第1章 概 述

1.1 研究背景与意义

随着信息技术的飞速发展，人们对计算资源的需求呈现出日益增长的趋势，从社交媒体到在线购物，以至于智能家居，都依赖于计算的支持。从2000年至今，全球互联网用户数量已经翻了一番，超过了40亿。面对这一巨大的数字化浪潮，各行各业对计算能力的需求也不断攀升。根据国际数据公司(IDC)的最新研究，全球数据量每两年翻一番，到2025年，全球数据总量将达到175ZB，相当于每秒钟产生2.5PB的数据。这表明传统的计算模式已经无法满足日益增长的计算需求，因此，人们开始寻求更加智能、高效的计算方法和资源管理策略。随之而来的是智能计算方法的崛起，作为一种新兴的计算范式，它通过引入人工智能和机器学习的技术，旨在提高计算系统的自适应性、灵活性和效率，以更好地满足各种计算任务的需求。在当前的计算环境中，我们面临着多样化、复杂化和大规模化的计算需求。传统的资源管理方法往往无法适应这种变化，无法有效地利用计算资源，导致资源浪费和性能下降。因此，研究智能计算方法及其资源管理应用显得尤为重要。

在当前研究中，首要问题是解决多目标资源管理的挑战。随着计算任务多样性和复杂性的增加，传统方法难以在不同目标之间找到平衡。

智能计算方法通过引入优化算法和多目标优化技术，创新性地应对多目标资源管理问题，实现了更有效的资源配置和利用。同时，随着移动设备的广泛普及和移动应用的蓬勃发展，移动边缘计算成为新兴计算范式，提高了计算效率并降低了时延。然而，在这高度动态的环境中智能地管理计算资源，以满足实时业务和应用智能的需求，仍是亟待解决的问题。

此外，智能计算方法在面向混合云的资源管理中发挥着关键作用。混合云环境融合了公有云和私有云的优势，具有更大规模和更灵活的计算资源。通过智能计算方法，实现混合云环境下计算资源的动态调度和优化，提升了系统的可靠性和稳定性。在这个计算时代，我们迫切需要发挥智能计算在混合云环境中的潜力。另外，面向云边协同计算的资源管理也备受关注。在这一框架下，云中心和边缘计算节点协同工作，实现计算任务的分布式处理。智能计算方法的引入为协同计算提供了更灵活、高效的调度和优化策略，以更好地适应不同计算需求和网络状况。在迅速发展的计算环境中，智能计算为云边协同计算带来了新的可能性。

在智能交通领域，专注于车载边缘计算的资源管理显得尤为重要。在车载边缘计算环境中，智能计算方法的应用不仅促进了车辆之间的信息交互和协同计算，还提升了交通系统的智能化水平和整体效率。此外，面向能源优化的资源管理成为研究的另一关键方向。随着能源问题日益凸显，当务之急是在计算过程中实现能源的有效利用。通过智能计算方法，运用优化算法和智能调度，成功实现计算资源的能源优化，有效降低对环境的不良影响。这一研究方向在当前科技发展中具有重要的实际意义。

本研究主要涉及以下问题。首先，在多目标资源管理方面，通过智能计算方法解决多目标优化问题；其次，在移动边缘计算环境中，研究智能计算方法在资源管理中的应用，以提高实时性和效率；再者，在混合云环境中，通过智能计算方法实现计算资源的动态调度；然后，在云边协同计算中，探索智能计算方法在任务调度和资源协同优化中的应用；最后，在车载边缘计算和能源优化方面，通过智能计算方法提升交通系统的智能性和计算资源的能源效率。综上所述，本书将从面向多目标资源管理问题的求解、面向移动边缘计算的资源管理、面向混合云的资源管理、面向云边协同计算的资源管理、面向车载边缘计算的资源管理、面向能源优化的资源管理六个方面展开研究，旨在通过智能计算方法及其资源管理应用，更好地解决当前计算环境中面临的多样化、复杂化和大规模化的计算需求，推动计算科学的发展和应用。

1.2 国内外研究现状

1.2.1 确定式方法

确定式方法主要指按照一种确定的方式来进行选择或推理的一类算法，这类算法的计算结果不受随机因素的影响，可以精确地求解优化问题和函数计算。确定式方法可以分为几个不同的类型：第一类为基于数学模型的算法，如线性规划和非线性规划等；第二类为基于动态规划的算法，如最短路径算法和背包问题算法等；第三类为基于贪心策略的算法，如最小生成树算法和哈夫曼编码算法等。

基于数学模型的确定式方法是指根据数学模型的结构和性质，设计出能够有效求解该模型的计算方法。Zhang 等基于双层混合整数规划（BiMIP）模型，提出了使系统容量规划和运行成本最小化，以及基站运行成本最小化的双层联合优化问题，提出了一种具有 ESS 规划需求的大型 PV 集成 5G 基站 SES 系统双层联合优化问题。Yann 等结合用户的 PV 和 ESS，提出了一个非线性的优化问题，目标是优化用户的用电成本，并使用动态规划进行求解。针对能量调度管理，Aluisio 等对 EV 日常运行和 PV 产量预测的运营成本进行了研究，提出了一种包含 EV 的直流微电网运行优化规划的方法。Luo 等重点从多类型充电设施出发，考虑其年度社会总成本，建立了混合整数二阶段规划优化问题。Tint 等则在 TDMA 时分多址系统中，利用凸优化研究了多用户接入单个边缘服务器场景下任务卸载的最优策略。为了解决边缘服务器计算资源有限和传输干扰等问题，Huang 等针对多移动设备场景下的卸载问题，考虑信道和计算资源成本的同时，通过制定有效的资源分配策略，最小化用户的总时延。Tao 等以任务卸载比例作为优化解，在传输时延、计算时间和执行单任务所消耗的功率受到限制的条件下，构建了以最小化系统功耗为优化目的的凸优化模型，并求解出任务卸载的最佳比例，提出了一种有效节省系统功率开销的优化任务卸载策略。Lyu 等针对无人机使能的移动边缘计算系统，利用块坐标下降算法联合优化带宽分配、任务卸载时间分配和无人机轨迹来最大化整个系统的计算效率，整个问题被分解为三个次优化问题，并采用连续凸优化技术求解。Guo 等提出并设计了一种新的基于李雅普诺夫优化的部分计算卸载算法，以求解由单个边缘服务器和多个用户组成的移动边缘计算系统的数据分区计算任务卸载问题。Yan 等提出了一种移动边缘计算系统的服务定价方案，通过一个低复杂度的算法联合优化基站的定价和服务缓存，以协调服务缓存决策并控制蜂窝网络中

无线设备的任务卸载行为。为了最小化边缘计算系统的成本，Wu 等提出了一个以 IEEE 802.11p 作为车辆间通信传输协议的最优任务卸载方案，其中综合考虑了传输延迟、计算延迟，以最大化系统的长期回报。结合 Dai 等将负载均衡与卸载问题化为一个整数非线性规划问题，以最大化系统效率，实验表明该策略在系统效用方面显著优于基准策略。Li 等提出了基于线性化的分支定界算法和最近舍入整数算法的计算卸载算法来解决静态和动态任务的原始问题。这类方法是一种可以明确表示知识真假的方法，利用了解空间的特性去构建目标函数和约束条件，然后用数学分析和算法来求解最优解或近似最优解。

基于动态规划的确定式方法是一种通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。Abrishami 等基于局部关键路径算法提出了 IaaS 云局部关键路径算法（IaaS Cloud Partial Critical Paths，IC-PCP）和带截止日期分配的 IC-PCP 算法（IaaS Cloud Partial Critical Paths with Deadline Distribution，IC-PCPD2）。Calheiros 等在此基础上提出了考虑副本的增强型 IC-PCP 算法（Enhanced IC-PCP with Replication，EIPR），利用租赁资源的空闲时间块设置任务副本，减轻任务截止期的压力，最大化资源利用率。徐等提出了一种分布式协同控制策略，用于协调 ESS 以维持供需平衡并最大限度地减少与充电/放电效率低下相关的总功率损失。Zhao 等提出了一种基于李雅普诺夫优化方法的新的卸载框架，用于跨边缘协作计算，该框架可以在保证设备电池能量可靠的同时，最小化任务卸载的时间。Wang 等针对物联网中的软件定义移动边缘计算问题，提出了一种基于分布式深度学习的算法，该算法可以同时进行计算卸载和资源分配，目的是在分布式密集物联网环境中，最小化加权的延迟和功耗效用。Jeong 设计了一种轻量级的迁移系统，该系统可以将 DNN 层从资源受限的移动设备迁移到边缘服务器，从而在支持 Web 的设备上实现 DNN 计算的边缘化。为此，他提出了一种 DNN 分区算法，该算法可以有效利用边缘资源并减少系统响应时间。Kang 等设计了一种轻量级的调度程序，该调度程序可以适应各种 DNN 模型结构，并以神经网络层的粒度自动划分 DNN 模型，使 DNN 应用可以部分执行在物联网设备端或云中心。Liu 等将多个计算密集型车辆应用卸载到路侧单元中，并将每个应用细分为具有任务依赖性的多个任务，提出一种高效的任务卸载调度算法，以最小化多个应用程序的平均完成时间。在分解策略求解高维多目标问题中，多种分解方法的提出都是基于将原始问题进行分割，Jensen 首次提出辅助目标的概念，认为辅助目标不仅能够引导粒子往良好解集的方向搜索，而且能够帮助算法摆脱局部最优。这类算法是一种利用动态规划思想来求解优化问题的算法，将原问题分解为相对简单的子问题，并将子问题的解保存起来，以避免重复计算的方法。

