摘　要

程序自动生成一直是软件工程领域研究的重点，如何让机器理解代码和自动编写程序也是人工智能领域的期望。在算法设计这一过程中，面对不同的问题需要考虑不同问题的特点、求解目标以及约束条件，从而难以实现算法的可重复构造。以最优化问题为例，其在生活和工程中的应用无处不在，在面对结构相似的但数据不同的最优化问题时不得不重新设计算法。通过研究此类优化问题之间的共性，可以使算法设计过程可重用性更高，帮助人们减少在面对相似问题的算法设计过程中的脑力活动，提高算法设计的效率和质量。强化学习在求解组合最优化问题时，通过“交互-试错”学习方式可以学习相同类型问题之间的算法求解策略，为这些问题可以找到一个可复用的“元算法”算法设计模式，有效避免传统理论设计的算法需要大量的专业知识进行手工的反复试验问题。

本文基于所在研究团队研发的软件形式化与自动化开发的开发方法 PAR 及其支撑平台——PAR 平台，基于 PAR 方法中统一的算法设计方法，为求解图最优化问题构造了一种可重用的算法设计理论和框架，并将其与强化学习融合起来以实现算法程序的自动构造。以最短路径和旅行推销员问题为例，通过强化学习方法学习到的以 PAR 方法为基础的分划递推的算法构造策略，并通过强化学习中智能体的动作来捕获当前算法的正确程度，不断的调整算法程序构造策略，为最优化问题逐步构建成一个合适的解决方案。以此为基础，开发了最优化问题 Radl 算法生成系统，并对系统以及生成的算法的可靠性进行了理论分析。

本文主要工作如下，具有一定的创新性：

1. 以强化学习的感知和试错的学习方式为核心，结合基于 PAR 方法提出的图最优化问题的可重用算法设计理论以及算法框架，将算法程序的构造过程交由强化学习中智能体来自主探索算法程序设计策略，最终实现了图最优化问题的算法程序的自动构造，提高了算法程序的开发效率和可复用性。

2. 基于强化学习这一途径，为实现图最优化问题的算法程序设计可重用以及程序自动生成这一目标，设计并实现了解决图类型最优化问题 Radl 算法程序自动生成系统，进一步提高了 PAR 平台中算法程序开发的自动化程度。

关键词：PAR 方法；程序生成；算法设计；自动化；强化学习

Abstract

Automatic generation of programs has been the focus of research in the field of software engineering, and how to make machines understand the code and write programs automatically is also an expectation in the field of artificial intelligence. In the process of algorithm design, it is difficult to achieve reproducible construction of algorithms by considering different problem characteristics, solution goals and constraints for different problems. In the case of optimization problems, for example, which are ubiquitous in life and engineering, algorithms have to be redesigned when faced with optimization problems with similar structures but different data. By examining the similarities between such optimization problems, we can make the algorithm design process more reusable, assist people in reducing mental activities while designing algorithms that face similar problems, and improve the efficiency and quality of algorithm design. In solving combinatorial optimization problems, reinforcement learning can learn algorithmic solution strategies between the same types of problems through " interaction-trial-and-error" learning, and find a reusable "meta-algorithm" algorithm design model for these problems, effectively avoiding the problem of traditional theoretical design algorithms that require a lot of expertise for manual and repeated experiments.

In this article, we construct a reusable algorithm design theory and framework for solving graph optimization problems based on the unified algorithm design approach in the PAR method, and integrate it with reinforcement learning to automate the construction of algorithmic programs. Taking the shortest path and traveling salesman problems as examples, we use the PAR-based recursive algorithm construction strategy learned by the reinforcement learning method to capture the correctness of the current algorithm and continuously adjust the algorithm construction strategy to build a suitable solution for the optimization problem. Based on this, a Radl algorithm generation system for optimization problems is developed, and the reliability of the system and the generated algorithms is analyzed theoretically.

The main work of this paper is as follows, which has certain innovations:

1. The construction of the algorithm program is left to the intelligent body in reinforcement learning to explore the algorithm program design strategy autonomously, and finally the automatic construction of the algorithm program for graph optimization

problems using the reusable algorithm design theory and algorithm framework for graph optimization problems based on the PAR method.

2. In order to achieve the goal of reusable algorithm program design and automatic program generation for graph optimization problems, which further improves the automation of algorithm program development in the PAR platform, we designed and implemented an automatic generation system for solving Radl algorithm programs for graph optimization problems based on the reinforcement learning approach.

Key words: PAR Method; Program Generation; Algorithm Design; Automation; Reinforcement Learning

目　　录

第 1 章　绪论

1.1 研究背景

当前算法问题中主要可分为已有算法解和无常规算法解两类问题，在最优化问题中便存在着许多无算法解的问题，以至于在某些条件下难以求解这一问题。而算法程序自动生成是计算机科学和信息技术领域中的重要研究课题，其研究成果对学术界和工业界都具有重要意义[1]。但在算法程序开发过程中，用来求解不同问题之间的算法往往是以一种孤立的形式存在，无法有效的建立起它们之间的联系，从而难以实现算法程序的可重复使用。算法程序自动生成的应用非常广泛，例如在计算机科学领域中，它可以被用于生成程序补全工具[2]、自动程序调试、自动代码生成[3,4]等工具。随着“人工智能”这一领域的兴起，人们开始追求算法程序的可重复生成，以实现算法程序生成的部分自动化、完全自动化[5-8]和软件自动化[9]等这一更高的科学目标，而算法在这其中占据着重要的地位。目前算法程序开发可考虑的方法纷繁复杂，使用哪种方法来进行开发大多情况下依赖于人的先验知识来进行手工开发，并且算法程序的设计工作是依靠大脑而进行的一项创造性活动，这给算法程序自动生成带来了不小的挑战和困难。

软件自动化是二十一世纪计算机科学领域中研究的重点对象，图灵奖获得者 James Gray 提出的新世纪必须解决的问题就包含“程序设计自动化”这一重要问题[10]。目前软件自动化中的核心内容算法程序自动化已经取得了一定的研究进展。由科学家 D.R.Smith 开发的算法程序自动化系统 CYPRESS[11]成功的运用了一种算法设计方法分治法，并且将其形式化地运用到其系统中，成功的实现了一组分类程序的自动化设计。在 NDAUTO 系统中算法程序开发被设计为人机交互的合作模式，当系统无法解决问题时便向人提出请求帮忙解决，NDAUTO 旨在减少人的干预。由南京大学徐家福等人牵头研制的算法设计自动化系统 NDADAS(ND Algorithm Design Automation System)[12]在原有 NDAUTO 系统基础上实现了一套完整的问题规约分析机制，将非算法性的软件规约自动或半自动的转换为算法性的软件规约，最后再选取合适的算法设计方案自动生成程序代码。此外还有基于程序变换的通用算法程序生成系统 Designware[13]、基于演绎的 LSIP 系统、基于归纳推理的 MagicHaskel[14]系统等等。尽管上述系统实现了一定程度的算法自动化，但大多都局限于特定知识领域的问题，无法将模型进行泛化，从而无法有效解决复杂多变的实际问题。

