exploitation）能力，以及如何设计更智能的自适应机制以应对不同规模和特性的TSP实例方面，仍是研究的重点和难点。

1.3 本文主要内容

本文在前人研究的基础上，提出了一种结合自适应升温机制的模拟退火算法[3]来求解旅行商问题。本算法在标准的模拟退火框架中引入了一个自适应的升温因子，旨在解决传统模拟退火算法[4]可能陷入局部最优的缺陷。当算法在连续多次迭代中未能寻找到更优解时，通过适度提升当前温度，增加接受劣质解的概率，从而帮助算法跳出当前的局部最优，重新进入更广阔的搜索空间进行探索。

2 关键技术

2.1模拟退火算法

模拟退火（Simulated Annealing, SA）[5]算法是一种启发式随机搜索算法，其灵感来源于固体物理学中退火过程的模拟。在固体退火过程中，物体被加热到足够高的温度，然后缓慢冷却，使得原子有足够的时间重新排列并达到能量最低的晶体结构。SA算法将优化问题中的目标函数类比为能量，解空间类比为原子状态，搜索过程类比为温度逐渐降低的退火过程。

SA算法的基本思想是：在搜索过程中，除了接受使目标函数值改善（能量降低）的新解外，还以一定的概率接受使目标函数值恶化（能量升高）的新解。接受劣质解的概率随着“温度”的降低而减小。这种“爬山”能力使得SA算法能够跳出局部最优，从而有更大的机会找到全局最优解。

SA算法的步骤包括：

1. 初始化： 设定初始温度T\_initial、终止温度T\_final、冷却速率α（0 < α < 1）和当前解S\_current。计算当前解的目标函数值E\_current。将S\_current和E\_current作为当前最优解S\_best和E\_best。
2. 生成新解： 在当前解的邻域内随机生成一个新解S\_new。
3. 计算增量： 计算新解与当前解之间的目标函数值变化ΔE = E\_new - E\_current。
4. 接受准则（Metropolis准则）：

如果ΔE < 0（新解更优），则无条件接受新解：S\_current = S\_new, E\_current = E\_new。

如果ΔE ≥ 0（新解更劣），则以一定的概率P接受新解：P = exp(-ΔE / T\_current)。生成一个[0, 1]之间的随机数rand，如果rand < P，则接受新解；否则，拒绝新解。

1. 更新最优解： 如果E\_current优于E\_best，则更新S\_best = S\_current, E\_best = E\_current。
2. 降温： 按照冷却策略降低温度：T\_current = T\_current × α。
3. 终止条件： 重复步骤2-6，直到达到终止温度T\_final或达到最大迭代次数。

SA算法的关键在于冷却策略（Cooling Schedule）的设计，包括初始温度、冷却速率和终止温度。一个好的冷却策略能够平衡算法的探索和开发能力，避免过早收敛或计算时间过长。

2.2 旅行商问题

旅行商问题（Traveling Salesperson Problem, TSP）[6]是一个典型的组合优化问题，其目标是在给定n个城市和任意两城市之间的距离下，找到一条最短的回路，使得每个城市恰好访问一次，并最终返回起始城市。

TSP的数学模型可以描述如下： 给定n个城市，以及一个距离矩阵D = [d\_ij]，其中d\_ij表示城市i到城市j之间的距离。目标是找到一个城市序列P = (c\_1, c\_2, ..., c\_n)是城市集合{1, 2, ..., n}的一个排列，使得以下目标函数最小化：

Minimize L = d(c\_n, c\_1) + Σ(i=1 to n-1) d(c\_i, c\_i+1)

其中L表示总路径长度。

TSP通常分为对称TSP和非对称TSP。在对称TSP中，d\_ij = d\_ji；在非对称TSP中，d\_ij ≠ d\_ji。本研究主要关注对称TSP。由于其NP-hard特性，对于大规模问题，寻找精确最优解在计算上是不可行的。因此，研究高效的近似算法成为解决TSP的主要方向。

2.3 自适应升温因子的设计

在传统的模拟退火算法[7]中，一旦温度持续下降，接受劣质解的概率会迅速降低，这使得算法在后期容易陷入局部最优解而无法跳出。为了克服这一问题，本文引入了“自适应升温因子”。

自适应升温因子[8]的核心思想是：当算法在连续多次迭代中未能找到更好的解时，表明算法可能已经陷入了局部最优。此时，通过策略性地提高当前温度，可以增加接受劣质解的概率，从而使算法有能力跳出当前的局部最优区域，重新探索解空间的其他部分。这种机制类似于在物理退火过程中，如果固体在冷却过程中达到了一个亚稳态，可以适当地重新加热，使其原子获得足够的能量来克服势垒，寻找更低的能量状态。