基于贪心策略的确定式方法是一种在每一步都做出当前最优选择的算法。在异构计算环境中，任务调度是一类重要的优化问题。Topcuoglu 等在最早完成时间（Earliest Finish Time，EFT）算法的基础上，提出了一种考虑资源异构性的启发式方法，称为异构最早完成时间（Heterogeneous Earliest Finish Time，HEFT）算法，该算法通过两个阶段的任务排序和处理器选择，极大提高了资源利用率和任务执行效率。Yan 等提出了一种解决插电式 EV 功率分配和充电协调的两阶段方案，解决了充电站能量管理的两个问题。在智能电网中，具有分布式能源资源（Distributed Energy Resources，DER）的用户可以向电网卖电，从而获得收益。Hopkins 等则是考虑了这种场景，构造了一个线性规划问题来最大化用户的收益，同时满足电网的需求和用户的限制。在移动边缘计算（Mobile Edge Computing，

MEC)中,多个用户可以通过多个小基站(Small Base Station,SBS)共享一个边缘服务器的计算资源,从而提高移动应用的性能。然而,这种场景也带来了无线通信的干扰问题,影响了任务卸载的效率。Wang 等针对这一问题,提出了一种分步优化的卸载决策,计算资源分配以最小化系统能耗和时延的加权和。同时,他们利用图染色法解决无线资源分配的问题,并最小化用户间干扰。Han 等使用三个流水线对 DNN 模型进行深度压缩:剪枝、量化训练和哈夫曼编码。在多目标优化问题中,Li 提出了一种在标准优势关系中增加偏见的机制,并认为这种基于偏差阈值的控制策略可以克服帕累托优势缺乏对约束条件的偏好。Shi 等针对组合优化问题,提出将原目标分解为两个具有可控相关性的子目标。这类算法的计算结果不受随机因素的影响,可以精确地求解优化问题和函数计算。

1.2.2 启发式方法

随着科技的迅猛发展,计算机科学和人工智能领域面临着越来越复杂的问题。这些问题可能涉及大规模的数据、高度复杂的决策树,以及难以建模的实际环境。传统的穷举式求解方法在这些情境下变得不切实际,因为它们在搜索整个解空间时需要耗费巨大的计算资源和时间。正是在这一背景下,启发式方法作为一种基于经验和直觉的问题解决策略应运而生。其灵感来自人类在解决问题时的启发性思维方式,通过引入经验、直觉和启发性规则,启发式方法试图在有限时间内找到一种满意的解决方案,而不是追求完全精确的解。

启发式方法是一种基于经验、直觉和问题特性的启发性思维方式,用于在有限时间内找到问题解决方案的有效策略。这种方法通常应用于那些由于庞大的搜索空间、高度复杂的问题结构或实时性要求,使得传统的精确求解方法难以应对的情境。在启发式方法中,“启发式”一词强调了问题解决过程中的灵活性和智能性。与传统的穷举式方法不同,启发式方法通过引入一系列启发性规则,这些规则可能基于领域知识、经验或问题的结构,有目的地引导搜索过程。这使得启发式方法能够在大规模的问题空间中,通过聪明的搜索策略,寻找到一个可行解或接近最优解。启发式方法的应用领域非常广泛,包括但不限于优化问题、规划问题、机器学习、模式识别、智能交通系统等。其灵活性使得启发式方法能够适应各种问题的特性,而其高效性则使其成为解决实际问题时的一种重要工具。总体而言,启发式方法在求解那些由于问题的复杂性和多样性而无法直接采用传统方法求解的实际问题时,为我们提供了一种创新而高效的思维方式。在不断演化的科技时代,启发式方法的发展和应用有望为我们提供更为智能和适应性强的问题解决途径。

启发式方法的核心思想在于模拟人类智能的启发性思维方式,通过引入经验、直觉和问题特定的启发性规则,有目的地搜索问题解空间,以在有限时间内找到一种满意的解决方案。主要包括:①模拟人类智能的启发性思维方式。启发式方法的灵感来源于观察和理解人类在解决问题时的思考过程。人们在面对问题时往往会运用积累的经验、基于直觉的判断以及对问题的特定了解。启发式方法试图通过对这种启发性思维方式的模拟,将人类的智能引入计算机问题求解中。②基于经验和直觉的引导搜索。启发式方法通过引导搜索过程,避免对整个解空间进行穷尽式的搜索。它依赖于先前的问题求解经验、领域专业知识,以及问题本身的特性,通过这些信息引导搜索算法更有针对性地探索可能的解决方案。③灵活性与问题自适应性。核心思想还包括启发式方法的灵活性,即其能够适应各种不同类型的问题。启发性规则通常能够根据具体问题的特点进行调整和优化,使得启发式

方法在处理不同问题时表现出良好的适应性。④关注解空间的局部优化。与追求全局最优解的精确方法不同，启发式方法通常关注在当前情境下看似最优的解决方案。这种局部优化的策略可以加速搜索过程，尤其适用于大规模、复杂问题的求解。⑤处理不确定性和复杂性。启发式方法被设计用于应对现实问题中的不确定性和复杂性。通过结合人类智能的模拟、经验的应用和问题特定的启发性规则，它能够在实际问题的挑战性环境中找到相对有效的解决方案。总的来说，启发式方法的核心思想在于通过模拟启发性思维，结合经验和问题特性，以一种更加灵活和高效的方式进行问题求解。这种方法在应对实际问题时体现出了独特的优势，使得其在各个领域中得到广泛应用。

启发式方法并非单一的算法或策略，而是包括多种不同的方法和技术。启发式方法主要包含以下几类：①贪婪算法。贪婪算法是一种简单而直观的启发式方法，它在每一步选择当前看似最优的解，而不考虑全局最优。这种方法特别适用于那些问题，其中每个步骤的决策只对局部解的质量有影响，如某些最短路径问题或分配问题。②模拟退火。模拟退火是一种基于物理冷却过程的启发式方法。通过引入随机性，模拟退火算法在搜索过程中接受可能劣于当前解的解，以避免陷入局部最优。这种方法在全局优化和避免陷入局部最优解的问题中表现出色。③遗传算法。遗传算法模拟了生物进化的过程，通过交叉、变异和选择的操作来生成新的解。这种启发式方法在搜索空间较大且复杂的问题中表现出色，如优化问题、机器学习模型的超参数优化等。④禁忌搜索算法。禁忌搜索是一种基于状态空间的搜索方法，通过模拟问题的状态转移过程来寻找解。这在解决许多问题中很有效，如在博弈中的对弈树搜索、路径规划等。⑤蚁群算法。蚁群算法模拟了蚂蚁在寻找食物过程中的协作行为。每只蚂蚁根据信息素的浓度选择路径，从而形成全局优化。蚁群算法在解决网络优化、路径规划等问题中具有良好的性能。⑥粒子群优化算法。粒子群优化算法模拟了鸟群或鱼群的群体行为。每个“粒子”代表解空间中的一个潜在解，通过个体最优和群体最优来引导搜索过程。这在解决连续空间的优化问题时表现出色。⑦人工免疫系统算法。人工免疫系统模拟了生物免疫系统的进化过程，通过模拟抗体的生成和进化来寻找解。这种方法在模式识别和异常检测等领域有着广泛的应用。⑧混合启发式方法。多样性的启发式方法也可以通过混合不同的启发式策略来创造新的解决方案。通过整合贪婪算法、遗传算法和模拟搜索等方法，可以充分发挥各自的优势。每种方法都有其适用的场景和特点，使得研究者能够根据具体问题的性质选择最合适的启发式策略。

尽管启发式方法在解决实际问题时取得了显著成功，但仍然面临一些挑战。首先，其容易陷入局部最优解，无法保证找到全局最优解，这对于一些复杂的搜索和优化问题可能构成限制。其次，启发式方法通常包含参数，调整这些参数以达到最佳性能可能是一项具有挑战性的任务，因为不同问题可能需要不同的参数配置。此外，启发式方法的性能受到问题领域的影响，有时不够通用，需要根据具体问题进行调整和优化。另外，由于其基于经验和启发式规则，启发式方法的正确性通常难以形式化证明，这增加了对其可靠性的评估难度。最后，计算复杂度问题也存在，特别是在某些情况下，启发式方法的计算复杂度可能会急剧增加，对于大规模问题的处理可能面临困难。综合而言，启发式方法的应用需要谨慎权衡其优势和面临的挑战，以找到最适合解决具体问题的方法。

1.2.3 元启发式方法

元启发式方法主要指一类通用型的启发式方法，这类算法的优化机理不过分依赖于算法的组织结构信息，可以广泛地应用到函数的组合优化和函数计算中。元启发式方法可以分为几个不同的类型：第一类为基于进化的方法，如遗传算法和差分进化算法等；第二类为基于物理原理的元启发式方法，如模拟退火算法和水循环算法等；第三类为基于群体智能的元启发式方法，如粒子群算法和蚁群算法等。