最优化是应用很广的一个数学分支，很多问题都可以看成是某种形式的最优

化问题[15]。在数学和工程应用中，相似的最优化问题反复出现，例如求解路径问题的单源最短路径问题，巡回售货员问题等等。但每次面对问题形式相似、但问题中所需求解的数据不同的情况下，不得不为这些问题重新思考并提出新的算法解决方案[16]。基于图的最优化问题一直以来都受到工业应用界和学术理论界的广泛关注，为此类问题寻找一个共同的算法自动求解策略，进而实现算法程序的自动生成十分具有研究意义[17-19]。

1.2 研究现状

算法程序形式化是实现算法设计自动化的基础和前提条件，在算法程序自动化研究中，通常会与算法程序形式化相结合进行研究。算法程序形式化和自动化目前处于发展初期，国内外众多学者使用了不同的技术进行了研究。尽管部分研究工作较少探讨算法设计自动化，但其研究方法、途径和成果均构成了算法设计自动化研究的基础[20]。

基于演绎推理的方法将算法程序设计看作演绎过程，从给定的问题规约出发，使用各种规则进行演绎和搜索，借助演绎推理综合出程序。问题规约通常使用基于一阶谓词逻辑的前后置断言来表示。演绎本质上对构成问题求解逻辑的所有可行路径进行一个检索的过程，由于检索方法的低效，尚未开发出规模较大的复杂程序。

程序变换是一种形式化开发算法程序的方法，也是实现自动程序设计的重要途径之一。其具有简单的实现逻辑和便捷的修改扩充方式，通过严格的数理逻辑来实现程序之间的等价变换，为算法程序变换过程提供各种自动化的支持。徐家福教授等人牵头研制的算法设计自动化系统 NDADAS 以广谱语言 FGSPEC（Functional Graphical SPECification）为基础，使用演绎与转换相结合的途径来实现自动生成程序代码。Designware 系统则更加专注于应用范围更广但难度更大的通用算法程序的设计，它基于一个无特定领域的通用算法设计知识库，在此知识库的基础之上实现从规约到算法程序的开发。该系统及其依赖于与专家之间的交互过程，使用者需要一定的数学知识为基础，且算法设计策略的选择也需交由用户自行决策，并没有一个行之有效的指导策略，为算法程序自动化带来了困难。

归纳推理方法通过观察和分析问题的一般性质，将问题解决方案的程序设计方法作为问题的规约说明描述，利用归纳推理，通过对已知问题的解决方法的分析，归纳总结出解决一类问题的通用方法，从而设计出更加普适的程序。代表性工作是基于归纳推理的 THESYS 系统和 MagicHaskeller 系统等等。

目前许多学者主张将人工智能(AI)技术运用到算法程序自动生成和指导算法设计工作中去，以期望加快智能化的进程让 AI 完全实现可自主构造算法程序。这个领域涉及到许多不同的问题，包括如何自动设计算法程序，如何在不同应用

场景下优化算法的性能，以及如何保证算法的可靠性和正确性等问题。目前，与人工智能相结合的算法程序自动化生成已经取得了许多重要的研究进展。具体研究方向包含了元学习、深度强化学习、演化计算和可解释性人工智能等等。例如，UC Berkeley 的研究成果可以实现算法的自我优化、以及 DeepMind 的让算法“学习强化学习”和其最近在 Nature 上发表的新研究，使用强化学习发现了新的矩阵乘法。

在文献[21]中，UC Berkeley 的研究人员实现了一种可以在没有过多的人工干预的情况下，让算法程序能够自我优化的方法。它们基于强化学习这一方法，总结了算法程序设计过程其实需要循环迭代来验证其算法的正确性思想，通过强化学习中智能体的“交互-试错”模型搜索策略来让 AI 学习如何去不断的优化算法，在文献[22]中还提出了一种让算法学习强化学习的方法。经过实验证明，基于机器自我编程并不断优化的算法程序在运行收敛速度和计算复杂度，以及目标值的求解方面都优于传统手工设计的算法程序。

2015 年，DeepMind 团队在文献[23]中提出了一种能够模拟人类编程的初级程序“神经编程解释器（Neural Programmer-Interpreters, NPI）”，它能够实现自我学习并已实现了一些简单程序的编写，如快速排序算法，且生成的算法具有很强的泛化能力。

2022 年，DeepMind 在 Nature 发表的文献[24]中展示了最近在算法程序设计自动化这一研究领域的最新研究成果。它们将深度学习和强化学习融合在一起，提出了一种可实现算法设计的新型人工智能系统“AlphaTensor”。这一新型人工智能系统可以用于计算机系统中常用的矩阵乘法算法等基础算法中，且发现的新型算法都优于以往的算法。这一研究成果打破了数学领域中，关于两个矩阵相乘最快的算法的尘封已久的记录。通过将设计算法的过程转换为一场单人游戏，让强化学习的智能体充当玩家这一角色，通过一组三维数字张量来捕捉当前算法的正确程度，并允许智能体在的算法指令下，通过移动交互修改三维数字张量并最终置为零。在强化学习不断的训练过程中逐渐发现了历史上出现过的最快的矩阵乘法，随着时间的推移，并展现了其发现新算法的强大能力以及前所未有速度，这表明当前人工智能在算法设计这一领域已经部分超越了人类，且向着全面赶超人类这一方向不断前进。

人工智能面临的核心难题之一便是让机器能够自我学习，快速地从已有程序中生成新的程序，并在特定条件下自动执行这些程序来解决各种任务。在这之中关键之处就在于实现这些程序的算法能否交由 AI 来完成设计并不断优化。从目前人工智能在算法程序自动化领域的研究成果来看，强化学习作为迈向通用人工智能的重要途径，已经在算法程序自动生成这一领域已经有了一定的基础，能够加速 AI 实现算法程序设计自动化这一目标。

1.3 研究内容

现阶段的算法程序开发过程中基于传统算法理论可以考虑的算法设计方法纷繁复杂，没有行之有效的理论来指导算法程序自动化系统中应选取何种算法设计方法，导致系统的自动化程度较低，因此需要一种统一的算法设计理论来避免纠结于选择何种算法设计方法。本文基于实现强人工智能途径的强化学习算法为核心，结合以 PAR 方法为理论指导提出的最优化问题可重用算法设计理论和算法框架为基础，本文将开展的工作如下：

(1) 以强化学习方法为核心，并结合最优化问题可重用算法设计理论，将算法程序开发过程基于强化学习的五个基本要素将其抽象成为马尔可夫决策过程，用以解决图类型最优化问题的算法程序自动生成这一问题。

(2) 对一类最优化问题算法之间的共性进行研究和总结，采用强化学习方法对最优化问题算法设计过程进行自动化研究，结合 PAR 平台中各开发环节已有的转换系统，基于最优化问题算法设计理论的泛型算法结构，确定从问题规约到算法程序的程序生成规则。

(3) 结合传统算法自动化系统框架以及强化学习算法，设计并实现最优化问题 Radl 算法自动生成系统。

(4) 给定最优化问题算法实例规约，并对系统自动生成的最优化问题算法程序并进行正确性验证。

1.4 本文组织

本文共分六章进行阐述，具体组织如下：

第一章， 绪论。简要阐述本文的研究背景、研究现状和研究内容。

第二章， 相关技术。本章主要说明了为实现算法程序自动化生成这一目标所需的相关技术理论，包括形式化方法、程序智能合成、最优化原理和强化学习方法，以及本文实现算法自动生成的途径。

第三章， 最优化问题算法设计理论与 Radl 语言。本章主要介绍了算法程序设计语言 Radl、一种实现最优化问题可重用算法设计的框架、以及实现算法程序生成的泛型算法结构。

