软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2022,33(5):1774−1799 [doi: 10.13328/j.cnki.jos.006561]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于并行搜索优化的指控系统自适应决策方法*

王 璐, 霍其恩, 李青山, 王 展, 姜宇轩

(西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

通信作者: 李青山, E-mail: qshli@mail.xidian.edu.cn

摘 要: 指挥控制信息系统 (指控系统) 运行在动态变化的复杂环境中且任务需求时刻变更, 亟需一种自适应决策方法以动态产生调整系统的最优策略, 从而适应环境或任务变化, 确保系统长期稳定运行. 随着指控系统自身及其运行环境的持续复杂化, 自适应决策方法需具备应对多个非预期变化的在线权衡决策能力, 以避免造成冲突的调整后果或无法及时响应未知情况. 然而, 当前指控系统多采用基于先验知识、应对单一变化的自适应决策方法, 尚无法完全满足该能力需求. 因此, 提出了一种基于并行搜索优化的指控系统自适应决策方法. 方法采用基于搜索的软件工程思想, 将自适应决策问题建模为搜索优化问题, 并采用遗传粒子群算法, 实现针对同时发生的多个变化进行在线权衡的目标. 并且, 为解决该方法在指控系统中实际应用时存在的搜索效率保障、策略择优选择问题, 分别采用并行遗传算法和后优化理论, 对决策方法实现了并行化并建立了策略多指标排序法, 以确保方法的实用性.

关键词: 指挥控制信息系统; 自适应决策; 基于搜索的软件工程; 并行遗传算法; 后优化理论

中图法分类号: TP311

中文引用格式: 王璐, 霍其恩, 李青山, 王展, 姜宇轩. 基于并行搜索优化的指控系统自适应决策方法. 软件学报, 2022, 33(5): 1774–1799. http://www.jos.org.cn/1000-9825/6561.htm

英文引用格式: Wang L, Huo QE, Li QS, Wang Z, Jiang YX. Self-adaptation Decision-making Based on Parallel Search Optimization for Command and Control Information System. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1774–1799 (in Chinese). http://www.jos.org.cn/1000-9825/6561.htm

Self-adaptation Decision-making Based on Parallel Search Optimization for Command and Control Information System

WANG Lu, HUO Qi-En, LI Qing-Shan, WANG Zhan, JIANG Yu-Xuan

(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: The command and control information system (command and control system) runs in a dynamically changing and complex environment with constantly changed mission requirements. A self-adaptation decision-making method is urgently needed to dynamically generate the optimal strategy for adjusting the system, so as to adapt to changes in the environment or missions and ensure the long-term stable operation. At present, as the command and control system itself and its operating environment continue to become more complex, self-adaptation decision-making methods need to have the online trade-off decision-making ability to deal with multiple unexpected changes, so as to avoid conflicting adjustment consequences or failure to respond to unknown situations in a timely manner. Nevertheless, the current command and control system mostly adopts self-adaptation decision-making methods based on prior knowledge and responding to single changes, which cannot fully meet this capability requirement. Therefore, this study proposes a self-adaptation decision-making method for the command and control system based on parallel search optimization. This method uses search-based software engineering ideas to model the self-adaptation decision-making problem as a search optimization problem, and uses the genetic particle swarm algorithm to achieve the goal of online weighing against multiple changes that occur at the same time. In addition, in order to solve the

* 基金项目: 国家自然科学基金青年基金 (61902288); 国家自然科学基金 (61672401); 国家重点研发计划 (2019YFB1406404); 陕西省重点研发计划 (908014487064)

本文由"领域软件工程"专题特约编辑汤恩义副教授、江贺教授、陈俊洁副教授、李必信教授以及唐滨副教授推荐.

收稿时间: 2021-08-15; 修改时间: 2021-10-09; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

problems of search efficiency guarantee and strategy selection in the actual application of this method in the command and control system, this study uses parallel genetic algorithm and POST-optimization theory to parallelize the self-adaptation decision-making method and establish a strategy multi-index sorting method to ensure the practicality of the method.

Key words: command and control information system; self-adaptation decision-making; search-based software engineering; parallel genetic algorithm; POST-optimization

指挥控制信息系统软件[1](以下简称指控系统) 在国防与军队信息化建设中起到关键作用. 该类系统通过全面收集分析态势信息、动态管理分配作战资源, 为指挥人员提供对人力、物理、信息等资源的快速整合能力, 以辅助指挥人员实施最合适的战场决策. 当前, 指控系统的运行环境极易发生增加新的计算节点、关键传输链路失效、降效和赛博攻击等环境变化. 并且, 指控系统面临的任务需求具有动态性和阶段性, 在系统运行过程中会随时产生非预期的任务变更需求. 这些来自运行环境的变化与任务的变更需求, 如果未能被及时合理地响应, 严重时可导致系统整体失效. 因此, 指控系统亟需具备一种自适应能力, 使系统可通过调整自身组织结构或行为以适应动态变化的环境与作战任务.

为实现该自适应能力, 美军提出了“OODA” (observe-orient-decide-act) 理论, 建立了包括“观察、判断、决策和行动”4 个环节的自适应过程, 以实现系统对环境变化和任务变更需求的实时响应[2]. 其中, 观察环节是对系统运行环境或需求变更进行感知或观察, 从中搜集信息和数据; 判断环节是对当前的系统状态进行分析和评估, 并处理相关的信息和数据; 决策环节是根据运行环境信息和当前系统状态制定调整策略, 并选择合适的策略; 行动环节则是根据所选策略实施对系统的调整. 由此可见, 决策环节产生的调整策略最终决定了系统如何进行自适应. 该环节是决定能否成功调整指控系统以适应当前环境或变更任务的关键, 在指控领域的决策环节存在如下两方面迫切的能力需求.

首先, 由于指控系统运行在高动态环境下, 环境变化频繁发生, 且同一时刻可能同时存在多个环境变化. 考虑到指控系统本身是一个具有复杂运行机理的物理系统, 根据现实世界的规律与实践经验, 同一时刻发生的变化可能存在一定的关联关系. 这种关联关系会导致每个变化各自的调整策略极可能存在潜在冲突[3]. 例如, 为响应同时发生的多个变化, 系统将采取多个调整策略以不同方式调整同一个作战资源. 为解决上述问题, 则需自适应决策方法能够在具有冲突关系的变化间进行权衡, 产生折中方案.

其次, 由于系统运行环境瞬息万变且指控系统内部状态和行为空间爆炸, 在系统运行前几乎无法穷举所有可能的调整策略, 也无法预测策略的实际有效性. 这种策略的不确定性及效果的不可预测性, 导致自适应决策方法仅凭预定策略无法正确应对已知变化, 更无法处理未有相关策略的未知变化. 因此, 自适应决策方法必须具备根据变化情况, 动态产生调整策略的在线决策能力.

然而, 现有指控系统中采用的自适应决策方法并不能完全满足上述两方面的能力需求. 例如, 基于规则/策略的方法, 需在系统运行前预定义自适应策略, 无法支持在线决策. 基于目标、效用函数和优化函数的决策方法, 每次仅处理单一变化, 容易产生出互相冲突的自适应策略. 虽然也存在部分研究通过定义变化间的权重关系产生折中策略, 但由于变化间的优先关系在系统运行过程中会发生演化, 因此该类研究无法保证其始终有效[4]. 而通用软件的自适应决策方法在应用于指控领域时, 也存在一定的不适用性. 例如, 基于马尔科夫链、贝叶斯网络等自适应决策方法虽具有应对变化不确定性的能力, 但该类方法需预先建立系统状态迁移模型, 这对指控系统而言复杂性较高. 而基于强化学习等决策方法需根据环境反馈反复试错产生最优策略, 不适用于指控系统这种高安全要求的应用领域.

本文拟采用基于搜索的软件工程 (search-based software engineering, SBSE) 理论, 将自适应决策视作在调整策略组成的搜索空间中, 评价并选择最优策略的搜索优化问题, 采用搜索优化方法建立一种满足在线权衡决策需求的自适应决策方法, 从而有效应对环境变化与任务需求变更. 该类决策方法可采用多目标搜索优化技术, 在不定义变化优先级的情况下, 实现对多种变化的权衡处理, 并可动态根据非预期变化情况在线搜索产生出相应的调整策略.

然而, 在应用该方法具体解决指控系统的自适应决策问题时, 仍存在效率提升与策略选择等实际应用问题. 首

先, 该方法的计算效率受搜索空间规模影响. 而指控系统对自适应过程的实时性要求较高. 因此, 必须保障该方法能够快速产生决策结果. 其次, 该类方法针对多目标进行权衡决策时, 会产生一个最优策略集合, 即前沿. 前沿内部的策略已无法再根据优化目标区分优劣性. 然而, 指控系统必须依据唯一的调整策略对系统结构、行为或参数实现准确调整. 因此, 必须保证该方法可从前沿中选择出当下最适用策略.

因此, 本文提出了一种基于并行搜索优化的指控系统自适应决策方法 (self-adaptation decision-making based on parallel search optimization for command and control information system), 通过将自适应决策问题转化为搜索优化问题并设计了多目标优化算法以满足指控系统在线决策及决策权衡的需求, 同时为提升决策效率设计了基于后优化理论的策略选择方法, 能够有针对性的选择当下最适用策略. 具体包括以下要点:

1) 自适应决策问题建模. 本文分析自适应决策问题特征, 定义了该类问题的自适应策略、策略空间、目标函数、适应度函数与约束函数, 将自适应决策问题建模为搜索优化问题. 本文设计的自适应决策建模方法可以根据指控系统特征, 快速建模独特的自适应决策问题模型, 将自适应决策问题转化为搜索优化问题, 针对指控系统环境变化动态生成策略空间.

2) 基于并行的多目标优化算法. 本文结合并行遗传算法、粒子群算法等理论, 建立了基于遗传粒子群 (genetic algorithm and particle swarm optimization, GAPSO) 的并行自适应决策方法, 并通过算法的并行化设计保障算法可快速产生调整策略. 通过此方法, 不仅能够针对指控系统复杂运行环境下的多个环境变化在线产生最优策略, 避免了由于仅考虑单个变化引起的策略冲突. 此外, 本文通过种群切割、引入迁移算子等方法实现了算法的并行化设计提升了决策效率.

3) 基于后优化理论的策略选择方法. 本文基于后优化理论, 针对不同决策问题在决策偏好、时效性约束等方面的不同特征, 建立了基于和谐性分析法 (elimination et choice translating reality, ELECTRE) 的多指标排序法, 能够有针对性地选择当下最适用策略, 从而指导指控系统实现自适应调整.

本文第 1 节介绍了相关背景知识. 第 2 节给出了本文的研究框架, 说明了各研究工作之间的关系. 第 3 节则分别介绍了本文的核心研究工作, 即自适应决策问题建模、基于并行 GAPSO 的自适应决策方法、及基于 ELECTRE 的多指标排序法. 在第 4 节中, 本文分别采用了指控系统典型自适应场景和大规模模拟场景对本文提出的方法及相关研究方法进行了对比实验, 并对实验结果进行了分析探讨. 第 5 节分析了相关工作. 第 6 节总结了本文工作.

1 背景知识

本节介绍基于搜索的自适应决策方法所采用的理论技术, 即基于搜索的软件工程、并行遗传算法与后优化理论.

1.1 基于搜索的软件工程

基于搜索的软件工程的思想是通过将传统软件工程中的问题映射为基于搜索的优化问题, 通过使用启发式搜索算法, 在可行解空间中找到最优解的方法 [5]. 相比启发式搜索算法, 现代启发式搜索算法定义智能搜索策略, 增强启发式算法搜索性能, 以一种智能的方式搜索问题的最优解或者近似最优解.

传统的软件工程的思想是通过针对问题设计解决算法以求得最优解, 基于搜索的软件工程是通过在可行解空间中针对问题构造适度函数以指引搜索方向, 经多次迭代得到最优解. 因此, 基于搜索的软件工程解决问题一般需要满足两个条件.

1) 设计出问题解决方案表达方式 (solution representation): 对所需解决问题的结果, 必须能通过相应的编码表示出来, 以构成搜索算法中的染色体, 进行相应的运算.

2) 设计出相应的适应度函数 (fitness function): 对解进行评价, 比较不同解之间的优劣. 在搜索解空间内, 适应度函数可以指引搜索的方向, 寻找满足条件的区域.

由于基于搜索的软件工程解决问题的方法主要由以上两步组成, 针对任何问题, 只要能够设计出问题解的表

示方式和适应度函数, 就可以应用该方法, 因此具有很强的普适性, 可以方便地应用到不同领域问题上. 另外, 针对同一问题的不同规模, 基于搜索的方法求解方式是不变的, 因此容易扩展到大规模的工程问题求解上去. 基于搜索的软件工程方法避免了传统方法的一些不足, 能在尽可能降低成本的前提下尽可能高效地自动化完成软件开发任务, 是软件工程领域中一种高效实用的新方法.

本文所要解决的问题为自适应决策问题, 自适应决策问题具体是指在系统面临一系列难以应对的环境变化时, 需要系统产生相应策略以自适应的调整行为和状态, 在这个过程中遇到的策略生成、策略权衡选择等一系列问题即为自适应决策问题. 依据上述的基于搜索的软件工程方法, 本文设计了将自适应决策问题映射为最优化问题的方法以实现获取自适应决策问题的最优解. 由于最优化问题是指通过选择一组决策变量, 在满足一系列有关的限制条件 (约束) 下, 使设计目标函数达到最优值. 因此, 在建模过程中同样需要明确自适应问题中的决策变量、目标函数以及约束条件等内容. 具体含义如下:

1) 决策变量是指影响目标函极值的变量, 也是优化算法搜索的对象.

2) 目标函数是指最优化问题优化的目标.

3) 约束条件是指限制决策变量取值的条件, 在数学描述中被称为约束函数, 依据限制条件不同可分为等式约束和不等式两种.

