分类号 _____ 编号_____

U D C_____ 密级_____

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

本科生毕业设计（论文）

题　　目： 基于大型语言模型的启发式算法自动设计

姓　　名： 黄德赐

学　　号： 12011916

系　　别： 计算机科学与工程系

专　　业： 计算机科学与技术

指导教师： 史玉回 讲席教授

2024 年 6 月 7 日

诚信承诺书

1. 本人郑重承诺所呈交的毕业设计（论文），是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2. 除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3. 本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4. 在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名：黄德赐

2024 年 6 月 7 日

基于大型语言模型的启发式算法自动设计

黄德赐

（计算机科学与工程系　指导教师：史玉回）

[摘要]：毕业设计课题探讨了利用大型语言模型自动设计启发式算法的可能性和效果，目的是提高设计启发式算法的效率。本课题使用基于Llama2 的大型代码语言模型 Code Llama 进行启发式算法设计。主要方法是为启发式算法设计任务设计合适的提示策略，其中典型的提示方法是上下文学习，将任务描述和演示以自然语言文本的形式表达，为启发式算法设计任务提供逐步的规划，同时使用思维链提示。通过使用零样本思维链提示，引导大型语言模型生成中间推理步骤以提高输出结果的性能，然后运行算法并得到输出结果的评价。通过实验发现基于大型语言模型的方法能够自动学习并生成具有启发性的算法结构，从而加速算法设计的过程。因此，通过借助大型语言模型，可以更好地生成具有创新性和性能优越性的启发式算法。

[关键词]：大型语言模型；启发式算法；代码生成

[**ABSTRACT**]: The project explores the potential and effectiveness of using large language models to automatically design heuristic algorithms, aiming to improve the efficiency of heuristic algorithm design. This project employs a large code language model based on Llama2, Code Llama, for heuristic algorithm design. The primary method involves designing suitable prompt strategies for heuristic algorithm design tasks. A typical prompting method is in-context learning, expressing the task description and demonstrations in natural language text, providing step-by-step planning for heuristic algorithm design tasks, and utilizing chain-of-thought prompting. By using zero-shot chain-of-thought prompting, the large language model is guided to generate intermediate reasoning steps to enhance the performance of the output results. The algorithms are then run, and the output results are evaluated. Experiments show that the method based on large language models can automatically learn and generate heuristic algorithm structures, thereby accelerating the process of algorithm design. Thus, by leveraging large language models, more innovative and performance-superior heuristic algorithms can be effectively generated.

[**Key words**]: Large Language Models, Metaheuristic Algorithm, Code Generation

目录

1. 引言

1.1 研究背景与意义

1.1 研究背景与意义

随着人工智能技术的飞速发展，大型语言模型（Large Language Models, LLMs）的研究与应用取得了显著的进展。这些模型通过在海量的文本数据上进行预训练，已经展示出在多种任务中的突破性水平，包括但不限于编程[1]、逻辑推理[2]、语言翻译、摘要生成、情感分析等。这些模型之所以强大，是因为它们能够捕捉复杂的语言结构和深层次的知识模式，进而在各种应用场景中发挥重要作用，从文本生成到语义理解，再到代码生成等。

在这样的背景下，大型语言模型不仅仅是工具，它们从人类知识的宝库中汲取灵感，能够做出近似于人类的决策。这些模型的上下文学习能力，特别是记忆能力和组合性能力的结合，使得它们成为了强大且友好的人机交互代理。它们能够理解和生成自然语言，提供信息检索，辅助决策制定，甚至参与创意写作和内容创作。

大型语言模型在预训练过程中涉及了多个领域的知识，这为它们提供了丰富的跨领域知识。这些跨领域的经验和直觉可以用于设计新的启发式算法中，为解决复杂问题提供新的思路和方法。同时，大型语言模型的快速迭代能力可以帮助算法设计者快速原型化他们的算法，根据初步结果不断调整和改进算法设计，这大大加快了算法的开发周期。

此外，大型语言模型还可以通过交互式学习进一步优化算法设计。通过与设计者的持续交流，模型可以学习到特定问题领域的特定需求，从而提供更加定制化的算法设计方案。这种能力使得大型语言模型在自动化设计启发式算法方面具有巨大的潜力。

因此，本课题探讨了利用大型语言模型来自动设计启发式算法的应用场景。通过结合大型语言模型的自然语言处理能力和先进的算法设计技术，我们可以提高设计启发式算法的效率，降低算法设计的门槛，使得非专家用户也能够设计出有效的算法来解决特定的问题。这不仅能够推动算法设计领域的发展，还能够为各行各业的问题解决提供更加智能化的工具和解决方案。

1.2 已有工作的问题

在现有的启发式算法设计研究中，尤其是在求解最优化问题时，存在一些显著的不足之处，这些问题限制了算法设计的效率和质量：

- 人工设计耗时、费力：传统的启发式算法设计通常依赖于专家的经验和直觉。这种方法虽然在某些情况下能够产生有效的解决方案，但其效率和创新性受到设计者个人水平和创造力的限制。当面对新的或复杂的优化问题时，人工设计算法不仅耗时而且费力，需要经过大量的试验和错误才能逐步改进算法性能。此外，这种方法很难快速适应问题特性的变化，对于需要快速迭代的场景不够灵活。

- 创新性有限：由于传统方法依赖于设计者的知识和经验，因此在算法创新上存在瓶颈。设计者可能受限于自己熟悉的算法模式和问题领域，难以跳出既定的思维框架进行创新。而预训练的大型语言模型则掌握了大量的跨领域知识，这使得它们在元启发式算法的设计中具有潜在的优势。大型语言模型能够提供新颖的设计思路，甚至可能创造出人类专家难以想象的算法结构。

- 泛化能力不足：许多现有的启发式算法是针对特定类型的问题设计的，它们可能在特定问题上表现良好，但在其他问题上则效果不佳。这种局限性导致算法的泛化能力不足，难以适应多变的问题场景和需求。

1.3 主要贡献

针对启发式算法设计中存在的问题，本课题提出了一种基于大型语言模型的启发式算法自动设计方法。设计了一个自动化设计框架，该框架以大型语言模型为核心，能够自动生成和优化启发式算法。与传统的人工设计相比，该框架可以显著提高算法设计的速度和效率。利用大型语言模型丰富的知识库和强大的模式识别能力，框架能够提出创新的算法设计。研究方法注重算法的泛化性，通过在多种问题类型上训练和测试算法，确保生成的启发式算法不仅在特定问题上有效，也能够适应广泛的应用场景。本课题提出了一套系统性的设计方法论，指导如何利用大型语言模型进行启发式算法的设计。这包括算法组件选择、参数调优以及算法结构优化等。通过自动化