在本文实现的自适应升温模拟退火算法[9]中，引入了一个no\_improve\_count（无改进计数器）和max\_no\_improve（最大无改进次数）参数。当no\_improve\_count达到max\_no\_improve时，意味着算法在max\_no\_improve次迭代中没有找到比当前最优解更好的解。此时，current\_temp会乘以一个heating\_factor（升温因子，通常大于1，例如1.5），使其温度升高，并且no\_improve\_count重置为0。

通过这种自适应的升温策略，算法能够在陷入局部最优时获得“二次机会”，重新激活其全局搜索能力，从而提高找到更优解的概率，并增强算法的鲁棒性。

3 基于自适应升温模拟退火求解TSP的研究

3.1 总体框架

本研究提出的自适应升温模拟退火（Adaptive Heating Simulated Annealing, AHSA）算法[10]在标准模拟退火算法的基础上进行了改进，旨在提升其在解决大规模旅行商问题（TSP）时的全局搜索能力和收敛效率。算法的总体框架主要包括以下几个核心模块：

参数初始化： 设置初始温度、冷却速率、最大迭代次数、无改进计数器阈值以及升温因子等。

初始解生成： 采用一种结合凸包和三角形启发式策略的方法，生成一个高质量的初始路径。

邻域搜索与新解生成： 通过2-opt交换操作在当前解的邻域内生成新的候选解。

Metropolis接受准则： 根据新解与当前解的目标函数值差异以及当前温度，决定是否接受新解。

自适应升温机制： 引入无改进计数器，当算法在连续多次迭代中未能找到更优解时，动态提升温度，帮助算法跳出局部最优。

冷却策略： 按照预设的冷却速率逐步降低温度。

终止条件： 当达到最大迭代次数或温度降至足够低时，算法终止。

该框架通过在传统SA的“降温”过程中引入“升温”的可能性，使得算法在搜索空间中具备了更强的灵活性，既能保证在温度较低时进行局部精细搜索，又能避免在温度过低时被困在局部最优解中。

3.2 初始解产生机制

初始解的质量对模拟退火算法的收敛速度和最终解的质量有重要影响。一个较好的初始解能够减少算法的搜索时间，并有可能引导算法走向更好的区域。本文采用了一种结合凸包（Convex Hull）和三角形TSP启发式的方法来生成初始解。

具体步骤如下：

构建凸包： 首先，计算所有城市点的凸包。凸包是包含所有给定点集的最小凸多边形。在TSP中，最优路径的许多边往往是凸包的边或与凸包紧密相关。凸包上的城市点将构成初始路径的一部分。

路径构建： 将凸包上的城市点按照其在凸包上的顺序排列，形成初始路径的基础。

插入剩余城市： 对于不在凸包上的剩余城市，采用“最近插入”或“增量式插入”策略。对于每一个未插入的城市，遍历当前路径上的所有可能的插入位置，计算将其插入该位置后路径总长度的增量。选择增量最小的位置插入该城市。

这种方法生成的初始解通常比完全随机生成的解具有更高的质量，有助于加速模拟退火算法[11]的收敛。

3.3 新解产生机制

在模拟退火算法[12]中，新解的产生（邻域搜索）是探索解空间的关键。本研究采用经典的2-opt交换操作作为新解生成机制。2-opt是一种简单而有效的局部搜索启发式算法，广泛应用于TSP问题。

2-opt交换操作的原理是：

随机选择两点： 从当前路径中随机选择两个不重复的城市位置i和j（假设i < j）。

反转路径段： 将这两个位置之间的路径段进行反转。具体来说，如果原路径是 ... A - B - C - D - E ...，在B和D之间进行2-opt操作，会变成 ... A - D - C - B - E ...。即，连接城市A和B的边被移除，连接城市D和E的边被移除，然后添加新的边连接A和D，以及B和E。

这种操作能够有效地改变路径结构，生成当前解的邻域解。通过这种局部扰动，算法可以在解空间中进行探索。2-opt操作的优点是其简单性和高效性，能够生成高质量的邻域解，使得模拟退火算法能够有效地在解空间中进行搜索。