1.2 并行遗传算法

遗传算法 (genetic algorithm, GA)[6]执行流程同动物种群的遗传机制类似, 其采用交叉、变异操作来达到模拟动物种群进化中个体的基因组改变的结果. 采用“优胜劣汰”的法则, 先保留较优的个体形成下一代种群, 最终通过迭代使种群状态处于最优或近似最优状态. 然而, 随着实际问题的复杂性和规模呈现出不断增加的趋势, 如何提升 GA 的求解速度以满足求解实际问题的需求已成为一个非常关键的问题.

随着计算机计算能力的提升和大规模并行计算框架的发展, 利用集群的计算能力提升 GA 的运行速度的想法促进了并行遗传算法 (parallel genetic algorithm, PGA)[7]的诞生. PGA 是指对 GA 进行并行化后形成一类进化算法, 适合解决大型复杂问题. 借助多种群并行进化, 使得 PGA 在避免 GA 陷入局部收敛问题的同时提升了种群的多样性, 并且全局搜索能力也有了较大提升. GA 并行化的设计主要在 GA 算法流程及 GA 结构进行改进, 算法流程由串行改为并行, 其结构也改进形成并行群体模型以适应并行过程, 以支撑多种群并行进化.

目前 PGA 的模型可以分为主从式、粗粒度、细粒度和混合模型共 4 大类型. 主从式模型是指选择、交叉和变异等全局操作由主节点串行进行, 而计算由各从节点机并行执行; 粗粒度模型是适应性最强、应用最广的遗传算法并行模型, 其原理是将种群划分至多个节点并行处理; 细粒度模型相比于粗粒度模型要求子群体的划分要更加细小, 理想状态是每个节点机只有一个个体; 混合模型是指将前 3 种模型混合形成模型结构. 四种模型中主从模型通信代价过高, 细粒度模型划分过于细致需要占用大量计算节点, 混合模型设计较为复杂, 而粗粒度模型需要的计算节点较少且各节点能独立完成遗传进化. 因此, 本文选用了粗粒度模型作为建立并行搜索方法的基础模型. 粗粒度模型具体是通过利用了 GA 算法结构在种群层的并行性, 实现多个子种群的并行进化. 假设并行计算集群中计算节点的个数为 n, 该模型就将 GA 的初始种群划分为 n 个子种群. 每个子种群将被分配到一个计算节点中, 独立采用 GA 进行运算并不断迭代. 在迭代过程中, 各个计算节点的子种群将采用迁移操作定期进行种群迁移, 可丰富种群的多样性并为种群引入新的优良个体, 以避免算法产生早熟收敛问题. 其中, 迁移操作是该类模型为 GA 加入的一种新进化操作. 该操作规定了种群进行迁移周期、个体数量、迁移策略和拓扑信息等.

1.3 后优化理论

完整的基于搜索的自适应决策方法除了通过多目标优化方法产生的前沿策略集合之外, 还需要解决后优化 (POST-optimization) 问题[8], 即缩小前沿集合规模, 从中选择出最优的解. 后优化问题主要针对优化算法的计算结果进行探索, 主要分为两种类型. 第 1 类后优化问题是确定优化算法参数 (即 right-hand-side value 和成本系数等) 的取值范围, 在对算法的解不会造成显著影响的前提下, 扰动最优解, 以探索是否能够获得新的最优解, 即使新的最优解超出可行解空间范围也可以接受. 第 2 类后优化问题则是缩小多目标优化算法获得前沿集合规模, 从中选

择出最优的解.

而由多目标函数优化算法产生的前沿集合与基于搜索的自适应决策方法产生的最优方案集合高度相符. 因此, “从方案集合中选择唯一策略”的问题本质上与第 2 类后优化问题“从前沿中选择最优解”一致. 因此, 本文将考虑采用与第 2 种后优化问题相关的理论技术建立本文的方案选择机制. 为方便论述, 后文提及“后优化问题”时, 仅特别指代“第 2 类后优化问题”.

目前, 根据现有文献的分析, 针对后优化问题的解决方法, 即后优化方法, 可被分为以下 4 种. 第 1 种是偏好排序法, 即通过对前沿中的解根据设定的偏好进行排序, 从而获得排序后的最优解[9]. 该类方法的实施难点就在于如何定义恰当的排序方法. 第 2 种图像法通过对前沿在值空间中的空间形态进行分析[10], 确定图像拐点 (Knee), 并认为处在拐点上的解一般是在多个目标间获得最佳权衡的解. 对于多目标优化问题, 前沿为超平面, 图像法需发掘其凸出或凹陷部分, 时间效率较低. 第 3 种聚类算法通过对前沿中的解进行聚类[11], 形成若干簇后, 从簇中取具有代表性的解为最优解, 但也需解决时间效率问题. 前 3 种方法解决后优化问题的时机都在于优化问题求解完成, 形成前沿后才开始的. 而第 4 种交互式方法, 则是在生成前沿过程中, 就不断与用户交流, 获得用户偏好以指导前沿生成, 该类方法不适合于实时性要求高的场景应用.

因此, 本文拟通过 ELECTRE (elimination et choice translating reality) 方法建立一种多指标排序方法以从多个生成策略种选择出唯一策略, 该方法与其他依赖于偏好排序的方法不同, ELECTRE 方法可考虑多种指标避免了完全按照偏好造成的风险, 相比于图像方法和聚类算法 ELECTRE 较为简单能够快速实现策略选择的任务. ELECTRE 方法是由 Bernard Roy 在 1965 年提出的一种多标准决策分析 (multi-criteria decision analysis, MCDA) 方法. 该方法是通过建立方案间的优先次序关系从而实现方案排序[12]. 通过和谐性检验和非和谐性检验来比较不同方案 x 和 y 的优劣性. 其中, 首先通过和谐性检验证明 x 至少与 y 一样好, 然后通过非谐性检验证明上述观点没有强烈有效的反对理由, 倘若方案 x 与 y 同时满足了这两项验证, 则认为方案 x 优于方案 y.

2 研究框架

本文的研究问题包括如何通过搜索优化技术建立自适应决策方法, 以及如何保证该决策方法可在实际指控系统中应用. 因此, 本文建立了如图 1 所示的基于搜索的自适应决策框架, 分别解决指控系统自适应过程中的“在线权衡决策问题”, 及在实际应用时突显出的“效率保障问题”和“策略择优问题”.

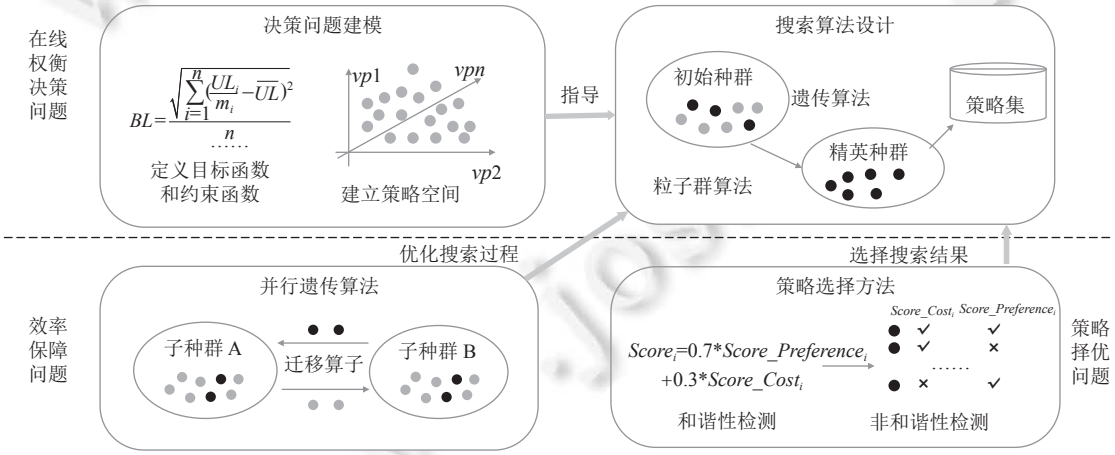


图 1 基于搜索的自适应决策框架

(1) 自适应决策问题建模与搜索算法设计, 解决在线权衡决策问题

本文首先分析了自适应决策问题的本质特征并与最优化问题进行了类比, 随后定义了自适应决策问题的决策变量、目标函数和约束函数, 将自适应决策问题建模为最优化问题. 在此基础上, 本文首先采用全局搜索能力强的

进化算法对空间进行初步搜索. 在经过多次迭代后, 本文挑选出精英个体, 并进一步采用收敛能力强的粒子群算法实现局部深入搜索, 获得最终结果.

(2) 基于并行 GAPSO 的自适应决策方法, 解决效率保障问题

在本文建立的决策算法中, 前期采用的进化算法承担了对策略空间的全局搜索任务. 因此, 提升该算法效率将有助于保障整体决策效率. 根据并行遗传算法思想, 本文建立了基于并行 GAPSO (genetic algorithm and particle swarm optimization) 的自适应决策方法, 采用迁移算子划分全局搜索空间并对多个种群进行并行搜索, 达到加速算法收敛目的. 并且, 本文采用 Spark 并行计算框架和多线程技术两种方式实现该算法, 以对比不同实现方式对算法时间效率的影响.

(3) 基于 ELECTRE 的多指标排序法, 解决策略择优问题

本文基于排序法中的 ELECTRE 方法建立了多指标排序法, 以从搜索结果中选择出唯一策略. 在设计排序指标时, 本文除了量化建模指挥员偏好外, 将充分考虑不同战场环境下的决策需求, 建立了作战偏好、资源态势、任务时效性等指控系统关切的特殊指标, 通过开展和谐性与非和谐性检验, 逐步精化搜索结果, 达到选择最优策略的目标.

3 基于并行搜索优化的自适应决策方法

3.1 自适应决策问题建模

根据上文将自适应问题映射为最优化问题的基本理论, 本节对其具体方法进行描述. 在自适应问题中影响其调整结果的变量为自适应策略, 因此, 本文将自适应策略作为自适应问题的决策变量. 在指控系统中自适应策略包含了大量系统信息及系统行为, 为更好的描述指控系统的自适应策略本文将自适应策略的组成部分定义为可变点 (variable point, vp), 即在自适应调整过程中对调整结果可以产生影响的可调整对象, 这些可调整对象不仅包含指控系统资源 (如计算节点 CPU、内存的利用率等)、指控系统的组织结构 (如组成作战单元数量、部署位置等)、组成单元的行为或参数 (如系统组成单元的功能等), 同时也包括指控系统运行中可调整的属性 (如网络带宽等).

由于可变点的取值情况不同, 因此可变点可被划分为离散可变点、连续可变点和量化可变点.

• 离散可变点是指可变的取值为离散数值. 如分布式计算节点只能以离散数值描述.

• 连续可变点是指可变点的取值时连续的. 例如内存利用率可以用 [0, 100%] 的连续数值进行描述.

• 量化可变点是指可变点中有些没有具体的数字化取值, 因此需要对这些可变点进行数字量化. 例如可以将 3 种不同的系统行为以{1, 2, 3}这 3 个数值代表, 实现可变点量化.

由于决策变量是指自适应策略, 因此将这些可变点组合起来就形成了自适应策略集合即决策变量取值, 在基于搜索的软件工程中也被称为解空间. 如图 2 所示为决策变量形成组合构成的解空间, 其中每个可变点都是解空间中的一条坐标轴, 解空间中的一个点表示可变点的一个组合.

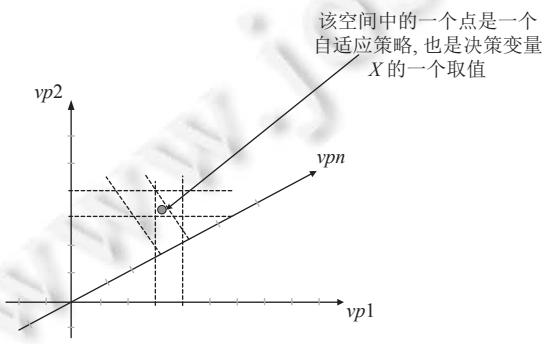


图 2 自适应决策问题的可行解空间

上述方式建立的解空间是通过在线分析可变点 (即系统状态及行为) 产生的, 不存在预定的关系, 同时由于解空间中的每个点均是独立的, 因此不同策略之间不存在冲突. 这种方式可以有效避免策略冲突, 动态选择出最优的自适应策略以实现在线决策.

(1) 目标函数

在指控系统中自适应决策问题的优化目标就是使需要自适应调整的系统能够适应软件变化, 以达到期望的系统目标. 在本文中软件变化具体是指由于环境变化和战场需求变更等因素引起的系统内部变化, 这些变化导致了系统无法正常稳定运行. 本文以期望调整后稳定的系统目标作为目标函数, 并将其分为直接相关函数和间接相关函数两部分.

首先, 需要明确在软件变化发生后期望的系统目标. 系统目标可以由两种方式获得. 一个是通过指控领域专家经验获取; 另一个是通过多次模拟战场环境变化和需求变更等场景进行多次实验, 观察受到影响的系统资源和行为, 受到影响的部分即为与此变化直接相关的目标函数. 例如, 在“负载均衡”这个场景下可能会发生“节点负载超载”变化, 为获取这个变化的直接目标函数, 需要重复进行实验, 通过分析系统运行日志可以发现“系统响应时间”受到了影响. 因此“响应时间”就是该变化的一个直接相关目标函数.

其次, 由于多种环境变化和需求之间可能存在联系. 因此, 不能简单考虑直接相关的目标函数, 通过对多种软件变化之间进行分析, 可明确相关的可变点信息. 受到这些可变点取值变化影响的系统目标, 则称为该变化的间接相关目标函数. 通过在自适应决策问题中引入间接目标函数, 可以在一定程度上避免由于软件变化之间的联系引起的反复调整等一系列系统振荡现象. 例如, 在“负载均衡”场景下, 与“节点负载超载”变化相关的可变点是节点的任务分配量, 同时这个可变点影响着系统的运行开销. 虽然系统运行开销暂时并未受到“节点负载超载”的影响, 但从长远的角度上, 将其纳入间接目标函数可避免由于调整不当引起的“系统运行开销过大”等软件变化.