设计，算法的更新和维护变得更加简单快捷。框架可以快速适应新的问题特性和需求变化，减少了人工介入的需要。本课题的研究成果为未来的研究提供了新的方向，包括如何进一步提升算法设计的自动化水平、如何结合特定领域的知识进行定制化设计，以及如何将自动化设计方法应用于更广泛的优化问题。

综合以上内容，本课题旨在推动启发式算法设计的自动化进程，提高设计效率，降低设计门槛，最终实现更智能、更快速、更创新的算法设计。

1.4 研究方案

本研究的主要目标是开发一种自动化流程，该流程能够利用大型语言模型根据给定的问题域和优化目标，自动生成和优化启发式算法的代码，以此来提高算法设计的效率，降低对专家知识的依赖，并推动算法设计的创新。

本课题的研究方案是通过一系列精心设计的步骤来实现启发式算法的自动设计。首先，选择一个适合本研究需求的大型预训练语言模型，该模型必须具备强大的代码生成和理解能力，以确保能够准确地生成启发式算法代码。接着，定义一个或多个启发式算法的代码框架，这些框架作为算法设计的蓝图，确保生成的代码既结构化又目标明确。在此基础上，收集和准备一组启发式算法的样例代码，这些样例对于训练语言模型至关重要，它们帮助模型理解算法设计的模式和结构。

进一步地，开发一套提示词策略，这些提示词将作为语言模型生成算法代码的指导，确保生成的代码能够针对特定的优化问题。随后，实现一个自动化代码生成流程，该流程以框架和提示词为输入，自动利用语言模型生成新的启发式算法代码。生成的算法代码将经过一套全面的评估体系进行测试和评估，以确保算法的准确性、效率和创新性。最后，根据性能评估的结果进行迭代优化，不断调整和改进生成的算法代码，以提高其性能和适应性。

本课题中选择 CEC2005[3]中的优化问题作为基准，以测试生成算法的性能。

2. 背景介绍

2.1 大型语言模型介绍

语言模型是让机器理解语言的主要方法之一。语言模型的目标是对单词序列的生成概率进行建模，以预测未来标记的概率。大型语言模型（Large Language Models, LLMs）指的是包含数百亿参数的 Transformer[4]语言模型，这些模型在大规模文本数据上进行训练，如 GPT3 和 Llama[5]。

使用大型语言模型通常需要庞大的计算资源，常用的应用大型语言模型的方法是使用公开可用的大语言模型资源进行实验研究。Llama 模型系列自发布以来，许多研究人员通过持续预训练或指导调整扩展了 Llama 模型[6]。Code Llama 是基于 Llama2 的大型代码语言模型系列, 提供对大型输入上下文的支持以及编程任务的零样本指令跟随能力[7]。

大型语言模型具备三种新的能力：上下文学习、逐步推理和指令微调。

- 上下文学习：不用额外训练即可接收已经为语言模型提供的许多指令，并可根据这些指令生成答案[8]。
- 思维链：当面对需要多个步骤进行推理的问题时，小型模型往往处理困难。大型模型可以通过“思维链”的方法，引导语言模型进行逐步推理，利用包含中间步骤的提示来逐步解决问题[2]。
- 指令微调：通过对任务数据集进行指令微调，大型语言模型能够理解并执行新的任务，这可以增强模型解决新任务的能力。

大型语言模型的上下文学习能力可以有效提取并记忆关键的描述信息。此外，模型融合的多个领域的知识有助于使用其他领域的经验和直觉，可能会产生更具创新性的启发式算法。大型语言模型可以支持算法设计的不断迭代，允许根据反馈迅速调整和自动进行算法优化。因此，本课题探索使用大型语言模型自动设计启发式算法的可能性。

2.2 启发式算法概述

启发式算法是解决优化问题的重要工具，特别是在那些对于精确算法而言计算成本过高的情况下。这类算法是一种基于经验和直觉的问题求解方法[9]，用于寻找解决复杂问题的近似解，通常在没有确切解决方案或确切解决方案难以计算的情况下使用。启发式算法通常比精确算法快，但不能保证找到最优解。通常用于对解的质量要求较为宽松的场景，或由于解空间太大导致难以使用精确算法的场景下。常见的启发式算法包括遗传算法，蚁群算法, 模拟退火算法等。遗传算法（Genetic Algorithms, GAs）是一种模拟自然选择和遗传机制的搜索算法，通过模拟生物进化过程中的交叉、变异和选择操作来优化问题解[10]。蚁群算法（Ant Colony Optimization, ACO）则受到蚂蚁寻找食物的启发，通过信息素的沉积和跟踪来寻找问题的最优解或近似解[11]。模拟退火算法（Simulated Annealing, SA）则是借鉴了材料退火过程中的物理现象，通过控制温度参数来逐步寻找解空间中的最优解[12]。

启发式算法的应用范围非常广泛，它们被成功应用于各种实际问题中。在调度领域，启发式算法可以帮助解决车间作业调度问题，优化生产流程，减少等待时间和提高生产效率。在航空和交通行业，它们被用来解决航班调度和路径规划问题，以应对延误和不确定性带来的挑战。此外，在网络领域，启发式算法也被用于数据包路由和网络流量管理，以提高数据传输的效率和稳定性。

资源分配和组合优化是启发式算法的另外两个重要应用领域。在资源分配问题中，算法需要考虑如何将有限的资源合理分配给不同的任务或项目，以达到最优的经济效益或社会效益。而在组合优化问题中，启发式算法则用于在庞大的可能解集中快速找到满足特定条件的组合。

设计有效的启发式算法不仅需要对特定问题领域有深入的理解，还需要掌握算法设计的一般原则。包括了解问题的约束条件、解空间的结构、以及如何平衡探索与利用之间的关系。参数调整是启发式算法中的一个关键环节，会直接影响算法的性能和最终解的质量。参数设置不当可能导致算法陷入局部最优解或在解空间中徘徊不前。

2.3 自动化设计启发式算法概述

设计启发式算法通常需要专家手动进行细致的调整以适应特定问题，才能够获得满意的解决方案[13]。尽管这种手动定制的方法在许多问题上取得了显著成效，但仍旧存在问题。首先，这个过程可能非常耗时，专家可能需要投入大量的时间去构思、开发和验证算法。其次，复杂的问题以及设计算法时所面临的广泛选择导致算法的设计容易出错。第三，手动定制过程中某些设计决策的依据不明确，导致难以捕捉和应用这些决策背后的原理和见解[14]。最后，在没有专家可用的情况下，手动定制算法变得不现实。

自动设计启发式算法的过程分成四个主要部分[13]。首先，设计空间包含了所有可能用到的构建模块，设计算法时可以选择的元素，比如基本的计算步骤、算法操作和参数设置。接着，设计策略帮助我们从设计空间中挑选和组合这些模块来创建算法。然后，性能评估策略说明如何衡量算法的表现，以及如何根据这些表现来调整和优化算法。最后，提出的不同的目标解决问题作为数据，帮助我们评估算法的性能。这四个部分构成了研究自动化算法设计的基础框架。