第四章， 最优化问题算法程序自动生成系统。主要介绍了结合传统算法程序自动化系统的框架以及行为主义强化学习算法，将基于最优化问题可重用算法设计理论以及泛型算法结构融入到强化学习核心机制中实现算法的自动生成，最终设计并实现最优化问题 Radl 算法自动生成系统。

第五章， 案例展示。从手工和非手工，已有和未知问题两方面进行系统测试。

第六章， 总结和展望。对本文工作进行总结，并对后续工作做出展望和建议。

第 2 章　相关技术

2.1 形式化方法及 PAR 方法

2.1.1 形式化方法

形式化方法以形式逻辑系统为基础，支持对计算系统进行严格的规约、建模和验证并为此设计算法，从而建立计算机辅助工具的方法[25]。运用形式化方法来设计软件和硬件，目的是通过数学分析来提升设计的可靠性和鲁棒性。

形式化方法最新的应用成果是在工业上，比如一些极其重要且对安全性要求极高的领域（几乎不允许差错），比如航空航天，高铁地铁等等。法国人 Abrial 将形式化方法应用到了这些领域，同时他还发明了 B 方法[26]，成功应用在法国自动驾驶地铁 14 号线。得益于他的贡献，使得罢工期间这条线最重要的两大系统：控制系统和通信系统几乎没有受到影响。随之该方法被广泛采纳，在世界各地的自动驾驶地铁中得到应用。例如上海开通的地铁 15 号线，这条全国最高等级的自动驾驶地铁通过了形式化方法的验证，保障了 15 号线自动驾驶功能的安全；又如中国的月球采样探测器“嫦娥五号”[27]，同样也采用了形式化方法进行验证保证了系统的时序安全性。形式化开发方法具有以下几个优点：

1. 精确性：形式化方法强调精确的描述和推导，可以避免自然语言表述的模糊性和歧义性，从而能够确保软件系统的正确性和可靠性。
2. 可验证性：形式化方法提供了明确的规范和验证手段，可以对软件系统进行形式化证明和验证，确保系统的正确性，减少漏洞和错误的可能性。
3. 重用性：形式化方法可以支持组件重用，通过重复利用已有的形式化组件来提升效率。

虽然在软件开发和程序设计中形式化方法还占据着一席之地，但其目前还存在着一些缺陷。

1. 形式化方法需要具备一定的理论基础，其中涵盖的数学理论限制了许多程序设计人员的应用。
2. 形式化开发方法需要开发人员进行严格的规范化开发，需要花费较高的人工成本，这也限制了其应用范围。
3. 形式化方法缺乏通用标准，不同的方法可能会产生不同的结果，这可能会导致一些不确定性和风险。

综上所述，虽然形式化开发方法具有许多优点，但也存在一些缺陷和限制。因此，在实际运用中，应该根据问题的特点和需求选择合适的开发方法和技术。

2.1.2 PAR 方法

由于上述形式化方法目前存在着一定的缺陷，所以在软件系统研发过程中不能完全依赖形式化方法，而需要根据实际情况合理地选择使用或部分使用。形式化开发方法的宗旨是为了提供更为优秀的理论、技术和工具，以进一步扩大其应用价值，并使其具备更深层次的科学研究意义。形式化方法正在将已经通过科学理论验证的软件开发方法融入到传统开发人员基于经验的开发方法中去，改变传统的开发过程使之科学化。将形式化方法应用到软件系统开发过程中去，也是解决现阶段“软件危机”的根本途径之一，影响程度重大且深远。下面简要阐述一下本团队提出的方法 PAR 方法。

PAR 方法和 PAR 平台[28-31] 简称 PAR，是一种基于形式化方法开发的软件开发和程序设计综合开发环境平台，PAR 平台整体架构如图 2-1 所示。PAR 由自定义规约描述语言 SNL、算法规约和描述语言 Radl、抽象程序设计语言 Apla、算法到程序自动转换工具、算法程序和软件系统的敏捷开发方法构成。在 PAR 方法中，提出了循环不变式的新定义和新的开发策略、统一的算法程序设计方法、新的算法表示方法，以及用于描述算法和程序的语言。提出了在软件模型设计和变换阶段实现泛型程序设计的方法，在建模语言中增加了新的泛型程序设计机制，降低了模型建立的复杂度、提高了模型的抽象性和可靠性，使得算法程序和软件开发的效率显著提高。

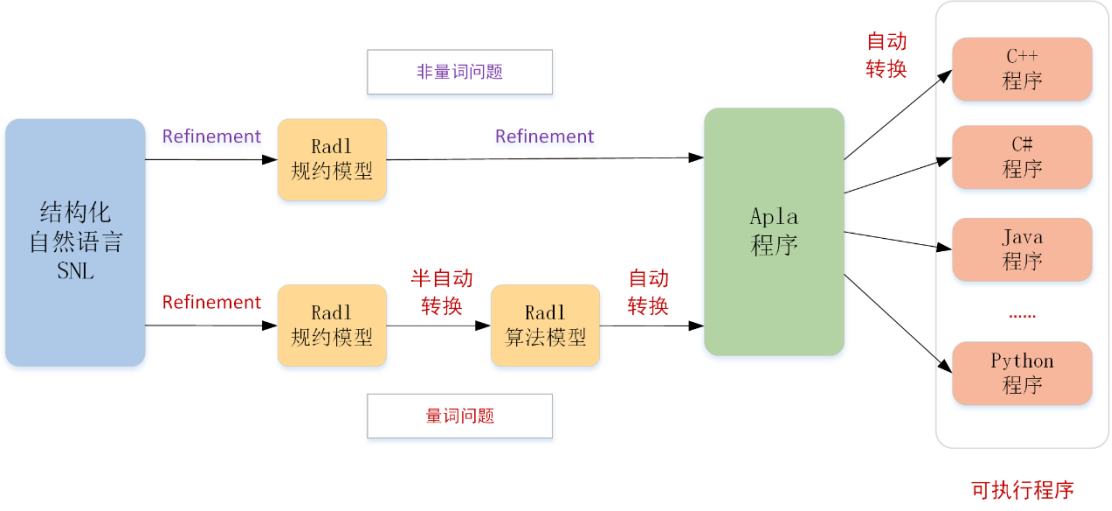

图 2-1 PAR 平台架构图

PAR 方法由建模语言以及可形式化推理验证的形式规则组成，涵盖了新的算法表示和建模语言，泛型程序设计方法和规约变换规则。PAR 方法是一种基于分划递推思想、量词变换规则和循环不变式新定义，充分利用数据抽象、功能抽象、软件重用、泛型、重载等成熟的程序设计技术形式化开发复杂的算法和程序的敏捷开发方法，基于此方法以及开发和证明了许多典型的算法[32-36]。

PAR平台可以支撑算法程序和软件开发的全过程，其包含了从SNL到Radl、Radl→Apla、Apla→可执行程序一系列的自动转换工具[37-40]。PAR 平台是基于MDA 并结合实用形式化方法（PAR 方法）开发的程序自动化系统平台。该平台上的一系列自动生成工具可以实现从需求模型到算法模型，再到抽象程序模型，最后到可执行程序模型的若干转换，转换的目标是生成可执行程序，通过以上方法可以减少人的脑力创造性劳动。