由于指控系统的运行环境复杂, 同一时刻可能同时存在多个环境变化和任务需求变更的发生, 自适应决策需要同时考虑多种变化. 因此, 需要将这些变化的直接相关目标函数和间接相关目标函数合并去重, 作为自适应决策的整体目标函数且这些目标函数之间不用定义任何的优先关系, 并将通过量化公式的形式定义. 具体公式内容需要根据不同系统具体的决策问题需要而建立.

(2) 约束函数

在自适应决策问题中, 约束函数仍旧是传统优化问题中的等式或不等式约束形式, 主要用于对可变点的取值范围进行限制, 但其产生的来源则包括如下两种.

• 可变点约束主要源于自适应调整系统运行时上下文对可变点取值的特殊要求. 例如, “集群计算节点数量”这个可变点的取值范围本身是明确的, 但如果系统管理人员临时规定了计算节点的可用数量, 则这个可变点的取值就将受到影响.

• 功能约束则源于用户对系统的功能期望. 自适应策略不仅要保证决策问题中目标函数的最优, 也要保证在基于该策略调整系统后, 系统必须对外提供的服务和服务水平不能受到影响. 也就是说, 该类约束函数主要用于保证可变点的取值不会影响系统为用户提供必备的对外功能.

3.2 基于并行 GAPSO 的自适应决策方法

根据上一节所述方法本文实现了在指控系统下的自适应决策问题建模. 如何从模型确定的可行解空间, 也就是可行自适应策略空间中选择最优自适应策略是本节将解答的问题.

目前, 工业界采用最多的多目标优化算法为遗传算法 (例如, nondominated sorting genetic algorithm-II, NSGA-II) 与群体智能算法 (例如, multi-objective particle swarm optimization, MOPSO)[13]. 遗传算法具有全局搜索、离散连续空间均可、整体移动、广泛搜索等优势, 但也存在收敛速度慢及陷入局部最优而早熟等问题. 而群体智能算法因为结构设计简单, 适合现实场景使用, 但其也存在与遗传算法相同的早熟问题, 且搜索精度较低、后期迭代速度慢等问题. 然而, 指控系统的自适应决策问题具有较强的实时性, 其决策效率是首先需要保证的要素, 如果不能快速产生决策结果, 决策过程中系统和运行环境极有可能继续变化, 那么所选择的调整方案极有可能已不适合新

的环境.

因此, 本文提出了一种基于遗传算法的并行遗传粒子群 (genetic algorithm and particle swarm optimization, GAPSO) 算法以搜索最优自适应策略并加快演化策略的生成速度, 提升搜索效率. 该算法主要由两部分组成, 分别是用于算法前期迭代的多目标优化算法 NSGA-II, 用于算法后期精确搜索的 NSGA-II 算法. NSGA-II 算法时间开销较小, 故用于实现算法前期对策略空间的全面覆盖搜索, 避免算法陷入局部最优. 因为可行自适应策略空间中的点是离散分布的, 所以在算法执行后期, 采用离散的 MOPSO 算法以对局部精英种群进行深入搜索, 缩短算法收敛时间. 此外, 为进一步提高搜索效率, 本文结合粗粒度并行遗传算法思想, 通过将大种群切分为若干子种群实现并行进化, 并通过引入迁移算子实现子种群间个体迁移, 保障各子种群的多样性. 算法流程如算法 1 所示.

算法 1. GAPSO 算法流程.

Input: 目标函数、约束函数、决策变量及算法参数, 迁移代数 T.

Output: 前沿集合.

1. NSGA-II 初始化. 在该步骤中获得算法参数, 产生算法运算所需要的初始群体.

2. 切分初始种群为 n 个子种群.

3. 对于每一个子种群 n, 在得到初始种群后, 根据适应度函数计算每个个体的适应度值. 对种群进行非支配排序, 拥挤度计算.

4. 对子种群中的个体进行交叉、变异操作, 计算个体的适应度值.

5. 由 3, 4 两步的中间种群组合得到下一代种群, 并对个体按照适应度值进行排序, 保留固定种群大小数目个体, 抛弃多余个体.

6. 获取当前迭代次数. 若迭代次数为 T 的倍数, 则采用迁移算子进行各个子种群之间的个体迁移; 若非 T 的倍数, 则转向步骤 7.

7. 判断是否达到 NSGA-II 的终止条件, 若满足, 则转步骤 8; 否则, 转步骤 3.

8. 确定种群大小、维度、每个维度上的取值及微粒的初始位置和初始速度, 微粒在每个维度上位置和速度的上下界限值.

9. 以前沿策略为初始粒子群, 按照适应度排序及拥挤度排序结果抛弃多余微粒.

10. 计算每一个微粒的适应度值, 进行非支配排序, 记录个体最优位置 *Pbest* (一组) 和群体最优位置 *Gbest* (一组), 将 *Gbest* 保存在最优集合中 (归档), 更新微粒的位置与速度.

11. 判断迭代次数, 决定是否结束算法, 若达到算法终止条件, 则将最优集合进行非支配排序, 输出前沿集合

如算法 1 所示, 并行 GAPSO 算法过程如下.

(1) 初始化. 在该步骤中确定 NSGA-II 算法和 MOPSO 算法执行所需要的群体特征. 主要为种群规模、维度、各维度上取值约束、粒子的初始化信息及粒子在各维度上位置和速度的取值约束.

(2) 子种群进化. 将初始种群划分为若干子种群. 每个子种群根据适应度函数计算每个个体的适应度值. 对个体进行非支配排序, 拥挤度计算, 并进行交叉、变异操作, 更新个体的适应度值. 不同子种群通过迁移算子交换优势个体, 不断进化迭代.

(3) 深度寻优. 达到 NSGA-II 的终止条件后, 确定种群大小、维度、每个维度上的取值及微粒的初始位置和初始速度, 微粒在每个维度上位置和速度的上下界限值以前沿策略为初始粒子群, 按照适应度排序及拥挤度排序结果抛弃多余微粒. 计算每一个微粒的适应度值, 进行非支配排序, 计算并记录每个个体的最优位置 *Pbest* 和群体最优位置 *Gbest*, 将 *Gbest* 保存在最优集合中 (归档), 并选择唯一的 *Gbest*, 随后更新每个微粒的速度.

(4) 算法终止. 判断迭代次数, 决定是否结束算法, 若达到算法终止条件, 则将最优集合进行非支配排序, 输出前沿.

结合上述算法流程, 给出本文 GAPSO 算法设计细节, 如表 1 所示, 对号表示相应算法需考虑该设计内容.

并行 GAPSO 算法设计存在着一定的挑战和难点. 首先, 由于遗传算法一般采用二进制编码方法对个体进行编码该编码方式易于实现变异等进化操作. 然而, 自适应决策问题的决策变量取值被离散化了, 如果采用二进制编码, 则容易产生无效取值, 且由于决策变量的取值随时可能会发生变化, 因此针对自适应决策变量的编码方式必须支持决策变量取值的灵活调整. 其次, 本文将在算法前期迭代将切分出若干子种群实现并行进化, 然而如果种群切分过大会导致收敛效率过低, 种群切分过小又会导致陷入局部最优解, 需要考虑在结果质量和收敛速度间寻找平衡, 可以接受的时间范围内尽可能的提供较优的结果. 为解决上述问题, 在算法中进行了相关的设计如表 1 所示, 部分内容细节如下所示.

表 1 算法细节设计

内容	说明	算法		本文设计
		NSGA-II	离散MOPSO	
个体编码	表征个体的方式	√	√	见“(1)编码方法”
初始种群	种群的初始化生成方式	√		随机生成方式
				在GA最优解附近随机生成
			√	以NSGA-II前沿为初始种群
适应度函数	评价个体优劣性的标准		√	将目标函数作为适应度函数
		√		Pareto支配关系和拥挤度
选择算子	规定保留父代种群哪些个体进入子代种群的操作			采用线性排序法选择个体, 并采用精英选择法直接保留最优个体
		√		带有精英策略的选择算子
交叉算子	规定个体间如何交互部分基因的操作	√		单点交叉即交换点位k之后的p个基因位
交叉率	规定采用交叉算子的个体规模	√		0.6至1设置得稍高以推进搜索进程
变异算子	规定修改个体哪些基因位取值的操作	√		采用单点变异随机选择一个基因位, 并采用均匀变异随机选择一个取值代替原值
变异概率	规定采用变异算子的个体规模	√		0.005至0.1, 实际中可设置较高, 以扩大搜索范围
速度更新操作	决定个体飞向最优解解的步长和方向		√	将标准速度更新操作直接应用于离散空间
位置更新操作	记录个体目前的取值情况		√	将标准位置更新操作直接应用于离散空间, 对个体位置进行近似取整
学习因子	调节个体向最优解学习的步长		√	采用通用取值2
随机函数	增加搜索随机性		√	取值范围[0, 1]二者相互独立
非支配集构造	依据多个目标函数选出每代较优解集		√	快速非支配排序法
外部存档维护	存放每次迭代后获得的非支配解		√	聚集密度法, 即采用拥挤距离评价个体, 定期剪辑非支配集
个体最优选取	记录个体所经历过的最好解			采用标准个体最优值更新方式
			√	对比历史最优位置与新位置非支配关系
全局最优选取	从所有个体最优值中选择最优的解			采用标准全局最优选择方式
			√	见“(2)群体最优选择”
惯性权重	保持个体运动惯性		√	见“(3)惯性权重设计”
终止条件	终止算法的条件	√	√	见“(4)终止条件设计”
迁移算子	定义子种群个体交换规则	√		见“(5)迁移算子设计”

(1) 编码方法

正在运行的系统中, 决策变量取值发生变化的可能性较大. 因此, 自适应决策变量在编码层面需要对变量取值的灵活调整提供支持. 本文特别提出了一种数组编码方法, 对决策变量, 即 GA 与 PSO 算法中的个体进行编码. 如图 3 所示, 决策变量中每一个可变点的所有取值被映射到一个数组. 数组的每一项对应一个取值. 决策变量的编码长度取决于自适应决策实际问题中的可变点数量.

(2) 群体最优选择

对于 MOPSO, 本文使用理想解法 (technique for order preference by similarity to an ideal Solution, TOPSIS) 用于群体最优的选取. 该方法在初次迭代时会预选选取每个目标函数的最优极值和最劣极值, 并以此作为理想最优值和理想最劣值. 该方法通过计算各个个体最优与理想最优值和理想最劣值之间的距离, 将离理想最优值最近且离理想最劣值最远的个体最优值为群体最优值. 此外, 通过比较最新的最优值与历史群体最优值的非支配排序关系, 以确定最新群体最优值. 该方法能够有效避免随机选择方法不准确性带来的影响及聚集密度等方法的计算复杂度等问题.

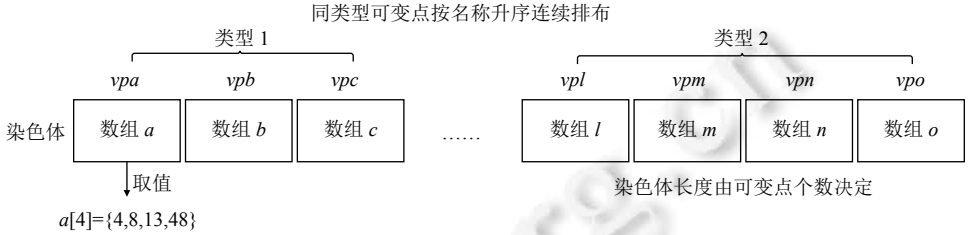


图 3 基于数组的染色体编码过程

(3) 惯性权重设计

对于 PSO 算法的惯性权重, 本文根据“随着算法迭代的进行, 惯性权重进行减小, 有助于保证算法前期迭代的全局搜索和后期迭代的局部收敛能力”的研究结论[14], 采用了根据迭代次数递减惯性权重值的方式. 计算过程如式 (1) 所示.

$$w = W_{ub} - \frac{times}{MAX\_ITER}(W_{ub} - W_{lb}) \tag{1}$$

其中, $times$ 表示的是当前迭代次数, W_{ub} 与 W_{lb} 分别表示为惯性因子的上下界, 在本文中规定 W_{ub} 取值为 1, 而 W_{lb} 取值为 0. MAX_ITER 表示迭代最大周期. 公式 (1) 中惯性权重将随着迭代次数的增加而减小.

(4) 终止条件设计

考虑到自适应决策问题需要尽快获得决策结果, 且一般实际工程问题并不特别要求获得真正意义上的最优解, 因此本文采用规定迭代次数, 达到时即终止算法.

当迭代次数达到规定的数值便停止迭代, 终止算法, 迭代次数的取值则是依据专家经验及系统历史情况确定. 本文的终止条件兼顾了时间开销和决策结果, 在实际场景中更具应用价值.

(5) 迁移算子设计

本文采用了并行遗传算法的思想, 引入迁移算子, 用于实现各个子种群中个体的迁移. 迁移算子主要作用是控制各个子种群间个体的迁移, 如式 (2) 所示.

$$Transport = \{\text{种群迁移率, 迁移周期, 迁移策略, 迁移拓扑}\} \tag{2}$$

式 (2) 中种群迁移率指子种群之间迁入新个体占原子种群的比重; 迁移周期指子种群之间进行个体迁移的时间间隔; 迁移策略是指子种群之间个体迁移策略, 一般包括如何选择向外迁移的个体、种群接收个体后的置换策略以及迁出个体的保留策略; 迁移拓扑是指种群之间个体的迁移路径.