算法的自动设计策略包括了局部搜索[15]、进化搜索[16]等无模型策略和分布估计[17]、贝叶斯优化[18]、强化学习[19]、大型语言模型[5]等基于模型的策略。通过在大规模语料库上预训练 Transformer 模型[4]，大型语言模型具备了知识记忆和组合能力。近期的研究已经提出了使用大型语言模型作为设计元启发式算法的策略[5]。研究通过上下文学习方式来应用大型语言模型：向模型提供文本形式的任务提示和一些算法的示例。基于这些信息，大型语言模型能够通过补全提示中的词序列来创造新的算法，这个过程不涉及额外的训练或梯度更新步骤。

3. 研究方法

本课题旨在深入探索并验证基于 Llama2 的 Code Llama 大型语言模型在启发式算法设计领域的应用潜力和效果。研究的核心焦点在于利用 Code Llama 模型的先进代码生成能力，通过精心构建的提示策略，来迭代地生成和优化启发式算法的代码。主要目标是展示如何通过大型语言模型来自动化启发式算法的设计过程，减少人工干预，同时保持或提高算法的性能。我们将采用一系列的实验方法来评估大型语言模型在不同类型优化问题上的表现。

3.1 提示策略

在预训练或微调之后，使用大型语言模型的主要方法之一是为解决任务设计合适的提示策略。一个典型的提示方法是上下文学习[8]，将任务描述与演示以自然语言文本的形式表达。同时，可以使用思维链提示[2]通过涉及一系列中间推理步骤来增强上下文学习。

上下文学习（In-Context Learning, ICL）是首次随 GPT-3[8]提出的，已经成为利用大型语言模型的常用方法。ICL 选择几个示例作为演示，然后将它们组合成设计的自然语言提示模板，经典的方法是构建具有输入-输出对的提示。

思维链（Chain-of-Thought，CoT）[2]是一种改进的提示策略，目的是提高大型语言模型在较为复杂的推理任务上的性能水平。相比于在 ICL 中构建输入-输出对的提示，CoT 在提示中加入中间推理步骤，构建输入-推理过程-输出对的提示。CoT 有少样本和零样本 CoT 两种常用设置, 常与 ICL 一起使用。少样本 CoT 将输入-输出扩充为输入-推理-输出，把推理步骤纳入其中。零样本 CoT 在提示中不包含人工添加的推理步骤，它直接生成推理步骤，然后使用生成的 CoTs 推导答案。大语言模型可以通过“让我们一步一步地思考”提示生成推理步骤，然后通过“因此，答案是”的提示引导得出最终答案[20]。

3.2 提示词设计方法

在本研究中，探索了一种创新的提示词设计方法，旨在最大化 Code Llama 等大型语言模型在启发式算法设计中的潜力。这种方法结合了上下文学习和思维链技术，以增强模型对设计任务的深入理解，并激发其创造性地生成解决方案。

首先为模型提供了一个清晰的算法设计问题定义，确保模型能够准确把握需要解决的优化问题。以 CEC2005 的 f1 问题[3]。为例，我们向模型展示了该问题的具体要求和目标。随后，提供了一个或多个遗传算法的示例代码，这些示例不仅展示了问题的解决方案，还构成了上下文学习的基础，帮助模型理解任务的关键要求和潜在的解决策略。

在模型成功生成初步的遗传算法之后，采用了零样本思维链提示方法来进一步优化算法设计。这种方法的核心在于引导大型语言模型“一步一步地思考”，从而确保模型能够明确并理解算法中每一步的逻辑和计算过程。这不仅帮助模型深入理解算法结构的核心内容，而且促进了对算法设计的深入分析和创新。

在优化提示的设计上，为模型提供了初始的遗传算法样例代码，并结合模型基于这些样例生成的初步算法代码。我们设计了一系列的提示词，旨在引导模型细致地比较样例代码和新生成的代码之间的差异。通过这种方法，我们使用模型识别出不同之处并评估这些变化对算法性能的具体影响。

进一步地，通过对比分析新旧代码的评估结果，指导模型理解哪些修改是合理和有效的。这种对比分析的过程至关重要，因为它不仅帮助模型识别出改进的方向，还让模型理解代码中哪些部分是关键性的，哪些部分可能需要进一步的调整或优化。

通过这种迭代的引导和分析，目标是让大型语言模型在设计遗传算法的过程中能够吸取精华，去除糟粕。这意味着模型将学会识别和保留那些对算法性能有显著提升的修改，同时放弃那些无效或负面的改动。随着迭代过程的不断深入，模型能够逐渐提升其设计的遗传算法的质量，最终达到一个更优秀的创新和优化的算法解决方案。这一过程不仅推动了算法性能的提高，而且加深了模型对遗传算法设计深层次原理的理解。

在模型完成所有逻辑推理后，使用结论性提示，如“因此，你最终建议的算法设计是...”，帮助模型整合所有推理步骤，生成最终的算法代码。这种提示促使模型将各个部分的推理整合起来，形成一个完整的、连贯的算法设计。

通过这种提示词设计方法，不仅提升了算法的性能，还使模型对算法设计任务有了更深刻的理解。这种方法的优势在于它能够引导大型语言模型在算法设计的迭代

过程中，生成更精确、效率更高的启发式算法。此外，这种方法还有助于提高模型的创造性，使其能够探索更广泛的设计空间，发现创新的算法结构。

3.3 算法自动设计流程

实验设计基于图 1 流程图中所示的结构，通过一系列步骤，从问题描述到算法性能评估，再到基于反馈的迭代优化，探索和优化大型语言模型生成启发式算法的过程。实验中使用启发式算法中的遗传算法作为初始代码样例进行迭代实验。形式为