基于进化的元启发式方法主要是通过模拟自然界中的优胜劣汰的进化法则，实现种群的整体进步，最终完成最优解的求解。Zhou 等提出了基于负荷平衡的多目标充电调度策略，采用改进的非支配排序遗传算法进行求解，有效平抑负荷峰谷差和降低三相不平衡度。Liang 等采用改进动态概率遗传算法以实现充电桩使用时间与充电负荷的均匀分布。Zhang 等在基于跳频的认知无线电网络中，利用遗传算法获得的近似最优的联合带外频谱感知和信道分配策略，以最大化次级用户的吞吐量。Chen 等提出了一种对微电网内的分布式能源和储能设备进行标准化建模的方法，并引入了一种优化的遗传算法来解决相关问题。Xu 等提出了一种基于遗传算法的数据放置策略，搜索近似的最优数据放置矩阵，从而最小化数据中心之间的数据传输。Guerrero 等对遗传算法进行了改进，提出了非支配排序遗传算法，用于解决多目标下的容器调度问题。Rafique 等提出了一种基于实编码遗传算法的优化模型来优化能源的调度。Pan 等考虑节能模糊作业车间调度问题，提出了一种带有反馈机制的双种群进化算法，以最小化模糊完成时间和模糊总能耗，并最大化最小一致性指数。Muhuri 等考虑模糊不确定环境下实时嵌入式系统的节能任务调度问题，提出了一种 ε 约束耦合节能遗传算法。基于进化的元启发式方法适用于复杂的搜索空间和多模态问题，具有较强的全局寻优能力。但它们的收敛速度较慢，需要大量计算资源，且存在局部最优解的风险。

基于物理原理的元启发式方法主要来自宇宙中的物理规则。Wu 等提出了一种以用户为中心的软件定义超密集网络边缘资源共享模型，通过最小化服务延迟来优化 MEC 与用户的服务过程，最后通过模拟退火算法来降低服务关联模型的时间复杂度并提供近似最优解。Chun 等提出了一种用于全局优化问题的增强头脑风暴正弦余弦算法，通过头脑风暴算法的增加种群粒子多样性和避免进化过程早熟收敛来改善正弦余弦算法中更新粒子的方式。Qin 等研究云工作流调度问题，在其执行时间保持在截止时间内的同时降低执行成本。同时，基于云工作流调度问题的特定知识，提出了一种新的基于知识的自适应离散水波优化算法。基于物理原理的元启发式方法能够跳出局部最优解，适用于随机性较强的搜索空间，具有一定的全局寻优能力。但它们在高维空间中搜索效率较低，对参数设置和初始解的选择较为敏感。

基于群体智能的元启发式方法是通过模拟群体的智慧，来实现全局最优解的获取。在该算法中，每一个群体都是一个生物种群，通过种群中个体之间的协同行为，进而完成个体无法完成的任务。Shao 等运用粒子群优化算法来解决电动汽车充电调度问题，其目标是在增加运营商收益的同时，实现电网侧的需求平衡，通过“削峰填谷”策略优化电网运行。Zhang 等以电网层负荷峰谷差最小和用户层充电费用最小为目标函数，采用改进鲸鱼算法合理安排车辆的充电时间，实现电网侧的削峰填谷且减少车主的充电费用。Liu 等从负荷

变化和车主的经济效益出发，构建动态多目标优化模型，并采用动态粒子群算法对模型进行求解。Wu 等提出了基于蚁群算法的元启发式方法在截止日期约束下最小化云中工作流的执行成本。Chen 等考虑了在严格期限约束下如何有效卸载基于 DNN 的智能物联网系统的问题，并提出了一种基于自适应粒子群算法的新型高效卸载策略，通过对 DNN 层进行分层划分操作的卸载决策，显著降低了物联网系统的整体运行能耗。Xie 等针对云边缘环境下的工作流调度问题，提出了一种新颖的定向和非本地转换的粒子群优化方法，该方法可以极大地减少任务维度和执行成本，对复杂应用的任务卸载有良好的效果。基于群体智能的元启发式方法模拟群体协作与竞争，适用于全局寻优和多维搜索问题，具有一定的稳健性。但它们易陷入局部最优解，收敛速度相对较慢，对参数设置较为敏感。

1.2.4　机器学习方法

机器学习方法让计算机能够通过数据学习和改进，而无须明确地编程。这些方法基于模式识别和推断的算法，使计算机能够从数据中学习并进行预测或决策，可以广泛应用于预测和评估电池的性能、健康状况和剩余寿命。目前用于锂电池状态估计的机器学习方法可以分为几个不同的类型：第一类为基于前馈神经网络(FNN)的算法；第二类为基于极限学习机(ELM)的算法；第三类为基于循环神经网络(RNN)的算法。

基于前馈神经网络(FNN)的算法通过任意数量的输入和输出实现非线性映射。它是应用最简单的神经网络(NN)之一。Dang 等提出了一种结合 ECM 和 FNN 的混合方法，该方法不使用查找表将电池 SOC(State Of Charge，剩余电量)与电池 OCV(Open Circuit Voltape，开路电压)相关联，而是使用 FNN 模型被训练来建立这种关联。FNN 结构由一个输入、OCV、具有 m 个神经元的隐藏层以及作为其输出的 SOC 组成。利用能够根据 OCV 输入关联 SOC 的 FNN 模型。Vidal 等建立了 FNN 来估计 LFP 电池的 SOC，然后使用无迹卡尔曼滤波器来提高 SOC 估计精度。Hannan 等介绍了一种使用离线优化算法系统地改变 FNN 结构的过程，以找到最佳的 FNN 结构。这项工作的重点是回溯搜索算法，这是一种优化算法。据作者称，与遗传算法 (GA)、粒子群优化 (PSO) 等其他算法相比，该算法更容易实现、更快且更稳健。

极限学习机(ELM)结构与 FNN 非常相似，但主要区别在于其训练算法，ELM 不使用反向传播，而是使用 Moore-Penrose 广义逆矩阵或伪逆矩阵。Du 等使用 ELM 根据实验数据对锂离子电池进行建模，然后使用 KF 近似估计 SOC。与 RBF 相比，ELM 方法显示出较低的计算负载和更好的 SOC 估计误差。此外，还比较了 4 种不同的 KF 算法：EKF、AEKF、UKF 和自适应无迹卡尔曼滤波器(AUKF)。这项工作中使用的环境温度为 25℃，ELM 和 RBF 所使用的神经元数量分别为 10 和 15。与 RBF 估计时间相比，ELM 的速度快了 50%，并具有更低的估计误差。此外，即使与 KF 的其他变体相比，使用 AUKF 进行 SOC 估计也提高了其准确性并减少了计算负载。Hossain Lipu 等使用引力搜索算法 (GSA) 来查找具有一个隐藏层的 ELM 中针对两种不同驾驶周期[US06 和北京动态压力测试 (BJDST)]的最佳神经元数量，不同的温度(25℃和 45℃)，然而，他们使用了似乎有限的数据集来验证 ELM 模型对于 xEV 应用的泛化能力。未使用不同的驾驶循环来训练和验证模型，而是仅使用相同驾驶循环数据的一部分来训练和验证模型，其中 70%用于训练，30%用于验证。

基于循环神经网络(RNN)的算法是一种以闭环方式使用过去信息的神经网络。只需将网络输出或中间状态作为输入传递,就可以使神经网络成为循环网络,这是目前最主流的锂电池状态估计方法。并创建 RNN 的变体来解决梯度爆炸和梯度消失等限制,如 LSTM、双向 LSTM (BiLSTM) 和门控循环单元 (GRU)。Chemali 等应用 LSTM 来估计 SOC,仅使用直接测量的电池信号,如端电压、负载电流和环境温度,而不需要与其他方法和估计滤波器相结合。Caliwag 等引入了向量自回归移动平均 (VARMA)和 LSTM 的组合来预测电动摩托车的锂离子电池电压和 SOC,其中不同的输入组合包括电机速度、输入功率和评估扭矩、电池电压、电流和温度。作者使用 CVS-40 在 0℃和 25℃ 下测试了模型,用于训练模型的数据可直接从驾驶摩托车中获得。Vidal 等介绍了一种通过使用 LSTM 和迁移学习来减少训练时间并进一步改进 SOC 估计的新方法,迁移学习和适当优化器的使用将是有前途的研究路径,应该探索并与其他方法相结合。

1.2.5 资源管理

资源管理在不同领域中都扮演着关键角色,它涉及诸如处理器、存储、网络带宽和其他资源的有效分配和优化利用。在现代云计算、边缘计算和分布式系统中,资源管理变得更为复杂,因为它需要考虑不同层次的资源,如数据中心、边缘节点和终端设备之间的交互。在这些环境下,资源管理需要考虑到各种动态条件和需求,以实现任务和工作负载的高效分配。此外,资源管理也能够优化资源利用并降低运营成本,与能源利用息息相关。