本文遗传算子设置如图 4 所示, 经过多次实验, 为保证并行效率本文将种群迁移率设置为 10%. 关于迁移策略, 目前的主要替换方式包括替换最差的个体或随机替换两类, 其中选择选择最优个体可加速算法的收敛效率, 然而选择随机个体迁移, 在一段时间后, 可能会因为扩展了种群的多样性而获得更好的效果. 本文综合考虑了上述两种方法, 并考虑到算法的收敛效率更为重要, 因此设计迁移策略为选择 8% 最优个体替换迁入种群的最差个体, 2% 随机个体替换迁入种群的 2% 随机个体; 同时, 为减少开销, 本文选择同步迁移的思路, 每隔 T 代迁移一次; 迁移拓扑采用单向环形拓扑.

引入迁移算子后, GAPSO 算法将划分出若干子种群, 通过采用多线程的技术并行搜索方案空间, 并通过迁移

算子实现个体迁移和信息交互, 最终获得最优解集, 实现并行搜索提高搜索效率.

3.3 基于 ELECTRE 的多指标排序法

针对指控系统策略的战场空间大、不确定因素多, 情况可变性强的特点, 在战时决策中既需要提高决策效率, 又要确保决策的科学性. 因此, 本文基于 ELECTRE (elimination et choice translating reality) 方法, 量化指挥员偏好、调整开销和调整时间等与指控系统相关的评价指标, 然后结合和谐性和非和谐性的概念, 设定了两种检验方法, 实现对前沿自适应策略集合的评估和排序.

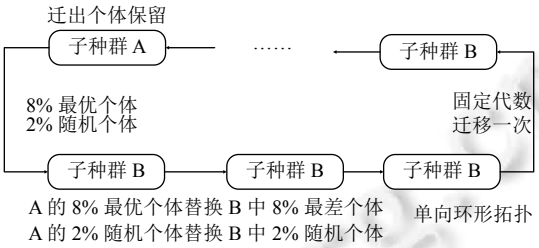


图 4 迁移算子设计

在和谐性检验方面, 本文量化了用户偏好和调整开销的评价指标, 在此基础上, 对比前沿调整策略的质量, 从而解决由于不同目标函数产生的结果不同导致的调整策略不唯一的问题, 进一步优化选择最佳策略.

其中, 在战场环境下的用户偏好主要指的是指挥偏好, 是对不同目标函数的设置的权重值, 通常在策略生成过程中进行; 在此基础上, 通过加权计算出方案 i 综合所有目标函数后的总得分, 即用户偏好得分 $Score\_Preference_i$.

调整开销表示自适应决策机制执行特定调整方案的演化重构开销. 包含资源消耗 $cost_{\text{resources}}$, 时间消耗 $cost_{\text{time}}$ 和范围影响 $cost_{\text{scope}}$ 等. 首先, 根据专家经验或实际测试计算得出开销之和表示调整方案 i 的调整开销 $Cost_i$; 然后, 通过综合考虑策略 i 的调整开销占所有策略的调整开销的比值进行计算出调整开销得分 $Score\_Cost_i$. 调整开销得分越大, 则表示该策略在所有前沿集合调整策略中更具有优势.

在上述两种评价指标的基础上, 本文结合自适应系统决策经验, 通过对不同的评价指标设定权重值, 计算策略总得分. 其中, 本文考虑指控系统的自适应决策需求并通过咨询专家经验, 认为在战场环境下策略选择过程中, 用户偏好对决策结果的影响力应大于调整开销, 为此本文将用户偏好的比重设置为 0.7, 调整开销比重设置为 0.3, 如公式 (3) 所示可计算最终的总得分 $Score_i$.

$$Score_i = 0.7 \times Score\_Preference_i + 0.3 \times Score\_Cost_i \tag{3}$$

在非和谐性检验方面, 本文设定了用户对评估指标和目标函数所能够接受的阈值, 其中包括对用户偏好、调整开销以及目标函数结果 3 个部分. 通过将阈值作为调整策略是否非和谐的衡量依据, 筛选出所有得分都在合理阈值之内策略, 并进一步通过和谐性检验方法选择最佳策略, 能够满足用户偏好, 有效提高调整策略的合理性和适用性.

在上述两个部分的策略评估方法的基础上, 本文建立了多指标排序法的主要流程如算法 2 所示. 该方法首先获取可变节点的状态信息, 计算出各调整策略的总得分, 然后通过和谐性检验和非和谐性检验依次筛选, 最终选择出唯一的最佳调整策略.

因此, 本文所提出的多指标排序方法能够适应较为复杂的战场环境, 针对多目标函数的自适应决策问题, 综合分析多种影响因素对调整策略的影响, 并对不同的调整策略进行量化评分, 从而选择出唯一策略. 该方法结合了用户指控偏好, 并利用专家经验处理实际战场指控的需求, 能够有效弥补人工参与的局限性, 有利于提高自适应决策结果的有效性和合理性.

4 实验验证与分析

本节介绍了本文对基于并行搜索的自适应决策方法在军事指挥控制系统领域中开展的实验及结果分析工作.

首先介绍实验的具体方法, 然后给出实验的实际结果并进行分析讨论.

4.1 实验设计

为验证方法的有效性, 本文尝试设计了 3 个实验对其进行验证.

算法 2. 多指标排序法.

Input: 前沿策略集; 目标的权重; 可变点调整开销; 和谐性指标阈值; 目标函数阈值;
Output: 最优方案 *Pi*.

1. for i=1 to $P.size$
2. 计算用户偏好得分 $Score\_Preference_i$
3. 计算调整开销得分 $Score\_Cost_i$
4. 根据式 (3) 计算方案总得分 $Score_i$
5. end
6. sort(P, $P.Score$); //将策略集按每个目标得分降序排列
7. for i=1 to $P.size$
8. for j=1 to 和谐性检验指标个数
9. if 第 i 条策略的第 j 个和谐性检测指标< 其第 j 个和谐性检验指标阈值
10. then 剔除此方案;
11. end
12. for j=1 to 目标函数阈值个数
13. if $P.score_j$< 第 j 个目标函数阈值
14. then 剔除此方案;
15. end
16. end
17. Print *Pi* //输出最优策略

4.1.1 RQ1. 本文提出的决策方法在指控系统的运行场景下是否能产生有效策略?

本文模拟了军事系统中的搜索子系统作为案例系统, 并在系统中选择典型场景, 测试方法的策略效果, 以此评价方法效果. 具体实验设计如下.

自适应控制平台. 为更好的实现对案例系统的自适应调整, 本文使用了本团队开发的自适应控制平台, 平台由开发层、运行层和基础设施层组成. 其中开发层提供了用于自适应演化逻辑设计阶段的系统设计开发工具集; 运行层提供了用于自适应演化机制运行阶段的基于 Agent 的演化运行支持平台, 由主控制平台和从平台两种类型组成; 基础设施层提供了支持软件正常运行所需的主流软硬件环境. 结构如图 5 所示.

首先, 在开发层平台会通过 Agent 包装工具将需要进行自适应演化的软件单元封装为 Agent 并提供对外的调用接口. 其次会通过集成规则设计工具生成初步的演化规则, 在本文的方法中对应的是自适应决策问题的可行解空间. 然后, 通过演化约束规则设计工具形成演化约束规则即本文中的约束函数, 并通过演化约束规则检查工具进行验证. 在运行层, 分为主控制平台与平台两种类型. 其中, 从平台负责实现对软件系统正常运行的支持, 主要负责 Agent 的维护以及通信等功能. 主控制平台除需要支持软件系统的正常运行外, 同时还需负责提供软件动态集成演化过程中所需的各项服务. 其核心组件为演化控制引擎, 演化控制引擎会解析并向各从平台分发集成演化规则, 实现对 Agent 自身结构或 Agent 间协作关系的调整, 从而实现软件的动态演化.

测试系统选择. 在指挥控制系统中对于战场环境的信息收集能力和处理能力有着较高的需求, 同时在具有较大的动态性和不确定性的战场环境下, 数据信息也呈现出复杂多变的特征. 因此, 本文以信息搜集入手设计了一个搜索子系统, 通过模拟复杂动态环境变化以验证本方法在指控领域的有效性. 考虑到动态的战场环境, 本文设计的

搜索子系统将 7×24 不断运行, 并通过关闭运行节点、负载调整等方式以模拟动态场景. 在这种环境下该系统为保证稳定运行将面临 3 方面的特殊需求. 具体而言, 首先其面临着节点毁伤、任务调整等多种突发性情况, 对于这些情况系统需要保证其稳定可靠性, 需要进行在线进行演化调整. 其次, 在调整过程中, 由于每个节点的物理信息是不同的 (CPU、内存等), 且服务实例需求不同, 形成的调整方案空间是庞大的. 例如, 现有节点上的 8 个相关服务实例需要进行调整, 此时, 有 10 个节点可供调整, 产生的方案空间将达到一亿的规模. 并且, 由于运行环境的复杂性, 是无法准确预测到系统的变化情况, 仅依靠人力进行调整是不可行的, 因此需要自主策略生成. 最后, 是关于策略权衡问题, 当系统发生多种变化时, 可能会产生多个对同一资源进行调度的策略进而产生新的问题出现, 因此需要针对这种情况对产生的策略进行权衡选择出合理的调整策略. 本文提出的方法能够解决在此案例系统下由于动态环境引起的突发性问题, 对其进行在线调整, 产生合理的调整策略.

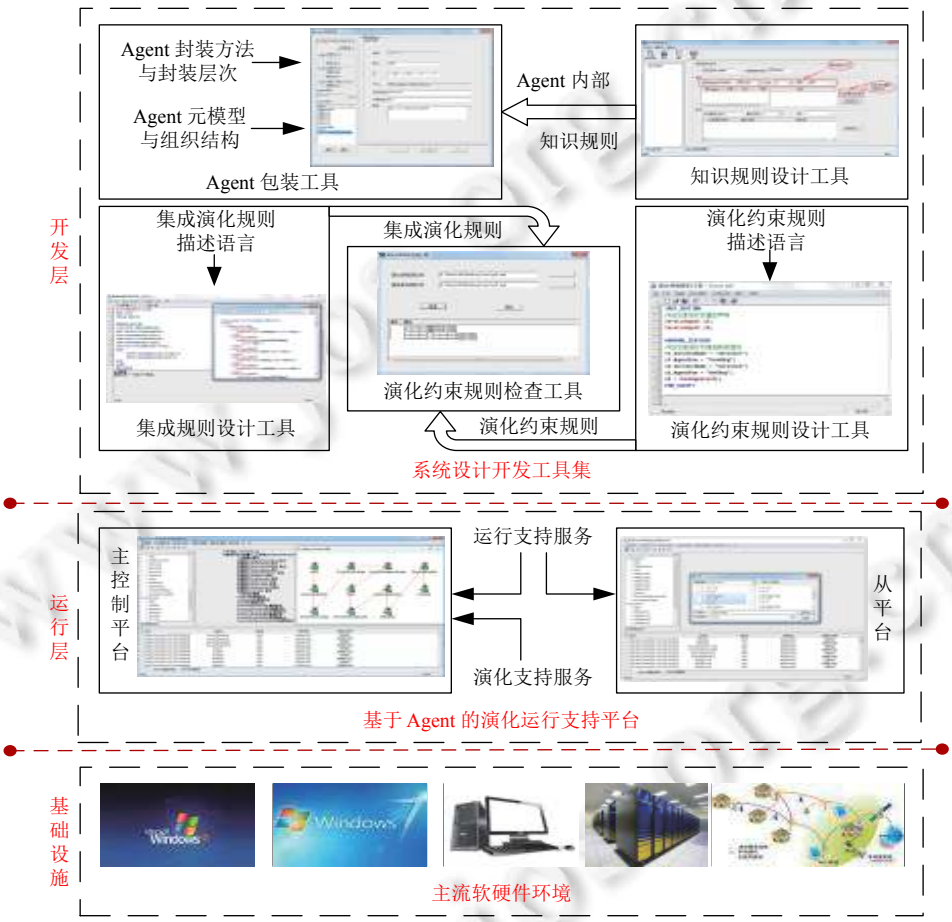


图 5 基于 Agent 的自适应软件控制框架

测试系统概述. 本文设计的搜索子系统包含了一个区域的组成信息、设备信息以及环境信息, 结构如图 6 所示. 系统包含对组成信息主题服务、设备信息主题服务以及环境主题服务. 每种类型的服务都提供视频搜索、图片搜索以及基本信息搜索 3 种功能. 在获取相关信息后, 会经过分析排序展示给指挥人员. 在系统的部署方面, 该系统采用了多节点分布式部署, 每个节点上有若干个服务, 根据需求每个服务均有若干个实例.

本文实验所设计的搜索子系统包含 13 个节点, 每个节点中包含若干数据处理实例和感知监控实例, 13 个节点的配置分为 3 类, 主要负责信息维护的节点, 主要负责信息通信节点服务以及主要负责信息分析的节点. 其中信息维护节点共有 7 个, 编号为 node1–node7. 信息通信节点共有 3 个, 编号为 node8 至 node10. 信息分析节点共有

3 个, 编号为 node11–node13, 具体节点配置信息如表 2 所示. 每个节点上含有基本的 8 个服务实例, 分别为负责视频搜索、图片搜索以及基本信息搜索 3 种类别的数据检索服务实例, 负责对组成信息、设备信息以及环境信息的数据处理服务实例, 以及两个负责感知信息变化调整的感知监控实例.

测试场景概述. 本实验选择了在军事环境中典型的毁伤替换场景作为测试场景. 毁伤替换场景是指在搜索子系统在运行的过程中, 某一个或者某些服务的实例可能由于某种原因而异常终止运行, 需要重新部署毁伤节点上运行的服务实例并通过寻找系统中可用的其他节点来替代已毁伤的节点, 继续执行该节点的任务, 以保证系统的正常运行. 本文在该场景下进行实验以验证方法, 具体流程如下, 本文在该过程采用了基于并行搜索的自适应决策方法将自适应决策过程映射为多目标优化的问题, 进行决策处理. 针对毁伤替换场景, 主要包含 4 个步骤.