基于 PAR 方法和 PAR 平台进行算法程序设计可分为以下几个步骤：

第一步：构造问题的形式化规约，基于形式语言精确描述算法所要做的工作。

第二步：保存问题结构规模不变，分划目标问题直至能直接求解。

第三步：构造问题的求解序列的递推关系$S_i = F(\overline{S_j})$，并定义函数和变量的初始条件，再将初始条件和递推关系组合表示为 Radl 算法。

第四步：结合 PAR 方法循环不变式的新定义以及新的开发策略，将 Radl 算法模型自动转换为 Apla 语言描述的算法程序。

第五步：进一步的将上述过程得到的 Apla 语言程序通过转换工具自动转换为高级语言可执行程序。

2.2 程序智能合成

2.2.1 人工智能

人工智能的概念从 1950 年左右开始被提出，在这一概念被提出不久之后，诞生了许多充满热情的研究学者以相关技术，并开始快速发展[41]。上世纪 70 年代左右，人工智能进入了以知识驱动为基础的应用研究，构建了可以用来解决特定领域问题的“专家系统”，例如 1972 年设计的 MYCIN 能够诊断血液传染病，知识库系统工程在当下引起了一波 AI 研究的热潮。早期的系统适用于更宽的问题选择和更难的问题时效果均不理想，基于专家系统的人工智能逐渐走向寒冬。

到了 20 世纪 90 年代，神经网络和机器学习技术的出现使得人工智能的发展进入了一个新的阶段。这些技术可以让计算机通过大量数据的学习，自动发现数据中的规律和模式。1997 年，IBM 的“深蓝”国际象棋程序打败世界冠军卡斯帕罗夫(Kasparov) [42]，成为人工智能史上的一个重要里程碑。

从人工智能发展的历史进程来看，人工智能经历了三大主义学派的演变，分别是符号主义、连接主义和行为主义[43]。符号主义将理解和认知这一人类独有的思维模式表示为符号，而智能则是符号的表征和运算过程，计算机由各个逻辑部件构成，其内部运算逻辑也符合物理符号系统。因此，通过构建符合人脑思维的智能形式逻辑符合和知识规则，从而实现用计算机中的各种符号系统来模拟人的认知和思考这一智能行为。联结主义又称连接主义，这一主义受到仿生学的启发，

尤其是对人类大脑如何运作的探索。连接主义通过构造和人脑相似的神经元的网络模型来学习和训练出符号人类思维的智能行为，通过不断的训练最终使人工神经网络产生智能。行为主义是一种模仿自然生物在环境中采取智能行为方法，该方法的思想源于进化论和控制论，基于控制论以及“感知-动作”行为模型原理，通过建立响应外部刺激的输入输出模型来训练机器学习算法，从而模拟人类和动物的行为。该学派认为智能这一智慧性知识取决于感知和行为，而非知识表示和推理。相比于智能是什么，行为主义对如何实现智能行为更感兴趣。在行为主义者眼中，只要机器能够具有和智能生物相同的表现，那它就是智能的。

学习一直是 AI 中最具挑战的领域。学习的重要性是庸置疑的，因为这种能力是智能行为的最重要的特征。专家系统可以通过执行大量且费用昂贵的计算来求解问题。不过不同于人类的是，如果专家系统第二次遇到相同或类似的问题，它通常并不记得上一次的解，而是再执行一次相同的计算过程。对于第三次、第四次甚至任何次的重复出现，它的处理方式都是如此——很难说这是一个智能的问题求解器的行为。解决这一问题的显而易见的方法是让程序自己能够学习，不管是从经验、类比、实例中，通过被“告知”如何去做，还是根据结果对其进行奖惩。

虽然让程序自己学习是一个很难的领域，但是很多程序说明这是可能的。一个早期的程序是被设计用来发现数学定律的自动数学家 (Automated Mathematician，AM) (Lenat 1977，1982)。在被赋予了集合理论的概念和公理后，AM 能够导出很多重要的数学概念，比如集合的基数、整数运算以及数论的很多结果。AM 通过修改其现有的知识库来猜想新的定理，然后启发式地从大量备择定理中搜寻“最佳者”。例如，科顿等人 (Cotton et al. 2000) 设计了一个程序，这个程序可以自动发明“有趣的”整数序列。

早期有影响的研究还包括温斯顿的研究，他根据积木世界中的一系列样例导出像“拱形”这样的结构概念(Winston 1975)。ID3 算法已经被证明可以成功地从样例中学习通用模式(Quinlan 1986)。Meta-DENDRAL 可以从具有已知结构的化合物数据样例中学习规则，用于解释有机化学中的大量光谱数据。Teiresias-专家系统的智能“前端”——可以把顶层的建议转化为新的规则放入知识库 (Davis 1982)。计算高手可以设计完成积木世界作的规划，方法是使用一个迭代过程不断地设计规划、测试规划，然后纠正在候选规划中发现的所有不足 Sussman 1975)。对基于解释的学的研究已经表明先验知在学习中的有效性 (Mitchell et al1986，DeJong and Mooney 1986)。

机器学习程序的成功表明存在一系列通用的学习原则，这些原则将使我们构造出具有在实际领域内进行学习的能力的程序。

2.2.2 程序合成

程序合成（Program Synthesis）是一种自动软件开发技术，其目的是基于给定的规范或需求，自动生成满足这些规范或需求的程序。程序合成通常使用形式化方法和人工智能技术，将问题规范转化为程序代码，以实现程序自动化生成。程序合成可以大大提高软件开发的效率和质量，减少开发成本和人力资源的浪费。它在许多领域都有广泛的应用，包括嵌入式系统、网络协议、数据库查询、编译器等。

程序合成的主要思想是将问题规范转化为程序代码，通过搜索算法或优化算法等方法，自动合成满足规范的程序。程序合成的最大优点是可以自动生成正确性证明，因为生成的程序是基于规范生成的，因此可以保证程序的正确性。程序合成技术的主要挑战在于如何将问题规范转化为程序代码，以及如何设计高效的搜索算法和优化算法以此来解决未知算法解的问题。程序合成的基本流程包括以下几个步骤：

1. 规范输入：程序合成的输入是一个形式化的规范，通常是逻辑公式或约束条件。规范可以是自然语言描述的需求，也可以是形式化的规格说明。
2. 搜索空间定义：程序合成需要定义搜索空间，即程序合成算法需要搜索的所有可能的程序代码。搜索空间的定义是程序合成的关键之一，因为搜索空间的大小直接影响程序合成的效率和精度。
3. 合成算法设计：程序合成需要设计搜索算法或优化算法，以在搜索空间中找到满足规范的程序。搜索算法的设计是程序合成的核心，因为搜索算法的效率和精度直接影响程序合成的结果。
4. 程序生成：程序合成算法通过搜索空间中找到满足规范的程序。程序生成的结果通常是一个程序代码，可以用于软件开发或测试。

程序合成技术主要使用形式化方法和人工智能技术。形式化方法是指使用数学语言和逻辑语言来描述问题和规范，以及定义程序的语法和语义。人工智能技术是指使用机器学习和深度学习等技术来自动学习程序的结构和语义。

程序合成技术的发展离不开机器学习、人工智能等技术的支持和发展。这些技术相互结合，形成了一个庞大的程序自动生成生态系统，实现了从代码生成到测试的全流程自动化。人工智能等技术高度发达，能够实现从自然语言描述的任务需求直接生成代码程序，有望让计算机彻底取代程序员的工作，实现在没有人工干预的情况下由计算机完全自主编写程序。未来，随着技术的不断发展，程序自动生成技术将进一步发展和完善，为软件开发和测试带来更多的便利和效率。