在面向多目标资源管理方面,Tan 等提出了一种合理的复合储能容量分配方法,利用自适应惯性加权粒子群算法,建立了设备成本最低、功率匹配最佳、可再生能源输出功率最平的多目标复合储能优化方案,以解决微电网多目标问题; Xu 应用了零和博弈来解决复合储能微电网的多目标优化运行; Hu 等研究了一种基于变异算子的粒子群算法来解决以运行成本、环境效益等为目标的多目标优化问题; Chen 等提出了对微电网中的分布式能源和储能单元进行标准建模,并采用了一种改进的遗传算法来求解问题。

在面向移动边缘计算资源管理方面,Cao 等提出一种新的卸载和资源分配方案,多个计算任务被卸载到同一个边缘服务器,在边缘服务器计算资源和延迟的约束下,这能够有效降低计算卸载能耗。Zhang 等研究了基于 MEC 的密集 C-RAN 中的任务卸载和资源分配问题,旨在优化网络能量效率,并提出了一个随机混合整数非线性规划问题,用于联合优化任务卸载决策、计算资源调度和无线资源分配; Huang 等针对多移动设备场景下的卸载问题,在考虑信道和计算资源成本的同时,通过制定有效的资源分配策略,最小化用户的总消耗。Zhou 等针对动态多用户移动边缘计算系统中计算卸载和资源分配的联合优化问题,考虑延迟约束和异构计算任务的不确定资源需求,提出了一种基于双深度 Q 网络的方法,来最小化整个移动边缘计算系统的能量消耗。

在面向混合云的资源管理方面,Yuan 等提出了一种基于 K-means 算法的数据布局策略,首先使用 BEA 算法对数据集依赖矩阵进行聚类变换,获得聚类矩阵,然后根据 K-means 算法将聚类矩阵进行集合划分,最后将数据集布局到对应的数据中心,有效减少数据的移动次数。Wang 等开发了一种依赖于动态计算相关性(DCC)进行数据分布的方法。这一方案通过把 DCC 值较高的数据集群集于同一数据中心,并根据实时情况动态分配新产生的数据集至最佳数据中心,从而显著降低了数据中心间调度数据的需求。Li 等通过

构建工作流级别的数据放置模型，提出了一个两阶段的数据放置策略，该策略通过离散粒子群优化算法将数据动态分配到适当的数据中心，从而比传统任务级别数据放置策略更加节约成本。

在面向云边协同计算的资源管理方面，Meng 等提出了一种基于粒子群优化算法的安全感知调度方法，用于跨异构云的实时资源分配。实验结果表明，该策略能够在调度和安全性能之间取得良好的平衡。Pham 等考虑不稳定资源的完成率和中断率，以反映云基础设施的不稳定性。同时，提出了一种新的进化多目标工作流调度方法，用于生成一组权衡的解决方案，其最大完成时间和成本都优于最新的算法。Topcuoglu 等在 EFT 基础上提出了异构最早完成时间算法，考虑了异构资源环境，极大提高了资源利用率。Fazio 等总结了微服务调度和资源管理中存在的一些未解决的问题，如微服务的弹性调度问题。

在面向车载边缘计算的资源管理方面，Xu 等提出了一种自适应计算卸载方法，以优化任务卸载延迟和资源利用率，最后，实验结果证明了方法的有效性。Khayyat 等综合考虑了多级车辆边缘云计算网络的计算卸载和资源分配问题，提出了一种分布式深度学习算法，以优化时延和能耗。仿真结果表明，该算法具有较快的收敛速度和较好的性能。Seid 等提出了空对地网络中基于无模型深度强化学习的协同计算卸载和资源分配方案。其目标是最小化任务执行延迟和能量消耗。Huang 提出了一个基于 DRL 的在线卸载框架 DROO，以优化任务卸载决策和无线资源分配，使之适应无线供电的 MEC 网络中时变的无线信道条件。

在面向能源优化的资源管理方面，Rajani 等提出了一种电动汽车充电站与配电系统能量管理的混合策略，应用吉萨金字塔建造算法和增强递归神经网络的混合算法，使能量调度高效运行。Biya 等采纳了一种充电站设计方案，该方案结合了最大功率点跟踪、比例积分微分(PID)和电流调节策略，以实施分散式能源管理。舒恺及其团队推出了一套依据系统功率状况的能源管理方案。这一方法按系统功率状况将直流微电网分成若干运行状态，每种状态下，系统中的各个单元将呈现出不同的操作模式。Mi 等为最大化利用 PV，提出了一种基于 ESS 装置充放电功率的能量管理策略。

第 2 章　面向多目标的资源管理

2.1　基于粒子群灰狼混合算法的多目标进化算法

近年来，多目标优化问题引起了广泛关注，其求解目标多、目标函数复杂，当前方法通常将所有目标加权后求解，但解集缺乏准确性。针对上述情况，本节首先根据目标分解框架——辅助目标和等价目标约束优化框架，该框架是将约束优化的问题分解为辅助目标和等价目标相结合的优化问题，同时动态调整所分解出的对应子问题的权值，使分解出的子问题求解趋向于等价目标求解。其次基于粒子群优化算法和灰狼优化算法的各自优势，提出参数自适应的粒子群灰狼混合算法，混合算法的优势集合了粒子群算法的较快收敛性和灰狼算法搜索过程的多样性，从而提高粒子进化过程的准确性。通过 IEEE CEC2017 数据集测试的结果表明：在调参合适的情况下，获得的函数最优值个数多于乌鸦搜索、受约束的模拟退火、带约束的水循环等经典算法，在维度 D 为 10 的情况下，28 个测试函数中 11 个测试函数表现最佳；在维度 D 为 30 的情况下，12 个测试函数表现最佳。

2.1.1　引言

约束优化问题在 20 世纪 90 年代被提出，广泛存在于信号处理、图像处理、微电网设计等领域。解决约束优化问题的传统优化方法包括解析法等，但是传统的优化方法无法解决高维的非线性、不可微分的复杂约束优化问题，而且传统的优化方法（如牛顿法、单纯形法等）需要遍历整个搜索空间，不能在短时间内完成，容易出现搜索的“组合式爆炸”。为了更好地解决约束优化问题，可将其转换为多目标优化问题进行求解，多目标优化问题（Multi-Objective Optimization Problems，MOPs）能够在多个目标上达到最佳效果，但 MOPs 的目标大多数为相互冲突的目标，可通过求解多目标问题的方式，得到最优的解决方案，因此，解决多目标问题不应该只考虑单一的最优解，还需要考虑一组最优解集合，力求均衡优化。多目标优化问题通常为多项式复杂程度的非确定性问题（Nondeterminism Polynomial Hard，NP-Hard），常用的解决方法是将 NP-Hard 问题转换成单目标优化或者多目标优化问题再进行解决。由于进化算法适合求解多项式复杂程度的非确定性问题，所以多目标进化算法（Multi-Objective Evolutionary Algorithms，MOEAs）能够通过进化算法来解决多目标优化问题。

进化算法（Evolutionary Algorithms，EAs）是一种基于群搜索的全局优化方法，适用于求解约束优化和多目标优化问题。它是一种启发式方法，通过观察生物种群的进化演变而成，有着一定的进化组织性。进化算法主要思想来自遗传算法，利用父代进行交叉变异计算得到子代种群，对比子代种群和父代种群的适应度值，选出适应度值更优的种群作为下一代进化的父代种群，循环往复得到最优的个体。许多专家学者对进化算法展开大量的研究，提出了许多改进方式，如基于支配、基于指标、基于目标分解策略的方法。基于支配的算法有 Deb 提出的非支配排序遗传算法（Nondominated Sorting Genetic Algorithm Ⅱ，

NSGA-Ⅱ),基于指标的算法主要有 Zitzler 等提出的 IBEA(Indicator-Based Evolutionary Algorithm)算法,基于目标分解的算法有 Zhang 等提出的 MOEA/D(Multi-Objective Evolutionary Algorithm based on Decomposition)算法。

目前,许多学者都将基于分解的 MOEA(Multi-Objective Evolutionary Algorithm)当作研究热点。相较于传统的进化算法,MOEA 的优势是:它能够同时处理多个目标并执行一次获得一组最优解;它还能够处理复杂函数,不受 Pareto 最优解集的形状制约。MOEA 的主要研究方向有三类:基于 Pareto 支配的 MOEA、基于指示器的 MOEA、基于分解优化的 MOEA。

分解优化的 MOEA 大致能够分为三部分:

(1) 标准的双目标问题形式。双目标包括原始目标函数和对于约束条件程度的度量函数。

(2) 标准的多目标问题优化。主要方法是将 COPs(Constrained Optimization Problems,约束优化问题)转换为 $n+1$ 类型的问题,其中 1 为优化问题,n 为约束函数。

(3) 广义多目标问题方法。将原始问题转换为无约束的多目标优化问题,其中至少要有一个目标函数与原始函数不同。

本节针对粒子群算法中的收敛过快和灰狼优化算法中多样性溢出的缺陷,提出了利用分解策略改进的混合算法 HECO-HPSGWO,该优化算法结合了 HECO 框架和参数自适应的 HPSGWO 算法。HECO 框架能够在进化过程中平衡各个目标的权重。HPSGWO 算法结合了 PSO 算法和 GWO 算法的优势。其中,PSO 算法进化收敛速度快,GWO 算法可以减少陷入局部优化的风险。