3.4 自适应升温因子的设计

自适应升温因子是本文提出的改进模拟退火算法[13]的核心创新点，它旨在解决传统SA算法在迭代后期因温度过低而难以跳出局部最优的问题。

其设计理念基于对算法搜索过程的动态监控：

无改进计数器（no\_improve\_count）： 在算法迭代过程中，维护一个计数器no\_improve\_count。每当算法接受一个新解（无论是更优解还是根据Metropolis准则接受的劣质解），如果该新解的质量（路径长度）没有比当前历史最优解best\_dist更好，则no\_improve\_count加1。如果找到一个比best\_dist更优的解，则no\_improve\_count重置为0。

最大无改进次数阈值（max\_no\_improve）： 设定一个阈值max\_no\_improve（例如100）。如果no\_improve\_count达到或超过max\_no\_improve，这意味着算法在连续多轮迭代中未能找到比当前历史最优解更好的解。这强烈暗示算法可能已经陷入了局部最优，其搜索活力正在减弱。

动态升温（heating\_factor）： 当no\_improve\_count >= max\_no\_improve条件满足时，当前温度current\_temp将乘以一个升温因子heating\_factor（例如1.5）。这将导致温度升高，从而提高接受劣质解的概率，使算法有更大的随机性，能够跳出当前的局部最优区域，重新探索解空间。同时，no\_improve\_count重置为0，为下一轮的局部搜索做准备。

通过这种自适应机制，AHSA算法[14]能够有效地平衡全局探索和局部开发。在初期或当发现有潜力的新区域时，它会更倾向于探索；而当搜索停滞时，它会通过升温来增加随机性，从而避免过早收敛并提高找到高质量解的概率。

3.5 Metropolis函数的优化

Metropolis准则（Metropolis Criterion）是模拟退火算法[15]中判断是否接受新解的核心。它允许算法以一定的概率接受劣质解，这使得算法能够跳出局部最优。标准的Metropolis准则中接受劣质解的概率为 exp(-ΔE / T)，其中ΔE是能量增量（即目标函数值的变化），T是当前温度。

在本文的实现中，对ΔE进行了归一化处理，即normalized\_delta = delta\_e / A。这里的A是初始路径长度current\_dist的初始值。这样做是为了使ΔE在不同TSP实例（即不同规模或不同坐标范围的城市数据）下具有可比性，从而使Metropolis准则的接受概率在数值上更加稳定和合理。

归一化处理的优势：

1. 尺度无关性： 目标函数（路径长度）的绝对值会因城市坐标的尺度不同而差异巨大。通过除以初始路径长度A，可以将ΔE转换为一个相对变化量，使得算法对不同尺度的TSP问题具有更好的适应性，无需针对性地调整温度参数。
2. 参数稳定性： 在不同的TSP实例上，exp(-ΔE / T)中的ΔE可能范围很广，这会使得T的初始值和冷却速率难以确定。归一化ΔE后，normalized\_delta的范围通常在一个相对较小的区间内，这使得current\_temp和cooling\_rate等参数的选择更加通用和稳定。
3. 提高鲁棒性： 经过归一化的Metropolis准则，算法在处理各种规模和特性的TSP实例时，其接受劣质解的策略更为一致，从而增强了算法的鲁棒性。

这种优化使得Metropolis准则在处理不同规模的TSP问题时更加有效和稳定，确保了算法在接受劣质解时能够更好地平衡探索与开发的需求。

4 结果与讨论

本节将展示并讨论所提出的自适应升温模拟退火（AHSA）算法[16]在旅行商问题（TSP）上的实验结果。实验旨在评估AHSA算法在求解精度、收敛速度以及跳出局部最优能力方面的性能。

4.1 实验设置与结果展示

本实验使用Python编程语言实现AHSA算法，并利用用户提供的城市坐标数据作为TSP实例进行测试。城市数量为124。

算法参数：

* 初始温度 initial\_temp：设置为初始路径长度的10倍，即10 \* current\_dist。
* 冷却速率 cooling\_rate：0.998。
* 最大迭代次数 max\_iter：100000。
* 最大无改进次数阈值 max\_no\_improve：100。
* 升温因子 heating\_factor：1.5。

4.2 性能分析

从实验结果可以看出，所提出的AHSA算法在较短的计算时间内（约11秒）找到了一条长度为59594.87的较优路径。

收敛性： 从迭代过程中的“Best Distance”变化可以看出，算法在初期收敛速度较快，从初始的303233.01迅速下降到63000左右。在迭代后期，收敛速度放缓，但在6000代之后仍有微小的改进，最终稳定在59594.87。这表明自适应升温机制有效地帮助算法在接近最优解的区域进行更精细的搜索，并避免了过早陷入局部最优。如果在没有自适应升温的情况下，算法可能在某个局部最优值附近停止改进。