$$\text{LLM}(I, f(x_1), \ldots, f(x_k))) \rightarrow x_{k+1} \tag{1}$$

其中，$f(x_k)$ 将第 k 个任务示例转换为自然语言提示，I 为任务描述，LLM 为使用大语言模型得到的输出结果。

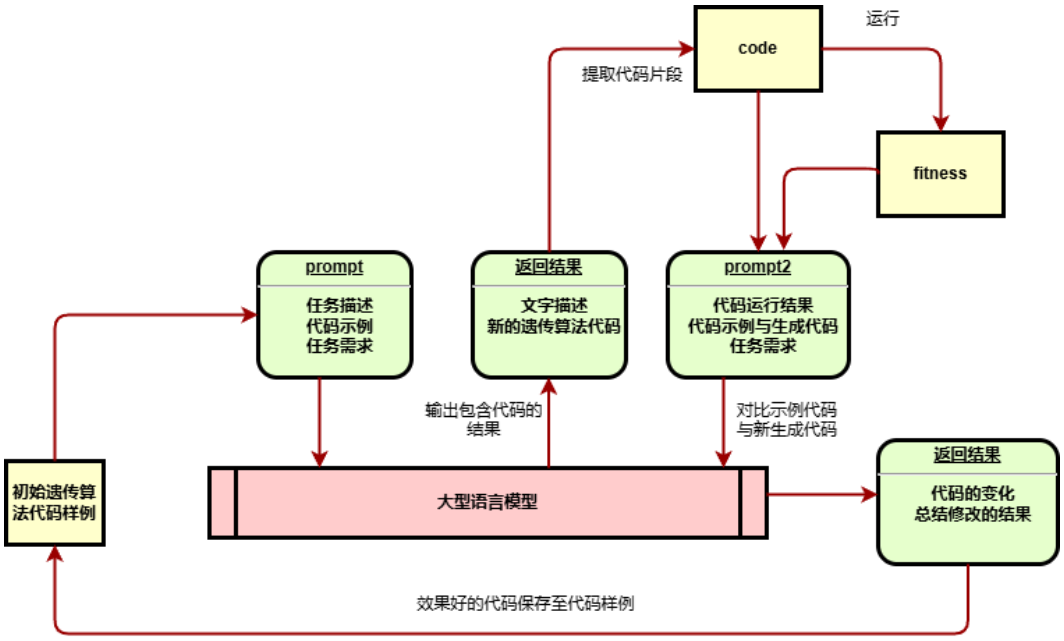

图 1 流程图

首先，定义一个启发式算法设计问题，详细描述算法需要解决的具体任务，为大语言模型提供一个明确的任务目标，确保生成的算法能够针对特定的问题进行优化。在本实验中定义的问题是解决 CEC2005 中的 f1 等最小化问题[3]，f1 问题的数学表达式为

$$f(x) = \sum_{i=1}^{D} z_i^2 + f_{\text{bias}}, \quad (z = x - o,\, x = [x_1, \ldots, x_D],\, o = [o_1, \ldots, o_D]) \tag{2}$$

其中 x 是决策变量，D 是 x 的维度，o 是偏移的全局最优值。具体地，D 的值设定为 30。x 的值的范围是 $[-100, 100]$。描述将以自然语言的形式输入到大型语言模型中。

我们首先准备三个不同但有效的初始遗传算法代码样例，这些样例将作为算法设计的起点。这些样例将被加入到一个存储代码的数组中，以便在迭代过程中进行选择和参考。我们将在每次迭代结束时，根据适应度评估的结果，保存最优的 5 个算法作为下一次迭代的初始样例。这样做可以确保我们的模型能够从多样化的算法中学习和生成新的算法，从而增加创新性。

遗传算法分为导入包、初始化参数、执行的方法和实现了用于解决优化问题的遗传算法的 solve() 函数，因此可以让 Code Llama 模型能够专注于对核心算法代码进行修改和创新。遗传算法遵循一个标准化的结构，包括导入必要的包、初始化参数、执行的方法以及一个专门为解决特定优化问题而设计的 solve() 函数。solve() 函数将利用 DEAP（Distributed Evolutionary Algorithms in Python）库来实现遗传算法的关键环节，包括初始化种群、评估个体、执行选择、交叉、变异等步骤，并最终返回最佳个体的适应度值。

solve() 函数定义了种群大小，遗传算法运行代数，交叉概率，变异概率等超参数。为了避免直接修改遗传算法的超参数，如种群大小、运行代数、交叉概率和变异概率等，我们将在提示中限制对这些参数的修改。这是因为我们希望模型能够专注于算法的核心逻辑和结构创新，而不是仅仅调整超参数来获得性能提升。为确保大语言模型对代码修改的有效性，超参数将在提示中被限制不能修改。

开始循环迭代时，从存储代码的数组中随机选取最优的 5 个算法之一作为初始遗传算法样例，这样可以避免大语言模型只参考一种算法而生成相似度极高的遗传算法导致失去创新性。首先输入大语言模型的第一个提示（Prompt），该提示包括算法问题的基本描述，初始遗传算法代码样例以及对大语言模型修改代码的限制。此提

示目的是引导大语言模型根据初始样例代码生成 1 个新的遗传算法代码。

对获得的新的遗传算法代码进行适应度计算，获得该遗传算法在最小化问题上的适应度作为评估遗传算法的结果。结合评估的结果，使用第二个提示（Prompt2）进行算法的迭代优化，目的是指导大语言模型调整和改进算法。采用零样本思维链提示优化算法设计，引导模型逐步思考，从而指导大语言模型理解该遗传算法的核心内容。通过引导大语言模型比较新遗传算法代码和遗传算法样例代码，根据二者的评估结果对比分析新代码与样例代码，总结代码的变化和修改结果并保留有效的部分，若最终评估结果好于样例代码，则将其保存至存储代码的数组中。

一次实验完成 200 轮迭代后，收集所有相关数据和测试结果，进行详细分析。为研究提供依据。通过上述流程，本课题将系统地评估大型语言模型在自动设计启发式算法方面的能力，探索和验证大型语言模型在启发式算法设计中的有效性。

4. 实验设置和结果分析

4.1 实验设置

4.1.1 实验环境

本课题采用 python3.10.14 版本搭建实验环境，因其广泛的应用、强大的库支持和成熟的社区，这为实验提供了一个稳定且可靠的平台。使用基于 Llama 2 模型的 Code Llama 大型代码语言模型系列，采用指令遵循模型 (CodeLlama-7b-Instruct-hf)[7]作为实验的大型语言模型。Code Llama 是专为代码生成和理解设计的先进模型，它能够处理复杂的编程任务，并生成高质量的代码。

最优化问题的测试来源于 CEC2005 测试集[3]，广泛用于评估优化算法的性能。它包含了一组具有挑战性的基准问题，这些问题设计用来测试算法的收敛速度、稳健性和效率。

4.1.2 提示词设置

在设计提示词的过程中，遵循了第 3.2 节中提出的提示词设计方法和第 3.3 节中概述的实验流程。我们的目标是确保提供给大型语言模型的指令不仅明确和清晰，而且还要能够精确地指导每一步的操作。这样做的目的是为了限制大型语言模型在算法修改上的自由度，确保它不会在指定的方法和参数之外做出不合理的更改。

为了提高大型语言模型在特定任务中的性能，在提示词中使用了前缀“您是这个任务（或领域）的专家”，这一策略根据[5]中的研究，已被证明是有效的。这种措辞可以引导模型以更高的专业标准来处理即将到来的任务，从而增强其输出的准确性和可靠性。