2.3 强化学习与 Sarsa 算法

在机器学习领域，有一类重要的任务和人生选择很相似，即序贯决策(Sequential Decision Making)任务。实现序贯决策的机器学习方法就是—强化学习(Reinforcement Learning)。决策和预测任务不同，决策往往会带来“后果”，因此决策者需要为未来负责，在未来的时间点做出进一步的决策。

强化学习的起源可以追溯到心理学家伊万·巴甫洛夫对动物条件反射的研究，控制论和动态规划等学科的兴起为强化学习提供了理论基础，并最终抽象为马尔可夫决策过程。强化学习经过长时间的发展，其理论和应用研究都发展迅速，比如 DQN 网络架构 (Deep Q-Network, DQN)[44]应用于雅达利(Atari)游戏[45]。2016 年，基于深度强化学习的 AlphaGo 击败了人类顶尖职业棋手，引起了全世界的关注[46]。AlphaGo 让普罗大众认识到人工智能，尤其是强化学习的实力和魅力。强化学习适用于具有序惯决策属性的问题，而在求解最优化问题过程中也需要针对不同的约束条件做出不同的选择，这与强化学习的行为选择具有天然的相似性。基于上述强化学习的求解策略，在求解最优化问题时，该方法超越了传统算法，不受人工经验的限制，可以自动发现求解问题的策略，对类似问题进行泛化求解，摆脱了传统算法设计在针对相同结构问题专门设计算法的弊端[47]。

强化学习框架如图 2-1 所示，相比于有监督学习中的“模型”，强化学习中的“智能体”强调机器不但可以感知周围的信息，还可以通过做决策来直接改变环境，而不只是给出一些预测信号。

图 2-1 强化学习基本框架

首先智能体感知当前状态 S_t，从动作空间A中选择动作a_t执行，环境根据智能体做出的动作来反馈相应的奖励r_{t+1}，并转移到新的状态S_{t+1}，智能体根据得到的奖励来调整自身的策略并针对新的状态做出新的决策，强化学习的目标是找到一个最优策略π^*如式（2-1）所示，使得智能体在任意状态和任意时间步骤下，都能够获得最大的长期累积奖赏[48]:

$$\pi^* = argmax_\pi \, E_\pi\{\sum_{k=0}^{\infty} \Upsilon^k r_{t+k} \mid S_t = S\}, \forall S \in S, \forall t \geq 0 \tag{2-1}$$

其中 π 表示智能体的某个策略，$\gamma \in [0,1]$为折扣率，k 为未来时间步骤，S 为状态空间[49]。

在强化学习多年的发展过程之中人们提出了许多具有代表性的算法，其中以 Q-learning 算法和 Sarsa(State-Action-Reward-State-Action)算法[50]为代表的算法最为典型。Q-learning 算法使用了一个称为 Q-table 的表格来存储不同状态和动作之间的行为价值Q值，其中Q值表示在特定状态下采取某种动作所获得的预期回报。Q-learning 算法采用的是异策略更新方式，异策略是指行为策略和评估策略不是同一个策略，即所做的与所说的并不一定对应。Q-learning 算法使用下一动作值函数的最大值来更新Q表的策略，而选取动作的策略则是是贪婪策略。Sarsa 算法同样将<状态-动作>价值函数存储于 Q-table 之中，其中 Sarsa 算法的行为决策部分和 Q-learning 相同，也是采用 ε-*greedy* 贪婪策略。不同之处在于 Sarsa 的更新方式是不一样的，Sarsa 是同策略的更新方式，它的行为策略和评估策略都是 ε-*greedy* 策略即说到做到。强化学习中的智能体 agent 需要做两件事：一是选择 action；二是学习 learning。在智能体选择一个动作后再与环境进行不断的交互，得到了一个由动作价值函数构成的一张 Q-table 表，用来存储所有的动作价值函数的Q值，通过不断的更新这张收益表并最终达到最大收益。强化学习算法中 Q-table 的更新方式可以简化为如图 2-2 所示：

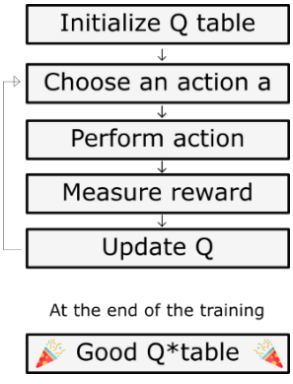

图 2-2 Q-table 更新流程图

以 Sarsa 算法的 Q-table 的更新方程为例，如式（2-2）所示：

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{2-2}$$

上述方程表示当前智能体在状态 s 下选取动作 a 后会得到一个奖赏 R，并会产生一个收益期望，用Q来表示。智能体的学习速度由参数 α 进行控制，获得奖赏的大小由折扣因子γ来进行控制，用来控制智能体在当前以及未来的期望收益中重要程度。

2.4 最优化原理

最优化原理也称最优性原理，指解决多阶段决策问题的理论。这个理论是美

国的贝尔曼在 1956 年提出的。这个原理指出多阶段决策过程具有这样的性质：一个过程的最优策略具有这样的性质，即无论其初始状态及初始决策如何，其以后诸决策对以第一个决策所形成的状态作为初始状态的过程而言，必须构成最优策略。简言之，一个最优策略的子策略，对于它的初态和终态而言也必是最优的。一个最优化问题的最优解可以通过分解为若干子问题，并且每个子问题都有最优解的方式来求得。这样的分解可以大大简化问题的复杂度，并且提高求解效率。

最优化原理使用数学语言来描述为如下形式：假设为了解决某一优化问题，需要依次作出 N 个决策$D_1, D_2, \cdots, D_n$，如若这个决策序列是最优的，对于任何一个整数$k, 1 < k < n$，不论前面 k 个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策$D_{k+1}, D_{k+2}, \cdots, D_n$也是最优的。

最优化原理是动态规划的基础。任何一个问题，如果失去了这个最优化原理的支持，就不可能用动态规划方法计算。能采用动态规划求解的问题都需要满足一定的条件：

(1) 问题中的状态必须满足最优化原理；

(2) 问题中的状态必须满足无后效性。

所谓的无后效性是指：“下一时刻的状态只与当前状态有关，而和当前状态之前的状态无关，当前的状态是对以往决策的总结”。

最优化原理在实际问题中有着广泛的应用，例如在工程设计中优化设计参数、在金融中优化投资组合、在运输和物流中优化配送路径等。同时，最优化原理也是许多高级数学和工程科学课程的重要内容，对于理解和应用最优化方法具有重要意义。

2.4.1 求解强化学习中的最优化原理

在上述 2.3 小节中介绍了强化学习的数学原理是基于序惯决策的马尔可夫决策过程，其也具有最优化原理的性质，求解强化学习意味着要寻找一个符合贝尔曼方程的最优策略，即最优性原理。

（1）贝尔曼方程

贝尔曼方程表示当前时刻状态的价值$v(s_t)$和下一时刻状态的价值$v(s_{t+1})$之间的关系。状态价值函数和动作值函数都可以使用贝尔曼方程来表示。

对于值函数来说，可以分为两个部分：第一个部分为即时奖励R_t，另一个部分为未来状态的折扣价值$\gamma v(s_{t+1})$。

$$v(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma v(s_{t+1}) | s_t = s] \tag{2-3}$$