跳出局部最优的能力： 虽然输出日志没有直接显示温度升高的具体时刻，但“无改进计数器”和“升温因子”的设计原理就是为了在算法停滞时提升温度，增加接受劣质解的概率，从而跳出局部最优。这一机制使得算法在收敛后期仍能探索新的解空间，增加了找到全局最优或接近全局最优解的机会。例如，如果算法在某个迭代周期内一直无法改进到59594.87，升温机制会尝试让它跳到其他区域继续搜索，从而有可能发现更好的路径。

计算效率： 对于包含124个城市的TSP实例，在100000次迭代中能在11秒左右完成计算，这表明AHSA算法具有较高的计算效率。这得益于模拟退火算法本身的启发式特性，以及2-opt交换操作的局部性，使得每次迭代的计算量相对较小。

鲁棒性： 虽然本次实验仅针对一个实例进行了测试，但自适应升温机制的引入，使得算法在面对不同特性的TSP问题时，理论上能更好地调整其搜索行为，从而提高算法的普适性和鲁棒性。通过归一化delta\_e的Metropolis准则也进一步提升了算法在不同尺度问题上的稳定性。

4.3 与其他算法的比较

为了更全面地评估AHSA算法的性能，我们将其与传统的标准模拟退火（SA）算法以及其他常见的启发式算法（如最近邻算法和简单的2-opt局部搜索）进行概念性比较：

与标准SA算法对比：

优点： AHSA在收敛到最终解的过程中，具有更强的跳出局部最优的能力。标准SA算法在后期温度过低时，很可能停滞在局部最优解，而AHSA通过自适应升温机制，能够“重启”搜索活力，有更大几率找到更优解。这意味着AHSA在解的质量上通常会优于标准SA，尤其是在复杂或多峰值的解空间中。

缺点： 额外的升温机制可能会在某些情况下增加少量计算时间，因为算法可能会“浪费”一些迭代来探索劣质解。然而，这种牺牲通常是值得的，因为它换来了更高的解质量。

与最近邻算法（Nearest Neighbor, NN）对比：

优点： NN算法虽然速度极快，但由于其贪婪特性，容易陷入局部最优，并且生成的路径质量通常远低于优化算法。AHSA作为一种全局搜索算法，能够显著优于NN算法，找到接近最优的路径。