此外，提示词中使用了开始标签和结束标签来标识文本内容的范围。这样的做法有助于清晰地定义每个任务的开始和结束，使得大型语言模型能够更容易地理解任务的界限，并保持对任务焦点的集中。

在描述算法需要解决的具体任务时，设置了详细的提示词，这些提示词不仅包括了算法的目标和要求，还涵盖了解决问题所需的关键步骤和考虑因素。我们确保这些提示词能够引导大型语言模型深入理解问题的本质，同时遵循我们为算法设计设定

的指导原则和限制。

为了进一步规范大型语言模型的文本输出格式，我们制定了一套标准化的输出模板。这套模板定义了输出的结构，包括算法的每个组成部分应该如何呈现，以及应该如何清晰地表达算法的逻辑流程和关键决策点。通过这种方式，不仅能够确保输出的一致性和可读性，还能够方便后续的性能评估和算法迭代。

描述算法需要解决的具体任务的提示词设置如下：

```
prompt:
You are an expert metaheuristic algorithm design. Your task is to
design a metaheuristic algorithm to solve an optimization problem. The
problem's description and mathematical formula are specified in
'<problem description> </problem description>'. The performance metrics
for the algorithm are listed in '<metrics> </metrics>'. You have access
to the existing initialization code in '<existing code> </existing
code>', which you must read carefully but cannot modify. To help you
understand algorithm design requirements, examples of metaheuristic
algorithms are provided in '<examples> </examples>'. The criteria that
your algorithm design must meet are outlined in '<demands> </demands>'.
Finally, you need to complete the algorithm you design, starting from
the code snippet provided in '<your code> </your code>'.
<problem description>
This is the f1 problem from the CEC_2005 competition.
It's a minimization problem with the mathematical expression $_{f\left(
x \right) =\sum_{i=1}^{D}{z_i^2}+f\_bias,\left(z=x-o, x=\left[
x_1,x_2,...,x_D \right],o=\left[ o_1,o_2,...,o_D \right]\right)}$,
where $x$ is the decision variable, $D$ is the dimension of $x$ and $o$
is the shifted global optimum. The value of $D$ is set to 30
concretely. The range of the value of $x$ is $\[-100, 100\]$.
</problem description>
<metrics>
fitness value: the value of the function equation ${f\left( x
\right)}$. Smaller fitness value indicates better algorithm
performance.
</metrics>
```

以下提示词旨在对大型语言模型进行精确的指导和限制，确保其在算法修改过程中遵循既定的范围和规则。这些提示词的设计初衷是为了避免模型在未指定的领域进行过度的自由发挥，从而确保生成的算法不仅具有创新性，同时也保持了有效性和实用性。

提示词首先明确了算法修改的范围，即大型语言模型只能在既定的算法结构和方法论内进行修改。这意味着模型不能随意更改算法以外的逻辑，而应专注于优化现有算法结构中的部分。进一步地，提示词对大型语言模型能够使用的特定方法和参数进行了限制。大型语言模型被指导只考虑和应用在提示词中明确指出的那些方法和参数，这样有助于保持算法的一致性，并确保每次迭代都围绕既定目标进行。为了确保算法的稳定性和可靠性，提示词特别强调了对不合理修改的限制。大型语言模型被要求避免那些可能破坏算法性能或导致算法无法正常工作的修改。

限制提示词设置如下：

```
prompt:
<demands>
1. Do not modify the values of population_size and generation_number in
the algorithm. In your result, the value of population_size must be 100
and the value of generation_number must be 200.
2. You should refer to the example I provided and output your algorithm
results. The example uses a genetic algorithm, Your implementation
should follow the structured example yet significantly improve upon it.
Build an algorithm using a combination of various efficient and diverse
operators.
3. You should not merely replicate existing examples. Ensure that the
result you output have made certain changes compared to the example.
The result you output should be the algorithm I requested, in the
format of starting with 'def SOLVE(toolbox, lower_bound, upper_bound):'
and end with 'best_ind = tools.selBest(pop, 1)[0] \n return
best_ind.fitness.values[0]'.
</demands>
```

引导大语言模型比较新遗传算法代码和遗传算法样例代码，评估新遗传算法代码相对于样例代码的改进和优势。在收集了性能数据后，使用模型深入分析新代码与

样例代码之间的差异。这不仅包括代码结构的变化，如增加了新的操作、改进了现有操作或优化了数据结构，还包括算法逻辑的调整，如改变了选择、交叉、变异等遗传操作的策略。通过对比分析，总结那些对性能有积极影响的代码修改。这可能涉及到对特定遗传算法组件的微调，或是引入了全新的概念和策略。最后引导大语言模型记录这些有效修改，用于下一次的迭代优化设计。

引导模型逐步思考，理解新算法的变化的提示词如下：

```
prompt:
<demands>
1.Let us think step by step. First write down the differences between
these two algorithms. Then compare the example code with the new code.
Write down you have done what modifications on the algorithm and tell
me why you do these modifications.
2.You should format your output as "My modifications are:", "My advice
is:" and then output one short but specific suggestion. Make a
conclusion on the modifications you have made based on the fitness of
example code and new code, finally keep the valid parts in your
algorithm.
</demands>
```

在少数情况下，大语言模型不遵循提示词的指令，修改了限制之外的参数数值，导致出现不合理的结果。我们使用正则表达式匹配所有关键参数的数值，当匹配到大语言模型生成算法修改了不合适的参数数值时，将废弃此次结果并自动修改提示词，在下一步对比比较新遗传算法代码和遗传算法样例代码时使用另一套提示词，再次强调不允许修改的参数以及参数数值。

```
prompt:
<demands>
Do not modify the values of {param[i].name} in the algorithm. The value
of {param[i].name} must be {param[i].value}
</demands>
```

4.2 评价指标

性能评估策略定义了如何衡量所设计算法的性能。要进行性能评估，首先应选择一个性能度量指标，然后根据该指标评估性能，并通过性能比较算法优劣。本实验中使用大语言模型进行遗传算法设计。

遗传算法的流程如图 2 所示。遗传算法的基本思想是根据问题的目标函数构造一个适应度函数，对于种群中每个个体进行评估、选择、交叉、变异。遗传算法的性能评估方法是通过多轮繁殖选择适应度最好的个体作为问题的最优解。适应度是评价个体优劣的函数 $f(x)$，实验中使用 CEC2005 中目标问题的函数，最小化优化问题的函数可表示为 $f(x) = -\min \sum C(\hat{y}, y)$ ，其中 C 为损失函数。