2.1.2　相关工作

约束优化问题在 20 世纪 90 年代被提出,约束优化问题通常是多项式复杂程度的非确定性问题(NP-Hard),NP-Hard 问题会产生很大的算法时间复杂度,对于解决约束优化问题,常用的方法是将约束优化问题转换成单目标优化问题或者多目标优化问题再进行解决。当带有约束优化的多目标进化算法被提出之后,运用带有约束优化的多目标进化算法来解决某些领域的问题引起了众多研究工作者的关注。刘敏等提出了记忆增强的动态多目标分解进化算法,通过设计基于子问题的串式记忆方法来获得最优解。Askarzadeh 提出了乌鸦搜索算法(Crow Search Algorithm,CSA),通过基于乌鸦的智能行为来解决约束优化问题。Hadi 提出带约束优化的水循环算法(Constrained Water Cycle Algorithm,CWCA),基于对水循环过程的观察来解决问题。很多实验研究都证实了多目标优化方法的效率和问题的复杂度紧密相关,但确保多目标 EAs(Evolutionary Algorithms)优于单目标 EAs 仍然是一个严峻的工作。由于多目标优化问题的复杂程度高并且难以均衡优化,所以通过分解多目标来降低问题的复杂度和达到均衡优化是一股热潮。例如,郑金华等提出基于权重迭代的偏好多目标分解算法;张磊等提出基于重新匹配策略的 ε 约束多目标分解优化算法;Xu 等通过带静态权值的加权法将问题分解为若干子问题;Wang 等通过动态权重的方法分解目标问题,在进化过程中有所侧重。为了更好地分解原目标,运用辅助目标,该目标的概念被 Jensen 首次提出。Jiao 等将辅助目标添加到原始问题,将 COP 问题转换为动态双目标优化问题。因为等价目标和辅助目标的运用能够使群体进化的过程更有侧

重点，从而有效地提高组合优化问题的性能。Xu 提出等价和辅助目标框架来解决多目标问题，将原始目标分解为等价目标和辅助目标，利用动态权重来调整搜索进程中的侧重方向。然而，框架所涉及的参数较多，在后续的算法使用过程中涉及的参数数量较大，会造成实验较高的复杂度。所以本节考虑和粒子群优化(Particle Swarm Optimization，PSO)算法相关的改进算法。EAs 算法普遍受到自然界生物的启发。粒子群优化算法受到自然界鸟群捕食的启发，PSO 因为有着参数少、收敛速度快等优势，所以在各种领域都取得了一定的成功。虽然 PSO 的收敛速度快，但是平衡全局的能力差，容易陷入局部优化。最近兴起的算法灰狼优化器(Grey Wolf Optimization，GWO)能够避免局部优化的困境，但由于过程较为复杂，导致搜索的时间较长。所以 Eenel 等提出了 HPSGWO，这是一种混合的 PSO-GWO 算法。

为了更好地解决多目标优化问题，本节将采取以下措施：

(1) 使用辅助目标和等价目标(Helper and Equivalent Objective)来分解已知的约束优化问题，从而缩短粒子搜索过程中跨越“鸿沟”的时间。

(2) 通过参数自适应的 HPSGWO 算法来解决分解后的子问题。

2.1.3 问题模型

定义 1(单目标约束优化问题) 数学模型中，只有一个目标的约束优化问题可以表述为

$$\begin{aligned} &\min f(\boldsymbol{x}), \quad \boldsymbol{x}=(x_1,x_2,\cdots,x_D)\in\Omega \\ &\text{s.t.}\begin{cases} g_i^I(\boldsymbol{x})\leqslant 0, & i=1,2,\cdots,q \\ g_i^E(\boldsymbol{x})=0, & i=1,2,\cdots,r \end{cases} \end{aligned} \tag{2-1}$$

式中，$\Omega=\{\boldsymbol{x}\mid L_j\leqslant x_j\leqslant U_j, j=1,2,\cdots,D\}$，为位于 R^D 中的有界域，D 为算法中粒子的维度，L_j 和 U_j 分别为算法中粒子取值的上限或者下限；$g_i^I(\boldsymbol{x})\leqslant 0$ 为不等式约束；$g_i^E(\boldsymbol{x})=0$ 为等式约束；q 和 r 为粒子的维度；Ω^*、Ω_I、Ω_F 分别为最优可行解、不可行解、可行解。

式(2-1)的目的是粒子处在一定的区间内，在满足等式和不等式的条件下，获取最小值。

定义 2(多目标的优化问题) 对于解决约束优化问题，常用的方法是多目标优化方法，多目标优化方法是将约束问题转换为无不等式或等式约束的问题。双目标约束优化问题的数学模型表述为

$$\min f(\boldsymbol{x})=(f(\boldsymbol{x}),v(\boldsymbol{x})) \tag{2-2}$$

式中，$f(\boldsymbol{x})$为原始目标函数；$v(\boldsymbol{x})$为违反约束的程度。

本节的约束违反程度由各约束违反程度之和来度量，$v(\boldsymbol{x})$的具体表达式为

$$v(\boldsymbol{x})=\sum_{i=1}^{q}v_i^I(\boldsymbol{x})+\sum_{i=1}^{q}v_i^E(\boldsymbol{x}) \tag{2-3}$$

式中，$v_i^I(\boldsymbol{x})$是指违反第 i 个不等式约束的程度，表达式为

$$v_i^I(\boldsymbol{x})=\max\{0,g_i^I(\boldsymbol{x})\}, \quad i=1,2,\cdots,q \tag{2-4}$$

$v_i^E(\boldsymbol{x})$是指违反第 i 个等式约束的程度，表达式为

$$v_i^E(\boldsymbol{x})=\max\{0,\mid g_i^E(\boldsymbol{x})-\varepsilon\mid\}, \quad i=1,2,\cdots,q \tag{2-5}$$

式中，ε 是指对等式约束所能允许的误差。

2.1.4　优化方法

本节所运用的 HECO-HPSGWO 优化算法是结合了 HECO 框架和参数自适应 HPSGWO 算法。HECO 框架能够在进化过程中平衡各个目标的权重。HPSGWO 算法是一种混合算法，采用了 PSO 算法和 GWO 算法，并且在算法进行时实时调参，提高准确性。

1. 辅助目标和等价目标的方法

HECO 是一种新型的框架，能够通过辅助目标来增加目标函数的搜索方向并通过等价目标来转换目标函数。因为等价目标的最优解集和原始的 COP 问题的最优解集一致，所以等价目标能够提供算法中粒子搜索的主要方向，并且等价目标的解集能够满足原始 COP 问题的所有约束条件。而辅助目标是由原目标衍生而来，在算法进化的过程中，提供多样的进化结果，防止算法陷入局部最优。由于 HECO 框架利用了动态权重来均衡优化等价目标和辅助目标，所以其性能超越了可行性规则和死亡罚函数的进化方式。下面介绍死亡罚函数和可行性规则以体现 HECO 框架的优点。

死亡罚函数如式(2-6)所示，其中 Ω_F 表示可行解，Ω_I 表示不可行解：

$$\min(\boldsymbol{x})=\begin{cases} f(\boldsymbol{x}), & \boldsymbol{x}\in\Omega_F \\ +\infty, & \boldsymbol{x}\in\Omega_I \end{cases} \tag{2-6}$$

用死亡罚函数处理约束时，当解为非可行解的时候，则直接丢弃，会造成某个适应度值优良而处在约束边缘化的解的丢失，造成对解集不合理的评判。

可行性规则的运用有以下原则：①适应度较小的解优于适应度较大的解；②可行解优于不可行解；③如果两个解集都为可行解或者不可行解，通过对比两者违反约束的值来选择，较小的约束违反优于较大的约束违反。

根据可行以上的规则，等价函数可以表示为

$$e(\boldsymbol{x})=\begin{cases} f(\boldsymbol{x}), & \boldsymbol{x}\in\Omega_F\cap P \\ v(\boldsymbol{x})+f_F(P), & \boldsymbol{x}\in\Omega_I\cap P \end{cases} \tag{2-7}$$

式中，$f_F(P)=\max\{f(\boldsymbol{x}),\boldsymbol{x}\in\Omega_F\cap P\}$，$\Omega_F\cap P\neq\varnothing$ 或 $f_F(P)=0$；P 为种群的所有粒子；Ω_I 和 Ω_F 分别为不可行解和可行解集合。

然而死亡罚函数和可行性规则的优越性中一个可行解总是优于任何一个不可行解。为了减少这种强加于可行解的偏好的影响，HECO 构造了一个新的等价函数。HECO 的规则如下：①如果最小化 $f(\boldsymbol{x})$ 的所有解都为可行解，并且 $f(\boldsymbol{x})$ 的解集和原始问题 $e(\boldsymbol{x})$ 的最优解集一致，则 $f(\boldsymbol{x})$ 称为等价目标；②如果最小化 $f(\boldsymbol{x})$ 的所有解集不满足原始问题的最优解集，则 $f(\boldsymbol{x})$ 可以称为辅助目标；③从约束违反程度 $v(\boldsymbol{x})$ 来分析，如果 $v(\boldsymbol{x})$ 的一个可行解不是最优的，解集与原始的 COP 最优解集不一致，那么 $v(\boldsymbol{x})$ 可以称为辅助目标。