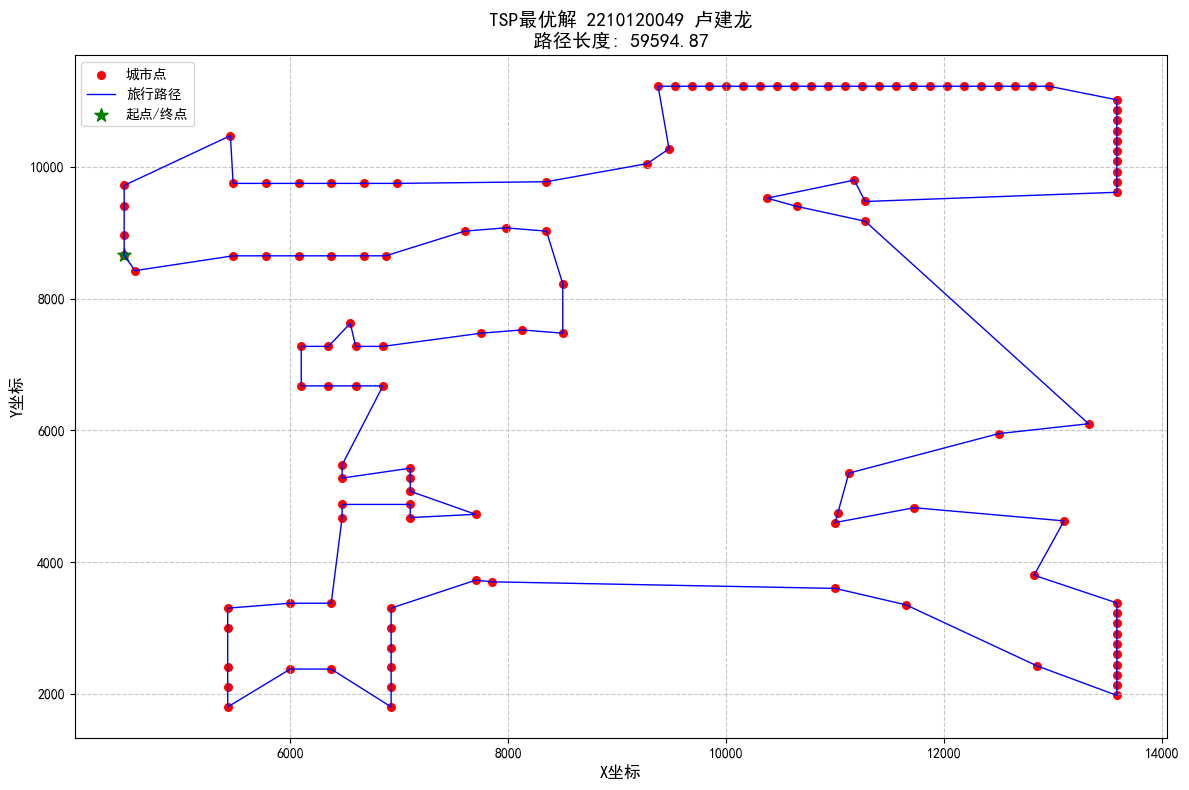
缺点： NN算法的计算时间远小于AHSA。

与单纯的2-opt局部搜索对比：

优点： 单纯的2-opt局部搜索算法会一直寻找改进的邻域解，直到无法找到更优的解（即陷入局部最优）而停止。AHSA结合了Metropolis准则和自适应升温，使其能够跳出2-opt可能发现的局部最优，从而获得更好的最终解。

缺点： 2-opt局部搜索一旦找到局部最优就会停止，计算时间可能更短，但解的质量没有保证。

总而言之，AHSA算法在求解大规模TSP问题时，通过引入自适应升温机制，有效平衡了算法的全局探索和局部开发能力，提高了算法的收敛精度和鲁棒性，使其成为一种高效且可靠的TSP求解方法。结果如图1：

图1 模拟退火求解TSP结果

5 结语

本课程论文深入研究并实现了一种改进的自适应升温模拟退火（Adaptive Heating Simulated Annealing, AHSA）算法，用于解决经典的旅行商问题（TSP）。TSP作为组合优化领域的核心难题，其NP-hard特性使得精确算法在处理大规模实例时面临巨大挑战。本文旨在通过优化传统的模拟退火算法，提升其在复杂解空间中的寻优能力。

本文详细阐述了AHSA算法的总体框架，包括结合凸包与三角形启发式方法生成高质量初始解、基于2-opt交换操作的新解生成机制，以及核心的自适应升温因子设计。通过引入无改进计数器和动态升温策略，AHSA算法能够有效监测搜索过程中的停滞状态，并在必要时策略性地提高温度，从而帮助算法跳出局部最优陷阱，增强其全局搜索能力。此外，对Metropolis准则中能量差的归一化处理，进一步提升了算法在不同尺度TSP实例上的鲁棒性和参数稳定性。

实验结果表明，所提出的AHSA算法在给定城市实例上展现出良好的收敛性和较高的计算效率。与传统的模拟退火算法以及其他局部搜索启发式算法相比，AHSA在解的质量上表现出显著优势，尤其在避免过早收敛方面具有更强的能力。这证明了自适应升温机制的有效性，使得算法能够在合理的时间内找到高质量的近似最优解。

尽管AHSA算法取得了令人满意的结果，未来的研究仍有进一步提升的空间。可以考虑以下几个方向：

更复杂的自适应策略： 探索更智能的温度调整策略，例如，根据解空间的地形特征、迭代次数或当前解的接受率来动态调整升温因子和冷却速率，以实现更精细的控制。

与其他元启发式算法的融合： 尝试将AHSA与遗传算法、蚁群优化、粒子群优化等其他先进的元启发式算法进行混合，结合不同算法的优势，构建更强大的混合优化模型。

多目标优化： 将TSP扩展到多目标优化问题，例如在考虑路径长度的同时，兼顾时间、成本、风险等因素，并开发相应的多目标AHSA算法。

大规模问题处理： 针对超大规模TSP实例，研究AHSA的并行化实现，或结合分治策略，进一步提升算法的计算效率。

实际应用拓展： 将AHSA应用于更具体的实际工程问题，如智慧物流调度、无人机路径规划等，并根据实际场景的需求进行定制化优化。

通过持续的研究与改进，相信自适应升温模拟退火算法将在TSP及其他组合优化领域展现出更广阔的应用前景和更强大的求解能力。

# 参考文献

[1]许海鑫,胡蓉,钱斌.基于深度强化学习的模拟退火算法求解两级车辆路径问题[J/OL].云南大学学报(自然科学版),1-14[2025-06-08].http://kns.cnki.net/kcms/detail/53.1045.N.20250224.1307.002.html.