因此，状态价值函数的贝尔曼方程为：

$$v(s) = R_s + \gamma \sum_{s' \epsilon S} P_{ss'} v(s') \tag{2-4}$$

式（2-4）称为马尔可夫奖励过程中的贝尔曼方程，它表明一个状态的价值由两部分组成：一部分是该状态的即时奖励，另一部分与后续所有可能的状态价值、对应的状态转移概率以及衰减系数相关。

（2）最优值函数

解决强化学习问题意味着要寻找一个最优的策略（让个体在与环境的交互过程中获得始终比其他策略都要多的收获），这个最优策略用π^*表示。一旦找到最优策略π^*，就意味着该强化学习问题得到了解决。寻找最优策略是一件比较困难的事情，但是可以通过比较两个不同的策略的优劣来确定一个较好的策略。

最优状态值函数可以用 v^*来表示，具体如下：

$$\pi^*(s) \to v^*(s) = \max_{\pi} v(s) \tag{2-5}$$

最优状态值函数$v^*(s)$表示在所有可能的策略中，选择能使状态s的价值最大的策略所对应的状态价值函数。在求解过程中，最优状态值函数对应的最优策略是智能体所能选择的使得价值最大化的动作序列。也就是说，最优状态值函数所表示的是在最优策略下，智能体所能达到的最大价值。

同理，最优动作值函数$q^*(s,a)$指的是所有策略下产生的众多动作价值函数中的最大者，具体如下：

$$q^*(s,a) = \max_{\pi} q_\pi(s,a) \tag{2-6}$$

最优值函数的确定，能够确保智能体在马尔可夫决策过程中能够实现最佳可能的表现。当我们获得最优值函数时，就能得到每个状态的最佳价值，从而使得马尔可夫决策过程的所有变量都成为已知量。这样一来，我们就能够很好地解决马尔可夫决策过程的问题了。

（3）最优策略

最优策略(Optimal Policy)的定义如下：

$$\pi \geq \pi' \; if \; v_\pi(s) \geq v_{\pi'}(s) \tag{2-7}$$

在状态s下，当策略π的价值函数优于任何其他策略π'的价值函数时，策略π即为状态s下的最优策略。关于马尔可夫决策过程的最优策略，有如下三个定理。

定理 1：对于任何马尔可夫决策过程问题，可能存在不止一个优化策略解，但一定存在一个最优策略π^*，其好于（至少相等于）任何其他策略，即$\pi^* \geq \pi$。

定理 2：所有最优策略下都有最优状态值函数，即$v_{\pi^*}(s) = v^*(s)$。

定理 3：所有最优策略下都有最优动作值函数，即$q_{\pi^*}(s,a) = q^*(s,a)$。

基于上述 3 个定理，寻找最优策略可以通过最优状态值函数$v_{\pi^*}(s)$或者最优动作值函数$q_{\pi^*}(s,a)$来得到。也就是说如果最优值函数已知，则可以通过获得马尔可夫决策过程的最优策略。

因此，可以通过最大化$q^*(s,a)$得到最优策略π^*，其具体定义如下：

$$\pi^*(a|s)=\begin{cases}1, & a=max\,q(s,a)\\ 0, & 其他\end{cases} \tag{2-8}$$

式（2-8）表示，在最优行为价值函数已知时，在某一状态s下，对于行为集里的每一个行为 a 将对应一个最优行为价值$q^*(s,a)$，最优策略$\pi^*(a|s)$将给予所有最优行为价值中的最大值对应的行为以 100%的概率，而其他行为被选择的概率为 0，也就是说最优策略在面对每一个状态时将总是选择能够带来最大、最优行为价值的行为。这同时意味着，一旦得到$q^*(s,a)$，最优策略也就找到了。

至此，最优策略的求解问题事实上已经转化为最优值函数的求解问题。如果已经求出最优值函数，那么最优策略是非常容易得到的，反之同理。通过最优策略求解问题的转换，可以将孤立的最优策略π^*、最优值函数$v^*(s)$、最优动作值函数$q^*(s,a)$联为一体。但在实际过程中，也可以不求最优值函数，而使用其他方法直接求解最优策略，如时间差分法中的 Sarsa 算法和 Q-learning 算法等，在第四章节系统强化学习核心算法设计中有介绍。

2.5 基于 PAR 方法的算法程序自动生成途径

符号主义追求的是如同数学定理般的算法规则，试图将人的思想、行动活动及其结果，抽象化为简洁深入而又包罗万象的规则定理。基于符号主义的程序生成专家系统主要利用一系列从特定专业知识中推导出的逻辑规则来回答或解决某一特定领域的问题，从而避免了大量的常识性问题[51]。由于其特定性质，专家系统仅适用于一个专业细分的知识领域。正由于其仅仅局限于某些特定情景，知识采集难度大、使用难度大导致其在别的领域未取得很好的成果。

符号主义靠人工赋予机器智能，连接主义靠机器自行习得智能，行为主义则通过与环境的感知反馈中获得智能，强化学习这一模拟智能的训练方法在连接主义和行为主义中都可以使用。强化学习使用“感知-动作”行为模式这一模式来学习智能行为，与其“交互试错”的学习策略来模仿人类的学习过程和决策推理过程，非常契合人类的行为心理学。

PAR 方法是从问题规约出发，对原始问题进行形式化规约的描述，通过形式化的量词规约变换规则对算法程序进行开发和验证的方法。PAR 平台中的 Radl 规约→Radl 算法程序生成过程还处于手动/半自动的开发过程中，主要原因是由于涉及到脑力创造性工作，其主要还是依赖于一般算法开发的思想过程及其涉及到人的先验知识，这是阻碍大部算法问题难以实现算法设计模式复用主要原因。在 PAR 方法中将需要人的脑力活动进行程序设计的工作定义为非机械性活动，反之在不需要脑力活动仅依靠计算机自动完成程序设计的活动定义为机械性活动。二者之间可以通过不断发展的新理论和新方法，在强化学习这种“感知-动作”模式下不断的实践、创新和尝试将机械性活动与非机械性活动慢慢融合起来，

最终实现一个完全可由机器进行的机械性开发活动，减少人对机器的干预。

虽然行为主义的强化学习在其特有的感知和模仿学习上取得了一定的类人智能程度，但其在解决 Radl 规约到 Radl 算法程序自动生成的过程中还存在着一定的缺陷。算法程序设计过程需要的求解空间状态非常大，算法生成策略纷繁复杂，在没有统一的算法设计方法的指导之下，强化学习的算法在其“交互-试错”的学习机制上也不一定能取得较好的效果。

综上所述，需要一种将 PAR 方法算法设计方法融合到强化学习策略中去的算法程序生成方法。通过融合三大主义学派的优点，将连接主义的“大脑”安装在行为主义的“身体上”，使机器能够在对环境做出本能的反应同时，还能够思考推理，才有可能将算法自动生成过程中的非机械性劳动转换为机械性劳动，简化算法程序设计过程，同时具备一定的可靠性、可验证性，并体现出其中的可复用的算法程序设计理论这一思想。