下面阐述 HECO 框架的步骤。先列出等价目标 $e(\boldsymbol{x})$，在此基础上添加辅助目标 $h_i(\boldsymbol{x})$，$i=1,2,\cdots,k$，k 为辅助目标的个数。以此来帮助等价目标更好地搜索粒子，即

$$\min\boldsymbol{f}(\boldsymbol{x})=(e(\boldsymbol{x}),h_1(\boldsymbol{x}),\cdots,h_k(\boldsymbol{x})),\quad \boldsymbol{x}\in P \tag{2-8}$$

基于加权求和的方法，分解目标问题可以转换为

$$\min f_i(\boldsymbol{x}) = w_{0i} e(\boldsymbol{x}) + \sum_{j=1}^{k} w_{ji} h_j(\boldsymbol{x}) \tag{2-9}$$

式中，$i=1,2,\cdots,\lambda$，$\boldsymbol{x} \in P$。

根据以上的式子，本节目标为每个 $f(\boldsymbol{x})$ 最后都收敛为等价目标，式(2-9)参数的部分规则表达方式为

$$\lim_{t \to +\infty} w_{0i,t} > 0 \tag{2-10}$$

$$\lim_{t \to +\infty} w_{ji,t} = 0 \tag{2-11}$$

式中，$i=1,2,\cdots,\lambda$，$j>0$；t 为运行的代数。

结合上述步骤本节先通过原始函数创建一个新的等价目标，设置种群为 P，设置 x_p^* 为种群中的最优粒子。x_p^* 的计算方法为

$$x_p^* = \begin{cases} \arg\min\{v(\boldsymbol{x}); \boldsymbol{x} \in P\}, & P \cap \Omega_F = \varnothing \\ \arg\min\{f(\boldsymbol{x}); \boldsymbol{x} \in P \cap \Omega_F\}, & P \cap \Omega_F \neq \varnothing \end{cases} \tag{2-12}$$

接下来，令 $\tilde{e}(\boldsymbol{x})$ 为 $f(\boldsymbol{x})$ 和最优个体 $f(x_p^*)$ 的适应度差值。$\tilde{e}(\boldsymbol{x})$ 的数学表达式为

$$\tilde{e}(\boldsymbol{x}) = | f(\boldsymbol{x}) - f(\boldsymbol{x}_p^*) | \tag{2-13}$$

当最优个体满足约束条件，并且 $f(\boldsymbol{x})$ 适应度的值越靠近 $f(x_p^*)$，则 $f(\boldsymbol{x})$ 满足约束条件的可能性越大，$\boldsymbol{x}$ 处于最优解集的可能性也越大。为了保证 $\boldsymbol{x}$ 能够满足约束条件，构造的等价函数需要加上一定量的约束违反的度量值 $v(\boldsymbol{x})$，以保证算法中的粒子能够在约束条件界限下。所以种群 P 上的等价函数可以构造为

$$e(\boldsymbol{x}) = w_1 \tilde{e}(\boldsymbol{x}) + w_2 v(\boldsymbol{x}) \tag{2-14}$$

式中，w_1、w_2 为权重，且 w_1、$w_2>0$，w_1、w_2 的作用是引导函数收敛为等价函数，由于 w_1，$w_2 \in (0,+\infty)$，所以等价函数的数量是无限的。

然后选择 $f(\boldsymbol{x})$ 作为辅助目标，辅助目标和等价目标的问题表达式为

$$\min f(\boldsymbol{x}) = (e(\boldsymbol{x}), f(\boldsymbol{x})), \quad x \in P \tag{2-15}$$

根据式(2-14)和式(2-15)，最小化问题可以分解为基于加权求和的 λ 个子问题，即

$$\min f_i(\boldsymbol{x}) = w_{1i} \tilde{e}(\boldsymbol{x}) + w_{2i} v(\boldsymbol{x}) + w_{3i} f(\boldsymbol{x}) \tag{2-16}$$

式中，$i=1,2,\cdots,\lambda$。

根据动态调整权值，式(2-16)中的 $f_i(\boldsymbol{x})$ 为辅助函数和等价函数的加权和，即

$$f_i(\boldsymbol{x}) = w_{1i} \tilde{e}(\boldsymbol{x}) + w_{2i} v(\boldsymbol{x}) + w_{3i} f(\boldsymbol{x}) \tag{2-17}$$

式中，$i=1,2,\cdots,\lambda$。

针对式(2-17)，为了能够使 $f_i(\boldsymbol{x})$ 收敛到一个等价问题，加权规则表述为

$$\lim_{t \to +\infty} w_{1i,t} > 0, \quad \lim_{t \to +\infty} w_{2i,t} > 0, \quad \lim_{t \to +\infty} w_{3i,t} = 0 \tag{2-18}$$

式中，t 为循环的代数。

在 HECO-HPSGWO 中，w_1 和 w_2 设计为线性增加，而 w_3 设计为线性递减。权重的规则表述为

$$w_{1i,t} = \frac{t}{T_{\max}} \cdot \frac{i}{\lambda} \tag{2-19}$$

$$w_{2i,t}=\frac{t}{T_{\max}}\cdot\frac{i}{\lambda}+\gamma \tag{2-20}$$

$$w_{3i,t}=\left(1-\frac{t}{T_{\max}}\right)\cdot\left(1-\frac{i}{\lambda}\right) \tag{2-21}$$

式中，λ 为子问题的数量；$\gamma\in(0,1)$，为关于约束条件的常量偏差值；t 为循环代数；$T_{\max}$ 为最大循环代数的值。

通过用动态权重调整辅助目标和等价目标的 HECO 框架中各个子问题的收敛情况，能够搜寻到更多的进化方向，同时它明确了等价函数在进化过程中的主方向不变。HECO 的运用结果能够展现多目标进化算法在解决约束优化问题的良好特性，从而促进 HPSGWO 算法以更准确地解决目标问题。

2. 粒子群优化算法

粒子群优化算法是 Kennedy 和 Eberhart 在 1995 年提出的一种模拟群体行为的智能优化算法。该方法在多目标优化问题求解中有很好的效果。粒子群优化算法是一种具有个体改进、种群合作和竞争机制的启发式方法，它受启发于自然界中成群鸟类的捕食行为。模拟过程中，将每只鸟看作一个粒子，“粒子”找寻食物的过程模拟了鸟群里的每一只鸟都在一片固定的大区域内随机地觅食，它们不知道目标（也就是食物的具体位置），而鸟群会通过叫声或者记号等行为来交换信息，接下来更多的鸟会向那个方向聚集，引导种群更新位置。PSO 算法模拟了一群鸟儿在寻找食物，代表着粒子在寻求一个最佳解决方案。每个粒子都有 4 个属性：当前位置、自身最佳位置、种群最佳位置、当前速度。对于粒子而言，当前位置是循环过程中本次所处的位置，自身最佳位置是目前迭代中距离最优解的位置，种群最佳位置是种群迭代中距离最优解的位置，当前速度是粒子迭代进行的步长。

每一个粒子都在决策空间中通过速度来更新自身的位置，以此让粒子靠近所要求的空间，每个粒子的最优位置和粒子群中最优粒子的位置信息共同引导着粒子群的搜索位置。第 i 个粒子在第 j 维度决策空间中的速度和位置的更新采用的表达式为

$$v_{ij}(t+1)=wv_{ij}(t)+c_1r_1(p_{ij}(t)-x_{ij}(t))+c_2r_2(p_{gj}(t)-x_{ij}(t)) \tag{2-22}$$

$$x_{ij}(t+1)=x_{ij}(t)+v_{ij}(t+1) \tag{2-23}$$

式中，$i=1,2,\cdots,N$；$j=1,2,\cdots,n$；w 为惯性权重；r_1 和 r_2 为信息随机数，它们的值采用的是自适应更新的方法，具体的自适应更新方法会在第 2.1.5 节进行详细介绍；c_1 和 c_2 为加速常数，分别决定了粒子向自身最优位置和种群最优位置的学习能力；$x_{ij}(t)$ 为之前粒子的位置；$p_{ij}(t)$ 为粒子经过的最优位置；$p_{gj}(t)$ 为种群粒子所经过的最优位置；$v_{ij}(t+1)$ 为当前粒子的速度。

由于粒子群优化算法起源于复杂适应系统，因此工程中需要采用粒子群算法的问题越来越多。粒子群算法的收敛速度快，所需要的参数数量少，结构通俗易懂，伪代码如算法 2.1 所示。

算法 2.1　粒子群算法

输入：随机的粒子种群为 POP_N，种群大小为 N，当前迭代次数为 FES，最大迭代次数为 FES_MAX

输出：粒子种群中的最优解

1：　初始化：在可行域内初始化粒子的位置和随机的初速度，并且计算粒子的自适应度，以及定义一个群体最佳位置并初始化为 0。

```
选择：对比种群中的粒子适应度，从所有粒子中选出最佳位置的粒子，将其位置信息设置为群体最佳位置。
while FES<FES_MAX：
    for i in POP_N：
        根据式(2-22)和式(2-23)更新粒子的速度和位置，计算粒子的适应度，从而产生新的种群；
    end for
end while
得到种群最优解集
```