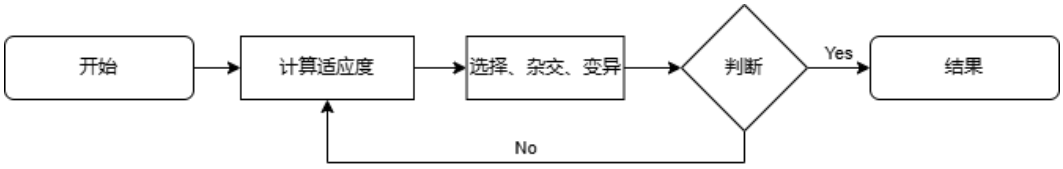

图 2 遗传算法的流程

实验中针对使用的 CEC2005 的 f1，f2,f6,f9 问题编写如下表中算法 1，算法 2，算法 3，算法 4 中的评估函数，求解遗传算法在当前最优化问题中种群最优个体的适应度大小。评估函数求得的适应度大小是当前遗传算法好坏的性能评估指标，在本实验中的最小化优化问题上，适应度越小即意味着遗传算法性能越优。

算法 1 CEC2005 f1 评估函数

输入: $bias$, o, $GENE\_LENGTH$
初始化 $tmp \leftarrow 0$, 初始化长度为 $GENE\_LENGTH$ 的数组 z
for $j = 0$ to $len(individual) - 1$ **do**
    $z[j] \leftarrow individual[j] - o[j]$
end for
for $j = 0$ to $len(z) - 1$ **do**
    $tmp \leftarrow tmp + z[j]^2$
end for
return $tmp - bias$

算法 2 CEC2005 f2 评估函数

输入: $bias$, o, $GENE\_LENGTH$
初始化 $tmp \leftarrow 0$, 初始化长度为 $GENE\_LENGTH$ 的数组 z
for $j = 0$ to $len(individual) - 1$ **do**
    $z[j] \leftarrow individual[j] - o[j] + 1$
end for
for $j = 0$ to $len(z) - 2$ **do**
    $tmp \leftarrow tmp + 100 \cdot (z[j]^2 - z[j+1])^2 + (z[j] - 1)^2$
end for
return $tmp - bias$

算法 3 CEC2005 f6 评估函数

输入: $bias$, o, $GENE\_LENGTH$
初始化 $tmp \leftarrow 0$, 初始化长度为 $GENE\_LENGTH$ 的数组 z
for $j = 0$ to $len(individual) - 1$ **do**
    $z[j] \leftarrow individual[j] - o[j] + 1$
end for
for $j = 0$ to $len(z) - 2$ **do**
    $tmp \leftarrow tmp + 100 \times (z[j]^2 - z[j+1])^2 + (z[j] - 1)^2$
end for
return $tmp - bias$

算法 4 CEC2005 f9 评估函数

输入: $bias$, o, $GENE\_LENGTH$
初始化 $tmp \leftarrow 0$, 初始化长度为 $GENE\_LENGTH$ 的数组 z
for $j = 0$ to $len(individual) - 1$ **do**
    $z[j] \leftarrow individual[j] - o[j] + 1$
end for
$result \leftarrow \sum(z^2 - 10 \cdot \cos(2\pi z) + 10) + f_{\text{bias}}$
return $result$

4.3 实验结果展示与分析

实验探索大型语言模型生成启发式算法的可能性。实验中使用 Code Llama 模型进行遗传算法的设计，并验证大语言模型能够针对不同的最优化问题设计出不同的遗传算法以达到接近全局最优解的效果。

本实验使用的最小化问题目标是找到一个使得适应度函数值最小的解决方案。CEC2005 测试集包含了用于评估和比较优化算法性能的 25 个标准函数集合[3]。用于测试的最优化问题来源于 CEC2005 中的 f1,f2,f6,f9 问题。

CEC2005 的 f1 问题表达式为：

$$f(x) = \sum_{i=1}^{D} z_i^2 + f_{\text{bias}} \tag{3}$$

CEC2005 的 f2 问题表达式为：

$$f(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j \right)^2 + f_{\text{bias}} \tag{4}$$

CEC2005 的 f6 问题表达式为：

$$f(x) = \sum_{i=1}^{D-1} \left(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2 \right) + f_{\text{bias}} \tag{5}$$

CEC2005 的 f9 问题表达式为：

$$f(x) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos(2\pi z_i) + 10 \right) + f_{\text{bias}} \tag{6}$$

其中 $z = x - o$, $x = [x_1, \ldots, x_D]$, $o = [o_1, \ldots, o_D]$。根据以上实验问题的表达式设计出相应的评估函数，作为验证遗传算法优劣的评价指标。

运行迭代设计的遗传算法并使用评估函数的结果如图所示，图 3,4,5,6 展示了使用大型语言模型设计遗传算法中最佳适应度随迭代次数演进的变化。X 轴代表了算法的迭代次数，Y 轴表示每轮迭代中保存的代码里评估结果最优的适应度值。最优适应度值下降说明了大语言模型生成了对当前最小化问题的更有效的遗传算法。