第 3 章　算法设计语言 Radl 与最优化问题算法设计理论

3.1 Radl 语言概述和总体结构

Radl 算法设计语言[52]由算法规约和程序规约组成，使用形式化方法的思想来描述算法和程序。Radl 可用来描述形式化推导的算法和程序规约及其之间的变换规则，以及描述 PAR 中的算法程序。Radl 语言具有与数学语言一样的逻辑推理和透明性，基于数理逻辑的可推导性使得用 Radl 描述的算法同样可以进行形式化的推导。Radl 语言充分考虑了各程序语言之间的共性，所包含的数据类型和程序处理机制与高级程序设计语言并无太大差异。Radl 中预定义的函数、数据结构和自定义结构在 PAR 平台中都已实现了构件化，可以像传统的标准数据类型一样使用。Radl 中的符号和表达式与 C、C++和 Java 等编程语言类型相似，使用高级语言的编程思维，提高了通用性从而降低了语言使用难度。结合形式化方法理论，依据数理逻辑进行算法设计，使得算法设计以及形式化验证过程的效率和正确性得到提升。

使用 Radl 语言描述的 Radl 算法程序能够确保算法的推导过程和逻辑开发过程的一致性，充分展现算法的核心思想。这种方式可以使得算法的描述简明扼要、逻辑严密、易于理解和开发，并且还支持形式化验证，从而保证算法的可靠性。使用形式化的算法描述语言 Radl 可以更好的去除自然语言中信息描述的二义性，使得算法描述更加精准[53]。Radl 语言描述的算法程序更加注重算法程序本身，突出算法程序的特性和思想，减少其他信息不必要的干扰，尽可能的保持算法程序的原有状态，展现其解决问题的能力，方法、步骤，而不过多的描述与算法程序不相干的细节[54]。

Radl 语言由两部分组成：算法规约语言和算法设计语言[55-57]。由此，Radl 既可以用来描述算法的规约，也可以用来描述算法程序。在本文中，简单介绍 Radl 语言的结构，详细资料可查阅文献[52]。

Radl 规约主要由三部分组成：变量标识符说明、前置断言、后置断言。

Radl 规约结构如下：

Algorithm:<算法名称>

|[变量声明：in:输入变量；out:输出变量；aux:辅助变量]|

{Q:前置断言：程序初始参数的条件}

{R:后置断言：程序求解之后满足的条件}

在 PAR 方法的指导下，一般算法都可以递推关系来表示，具体结构如下：

Radl 算法结构如下：

Algorithm: <算法名称>
{<算法规约>}
|[变量声明：in:输入变量；out:输出变量；aux:辅助变量]|
Begin: <变量条件初始化>
Termination: <递推终止条件>
A_I: <算法不变式>
Recur: <递推关系组>
End

3.2 Radl 语言的数据类型

Radl 语言在设计之初就考虑到了语言的应用性，提供了基础的标准数据类型，自定义简单类型（记录类型、数组类型、枚举类型）和预定义的 ADT 数据结构，如算法程序中常用的数据结构等类型，还引入了传统数学中的符号和数学表达式。

Radl 中的标准数据类型（含运算符）如下：

1. integer　　整形　　+，-，*，mod，div
2. real　　实型　　+，-，*，/
3. boolean　　布尔型　　∧，∨，¬，=(等价)，Cond，Cor
4. char　　字符型
5. 通用关系运算符　　≠，<，>，=，≤，≥

下面简要介绍一下表(List)和图(Graph)这两种泛化 ADT 的数据类型，以及它们的符号集合操作集的含义。

3.2.1 表(List)类型

只能存放相同数据类型的一组有序的元素序列可以把它称之为表，用一前一后两个变量 h 和 t 代表序列中头和尾的位置，并指向该位置所在的元素。

表(list)类型具体实例如下：

例 1：type list=list(integer, 20)　　//定义了一个可存 20 个整形数据的元素类型

var A:list;　　//A 为 list 类型，表中元素的个数仅依赖于机器内存的大小

var A:list(integer)　　//A 为 list 类型，表中元素的个数仅依赖于机器内存的大小

例 2：type list=list(char,20)　　//定义了一个可存 20 个字符型数据的元素类型

List 数据类型的程序运算逻辑符号表达式，以及符号集的具体说明和部分描

述如表 3-1 所示

表 3-1 符号集

符号集：	描述
[x]	表示一个元素的序列
[]	表示为空序列
S[i]	表示 S 中第 i 号元素，h≤i≤t

list 类型操作集的使用和具体描述，如表 3-2 所示：

表 3-2 操作集

操作：	描述
#(S)	计算序列 S 中所包含元素的个数
S[i..j]	产生 S 的子序列，h≤i≤t
S↑R	让序列 S 的尾和 R 的头连接合成一新序列
S[i]←→s[j]	让 S[i]和 S[j]交换
S=R	若 S 和 R 完全相同取值为 True，否则为 False
S[i..j]:=R[k..k+j-i]	S[i..j]被 R[k..k+j-i]替换
S:=R	S 被 R 替换，若 R 为空，则 S 被置为空
Choose(x, S, C)	从 S 中选取满足条件 C 的元素存入 x 中

3.2.2 图(Graph)类型

无向图是一个没有箭头和方向的图，其中的边是无序的连接两个顶点的线段，表示两个顶点之间的关系是相互的、对等的。在无向图中，两个顶点之间如果有边相连，则它们之间是互相可达的。由每条带权重的边集构成的有向图可看作二元组，即$G=(V,E/W)$，其中V为G的顶点集，E为边集，W为每条边的权重，E/W是带有权重边所构成的集合，即$E/W=\{(u,v)/w|(u,v)\in E,w\in W\}$。让$in(a)$表示某一顶点 a 的入边集，$out(a)$表示顶点 a 的出边集。对于无向图，其任意顶点 a 的入边集和出边集相等。于是有向图类型可定义为：

```
Type digraph=digraph((datatype,[size1]),(datatype,[size2]))
                =record
                V: set(vertex, [size1]);
                E: set(edge, [size2]);
End;
```

图数据类型实例如下：

```
Var G, G1: digraph;
    v, u: vertex;        //vertex 为自定义 record 类型
    e, e1: edge;         //edge 为自定义 record 类型
```

图数据类型符号集的具体说明，部分描述如表 3-3 所示：

表 3-3 符号集

符号	描述
%	空图

图数据类型操作集的使用和具体描述，如表 3-4 所示：

表 3-4 操作集

操作：	描述
e.w	边 e 的权值，e.w 的值可用赋值语句改变
e.h	e 的头顶点
e.t	e 的尾顶点
e.f	若 f 为 1，e 已被访问；f 为 0，e 尚未被访问 v.f 的值可用赋值语句改变
v.d	顶点 v 的值或标号，可用赋值语句改变
v.f	若 f 为 1，v 已被访问；f 为 0，v 尚未被访问 v.f 的值可用赋值语句改变
v=u	判断两顶点是否完全相同
v:=u	顶点 v（可以是 e.h 或 e.t）被 u 替换
e1=e	判断两条边是否完全相同
e1:=e	边 e1 被 e 替换
G.V	G 的顶点集
G.E	G 的边集
G + v	在 G 中加一节点 v
G + e	在 G 中给定结点间加边 e
Length(p)	图 G 的一条路径 P 的长度