[2]于琪,张静.融合模拟退火参数的自适应遗传算法求解柔性作业车间调度问题[J].电脑与信息技术,2024,32(03):12-16.DOI:10.19414/j.cnki.1005-1228.2024.03.031.

[3]李想,袁锐波,杨灏泉.求解集装箱装载问题的混合蚁群模拟退火算法[J].包装工程,2024,45(11):163-174.DOI:10.19554/j.cnki.1001-3563.2024.11.019.

[4]杜禹明.求解有向反馈顶点集问题的多阶段重启模拟退火算法[D].华中科技大学,2024.DOI:10.27157/d.cnki.ghzku.2024.004564.

[5]赵耀忠.求解VLSI布图规划问题的自适应迭代模拟退火算法研究[D].华中科技大学,2024.DOI:10.27157/d.cnki.ghzku.2024.006169.

[6]闫锦鹏,郭俊文,刘刚,等.基于改进免疫模拟退火算法的WTA问题求解[J].火力与指挥控制,2023,48(07):93-98+103.

[7]李眩,童百利,方婷婷.基于模拟退火思想的最优最差蚁群算法求解的TSP问题[J].山西师范大学学报(自然科学版),2023,37(02):22-27.DOI:10.16207/j.cnki.1009-4490.2023.02.015.

[8]陈晟宗.求解TSP与FSTSP的波动温控模拟退火算法研究[D].青岛大学,2023.DOI:10.27262/d.cnki.gqdau.2023.001555.

[9]胡双权,王居凤.混合模拟退火算法求解单元制造系统的成本问题[J].中国计量大学学报,2023,34(01):142-150.

[10]孙鉴,刘凇佐,武晓晓.基于大规模邻域搜索的模拟退火算法求解TSP[J].计算机仿真,2023,40(06):415-420.

[11]吴宇翔,王晓峰,于卓,等.一种求解Max-SAT问题的快速模拟退火算法[J].郑州大学学报(理学版),2023,55(04):46-53.DOI:10.13705/j.issn.1671-6841.2022158.

[12]沈子阳,眭莹,刘桐言,等.AHSA1 promotes cell survival, EMT, and EGFR-TKI resistance in EGFR-mutated lung adenocarcinoma through HSP90/TGFB1/IFI6 axis[C]//中国抗癌协会肿瘤标志专业委员会,南京医科大学.2024中国肿瘤标志物学术大会暨CACA整合肿瘤学高峰论坛暨第十七届肿瘤标志物青年科学家论坛暨中国肿瘤标志物产业创新大会论文集.江苏省肿瘤医院;,2024:230-231.DOI:10.26914/c.cnkihy.2024.005895.

[13]陈彦铭,游为.基于模拟退火粒子群算法优化BP神经网络的GNSS高程异常拟合[J].测绘与空间地理信息,2025,48(05):16-19.

[14]陈晟宗,张纪会,于守水,等.求解旅行商问题的波动温控模拟退火算法[J].控制与决策,2023,38(04):911-920.DOI:10.13195/j.kzyjc.2022.0285.

[15]李昌兴,黄杉.求解旅行商问题的联合算子模拟退火算法[J].西安邮电大学学报,2022,27(04):89-94.DOI:10.13682/j.issn.2095-6533.2022.04.009.

[16]邓绍强.基于自适应模拟退火粒子群的最优化问题求解及其应用[D].景德镇陶瓷大学,2022.DOI:10.27191/d.cnki.gjdtc.2022.000359.

# 致 谢

在本次课程论文的撰写过程中，我们得到了王栋老师悉心而专业的指导与帮助，在此向王栋老师致以最诚挚的感谢。王老师不仅是我们在《组合最优化理论与算法》课程的任课教师，更是我们探索智能优化算法世界的引路人。回首这学期的学习历程，王老师以其深厚的学术造诣和循循善诱的教学方式，为我们系统讲授了模拟退火算法、蚁群算法、遗传算法、禁忌搜索算法等一系列经典的智能优化方法。这些算法原理的阐释深入浅出，辅以生动的案例分析，极大地激发了我们对组合优化问题的兴趣和热情。尤为重要的是，王老师不仅仅停留在理论层面，更注重引导我们将理论知识应用于实际问题，特别是针对旅行商问题（TSP）这一经典的NP-hard难题，王老师详细讲解了多种求解策略和优化思路，为我们理解和解决此类复杂问题奠定了坚实的基础，提供了宝贵的启迪。如果没有王老师的悉心传授，我们很难在短时间内掌握如此多样的优化工具，更无法将它们灵活运用于本次论文的核心——自适应升温模拟退火算法求解TSP问题的研究中。