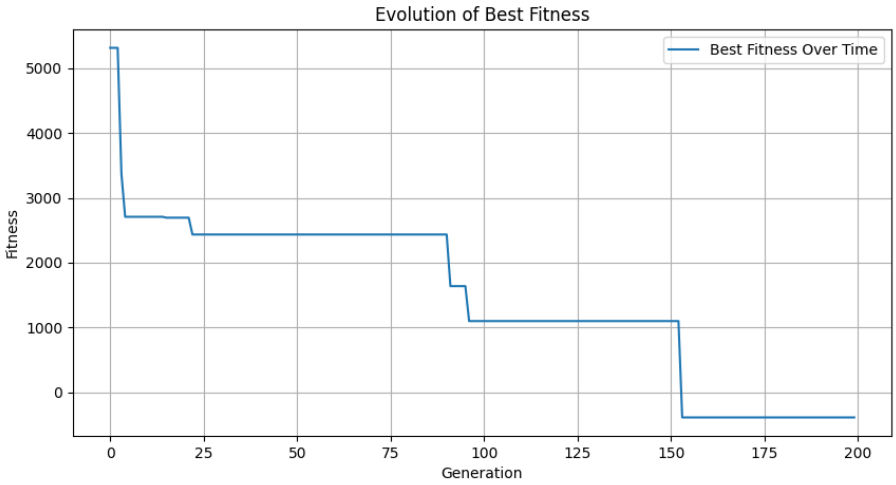

图 3 遗传算法 CEC2005 f1 中最佳适应度随迭代次数的变化

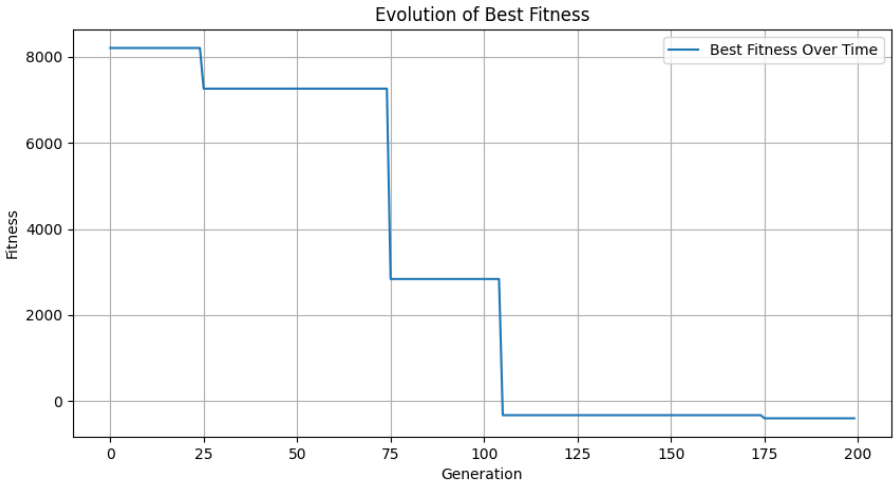

图 4 遗传算法 CEC2005 f2 中最佳适应度随迭代次数的变化

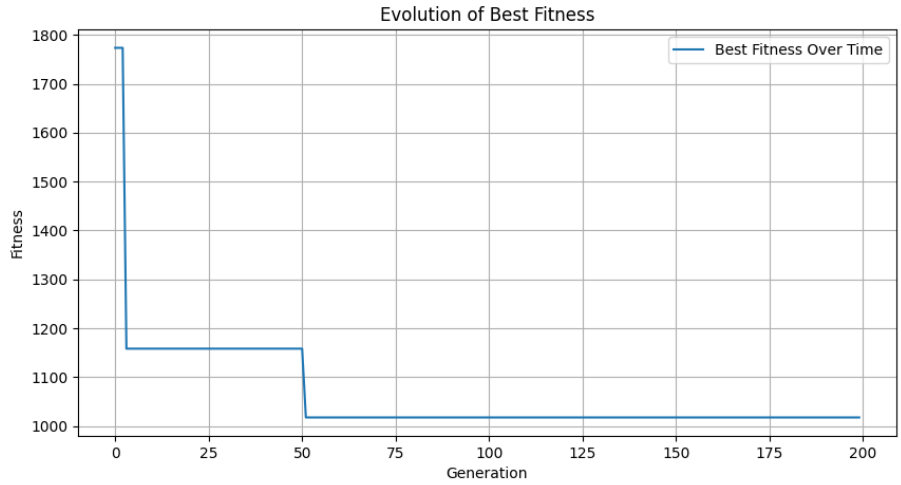

图 5 遗传算法 CEC2005 f6 中最佳适应度随迭代次数的变化

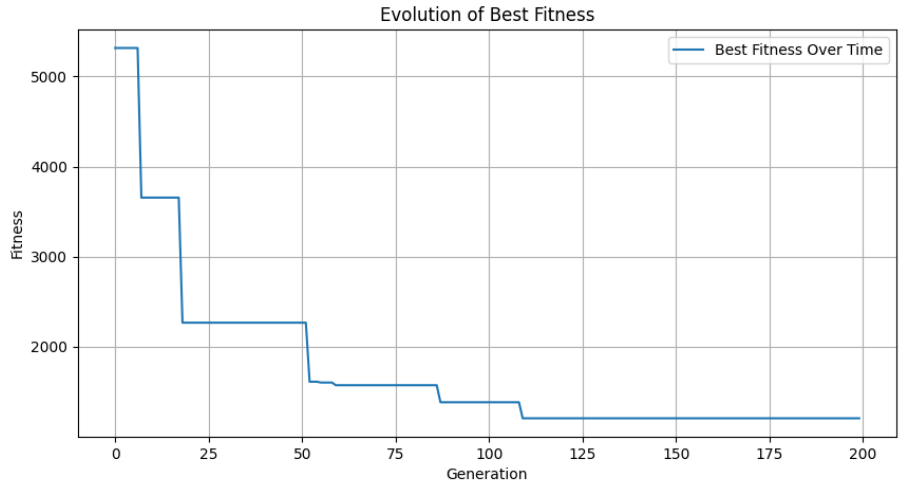

图 6 遗传算法 CEC2005 f9 中最佳适应度随迭代次数的变化

图中的折线显示了从第 1 代到第 200 代的最佳适应度的变化。可以看到，适应度值在前 100 代中快速下降，说明大语言模型在初期很快找到了较好的对初始遗传算法的修改方案，后期适应度下降开始放缓，因为每轮迭代最优的遗传算法接近局部最优解或全局最优解，直到最后收敛至全局最优解附近。

在上述实验中，表 1 展示了大型语言模型在处理 CEC2005 测试集中的四个不同优化问题（f1, f2, f6, f9）时的性能表现。如表所示，每个问题的初始适应度值和经过优化后的最终适应度值有显著的改进，表明模型能有效地减少适应度函数的值，接近理论上的最优解。问题 f1 和 f2 的最终适应度值接近其理论最优解，说明模型在这

些问题上表现出较强的优化能力。而问题 f6 和 f9 的最终适应度虽然没有达到理论最优，但相比于初始适应度也有了显著的改善。这些结果反映出大型语言模型在遗传算法的设计中不仅能快速找到较好的解决方案，还能逐渐优化算法以逼近最优解。此表也突显了大型语言模型在不同类型的优化问题上的适应性和效率，展示了其在自动化算法设计领域的潜力。通过迭代学习和优化，模型表现出对各种问题结构和解决策略的理解，能够根据问题的特性调整其策略，以达到最佳的性能表现。

表 1 遗传算法应用于 CEC2005 优化问题的性能比较

Question	Initial Fitness	Final Fitness	Theoretical Optimal Solution
F1	5653.95	-400.33	-450
F2	8843.45	-403.66	-450
F6	1957.85	1017.70	390
F9	5314.88	1206.01	390

以上图表均展示出初始迭代阶段适应度值下降快的结果，表明模型能够迅速适应并找到初步的改进方案。在中后期阶段，适应度值趋于平稳，模型生成的遗传算法已经接近或达到全局最优解。实验结果证明，大型语言模型在生成和优化遗传算法方面具有一定的优势。通过适当的提示和迭代优化，模型能够根据不同的最优化问题，自动生成高效的启发式算法。对于不同的最优化问题，分别观察图 3，4，5，6 可以发现适应度值的演变趋势有所不同，但总体表现出良好的优化效果。这表明大型语言模型具有较强的泛化性，能够处理多种复杂的优化问题。

5. 结论

本课题的研究成果揭示了利用基于 Llama2 的大型代码语言模型 Code Llama 在自动化启发式算法设计领域的潜力与实际效果。通过一系列设计的实验，不仅验证了自动化设计方法的可行性，还展示了其在提升遗传算法性能方面的巨大潜力。