3. 灰狼优化器

灰狼算法是由 Mirjalili 受到自然界灰狼群体捕猎的启发而在 2014 年提出的一种迭代式搜索智能算法。通过标准测试函数的验证，灰狼算法比粒子群算法、差分进化算法更加有优势。灰狼有很严格的社会等级统治制度，灰狼群体一共有 4 种等级的狼，分别为 α、β、δ、ω。α 是狼群中的领导者，它负责作有关捕猎、行动时间等决策，并在狼群发号施令。β 是第二阶级的狼群，它负责协助 α 狼作决策并传达命令。阶级底层的狼群是 ω，它不得不听从其他狼。灰狼群等级划分示意图如图 2-1 所示。

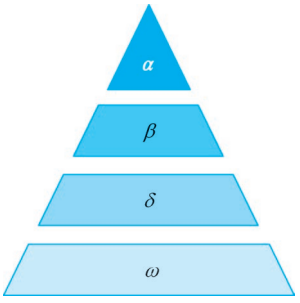

图 2-1 灰狼群等级划分示意图

灰狼群体中的这 4 种狼彼此配合进行捕猎，灰狼捕猎主要分为以下 3 个步骤：①追踪、追逐、靠近猎物；②追捕、包围、不断骚扰猎物直到猎物停止移动；③攻击猎物。根据灰狼狩猎的数学模型，可以得到方程

$$\boldsymbol{D}=|\boldsymbol{C}\cdot\boldsymbol{X}_p(t)-\boldsymbol{X}(t)| \tag{2-24}$$

$$\boldsymbol{X}(t+1)=\boldsymbol{X}_p(t)-\boldsymbol{A}\cdot\boldsymbol{D} \tag{2-25}$$

式中，t 为更新的代数；$\boldsymbol{X}_p$ 为猎物当前的位置；$\boldsymbol{X}$ 为灰狼的位置；$\boldsymbol{A}$ 和 $\boldsymbol{C}$ 为系统向量；$\boldsymbol{D}$ 为当前灰狼位置与猎物位置之间的距离。

$\boldsymbol{A}$ 和 $\boldsymbol{C}$ 的计算方法为

$$\boldsymbol{A}=2\boldsymbol{a}\cdot\boldsymbol{r}_1-\boldsymbol{a} \tag{2-26}$$

$$\boldsymbol{C}=2\cdot\boldsymbol{r}_2 \tag{2-27}$$

式中，$\boldsymbol{r}_1$ 和 $\boldsymbol{r}_2$ 均为[0,1]区间中随机产生的随机向量。

$\boldsymbol{a}$ 的值随着算法迭代次数的增加，从 2 线性减少到 0。我们令 $\boldsymbol{A}$ 的值范围为$[-2a, 2a]$，$\boldsymbol{C}$ 的值从[0,2]范围内随机产生。当$|\boldsymbol{A}|>1$ 时，$\boldsymbol{X}(t+1)$的值减少，说明灰狼正在靠近猎物。当 $\boldsymbol{C}>1$ 时，算法强调猎物的作用，$\boldsymbol{D}$ 的值也会增大，那么猎物和灰狼的距离则增大。$\boldsymbol{C}$ 的意义在于模拟大自然中灰狼捕食猎物时遇到障碍物对两者的影响。通过 $\boldsymbol{A}$ 和 $\boldsymbol{C}$ 值的随机性来模仿灰狼捕食猎物的过程中距离的不确定性。

由于灰狼狼群由 α、β、δ 三种狼来决定位置，所以利用这三种狼的位置能够判断出猎物的位置。三种狼与猎物直接距离的数学表达式为

$$\boldsymbol{D}_\alpha=|\boldsymbol{C}_1\cdot\boldsymbol{X}_\alpha(t)-\boldsymbol{X}(t)| \tag{2-28}$$

$$\boldsymbol{D}_\beta=|\boldsymbol{C}_2\cdot\boldsymbol{X}_\beta(t)-\boldsymbol{X}(t)| \tag{2-29}$$

$$\boldsymbol{D}_{\delta} = | \boldsymbol{C}_3 \cdot \boldsymbol{X}_{\delta}(t) - \boldsymbol{X}(t) | \tag{2-30}$$

式中，$\boldsymbol{X}_{\alpha}$、$\boldsymbol{X}_{\beta}$、$\boldsymbol{X}_{\delta}$ 分别为在当前迭代次数下，三种狼的最优个体。

$$\boldsymbol{X}_1 = | \boldsymbol{X}_{\alpha}(t) - \boldsymbol{a}_1 \cdot \boldsymbol{D}_{\alpha} | \tag{2-31}$$

$$\boldsymbol{X}_2 = | \boldsymbol{X}_{\beta}(t) - \boldsymbol{a}_2 \cdot \boldsymbol{D}_{\delta} | \tag{2-32}$$

$$\boldsymbol{X}_3 = | \boldsymbol{X}_{\delta}(t) - \boldsymbol{a}_3 \cdot \boldsymbol{D}_{\delta} | \tag{2-33}$$

$$\boldsymbol{X}(t+1) = \frac{\boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3}{3} \tag{2-34}$$

式中，$\boldsymbol{X}_1$、$\boldsymbol{X}_2$、$\boldsymbol{X}_3$ 分别为狼群中三种狼的当前位置；$\boldsymbol{X}(t+1)$为狼群中其他狼的位置。

综上，灰狼优化算法的伪代码如算法 2.2 所示。

算法 2.2　灰狼优化算法

输入：随机的粒子种群为 POP_N，种群大小为 N，当前迭代次数为 FES，最大迭代次数为 FES_MAX
输出：种群最优解 best_position。

```
初始化：在可行域内初始化灰狼的位置。
计算种群中灰狼个体的适应度值。
while FES<FES_MAX:
    选出种群中的灰狼 α、β、δ。
    for i in POP_N:
        种群中的粒子根据式(2-31)～式(2-32)进行更新；
        Best_position 根据粒子群的适应度进行更新；
    根据式(2-26)和式(2-27)更新算法中的 α、A、C 参数；
    end for
end while
```

4. 混合粒子群灰狼优化算法

灰狼算法自从提出后，就产生了各式各样的改进策略。例如，Saremi 提出通过种群动态操作来消除适应度差的个体；Nasrabadi 等利用反向学习改进灰狼算法；Zhu 等结合 DE 算法的全局搜索能力和灰狼算法的局部搜索能力；张新明等使用变异的反向学习方法，加强最差灰狼位置的学习，同时使用随机差分变异策略。

HPSGWO 算法结合了 PSO 算法和 GWO 算法的优势。PSO 算法能够搜寻快速，但在现实运用时容易陷入局部优化的困境中；GWO 算法可以减少陷入局部优化的风险，但是运行的时间将会延长。考虑到本节测试函数的复杂度和维度都较高，所以算法的运行速率不可忽视。PSO 算法在粒子进化的过程中，是通过一些随机参数和种群最优等位置进行计算来得到粒子位置，但会造成粒子跳到一个较劣的随机位置。GWO 具有高探索能力，粒子通过 GWO 进化会减少陷入劣性位置的可能性。所以本节采用 PSO 算法作为主线算法，但为了在种群进化过程中保持多样性，所以本节考虑在 PSO 算法进化过程中，通过迭代一定次数的 GWO 算法来降低收敛速度。HPSGWO 算法的伪代码如算法 2.3 所示。

算法 2.3　HPSGWO 算法

输入：(MAXITR，FES，Pop，FES_Pop，Prob)
输出：种群粒子

```
初始化种群粒子。
for i=1 to MAXITR:
    for j=1 to Pop:
        使用 PSO 来更新粒子的位置和速度;
        if rand(0,1) < prob:
            设置 a⃗、A⃗、C⃗ 值;
            for k=1 to FES:
                for m=1 to FES_Pop:
                    运行 GWO;
                    更新 α、β、δ 灰狼位置;
                更新 a⃗、A⃗、C⃗ 的值;
                end for
            end for
        end if
    end for
end for
```

5. HPSGWO 参数的调整

HPSGWO 算法中的参数对算法的收敛性起着重要作用。如果参数过大，则会导致收敛过快，陷入局部优化的困境；如果参数过小，则会导致收敛过慢，运行的时长过长。本节受到 L-SHADE 方法的启发，L-SHADE 方法调整参数是基于历史记忆的集合，通过对比父代和子代的适应度值的优劣来判定参数的情况。如果进化结果为优异，那么将参数的值保存进历史记忆的集合中，方便之后通过公式计算来更新参数。同理，我们也采取历史记忆集合更新的方式来获取 HPSGWO 算法中的部分参数值。如表 2-1 所示，设置了 λ 个位置来保存 HPSGWO 算法 c_1、c_2、w 的历史记忆值。在算法刚开始的时候，$M_{c1,i}$、$M_{c2,i}$、$M_{w,i}$（$i=1,2,\cdots,\lambda$）集合中的初始值都为 0.5，c_1、c_2、w 的值通过柯西分布计算得到。计算方法为

$$c_1 = \text{randc}(M_{c1}, 0.1) \tag{2-35}$$

$$c_2 = \text{randc}(M_{c2}, 0.1) \tag{2-36}$$

$$w = \text{randc}(M_w, 0.1) \tag{2-37}$$

在每一代中，c_1、c_2、w 通过历史记忆值来更新自身的值，M_{c1}、M_{c2}、M_w 是用来存储 c_1、c_2、w 的历史记忆值的集合。更新 c_1、c_2、w 的值，是通过父代和子代之间的数值比较来调整的，当父代个体经过 HECO-HPSGWO 算法更新后的子代个体，它的适应度值优于父代个体的适应度值，说明粒子经过算法后往更优的空间发展，所以这一代的 c_1、c_2、w 的值是促进粒子往更优空间搜索，因此这一代 c_1、c_2、w 被保存到记录参数优异的 S_{c1}、S_{c2}、S_w 集合中，S_{c1}、S_{c2}、S_w 集合是用来更新算法使得子代获得更优的 c_1、c_2、w 的值，并且 S_{c1}、S_{c2}、S_w 又通过式(2-38)～式(2-41)来计算得到 M_{c1}、M_{c2}、M_w，之后 c_1、c_2、w 又通过式(2-35)～式(2-37)得到。参数自适应的伪代码如算法 2.4 所示。