(1) 通过分析节点特点以及领域特征建立目标函数, 以选择出重新部署的节点;

(2) 为毁伤节点寻找替换节点进行接替;

(3) 由于选择进行替换的节点在重新部署后可能会产生诸多的问题, 也就是上文提到的策略可能会发生冲突, 为此还需对系统负载进行监控调整, 以实现权衡决策. 具体通过对毁伤服务重新部署并密切监控搜索子系统整体状态, 若负载较重, 实施负载均衡措施;

(4) 重新计算负载, 调整部署.

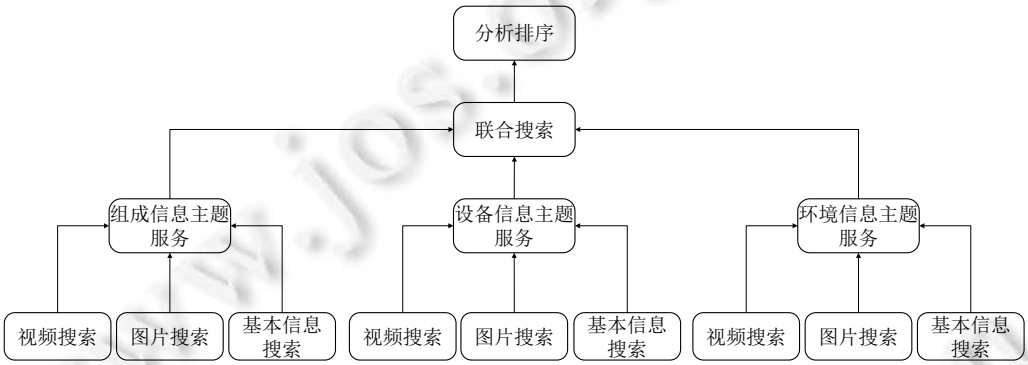


图 6 搜索子系统体系结构图

表 2 节点配置信息表

节点类别	CPU	内存	硬盘	网络
信息维护节点 (共7个)	型号:Intel(R)Core(TM) i3-2120 频率: 2.1 GHz 利用率: 0.85	类型: DDR2 速率: 19 Gb/s 利用率: 0.89	类型: RAID0 速率: 0.12 GB/s 利用率: 0.8 容量: 0.9 TB	丢包率: 0.42 利用率: 0.62 流入流量: 99321024 B 流出流量: 14423570 B
信息通信节点 (共3个)	型号: Intel(R)Core(TM) i3-2120 频率: 3.3 GHz 利用率: 0.75	类型: DDR3 速率: 20 Gb/s 利用率: 0.69	类型: RAID0 速率: 0.2 GB/s 利用率: 0.4 容量: 1 TB	丢包率: 0.12 利用率: 0.12 流入流量: 99348024 B 流出流量: 14183570 B
信息分析节点 (共3个)	型号: Intel(R)Core(TM) i4-2120 频率: 3.2 GHz 利用率: 0.75	类型: DDR3 速率: 22 Gb/s 利用率: 0.69	类型: RAID0 速率: 0.22 GB/s 利用率: 0.4 容量: 1 TB	丢包率: 0.12 利用率: 0.12 流入流量: 99358024 B 流出流量: 15153570 B

测试指标概述. 为了更好的衡量本方法在指控领域场景下效果, 本实验建立了两个度量指标具体如下.

• 指标 1: 每秒响应请求数. 系统在单位时间内的响应请求数量, 通过监听器可以获得秒钟内成果响应请求的数量. 每秒响应请求数通常是用作系统性能测试的度量标准之一, 在本实验中将其作为系统是否正常运行, 以及策略质量的标准. 如果系统的每秒响应请求数在正常范围内证明系统在正常运行, 并且通过可以通过每秒响应请求

数量的变化观察策略质量的情况.

• 指标 2: 节点负载量. 在本系统内节点的负载主要表现为 CPU 负载率、内存占用率、磁盘占用率, 为对本文方法产生的策略质量进行评估, 本文设计了节点负载量进行量化, 为上述 3 种指标设置权重以综合判断节点负载情况, 具体为 CPU 负载率的比重为 0.4, 内存占用率比重为 0.3, 磁盘占用率为 0.3.

4.1.2 RQ2. 本文方法的性能和健壮性如何? 能否适应指控领域的极端场景?

本文通过模拟大规模场景以及极端环境进行了方法的性能及健壮性测试, 通过设计大规模计算节点以模拟未来向大规模方向发展的战场趋势, 通过测试大规模环境下的算法响应时间以验证算法的性能. 其次为验证本方法在极端环境是否依然有效, 本实验通过改变在大规模计算节点的毁伤比例观察策略效果以验证本方法的健壮性.

测试环境设计. 本文设计模拟了一个大规模场景进行毁伤测试, 模拟了 4 700 个虚拟节点, 每个节点中包含一个感知监控服务以及一个数据处理服务, 实验环境如表 3 所示, 共有 15 台该类型计算机, 其中每台计算机均建立 2 台虚拟机节点, 在微服务平台下, 构成以 30 台计算节点的集群, 并通过 kubernet 管理工具创建了 4 700 个虚拟节点.

表 3 实验计算机信息

环境	参数	值
物理机环境	计算机型号	OptiPlex 7050
	处理器	Intel 酷睿i5 7500 3.4 GHz×4
	内存	8 GB
	硬盘	1 TB
虚拟机环境	虚拟机软件	VMware® Workstation 15 Pro
	内存	3 GB
	处理器核数	2
	硬盘	50 GB

测试指标概述. 为更好的对本方法在毁伤接替场景下进行评价, 本文建立两个度量指标具体如下.

• 指标 1: 算法响应时间. 自适应决策算法响应时间的长短可以在一定程度上反应算法的优劣. 在实际测试中, 记录自适应决策算法开始决策的时间为 $t1$; 自适应决策算法决策完成, 找到合适的替换节点的时间为 $t2$. $t2$ 与 $t1$ 之差即为自适应决策算法响应时间.

• 指标 2: 策略调整时间. 策略调整时间是指自适应决策方法产生的策略从策略下发开始到系统调整完成的时间, 例如有一调整策略发出, 此时记录的时间为 $t1$; 每秒遍历受影响的服务, 并记录对应服务下的实例数目. 当实例数目发生改变时, 即代表新的实例部署成功开始正常对外提供服务, 记录当前时间为 $t2$. $t2$ 与 $t1$ 之差即为节点毁伤的调整时间.

4.1.3 RQ3. 本方法与其他基于搜索的方法和其他军事指控的自适应决策方法有何区别?

本文选择将基于并行搜索的自适应决策方法与目前自适应领域中常用的自适应决策方法进行比较, 从基于搜索的自适应决策方法以及其他军事指控的自适应决策方法层面进行类别划分, 选择了包括 Coker 团队提出的基于决策树形式的复合自适应决策方法[4]、中国电科 28 研究所金欣等人提出的基于规则的指控功能模块自适应加载方法[15]、空军工程大学焦志强等人提出的基于效用的指控策略优势评估决策方法[16]. 具体实验设计如下.

对比方法选择. 选取上述方法的主要原因总结如下.

• 在基于搜索的方法中选取了 Coker 团队提出的基于决策树形式的复合自适应决策方法, 该方法是目前唯一相关的基于搜索的软件工程方法与自适应决策相结合的方法, 主要考虑软件变化之间的关系且存在动态优先级变化的特点, 采用静态离线的方式实现基础的策略生成. 因此, 本文选择该复合自适应决策方法与本文的并行搜索方法进行对比.

• 在指控领域下对比方法选择了中国电科 28 所金欣团队提出的基于规则的方法以及来自空军工程大学焦志

强团队提出的基于效用的方法. 其原因是在指控领域种应用最广泛的自适应方法即为基于规则、基于效用等方法, 这些方法经过多年研究证实了其在指控领域的适用性. 其次, 这两种方法的研究工作来自于中国电科 28 研究所以及空军工程大学, 其研究背景与指控领域有着很高的相关性, 能与本文提出的方法形成更可靠、更有效的对比. 具体来说, 基于规则的自适应决策方法是依据统外部环境发生变化, 从已有的专家知识库和规则库选择合适策略做出决策, 因其简单高效的运行效率、较强的可解释性等优势成目前在指控系统应用最广泛的方法, 因此本文选取基于规则的自适应决策方法进行对比以测试本文方法在执行效率上是否能优于该经典方法. 基于规则的方法的静态性使得策略最优化问题上效果不佳, 而基于效用的自适应决策方法, 根据对一定客观现实的主观判断结构把专家意见和分析者的客观判断结果直接而有效地结合从而定义效用, 依据效用值的大小选择最优决策以实现系统自适应. 因此, 本文通过对比基于效用的方法以测试本文方法在策略寻优上的效果.

测试环境设计. 为了确保本文方法在军事指控领域存在实际的应用价值, 本文选择将对比实验的运行环境依照上述问题一中所采用的搜索子系统的测试环境进行配置, 有效验证了本文方法在军事指控领域的可用性. 同时, 利用同样的测试环境进行实验, 可以解决由于不同测试环境导致的对比实验结果存在误差的问题, 有利于确保实验结果的可靠性.

测试指标概述. 在此基础上, 本文准备了策略效率和策略效果两类对比测试, 模拟上述方法的在具体场景下的应用情况, 旨在测试评估各类方法的有效性和优劣性. 同时, 为了更好的量化实验效果, 本文选取了决策过程耗时和策略质量值两种评估指标, 分别对方法的效率和效果进行测试.

• 指标 1: 决策过程耗时. 从自适应决策方法从开始执行到完成的完整自适应决策过程的耗时.

• 指标 2: 策略质量值. 调整方案在可行解空间中所存在的位置坐标距离理想点的距离.

其中, 为了对比决策方法的效率, 本文拟计算自适应决策过程的耗时. 在实际测试中, 在决策方法执行前后进行时间日志输出处理, 记录自适应决策算法开始决策的时间为 $t1$; 自适应决策算法决策完成, 找到合适的调整方案的时间为 $t2$.$t2$ 与 $t1$ 之差即为自适应决策过程的耗时.

为对比决策方法的效果, 本文拟分析决策方法产生的策略的优劣性. 当前多数策略评估方法都采用由 Hwang 和 Yoon 提出的 TOPSIS[17], 因不同策略最终都是对可变点的调整, 根据本文之前的设计, 不同的策略都可以映射到可行解空间中. 因此, 本文拟将不同调整方案映射到可行解空间中, 并分析在可行解空间中距离理想最优点以及理想最劣点的欧式距离, 并作为策略质量值分析策略的优劣性, 进而对比决策方法的能力. 该方法认为距离理想最优点最近而距离理想最劣点最远, 即策略质量值最大的坐标位置所对应的调整方案为最优.

4.2 方法测试

4.2.1 面向指控领域的自适应决策方法测试 (RQ1)

在毁伤替换场景中, 首先需要根据节点信息以及环境特点等因素将服务实例重新部署过程映射为多目标优化过程, 根据结果将毁伤的实例重新部署到选择的节点上, 由于各个节点的性能不同, 重新部署的服务可能会造成节点负载过重的问题, 因此需要对部署后的实例进行负载进行调整以使系统达到更佳的状态. 具体实验过程及结果分析如下.

(1) 将自适应决策过程映射为多目标优化问题

在毁伤替换场景中, 影响系统选择替换节点的因素主要包括 3 方面, 节点本身的底层环境信息、节点之间的交互性以及军事系统的领域特点等, 需要综合考虑上述因素选择替换节点, 建立目标函数来搜索最优的可替换节点. 具体考虑的指标如下:

• 节点的底层环境信息, 即节点的 CPU、内存、磁盘的使用情况和可用情况以及总的能力. 这些信息是保证节点可以正常执行任务的必要条件.

• 节点之间的通讯能力. 对于军事信息系统来说, 其运行环境的复杂性和高实时性的要求使得系统内部的节点之间需要经常进行交互来保证系统的正常运行, 因此, 节点之间的通讯能力对于节点正常的执行任务也显得尤为重要.

• 节点的可靠性和安全性. 由于军事领域中的军事信息系统的特殊性, 其数据和节点需要高度的保密机制和安全机制也是不可缺少的考虑因素.