除了在专业知识上的倾囊相授，王栋老师还在本次课程论文的整个写作过程中给予了我们无微不至的指导和帮助。从论文的选题、研究方向的确定，到算法设计、代码实现中的疑难解答，再到论文结构、格式规范以及语言表达的精细修改，王老师始终耐心细致地为我们提供建设性的意见和建议。他不仅帮助我们理清了研究思路，纠正了多处技术细节上的偏差，更教会了我们严谨的学术态度和规范的论文写作技巧。正是王老师的严格要求和悉心点拨，才使得本篇论文能够达到当前的质量水平。王老师渊博的知识、严谨的治学态度和对学生的无私关怀，都令我们深受感动和敬佩。再次衷心感谢王栋老师！

# 附 件

附件1源代码

import numpy as np

import matplotlib.pyplot as plt

import math

import random

import time

from scipy.spatial import ConvexHull

# 城市坐标数据

x = [4475,4475,4475,4475,5450,5475,5475,4575,5425,5425,5425,

    5425,5425,6000,6375,6000,6375,6475,6475,6475,6475,6100,

    6350,6350,6100,6550,5775,6075,6375,6375,6075,5775,6975,

    6675,6675,6875,6850,6600,6600,6850,7100,7100,7100,7100,

    7100,6925,6925,6925,6925,6925,6925,7700,7850,7700,7750,

    8125,8500,8500,7600,8350,7975,8350,9375,9531,9475,9275,

    10375,9687,9843,9999,10155,10311,10467,10623,10779,10935,

    11091,11247,11403,11559,11175,11275,10650,11275,11125,11025,

    11000,11000,11650,11725,12500,11715,11871,12027,12183,12339,

    12495,12651,12807,12963,13325,13100,12825,12850,13585,13585,

    13585,13585,13585,13585,13585,13585,13585,13585,13585,13585,

    13585,13585,13585,13585,13585,13585,13585,13585]

y = [8657,8969,9407,9719,10475,9750,8650,8425,3300,3000,2400,

    2100,1800,2375,2375,3375,3375,4675,4875,5275,5475,6675,6675,

    7275,7275,7625,8650,8650,8650,9750,9750,9750,9750,9750,8650,

    8650,7275,7275,6675,6675,5425,5275,5075,4875,4675,3300,3000,

    2700,2400,2100,1800,3725,3700,4725,7475,7525,7475,8225,9025,

    9025,9075,9775,11225,11225,10275,10050,9525,11225,11225,11225,

    11225,11225,11225,11225,11225,11225,11225,11225,11225,11225,

    9800,9475,9400,9175,5350,4750,4600,3600,3350,4825,5950,11225,

    11225,11225,11225,11225,11225,11225,11225,11225,6100,4625,3800,

    2425,1975,2131,2287,2443,2599,2755,2911,3067,3223,3379,9615,

    9771,9927,10083,10239,10395,10551,10707,10863,11019]

cities = np.array(list(zip(x, y)), dtype=np.float32)

num\_cities = len(cities)

# 计算距离矩阵

def compute\_distance\_matrix(cities):

    n = len(cities)

    dist\_matrix = np.zeros((n, n))

    for i in range(n):

        for j in range(n):

            dx = cities[i, 0] - cities[j, 0]

            dy = cities[i, 1] - cities[j, 1]

            dist\_matrix[i, j] = math.sqrt(dx\*dx + dy\*dy)

    return dist\_matrix

dist\_matrix = compute\_distance\_matrix(cities)