表 2-1 历史记忆 M_{c1}、M_{c2}、M_w

参　数	1	2	…	λ−1	λ
c_1	$M_{c1,1}$	$M_{c1,2}$	…	$M_{c1,\lambda-1}$	$M_{c1,\lambda}$
c_2	$M_{c2,1}$	$M_{c2,2}$	…	$M_{c2,\lambda-1}$	$M_{c2,\lambda}$
w	$M_{w,1}$	$M_{w,2}$	…	$M_{w,\lambda-1}$	$M_{w,\lambda}$

算法 2.4　HPSGWO 参数自适应算法

输入：$(M_{c1,i,G+1}, M_{c2,i,G+1}, M_{w,i,G+1}, S_{c1}, S_{c2}, S_w)$
输出：$(M_{c1,i,G+1}, M_{c2,i,G+1}, M_{w,i,G+1})$
1：　**if** $S_{c1} \approx \varphi$ and $S_{c2} \approx \varphi$ and $S_w \approx \varphi$：
2：　　**if** $M_{c1,i,G} = \perp$ or $M_{c2,i,G} = \perp$：
3：　　$M_{w,i,G} = \perp$；
4：　　**else**：
5：　　$M_{c1,i,G} = \text{mean}_{WL}(S_{c1})$；
6：　　$M_{c2,i,G} = \text{mean}_{WL}(S_{c1})$；
7：　　$M_{w,i,G} = \text{mean}_{WL}(S_{c1})$；
8：　　i++；
9：　　**if** i>λ,i=1;**end if**
10：　　**end if**
11：　　**else**：
12：　　$M_{c1,i,G+1} = M_{c1,i,G}$；
13：　　$M_{c2,i,G+1} = M_{c1,i,G}$；
14：　　$M_{w,i,G+1} = M_{c1,i,G}$；
15：　**end if**

在算法 2.4 中，$i(1 \leqslant i \leqslant \lambda)$是指参数更新的位置，在算法 2.4 循环刚开始的时候，i 的初始值为 1，如果 $i > \lambda$，那么 i 重置为 1。算法 2.4 的规则为

$$M_i = \text{mean}_{WL}(S_i), S_i \neq \varnothing \tag{2-38}$$

$$\text{mean}_{WL}(S) = \frac{\sum_{i=1}^{|S|} w_i S_i^2}{\sum_{i=1}^{|S|} w_i S_i} \tag{2-39}$$

$$w_i = \frac{\Delta f_i}{\sum_{l=1}^{|S|} \Delta f_l} \tag{2-40}$$

$$\Delta f_i = | f(x_{i,G}) - f(x_{i-1,G-1}) | \tag{2-41}$$

式(2-38)中的 $\text{mean}_{WL}(S_i)$对于更新参数来说很重要，它用来计算 M 的值。更新参数的原则是实时改变参数，这个原则会降低算法的收敛度，不易陷入局部优化，对于多维问题较为友好。图 2-2 所示为 HECO-HPSGWO 算法流程图。

6. 线性种群大小缩减

合适的种群大小可以有效地提高算法的性能，将种群大小缩减的方法融合到 HPSGWO 中能够提高性能。线性种群大小缩减(Linear Population Size Reduction，LPSR)是用于动态调整种群大小的适应度评估数的函数。初始的种群大小为 N_{init}，算法结束后的种群大小为 N_{min}。在每一代 G 中，种群大小 N_G 的计算方法为

$$N_G = \text{round}\left[\left(\frac{N_{min} - N_{init}}{\text{MAX\_FES}}\right) \cdot \text{FES} + N_{init}\right] \tag{2-42}$$

式中，FES 为当前适应度评估的数量；MAX_FES 为适应度评估的最大数量的值。当 $N_G < N_{G-1}$ 时，$(N_{G-1} - N_G)$个性能最差的粒子则被从种群中剔除。

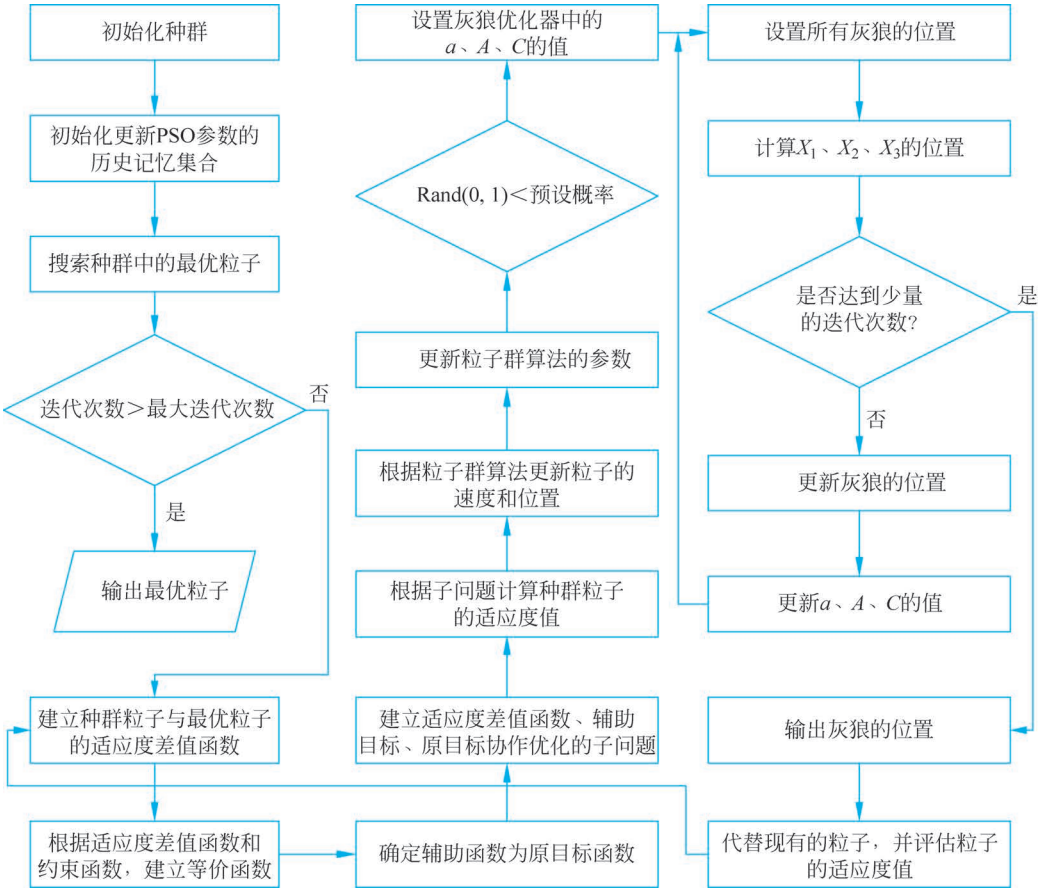

图 2-2 HECO-HPSGWO 算法流程图

2.1.5 实验

1. 实验环境

实验仿真环境为 64 位 Ubuntu 18.04.5 LTS Server 操作系统，内存为 96GB，CPU 为 Intel Xeon Gold 6244，GPU 为 2 个 Nvida Quadro RTX 6000，Python 版本为 3.7。

本节通过 IEEE CEC2017 函数来评估 HECO-HPSGWO。CEC2017 基准是 CEC2010 基准的升级版本。维度分别为 10、30、50。函数的解决方案排序方法与 CEC2006 中的类似：①在不可行解之前排序可行解；②根据函数值 $f(x)$对可行解进行排序；③根据违反所有约束的平均值对不可行解进行排序。CEC2017 为所有算法的排名提供了一套规则。比较的算法遵循相同的规则，如下所示。

（1）基于平均值对算法进行排序的程序：①基于可行性比率对算法进行排序；②根据平均违规量对算法进行排序；③基于目标函数平均值的排名算法。

（2）基于中值解的排名算法的过程：①可行的方案优先于不可行的方案；②可行的方案根据目标函数的值进行排名；③不可行的方案根据不可行的方案对约束的违反情况进行排名。

（3）对多个问题的所有算法进行排名：对于每个问题，算法的排名分别根据平均值和中值解确定。