针对上述 3 种因素, 本文在毁伤替换的搜索过程中建立了 CPU、内存、磁盘、网络及安全 5 种目标函数, 如下.

$$x_i.CpuAbility = x_i.CPU_{\text{type}} \times \left(\frac{x_i.CPU_{\text{frequency}} - \min\left(CPU_{\text{frequency}}\right)}{\max\left(CPU_{\text{frequency}}\right) - \min\left(CPU_{\text{frequency}}\right)} + (1 - x_i.CPU_{\text{used}}) \right) \tag{4}$$

如公式 (4) 所示 $x_i.CpuAbility$ 表示节点 x_i 的 CPU 能力. 其中, $x_i.CPU_{\text{type}}$ 表示节点 x_i 的 CPU 类型, $x_i.CPU_{\text{frequency}}$ 表示节点 x_i 的 CPU 处理频率, $x_i.CPU_{\text{used}}$ 表示节点 x_i 的 CPU 利用率.

$$x_i.MemoryAbility = x_i.Weight_{\text{Type}} \times \left(\frac{x_i.Memory_{\text{speed}} - \min\left(Memory_{\text{speed}}\right)}{\max\left(Memory_{\text{speed}}\right) - \min\left(Memory_{\text{speed}}\right)} + \frac{1 - x_i.Memory_{\text{used}}}{\max(1 - Memory_{\text{used}}) - \min(Memory_{\text{used}})} \right) \tag{5}$$

如公式 (5) 所示 $x_i.MemoryAbility$ 表示节点 x_i 的内存能力. 其中, $x_i.Weight_{\text{Type}}$ 表示节点 x_i 的内存类型权重, $x_i.Memory_{\text{speed}}$ 表示节点 x_i 的内存读写速度, $x_i.Memory_{\text{used}}$ 表示节点 x_i 的内存利用率.

$$x_i.DiskAbility = x_i.Weight_{\text{Type}} \times \left(\left(x_i.Disk_{\text{speed}}\right) / \left(\begin{array}{l} \max\left(Disk_{\text{speed}}\right) - \left(\min\left(Disk_{\text{speed}}\right)\right) + \\ \dfrac{(1 - x_i.Disk_{\text{used}}) - \min(1 - Disk_{\text{used}})}{\max(1 - Disk_{\text{used}}) - \min(1 - Disk_{\text{used}})} + \\ \dfrac{x_i.Disk_{\text{size}} - \min(Disk_{\text{size}})}{\max(Disk_{\text{size}}) - \min(Disk_{\text{size}})} \times \varepsilon \end{array} \right) \right) \tag{6}$$

如公式 (6) 所示 $x_i.DiskAbility$ 表示节点 x_i 的磁盘能力. 其中, $x_i.Weight_{\text{Type}}$ 表示节点 x_i 的磁盘类型权重, $x_i.Disk_{\text{speed}}$ 表示节点 x_i 的磁盘平均读写速度, $x_i.Disk_{\text{used}}$ 表示节点 x_i 的磁盘利用率, $x_i.Disk_{\text{size}}$ 表示节点 x_i 的磁盘大小.

$$x_i.NetAbility = (1 - x_i.packageFailure) \times (1 - x_i.Rn_{\text{used}}) \times x_i.Through \tag{7}$$

如公式 (7) $x_i.NetAbility$ 表示节点 x_i 的网络通讯能力. 其中, $x_i.packageFailure$ 表示节点 x_i 的丢包率, $x_i.Rn_{\text{used}}$ 表示节点 x_i 的网络利用率, $x_i.Through$ 表示节点 x_i 的输入输出总流量, 即吞吐率.

$$x_i.Safe = x_i.SA \times (x_i.IMTTF) \tag{8}$$

如公式 (8) $x_i.Safe$ 表示节点 x_i 的安全能力. 其中, $x_i.Safe$ 表示节点 x_i 的节点区域安全评估等级, $x_i.IMTTF$ 表示节点 x_i 的失效前正常工作的平均时间.

(2) 毁伤节点实例重新部署决策

为模拟毁伤接替场景, 本文关闭了搜索子系统中的节点 node7 以模拟节点毁伤, 该节点共部署实例 8 个, 包含数据处理服务实例 3 个, 实例编号为 I0、I1、I2, 数据检索实例 3 个, 实例编号为 I3、I4、I5, 以及感知监控服务实例 2 个, 实例编号为 I6、I7.

该节点毁伤之后, 感知机制无法获取该节点信息, 判断该节点毁伤并发布“毁伤替换”事件. 系统收到“毁伤替换”事件及相关信息, 触发决策行为. 此时, 为了保障搜索子系统的搜索结果的实时性, 自适应决策需要在其他节点上重新部署毁伤节点上的服务实例. 自适应决策根据其他节点状态, 考虑节点整体的资源利用率, 依次为毁伤节点上的服务实例寻找部署节点, 根据目标函数计算得到各节点在 CPU、内存、磁盘、网络及安全 5 个维度上的得分, 计算得分片段如表 4 所示.

通过计算得出了实例重新部署在各个节点中的得分, 由于 I0、I1、I2 为数据处理服务实例对 CPU 及内存的需求更为迫切, 因此决定将其部署在 node12 节点上. I3、I4、I5 为数据检索服务对内存以及磁盘需求较高, 因此决定将其部署在 node13 节点上, I6、I7 为感知监控服务实例, 对网络和安全需求较高, 因此决定将其部署在 node9 节点上.

(3) 部署节点负载均衡调整决策

由于计算节点性能各不相同, 可部署的实例种类及数量不尽相同. 考虑到节点负载能力有限, 若感知机制监控

到当前系统有负载较重的趋势, 自适应决策将触发负载均衡重新规划部署方案.

在进行负载均衡调整之前, 需要考虑节点利用率以及服务运行效率这些与负载均衡相关的数据, 建立目标函数来搜索最优的负载调整方案. 具体目标函数建立如下.

节点利用率得分S_{use}综合考虑各个节点的各项资源的利用率得分如公式 (9) 所示.

$$S_{\text{use}}=\sum_{i=1}^{n}\left(\alpha S_{cpu_i}+\beta S_{mem_i}+\gamma S_{disk_i}\right) \tag{9}$$

其中, n 为节点个数. 单个节点的利用率得分需要考量的资源有 CPU 利用率得分S_{cpu_i}、内存利用率得分S_{mem_i}和磁盘利用率得分S_{disk_i},α, β, γ 分别为三者利用率权重. 其中各资源利用率的计算方法为已分配给容器的资源与总资源之比.

服务运行效率S_{cap}分为两个方面: 其一为单个服务实例的运行效率得分S_{cap_i}, 其二为服务实例之间通信的效率得分$S_{b_cap_i}$, 如公式 (10) 所示.

$$S_{cap}=\sum_{i=1}^{n}S_{cap_i}+\sum_{i=1}^{m}\omega_i S_{b\_cap_i} \tag{10}$$

其中, n 表示服务实例的个数, m 则表示服务实例间通信关系总数. 单个服务实例的运行效率得分S_{cap_i}总共考虑 6 个方面, 分别考虑 CPU 配额、内存类型、磁盘类型、网络失败率、安全等级以及仪器安全性 (IMTTF) 指标, 各方面得分的计算方式为节点实际信息与实例预期值 (节点预期配置信息表) 的比. 服务实例之间通信的效率得分$S_{b_cap_i}$的计算方法为遍历已知服务实例间的通信关系, 若有通信关系的双方部署于同一个节点, 则其得分为 1, 否则得分为 0, 其权重ω_i则是此通信关系的强度.

表 4 服务在各个节点得分片段示例

实例名称	部署节点	CPU得分	内存得分	磁盘得分	网络得分	安全得分
I0	node1	0.61	6.21	5.10	3.21	16400
	node2	0.71	6.30	4.97	4.32	24100
	node3	0.63	5.94	4.13	5.41	18400
	…	…	…	…	…	…
	node12	0.74	6.40	4.06	8.87	16500
	node13	0.65	6.31	5.76	8.39	36000
I1	node1	0.65	6.34	5.12	4.21	16400
	node2	0.73	6.36	5.06	4.66	24100
	node3	0.65	6.13	4.53	5.55	18400
	…	…	…	…	…	…
	node12	0.79	6.64	4.12	8.35	16500
	node13	0.73	6.40	5.15	7.87	36000
…	…	…	…	…	…	…
I7	node1	0.54	6.24	5.54	4.65	16400
	node2	0.65	6.35	5.13	5.14	24100
	node3	0.61	5.92	4.67	6.21	18400
	…	…	…	…	…	…
	node12	0.77	6.41	4.27	8.12	16500
	node13	0.74	6.32	6.74	7.62	36000

随后, 自适应决策方法根据目标函数, 计算搜索子系统在剩余 12 个节点上的最佳部署策略. 若 12 个节点暂时无法满足搜索子系统的资源需求, 系统将根据选择减少一些次要服务的实例数量甚至关闭一些次要服务, 以保证主要搜索任务的进行; 最后, 产生适合调整部署的策略集合. 经过自适应决策方法得出了在 6 个节点内调整的较优策略, 如表 5 所示.

(4) 基于后优化理论的负载均衡策略选择

自适应机制在执行调整策略时, 必须依据唯一的策略方案对系统结构和行为作出调整. 因此, 完整的自适应决

策方法还需要在上述方法产生的前沿策略集合的基础上, 根据不同决策环境下的决策需求与选择指标, 结合作战偏好、资源态势等指标进行评估排序, 产生最优的唯一调整策略. 具体评估计算方法如下.

表 5 较优实例调整策略

部署策略	Node2	Node4	Node8	Node9	Node12	Node13	节点利用率得分	服务运行效率得分
策略1	I2	I1	I3、I5	I6	I0、I7	I4	1.3334	0.8857
策略2	I3	I1	I2、I5	I6	I0、I7	I4	1.3283	0.9705

用户偏好得分 $Score\_Preference_i$ 主要是通过指挥者对不同目标函数设置的权重, 加权计算方案 i 在所有目标函数的综合得分, 即用户偏好得分, 如公式 (11) 所示. 其中, $Preference_i$ 为用户偏好权重, $Score_i$ 为不同目标函数得分.

$$Score\_Preference_i = \sum_i^n (Preference_i \times Score_i) \tag{11}$$

调整开销得分 $Score\_Cost_i$ 综合考虑策略 i 的调整开销占所有策略的调整时间开销的比值进行计算, 如公式 (11) 所示. 其中, $Cost_i$ 是特定策略调整时间开销, $\sum_{i=1}^n Cost_i$ 是所有策略的调整时间开销之和.

$$Score\_Cost_i = \frac{1}{2}\left(1 - \frac{Cost_i}{\sum_{i=1}^n Cost_i}\right) \tag{12}$$

部署策略总得分 $Score_i$ 综合考虑时间调整开销和用户偏好得分, 具体计算如公式 (12) 所示.

$$Score_i = 0.7 \times Score\_Preference_i + 0.3 \times Score\_Cost_i \tag{13}$$

以本次场景验证实验为例, 在上述策略集合中, 假设节点资源有限, 应该较多地考虑节点资源利用率, 策略 1 的得分情况略微高于策略 2, 但若想更好地提供服务, 策略 2 显著优于策略 1. 因此, 考虑到上述两种策略在资源利用率指标上差距不大, 本文将服务运行效率的用户偏好权重设置为 0.7, 将节点利用率的用户偏好权重设置为 0.3, 计算不同策略的用户偏好得分. 然后, 将每个服务实例的调整行为时间开销设置为 1, 策略 1 调整了 5 个服务实例, $Cost_1$ 为 5; 策略 2 调整了 5 个服务实例, $Cost_2$ 为 5. 根据式 (12) 可得策略 1 的开销得分为 $Score\_Cost_1 = 0.3$, 策略 2 的开销得分为 $Score\_Cost_2 = 0.2$. 最后, 再根据式 (13) 计算得出最后策略总得分如表 6 所示.

表 6 策略评估计算表

部署策略	节点利用率得分	服务运行效率得分	用户偏好得分	调整开销	开销得分	总得分
策略1	1.3334	0.8857	1.02001	5	0.25	0.789007
策略2	1.3283	0.9705	1.07784	5	0.25	0.829488

因此, 根据总得分, 本文选取了服务质量整体表现更为优秀的策略 2 作为系统部署调整的策略.

为更好的说明本方法能够保证系统正常平稳运行, 本实验进行每秒请求响应数量的测试, 通过向毁伤节点中的 3 种类型的服务发送大量请求观察每秒请求响应数量, 如果每秒请求数量保持在正常范围内则证明系统处于平稳运行状态. 实验结果如图 7 所示. 其中横轴表示进行自适应决策过程的时间, 单位为 s, 纵轴表示在该时间点请求响应的个数.

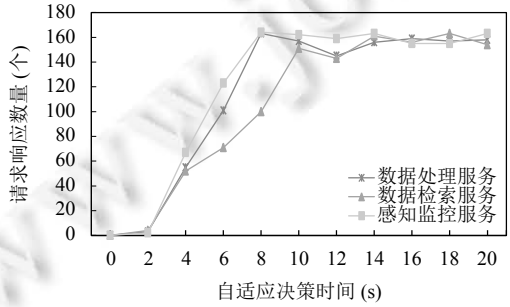


图 7 每秒请求响应数量变化图

从测试结果可以看出, 在自适应决策过程开始的一段时间内请求响应数量仅保持在个位数, 这是由于算法需要一定的执行时间, 而且将毁伤节点上的实例重新部署到其他节点上并进行调整需要一定的时间. 这个时间从图中可以看出在 7–8 s 间, 请求响应数量恢复正常范围内并趋于稳定, 这是有由于毁伤节点上的实例调整部署完毕, 系统恢复平稳运行状态, 这个时间间隔维持在了 10 s 以内, 能够满足指控领域的需求, 因此方法能够有效进行自适应调整.

同时为验证本方法在进行策略调整时最终产生的策略是有效的, 本文对进行调整的节点进行了节点负载量监控, 主要监控了参与调整的 node2、node4、node8、node9、node12 以及 node13 这 6 个节点的节点负载量, 实验结果如图 8 所示. 其中横坐标表示进行策略调整的时间, 单位为秒, 纵轴表示节点的负载量.

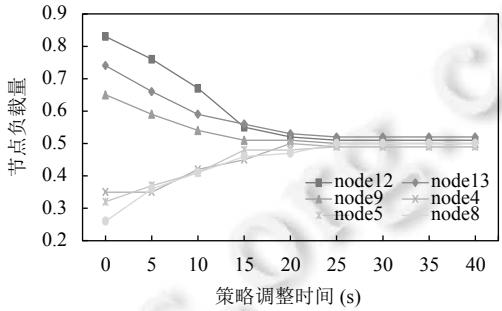


图 8 节点负载量变化图

从测试结果可以看出, 策略调整之初由于将毁伤节点集中部署在了 node9、node12 以及 node13 节点上导致了这 3 个节点的负载过高, 偏离了正常负载范围, 而 node4、node5 以及 node8 节点负载过低没有得到充分利用, 从图中我们能够看到偏离正常负载范围的节点在经过执行调整策略后负载逐渐合理并趋于平稳. 这证明了本方法产生的策略能够有效调整系统, 使得系统平稳运行.

从上述实验可以看出, 本文提出的自适应决策方法能够在指控领域毁伤接替场景下实现自适应决策问题到最优化问题的转化, 同时可以针对多种因素实现策略的权衡决策, 最后通过后优化理论产生唯一策略.

4.2.2 性能与健壮性测试 (RQ2)

为验证本方法的性能及健壮性, 本文设计了在大规模环境下算法响应时间的测试实验以及不同毁伤程度下的产生策略的效果, 具体实验如下.