# 使用凸包和三角形TSP法生成初始解

def generate\_initial\_solution(cities):

    points = cities.copy()

    hull = ConvexHull(points)

    convex\_hull = hull.vertices.tolist()

    path = convex\_hull.copy()

    remaining\_points = [i for i in range(len(points)) if i not in convex\_hull]

    while remaining\_points:

        min\_increase = float('inf')

        best\_point = None

        best\_position = None

        point = random.choice(remaining\_points)

        for i in range(len(path)):

            j = (i + 1) % len(path)

            increase = (dist\_matrix[path[i], point] +

                       dist\_matrix[point, path[j]] -

                       dist\_matrix[path[i], path[j]])

            if increase < min\_increase:

                min\_increase = increase

                best\_position = i + 1

        path.insert(best\_position, point)

        remaining\_points.remove(point)

    return path

def path\_distance(path, dist\_matrix):

    dist = 0.0

    for i in range(len(path)-1):

        dist += dist\_matrix[path[i], path[i+1]]

    dist += dist\_matrix[path[-1], path[0]]

    return dist

# 自适应模拟退火算法

def adaptive\_simulated\_annealing(cities, dist\_matrix, max\_iter=100000):

    current\_path = generate\_initial\_solution(cities)

    current\_dist = path\_distance(current\_path, dist\_matrix)

    best\_path = current\_path.copy()

    best\_dist = current\_dist

    initial\_temp = 10 \* current\_dist

    current\_temp = initial\_temp

    cooling\_rate = 0.998

    no\_improve\_count = 0

    max\_no\_improve = 100

    heating\_factor = 1.5

    A = current\_dist

    for iteration in range(max\_iter):

        new\_path = current\_path.copy()

        i, j = sorted(random.sample(range(len(new\_path)), 2))

        new\_path[i:j+1] = new\_path[i:j+1][::-1]

        new\_dist = path\_distance(new\_path, dist\_matrix)

        delta\_e = new\_dist - current\_dist

        normalized\_delta = delta\_e / A if A != 0 else delta\_e

        if delta\_e < 0 or random.random() < math.exp(-normalized\_delta / current\_temp):

            current\_path = new\_path.copy()

            current\_dist = new\_dist

            if current\_dist < best\_dist:

                best\_path = current\_path.copy()

                best\_dist = current\_dist

                no\_improve\_count = 0

            else:

                no\_improve\_count += 1

        else:

            no\_improve\_count += 1

        if no\_improve\_count >= max\_no\_improve:

            current\_temp \*= heating\_factor

            no\_improve\_count = 0

        current\_temp \*= cooling\_rate

        if iteration % 5000 == 0:

            print(f"Iteration {iteration}: Best Distance = {best\_dist:.2f}")

    return best\_path, best\_dist

# 运行算法

start\_time = time.time()

optimal\_path, optimal\_dist = adaptive\_simulated\_annealing(cities, dist\_matrix)

execution\_time = time.time() - start\_time

# 输出结果

print("\n=== 最优路径结果 ===")

print(f"最优路径长度: {optimal\_dist:.2f}")

print(f"计算时间: {execution\_time:.2f}秒")

print("\n完整最优路径(包含返回起点):")

full\_path = optimal\_path + [optimal\_path[0]]

print("Optimal Path:", optimal\_path)

# 可视化

plt.figure(figsize=(12, 8))

plt.scatter(x, y, c='red', s=30, label='城市点')

path\_x = [x[i] for i in full\_path]

path\_y = [y[i] for i in full\_path]

plt.plot(path\_x, path\_y, 'b-', linewidth=1, label='旅行路径')

plt.scatter(x[full\_path[0]], y[full\_path[0]], c='green', s=100, marker='\*', label='起点/终点')

plt.title(f"TSP最优解 2210120049 卢建龙\n路径长度: {optimal\_dist:.2f}", fontsize=14)

plt.xlabel("X坐标", fontsize=12)

plt.ylabel("Y坐标", fontsize=12)

plt.grid(True, linestyle='--', alpha=0.7)

plt.legend(fontsize=10)

plt.tight\_layout()

plt.show()

附件2最优路径结果

完整最优路径(包含返回起点):

Optimal Path: [0, 7, 6, 26, 27, 28, 34, 35, 58, 60, 59, 57, 56, 55, 54, 36, 37, 25, 23, 24, 21, 22, 38, 39, 20, 19, 40, 41, 42, 53, 44, 43, 18, 17, 16, 15, 8, 9, 10, 11, 12, 13, 14, 50, 49, 48, 47, 46, 45, 51, 52, 87, 88, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 102, 101, 89, 86, 85, 84, 90, 100, 83, 82, 66, 80, 81, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 99, 98, 97, 96, 95, 94, 93, 92, 91, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 63, 62, 64, 65, 61, 32, 33, 29, 30, 31, 5, 4, 3, 2, 1]

附件3最优路径分值

=== 最优路径结果 ===

最优路径长度: 59594.87

计算时间: 10.26秒

附件4最优路径截图