在本课题中，采用了 Code Llama 这一大型语言模型作为自动化设计的核心工具。实验设计遵循了科学严谨的方法论，确保了实验结果的可靠性和有效性。实验的执行过程包括了问题的建模、算法的生成、性能的评估以及结果的分析等多个环节。

实验结果清楚地表明，大型语言模型能够自动生成和优化遗传算法，尤其在算法设计的初期阶段，能够显著加快算法性能的提升。这一点在处理复杂的优化问题时表现得尤为突出，模型能够快速提出创新的解决方案。

通过精心设计的提示策略，成功地引导了 Code Llama 进行有效的迭代生成。这些策略不仅帮助模型理解了问题的核心要求，还确保了生成的算法在结构和逻辑上的合理性。提示策略的使用显著提高了自动化设计过程的效率和成功率。

在给定的最小化问题上，由 Code Llama 自动生成和优化的遗传算法展现出了优越的性能。相比于初始的启发式算法示例，使用大语言模型成功设计出评估结果更优秀的启发式算法。

本课题的研究成果不仅展示了大型语言模型在自动化启发式算法设计领域的应用潜力，还为未来的研究提供了新的方向。随着技术的不断进步和模型能力的进一步提升，大型语言模型将在解决更多复杂优化问题中发挥更大的作用。尽管自动化设计展现出巨大的潜力，但认识到这一领域仍面临许多挑战，包括如何进一步提高模型的泛化能力、如何处理更大规模的问题、以及如何确保生成算法的可用比例和可解释性等。这些挑战将是未来研究的重点。

综上所述，本课题深入探讨了基于大模型的自动化启发式算法设计。期待这一研究能够有助于对大型语言模型能力的探索和理解，以及挖掘大型语言模型的更多能力。大型语言模型的相关研究和自动化设计将是人工智能和优化算法发展的关键。

参考文献

[1] CHEN M, TWOREK J, JUN H, et al. Evaluating Large Language Models Trained on Code[Z]. 2021. arXiv: 2107.03374.

[2] WEI J, WANG X, SCHUURMANS D, et al. Chain-of-thought prompting elicits reasoning in large language models[C]. in: vol. 35. Advances In Neural Information Processing Systems, 2022: 24824-24837.

[3] SUGANTHAN P N, HANSEN N, LIANG J J, et al. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization[R]. 2005. KanGAL Report, 2005: 2005.

[4] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]. in: vol. 30. Advances In Neural Information Processing systems, 2017.

[5] ZHAO W X, ZHOU K, LI J, et al. A Survey of Large Language Models[Z]. 2023. arXiv: 2303.18223.

[6] TOUVRON H, LAVRIL T, IZACARD G, et al. LLaMA: Open and Efficient Foundation Language Models[Z]. 2023. arXiv: 2302.13971.

[7] ROZIÈRE B, GEHRING J, GLOECKLE F, et al. Code Llama: Open Foundation Models for Code [Z]. 2024. arXiv: 2308.12950.

[8] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[C]. in: vol. 33. Advances in Neural Information Processing Systems, 2020: 1877-1901.

[9] WONG W, MING C I. A review on metaheuristic algorithms: recent trends, benchmarking and applications[C]. in: 2019 7th International Conference on Smart Computing & Communications (ICSCC). 2019: 1-5.

[10] HOLLAND J H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence[M]. MIT Press, 1992.

[11] DORIGO M. Positive feedback as a search strategy[R]. Technical Report, 1991: 91-16.

[12] VAN LAARHOVEN P J, AARTS E H, van LAARHOVEN P J, et al. Simulated annealing[M]. Springer, 1987.

[13] ZHAO Q, DUAN Q, YAN B, et al. Automated Design of Metaheuristic Algorithms: A Survey[J]. Transactions on Machine Learning Research, 2023.

[14] SWAN J, ADRIAENSEN S, BROWNLEE A E, et al. Metaheuristics “in the large”[J]. European Journal of Operational Research, 2022, 297(2): 393-406.

[15] LOURENÇO H R, MARTIN O C, STÜTZLE T. Iterated local search[G]. in: Handbook of Metaheuristics. Springer, 2003: 320-353.

[16] EIBEN A E, SMITH J E. Introduction to evolutionary computing[M]. Springer, 2015.

[17] LÓPEZ-IBÁÑEZ M, DUBOIS-LACOSTE J, CÁCERES L P, et al. The irace package: Iterated racing for automatic algorithm configuration[J]. Operations Research Perspectives, 2016, 3: 43-58.

[18] YE F, DOERR C, WANG H, et al. Automated configuration of genetic algorithms by tuning for anytime performance[J]. IEEE Transactions on Evolutionary Computation, 2022, 26(6): 1526-1538.

[19] MENG W, QU R. Automated design of search algorithms: Learning on algorithmic components [J]. Expert Systems with Applications, 2021, 185: 115493.

[20] KOJIMA T, GU S S, REID M, et al. Large language models are zero-shot reasoners[C]. in: vol. 35. Advances in Neural Information Processing Systems, 2022: 22199-22213.

致谢

在这段学术旅程的尾声，我满怀感激之情，向所有给予我帮助和支持的人表达最深的谢意。

首先，我要特别感谢我的导师史玉回教授。史教授不仅在学术上给予我宝贵的指导，更在精神上给予我莫大的鼓励。在我遇到研究瓶颈和挑战时，他的耐心和智慧总是引领我找到解决问题的方法。他的严谨治学态度和对学术的无限热爱，对我影响深远，我将终生受益。

其次，我要衷心感谢赵琪老师。赵老师在实验设计和论文撰写方面给予了我巨大的帮助。他的专业知识深厚，对细节的关注和对质量的追求，使我的研究工作得以顺利进行。每当我遇到难题，赵老师的细心指导总能帮助我找到突破口，他的严谨和专业让我受益匪浅。

我还要感谢我的同学和朋友们。在这段研究生涯中，他们的陪伴和支持是我前进的动力。我们一起讨论问题，分享知识，互相鼓励，共同成长。他们的友谊和互助，使我的研究生活充满了乐趣和温暖。

此外，我还要感谢我的家人。他们一直是我最坚强的后盾。在我忙于学业和研究时，家人的理解和支持让我能够全心投入。他们的爱和鼓励是我不断前进的力量源泉。

最后，我要感谢所有给予我帮助和启发的人。每一位教授、同学、朋友，以及所有在这段旅程中给予我支持的人，都值得我深深的感激。没有你们，我的学术之路不会如此丰富多彩。

在未来的道路上，我将继续努力，不断学习，不辜负大家的期望和帮助。再次感谢所有给予我帮助的人，愿我们的未来都能充满希望和成功。