(1) 大规模环境下算法性能测试

为了验证方法在大规模场景下依然具有实时性, 本文进行了自适应决策算法响应时间测试, 通过调整节点数量观察算法的响应时间. 本实验中从 200 个节点开始进行测试, 并依次增加 100、200、300、400、500、600、700、800、900 个节点直到 4 700 个节点, 观察算法响应时间的变化情况, 针对每个规模进行 10 次实验, 并取平均值作为实验结果. 节点数与自适应决策响应时间如图 9 所示, 其中横轴表示节点的总数, 纵轴表示响应的时间, 时间单位为 ms.

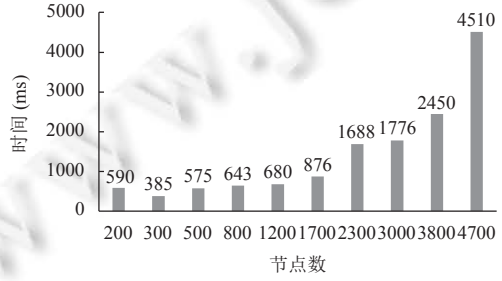


图 9 节点数与自适应决策响应时间

从测试结果可以看出, 本文提出的自适应决策方法, 随着计算节点的快速增加, 算法的响应时间始终保持在一个可接受的范围内. 其中, 算法响应时间在 3 800 个可选择节点下超过了 2 s, 在 1 700 个模拟节点以下算法响应时间均不超过 1 s, 这说明尽管随着指控领域计算规模的变大, 本方法依然能够满足军事系统的高实时的需求 .

(2) 极端环境下方法健壮性测试

考虑到在极端环境下, 节点毁伤比例将会到达很高的比例, 为验证在此种极端环境下的健壮性, 本文通过调整节点毁伤比例观察系统节点毁伤调整时间变化情况. 本实验中将测试节点毁伤比例测试区间确定为 10%–60%, 每次节点毁伤比例增加 10%. 针对不同节点毁伤比例进行 10 次实验, 并取每次时间的平均值为节点毁伤调整时间. 节点毁伤比例与调整时间如图 10 示, 其中横轴表示节点毁伤比例, 纵轴表示节点毁伤调整时间, 时间单位为 s.

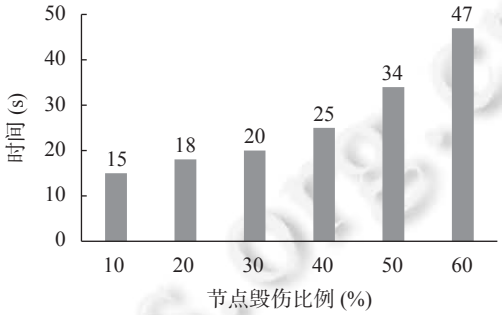


图 10 调整时间随节点毁伤比例变化图

从测试结果可以看出, 在节点毁伤比例处于 10%–60% 时, 系统运行正常, 节点毁伤调整速度较快, 调整时间不大于 1 min, 能够保证军事系统能够持续可靠运行. 因此, 本文所提出的方法能够产生有效的调整策略, 保证系统在极端环境下平稳运行.

4.2.3 对比实验测试 (RQ3)

本文准备了策略效率和策略效果两类对比测试, 模拟了本文方法与其他相关方法的在实际场景下的应用情况, 旨在测试评估各类方法的有效性和优劣性. 具体实验如下.

(1) 决策效率测试

在决策方法的效率测试中, 本文将决策过程耗时 (指标 1) 作为衡量标准, 测试评估本文串行版本的自适应决策方法以及并行搜索的自适应决策方法、Coker 提出的复合自适应决策方法[4]、基于规则的自适应决策方法[15]以及基于效用的自适应决策方法[16]在毁伤替换场景下的效率. 其中, 本文串行版本的自适应决策方法是指不使用并行优化机制在单线程内串行执行本方法. 并行版本是指使用并行优化机制通过划分种群以多线程的方式执行本方法, 使用两种版本的方法是为了验证并行有效机制的有效性. 首先, 参照上述问题一的配置环境进行对比测试, 然后, 针对不同节点毁伤比例分别进行 20 次试验, 并取每次的平均值作为决策过程耗时. 节点毁伤比例与决策过程耗时如图 11 所示, 其中横坐标表示节点毁伤比例, 纵轴表示节点毁伤调整时间, 时间单位为 ms.

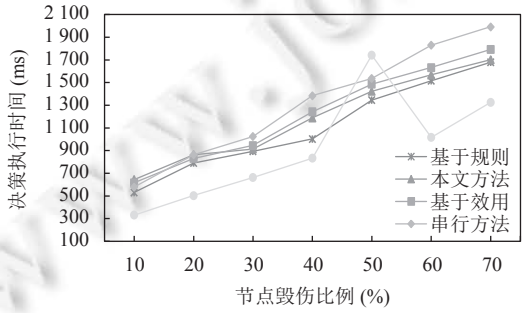


图 11 决策过程耗时随毁伤节点比例变化图

在基于搜索的方法对比测试结果中可以看出, 本文提出的自适应决策方法和 Coker 提出的复合决策方法, 随着毁伤比例的增长, 决策过程耗时也不断增长, 但都不超过 2 s. 可以明显看出, 并行方法的决策过程耗时普遍低于串行方法. 在大部分情况下, Coker's 决策方法耗时普遍低于串行和并行方法, 这是由于 Coker's 决策方法作为一种静态决策方式本身就比动态决策方法耗时短所决定的. 而在毁伤节点为 50% 时, 静态决策方法的决策过程耗时大幅度增长, 高于串行和并行方法, 这是由于静态离线方法的规则库具有一定的局限性, 所有的策略与事件映射都需要预先设置, 当不存在 50% 毁伤情况下的调整策略时, 需要搜索遍历完整的可行解空间, 在搜索失败后选取与其情况类似的毁伤 40% 时的调整策略.

在与指控领域的其他方法对比测试结果中可以看出, 随着节点毁伤比例的增长, 本文提出的基于并行搜索的自适应决策方法的决策执行时间会增大, 且其在整个过程中的决策执行时间基本高于基于规则的自适应决策方法并与基于效用的自适应决策方法的决策执行时间相差不大. 本文提出的方法的决策执行时间在节点毁伤比例为 10%–30% 基本高于基于规则的自适应决策方法, 这是由于基于规则的自适应决策方法判断的规则往往通过静态的方式进行, 决策时间偏少. 而当节点毁伤比例高于 40% 时, 由于整个系统毁伤比例过高, 节点数目减少, 为了完成系统需求, 每个节点必须增大其自身线程的数量, 从而导致其决策执行时间折线图突然猛增, 但是仍然低于基于效用的自适应决策方法.

(2) 策略效果测试

在决策方法的效果测试中, 本文将策略质量值 (指标 2) 作为衡量标准, 测量评估上述除串行方法以外的 4 种决策方法生成策略的优劣. 首先, 本文在上述决策过程耗时测试环境的基础上, 针对不同节点毁伤比例分别进行 20 次试验, 分别计算不同策略每次的策略质量值并取平均值. 节点毁伤比例与策略质量数耗时如图 12 所示, 其中横坐标表示节点毁伤比例, 纵轴表示策略质量数.

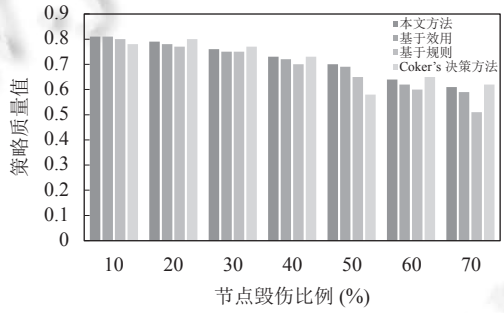


图 12 策略质量值随毁伤节点比例变化图

在基于搜索的方法对比测试结果中可以看出, 本文提出的自适应决策方法和静态方法的质量值在毁伤替换场景下不同毁伤比例下的策略效果相似, 与理想点质量值存在一定差距, 且随毁伤比例的增加而不断降低. 在毁伤 50% 情况下, 静态方法的质量值大幅度降低, 且低于毁伤 40% 和 60% 时的质量值. 是由于在毁伤 50% 情况下, 静态方法搜索可行解空间失败, 然后选取毁伤 40% 的调整方法, 然而由于毁伤比例的增加, 同样的调整方案并不适用于毁伤 50% 的情况, 策略效果低于毁伤 40% 的策略效果. 因此, 在实际应用时, 采用基于并行搜索优化的自适应决策方法可以在牺牲一定的决策过程耗时的条件下生成更稳定、可靠的策略, 而采用静态方法实现自适应决策的过程中可能会获得更快速的效果, 然而也有一定概率规则匹配失败, 严重影响软件运行, 不满足实际军事应用场景的需要.

在与指控领域的其他方法对比测试结果中可以看出, 3 类方法的策略质量值均随着节点毁伤比例的增大而减少. 这说明节点毁伤比例对策略质量存在着直接影响, 这是因为随着节点毁伤比例的增大, 策略寻优的过程变得愈发复杂. 本文提出的基于搜索的自适应决策方法在不同的毁伤比例下, 策略质量均优于基于规则和基于效用两种方法. 在毁伤比例小于等于 40% 的情况下, 3 类方法的策略质量值之间差异较小, 在毁伤大于等于 50% 情况下, 基

于规则的策略质量数则出现大幅降低, 显著低于其他两类的方法. 这是因为基于规则的方法是不能依据当前场景动态产生规则, 其策略质量也就有所下降. 在策略质量值接近的情况下, 本文方法在策略耗时上低于基于效用的方法, 这是因为基于效用的方法在执行过程中每次都需经过 5 次计算, 因此比较耗时. 依据上述对比, 可以得出, 本文方法产生的策略质量和策略执行耗时上明显优于其他两种方法, 证明了本文方法策略质量和时间开销上的优越性.

5 相关工作

本文主要分析了目前主流自适应决策方法的研究现状及其在指控系统中的应用情况, 并重点将本文工作与目前存在的基于搜索的自适应决策方法进行了对比分析.

5.1 自适应决策方法及其在指控领域的应用情况

Lalanda 在《Autonomic Computing》一书中, 按照自适应决策方法所采用的知识类型, 将目前面向通用领域的主流决策方法分为基于规则/策略、模型、效用函数、目标共 4 种[18].

基于规则/策略的自适应决策方法通过规则匹配或规则推理的方式, 获得恰当的自适应策略. 由于该方法的处理效率极高[19], 因此该方法已被广泛应用于指控系统中, 直到目前都在指控系统中扮演重要角色. 中电 28 所金欣等人提出了一种基于规则的功能模块动态加载方法, 实现了指控系统多样化需求的灵活重构[15]. 但是该类方法存在一个致命的缺陷问题, 即需要静态定义所有的候选策略, 并预先确定调整方案和事件间的映射关系. 针对该问题, 目前国内外也存在不少改进的思路, 主要是通过对策略本身实现在线修改与精化调整.

基于目标的自适应决策方法, 以系统目标作为决策目标, 将策略对目标的满足程度作为比较不同策略优劣性的衡量标准, 从而选择出最符合系统目标要求的自适应策略. 相关研究工作可分为两类. 一类是通过建立自适应需求模型, 并以此为依据建立自适应策略[20]. 而另一类则通过衡量已有自适应策略对目标的满足程度, 从而评判并选择出最优的自适应策略[21]. 在指控领域, 存在部分采用蚁群算法、与或树搜索方法开展的相关研究工作, 但大多数方法仍只考虑对单一目标问题或通过固定优先级的方式处理多目标问题.

效用函数指系统进行自适应调整后所能获得目标完成情况 (即收益) 与自适应调整导致的开销之间 (即成本) 的关系函数. 基于效用函数的自适应决策方法, 以系统效用作为决策目标, 采用效用函数衡量不同自适应策略调整系统后所能增加的系统效用, 从而选择出最优自适应策略[22,23]. 与基于目标的方法类似, 在指控领域多数研究采用主成分分析法、层次分析法或模糊综合评价法计算专家对效用的评分, 并据此选择最优策略. 该过程仍需依赖较多的专家经验, 无法有效在线应对未知变化.

基于模型的方法本身根据对模型的相关性, 又可以分为模型相关方法和模型无关方法 (model free method). 基于模型的方法, 通过建立模型并获取系统状态信息, 实现对系统状态趋势的预测与分析, 常用的模型包括深度神经网络[24]、强化学习[25]等. 近年来新兴的基于马尔科夫决策过程[26]、基于贝叶斯网络的自适应决策方法[27]本质上也属于基于模型的决策方法. 这类方法需要建立完整系统模型且不断试错, 开销较大, 目前并未被广泛应用于指控领域.

5.2 基于搜索的自适应决策方法

目前, 基于搜索的软件工程已成为软件工程领域的研究热点之一. 然而, 将该理论应用于自适应软件的研究仍处于探索阶段[28], 主要工作多关注于自适应软件系统的框架设计[29−31]、体系结构的优化[32−34]、软件测试[35−37]、开发过程和工作量估计[38]以及程序演化和修复[39,40]等方面. 例如, MOSES 框架是一个面向服务计算的自适应系统优化框架[31], Duse 提出的关于自适应软件体系结构和基于搜索软件工程结合的研究[6]等. 当前, 鲜有将自适应决策与基于搜索的软件工程结合的研究.

与本文最相关的研究工作是由美国卡内基梅隆大学 (CMU) 的 Coker 团队[4]提出的, 基于决策树形式的复合自适应决策方法. 该方法考虑到软件变化之间的存在关系且存在动态优先级变化的特点, 采用进化编程结合基础的自适应策略进行策略生成. 然而, 该方法整体过程采用静态离线的运行方式[8], 无法处理实时性较强的决策问题. 而本文提出的决策方法可以动态的形成方案空间, 依据实时的软件变化实现在线决策, 能够更好地适用于具备较大动态性和不确定性的自适应软件系统.

6 结 论

作为国防与军队信息化建设中的重要系统, 指控系统亟需建立应对动态运行环境和多变任务需求的自适应过程, 以确保系统可长期有效稳定运行. 自适应决策方法作为该过程中的重要环节, 应具备在线权衡决策能力. 因此, 本文提出了一种基于并行搜索优化的指控系统自适应决策方法. 该方法根据自适应决策问题特征, 将自适应决策问题建模为搜索优化问题, 并采用遗传算法与粒子群算法, 实现针对同时发生的多个变化进行在线权衡的目标. 并且, 本文采用并行遗传算法和后优化理论, 结合作战需求与指挥员偏好, 分别实现了并行化的自适应决策方法与策略多指标排序法, 以解决该方法在指控系统中实际应用时存在的搜索效率保障、策略择优选择问题. 实验结果表明, 本文提出的决策方法在模拟环境下能够在线产生应对毁伤接替典型指控场景的权衡策略, 达到有效指导系统开展调整行为, 实现对环境变化与任务变更的动态响应的目标.

当然, 本文所提出的决策方法依然存在很多值得探讨的工作: (1) 本文所提出的方法在自适应建模阶段对专家意见有一定依赖性, 需要专家预先给出建议的目标函数、约束函数以及可变点的取值范围. 本文后续考虑构建指控领域知识图谱以辅助自适应问题建模, 并通过实验方式获取可变点的取值与效果的关系, 通过算法计算阈值. (2) 随着系统规模和不确定的上升, 导致方案空间将不断增大. 如何通过优化算法提高收敛效率以满足系统的实时性需求, 是本文后续需要考虑的一个问题. (3) 本文后续考虑建立策略模板, 对搜索产生的结果进行建模, 以实现在不同运行环境下的策略迁移, 减少系统开销. (4) 为保证策略的有效性, 本文后续希望通过引入强化学习, 根据反馈信息对调整策略进行在线修正.

References:

[1] Sun YX, Zhou XZ, Tang BJ, Xu S, Zhu Z. Research on development path and future prospect of intelligent command and control system. Fire Control & Command Control, 2020, 45(11): 60–66 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-0640.2020.11.012]

[2] Pei Y, Xu BQ. Development course and trend of the US C^{4}ISR system. Systems Engineering and Electronics, 2005, 27(4): 666–671 (in Chinese with English abstract). [doi: 10.3321/j.issn:1001-506X.2005.04.024]

[3] Wu T, Li QS, Dai Q, Mao XB. Self-adaptive decision method for dynamic evolution of command and control information system. Command Information System and Technology, 2018, 9(5): 43–50 (in Chinese with English abstract). [doi: 10.15908/j.cnki.cist.2018.05.007]

[4] Coker Z, Garlan D, Le Goues C. SASS: Self-adaptation using stochastic search. In: Proc. of the 10th IEEE/ACM Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems. Florence: IEEE, 2015. 168–174. [doi: 10.1109/SEAMS.2015.16]

[5] Wang L. Software adaptation mechanism based on multi-agent theory and parallel search [Ph.D. Thesis]. Xi'an: Xidian University, 2018 (in Chinese with English abstract).

[6] Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. Boston: Addison-Wesley Longman Publishing Co., 1989. 229–230.

[7] Wang CD, Zhang YY. Adaptive pseudo-parallel genetic algorithm based on real coding. Journal of Xi'an Jiaotong University, 2003, 37(7): 707–710 (in Chinese with English abstract). [doi: 10.3321/j.issn:0253-987X.2003.07.012]

[8] Vafaeyan S, Thibault J. Selection of pareto-optimal solutions for process optimization using rough set method: A new approach. Computers & Chemical Engineering, 2009, 33(11): 1814–1825. [doi: 10.1016/j.compchemeng.2009.07.007]

[9] Engau A, Wiecek MM. 2D decision-making for multicriteria design optimization. Structural & Multidisciplinary Optimization, 2007, 34(4): 301–315. [doi: 10.1007/s00158-006-0078-y]

[10] Luque M, Ruiz F, Miettinen K. Global formulation for interactive multiobjective optimization. Or Spectrum, 2011, 33(1): 27–48. [doi: 10.1007/s00291-008-0154-3]

[11] Rosenman MA, Gero JS. Reducing the pareto optimal set in multicriteria optimization (with applications to pareto optimal dynamic programming). Engineering Optimization, 1985, 8(3): 189–206. [doi: 10.1080/03052158508902489]

[12] Figueira J, Greco S, Ehrogott M. Multiple criteria decision analysis: State of the art surveys. New York: Springer, 2005. 133–153. [doi: 10.1007/b100605]

[13] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002, 6(2): 182–197. [doi: 10.1109/4235.996017]

[14] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc. of ICNN '95-Int'l Conf. on Neural Networks. Perth: IEEE, 1995.

1942–1948. [doi: 10.1109/ICNN.1995.488968]

[15] Jin X, Lu XM, Duanmu ZY, Yan JJ, Zhao KJ. Software defined method on service-based flexible reconfiguration of command & control system. In: Proc. of the 3rd China Command and Control Conf. . Beijing: National Defense Industry Press, 2015. 387–392 (in Chinese with English abstract).

[16] Wang X, Zhang JY, Wan LJ, Jiao ZQ. Holonic-C2 organization decision allocation and evolution method. Journal of National University of Defense Technology, 2020, 42(6): 157–166 (in Chinese with English abstract). [doi: 10.11887/j.cn.202006020]

[17] Ferreira JC, Fonseca CM, Gaspar-Cunha A. A new methodology to select the preferred solutions from the pareto-optimal set: Application to polymer extrusion. AIP Conf. Proceedings, 2007, 907(1): 861–866. [doi: 10.1063/1.2729621]

[18] Lalanda P, McCann JA, Diaconescu A. Autonomic Computing. London: Springer, 2013. 153–213. [doi: 10.1007/978-1-4471-5007-7] [doi: 10.1007/978-1-4471-5007-7]

[19] Damianou N, Dulay N, Lupu EC, Sloman M. Ponder: A language for specifying security and management policies for distributed systems. Tribology, 2000, 2(3): 179–180.

[20] Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel JM. RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proc. of the 17th IEEE Int'l Requirements Engineering Conf. Atlanta: IEEE, 2009. 79–88. [doi: 10.1109/RE.2009.36]

[21] Welsh K, Sawyer P. Understanding the scope of uncertainty in dynamically adaptive systems. In: Proc. of the 16th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2010. 2–16. [doi: 10.1007/978-3-642-14192-8_2]

[22] Franco JM, Correia F, Barbosa R, Zenha-Rela M, Schmerl B, Garlan D. Improving self-adaptation planning through software architecture-based stochastic modeling. Journal of Systems & Software, 2016, 115: 42–60. [doi: 10.1016/j.jss.2016.01.026]

[23] Cámara J, Lopes A, Garlan D, Schmerl B. Adaptation impact and environment models for architecture-based self-adaptive systems. Science of Computer Programming, 2016, 127: 50–75. [doi: 10.1016/j.scico.2015.12.006]

[24] Wang Z, Yan M, Liu S, Chen JJ, Zhang DD, Wu Z, Chen X. Survey on testing of deep neural networks. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1255–1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5951.htm [doi: 10.13328/j.cnki.jos.005951]

[25] Belhaj N, Belaïd D, Mukhtar H. Self-adaptive decision making for the management of component-based applications. In: Proc. of the OTM Confederated Int'l Conf. on the Move to Meaningful Internet Systems. Rhodes: Springer, 2017. 570–588. [doi: 10.1007/978-3-319-69462-7_36]

[26] Goldman RP, Musliner DJ, Krebsbach KD. Managing online self-adaptation in real-time environments. In: Proc. of the 2nd Int'l Workshop on Self-adaptive Software: Applications. Balatonfüred: Springer, 2003. 6–23. [doi: 10.1007/3-540-36554-0_2]

[27] Bencomo N, Belaggoun A, Issarny V. Dynamic decision networks for decision-making in self-adaptive systems: A case study. In: Proc. of the 8th Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems. San Francisco: IEEE, 2013. 113–122. [doi: 10.1109/SEAMS.2013.6595498]

[28] Harman M, Mansouri SA, Zhang YY. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012, 45(1): 11. [doi: 10.1145/2379776.2379787]

[29] Cheng BHC, Ramirez AJ, McKinley PK. Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty. In: Proc. of the 1st Int'l Workshop on Combining Modelling and Search-based Software Engineering (CMSBSE). San Francisco: IEEE, 2013. 1–6. [doi: 10.1109/CMSBSE.2013.6604427] [doi: 10.1109/CMSBSE.2013.6604427]

[30] Pascual GG, Pinto M, Fuentes L. Run-time adaptation of mobile applications using genetic algorithms. In: Proc. of the 8th Int'l Symp. on Software Engineering for Adaptive and Self-managing Systems. San Francisco: IEEE, 2013. 73–82. [doi: 10.1109/SEAMS.2013.6595494]

[31] Cardellini V, Casalicchio E, Grassi V, Iannucci S, Presti FL, Mirandola R. MOSES: A framework for QoS driven runtime adaptation of service-oriented systems. IEEE Trans. on Software Engineering, 2012, 38(5): 1138–1159. [doi: 10.1109/TSE.2011.68]

[32] Esfahani N, Elkhodary A, Malek S. A learning-based framework for engineering feature-oriented self-adaptive software systems. IEEE Trans. on Software Engineering, 2013, 39(11): 1467–1493. [doi: 10.1109/TSE.2013.37]

[33] Andrade SS, de A. Macêdo RJ. A search-based approach for architectural design of feedback control concerns in self-adaptive systems. In: Proc. of the 7th IEEE Int'l Conf. on Self-adaptive and Self-organizing Systems. Philadelphia: IEEE, 2013. 61–70. [doi: 10.1109/SASO.2013.42]

[34] Chen T, Li K, Bahsoon R, Yao X. FEMOSAA: Feature-guided and knee-driven multi-objective optimization for self-adaptive software. ACM Trans. on Software Engineering and Methodology, 2018, 27(2): 5. [doi: 10.1145/3204459]

[35] Walcott KR, Soffa ML, Kapfhammer GM, Roos RS. TimeAware test suite prioritization. In: Proc. of the 2006 Int'l Symp. on Software Testing and Analysis. Portland: ACM, 2006. 1–12. [doi: 10.1145/1146238.1146240]

[36] Michael CC, McGraw G, Schatz MA. Generating software test data by evolution. IEEE Trans. on Software Engineering, 2001, 27(12): 1085−1110. [doi: 10.1109/32.988709]

[37] Wappler S, Wegener J. Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. In: Proc. of the 8th Annual Conf. on Genetic and Evolutionary Computation. Seattle: ACM, 2006. 1925−1932. [doi: 10.1145/1143997.1144317]

[38] Barreto A, de O. Barros M, Werner CML. Staffing a software project:A constraint satisfaction and optimization-based approach. Computers and Operations Research, 2008, 35(10): 3073−3089. [doi: 10.1016/j.cor.2007.01.010]

[39] Le Goues C, Nguyen T, Forrest S, Weimer W. GenProg: A generic method for automatic software repair. IEEE Trans. on Software Engineering, 2012, 38(1): 54−72. [doi: 10.1109/TSE.2011.104]

[40] Kim D, Nam J, Song J, Kim S. Automatic patch generation learned from human-written patches. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 802−811. [doi: 10.1109/ICSE.2013.6606626]

附中文参考文献:

[1] 孙宇祥, 周献中, 唐博建, 徐爽, 朱紫. 智能指挥与控制系统发展路径与未来展望. 火力与指挥控制, 2020, 45(11): 60−66. [doi: 10.3969/j.issn.1002-0640.2020.11.012]

[2] 裴燕, 徐伯权. 美国C4ISR系统发展历程和趋势. 系统工程与电子技术, 2005, 27(4): 666−671. [doi: 10.3321/j.issn:1001-506X.2005.04.024]

[3] 吴桐, 李青山, 戴清, 毛晓彬. 指挥控制信息系统动态演化的自适应决策方法. 指挥信息系统与技术, 2018, 9(5): 43−50. [doi: 10.15908/j.cnki.cist.2018.05.007]

[5] 王璐. 基于多智能体并行搜索的软件自适应机制 [博士学位论文]. 西安: 西安电子科技大学, 2018.

[7] 王成栋, 张优云. 基于实数编码的自适应伪并行遗传算法. 西安交通大学学报, 2003, 37(7): 707−710. [doi: 10.3321/j.issn:0253-987X.2003.07.012]

[15] 金欣, 陆晓明, 端木竹筠, 闫晶晶, 赵克俭. 基于软件定义的服务化指控系统灵活重构方法. 第三届中国指挥控制大会. 北京: 国防工业出版社, 2015. 387−392.

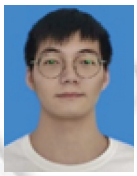
[16] 王勋, 张杰勇, 万路军, 焦志强. Holonic-C2组织决策分配及演化方法. 国防科技大学学报, 2020, 42(6): 157−166. [doi: 10.11887/j.cn.202006020]

[24] 王赞, 闫明, 刘爽, 陈俊洁, 张栋迪, 吴卓, 陈翔. 深度神经网络测试研究综述. 软件学报, 2020, 31(5): 1255−1275. http://www.jos.org.cn/1000-9825/5951.htm [doi: 10.13328/j.cnki.jos.005951]

王璐(1991−), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为软件演化与自适应, 智能化软件运维.

王展(1996−), 男, 硕士, 主要研究领域为自适应软件, 微服务负载均衡.

霍其恩(1998−), 男, 硕士, 主要研究领域为自适应软件, 微服务软件运维.



姜宇轩(1999−), 男, 硕士, 主要研究领域为自适应软件, 故障诊断技术.

李青山(1973−), 男, 博士, 博士生导师, CCF 杰出会员, 主要研究领域为面向智能体的软件工程, 软件演化与自适应.