3.3 Radl 语言的应用实例

下面举例一个 Radl 的简单示例。

例：给定一个以及排序好的一维数组 a[n]，请使用二分查找法查找给定元素 key 所在数组的位置，若无则输出 *n*。

根据上述问题的描述，结合 Radl 规约语言的描述规则和问题中的算法思想可得出该问题的规约，用 Radl 语言表示为：

```
Algorithm: BinarySearch;
|[ in n:integer; out p, a :array[0,..n-1];aux m, l, i, j:integer;]|
{Q: ∀k: 0≤k<n: a(k)≤a(k+1)}
{R: p(a,0,n)=(( ∃k:0≤k<n: a(k)=key)→(i=k)∨(i=n))}
```

对于上述规约，我们可以基于 PAR 方法的几个算法开发步骤对上述问题进行求解，通过 PAR 方法的分划递推思想对问题进行不断的拆分以及求解，最终可以求得满足算法解所需的递推关系组，最后基于递推关系就可以设计出该问题的 Radl 算法，经 PAR 平台生成的算法如下：

```
Algorithm: BinarySearch;
|[ in n:integer; out p, a :array[0,..n-1];aux m, l, i, j:integer;]|
Begin: m, l:=0,n-1;
Termination: m≥1;
A_I:
Recur:
    j=(m+1)/2;
    p(a, m, l)=(i=j if a(j)=key);
    p(a,m,j-1),if a(j)<key;
    p(a,j+1,l), if a(j)>key)∨(i=n);
End
```

上述 Radl 算法案例很好的展现了 PAR 方法以及 PAR 平台在算法设计这一过程中具有的优势，其充分体现了严格的数学逻辑思维，展现了算法的逻辑透明性。Radl 语言是辅助 PAR 方法进行算法程序设计的前置语言，所以其重要性不可或缺。Radl 主要用于描述问题的规约以及进行算法的设计。Radl 语言通过其极为严格的形式化描述使得其在去除自然语言描述算法的二义性、以及算法的处理逻辑性上都特别成功。相较于其他计算机程序设计语言，Radl 语言具有与数学语言一样的逻辑推理和透明性，基于数理逻辑的可推导性使得用 Radl 描述的算法同样可以进行形式化的推导，更面向数学运算理念。

3.4 最优化问题算法设计理论和泛型算法结构

在面对未知算法解的问题求解过程时，可供使用和参考的算法程序有许多，但这些算法程序之间往往缺乏一丝联系以至于它们都是孤立的存在着。对于尚未知道算法解的问题以及不同的算法程序问题，需要在考虑诸多的算法求解方法后才能针对具体的问题采用具体的策略来开发程序[58]，这就导致在面对未知算法解的问题时就需要去重新设计算法，而且由于缺乏统一的算法求解方法和算法框架，也导致现阶段各式各样的算法设计方法无法适用相同类型问题之间的程序生成的要求。事实上，相同类型的问题之间一般存在一些共性，而解决同一类型问题的不通算法之间也存在着一些相似的共性，如果能将解决同一类型问题的不同算法程序之间的这些共性抽象化出来形成统一的算法设计思想和算法框架结构，便可以在更高的抽象层次间实现一定的算法重用，从而提高算法程序设计的自动化程度[59-61]。为了求解未知算法解的问题领域中的图最优化问题的算法程序生成自动化的这一需求，结合统一的开发高效算法程序的算法设计方法 PAR 方法，提出了解决部分特殊情况下的图最优化问题的一种算法设计理论和泛型算法结构框架，并以此理论运用到具体问题的算法设计实例中去。

3.4.1 最优化问题算法设计理论

最优化问题将实际生活中待解决的问题，通过函数分划从而转化为求解这些目标函数问题的最值问题，一般情况下问题具有 n 个输入，问题的解是这 n 个输入的某些子集的集合，而且这些子集一般情况下需要满足一些限制条件。满足这些限制条件的子集称为可行解，这些解可能存在一个或者多个，或者也可以没有可行解。而找到其中满足约束条件下取得最值的解，可称为最优解。描述最优化问题我们需要借助数学逻辑表达，其求解模型公式如下：

$$\min f(X)\ or \max f(X), X \in D \tag{3-1}$$

1. D 是问题的解空间。
2. X 是 D 中的一个可行解，由多个可行解构成的集合$X = \{x_1, x_2, \dots, x_n\}$，表示一组满足问题限制条件的决策变量。
3. 函数$f(x)$称为目标函数，或者代价函数。在所有的可行解 X 中，找到一个在解空间 D 中的最小化（或者最大化）目标函数$f(x)$的可行解被称为最优解。

当前存在许多没有算法解的问题，以图类型问题中的图最优优化问题居多，为解决这一类可能存在没有算法解的图最优化问题程序生成这一问题，需要将 PAR 方法的算法求解思想与强化学习这一由人类教会机器学习的问题求解方式与之相结合，去探索未知算法解问题的求解方式。此理论基于 PAR 方法，针对图最优化问题这一领域进行求解，其核心思想是将待求解的问题划分为 N 个子

问题P_{n-1}和 P_n（$P_i \subseteq P(A), i = 1,2,\dots,n$，A 是问题的输入集），现假定$P_{n-1}$的解已求出，考虑加入$P_n$后问题解的变化。同理，求解$P_{n-1}$时假定$P_{n-2}$的解已求出，再判定$P_{n-1}$加入后问题解的变化。根据问题的结构，通过连续添加问题节点来构造可行解，并保持可行解，以满足问题的约束。以此类推，通过自顶向下将原问题细化为更小的子问题，直到不能再进一步细化为止，问题分划过程就此结束。再通过归并步骤依据问题的递推关系式自底向上逐步求解问题$P_1, P_2, \dots P_{n-1}, P_n$的解，直至得到问题的全局最优解。具体问题求解步骤可分为以下三步：

1.采用合适的方式将问题分划为 *N* 个子问题，直至问题无法分划。

2.探寻各个子问题之间递推关系，并求解子问题的最优解。

3.根据递推关系自底向上逐步构造满足问题解的算法程序。

3.4.2 最优化问题泛型算法结构

通过对问题的描述和若干图最优化问题的研究，结合上述针对图最优化问题的算法设计理论，可总结出解决图最优化问题的一种高效泛型算法结构，利用该结构可实现图最优化问题算法程序的自主探索生成。基于 Radl 语言来描述该泛型算法结构，具体算法程序结构如下：

```
algorithm: Algorithmname
recurfunc solution1(k): datatype;
recurfunc solution2(k): datatype;
function decompose(S,V):datatype, datatype;
|[in V:datatype; k:datatype;out result :datatype;]|
begin: k=somevalue--1; solution1(0)=somevalue; solution2(0)=somevalue;
termination: k=0;
recur:
    solution2 (V_k)= decompose (solution2 (V_{k-1})+ V_k, V_k);
    solution1 (V_k)= decompose (solution1 (V_{k-1}), solution2 (V_k));
end
```

上述算法程序中各个变量所代表的含义如下：algorithm 标识符后的字符表明 Algorithmname 代指的是算法的具体名称，其经由系统词法语法分析输入的问题 Radl 规约后可知；in 标识符代表算法所需的输入变量参数；out 标识符代表算法求解输出的变量参数；aux 标识符表示算法求解过程中需要用到的临时变量参数；上述输入输出等参数经由系统分析输入的问题 Radl 规约后可知；begin 关键词用于初始化算法中各个变量，若需要控制递推关系语句组中的变量以及循环的终止条件和步长，则可以在 begin 语句出进行初始化定义；这里的 solution1 和 solution2 作为问题初始的已知条件，在每次分划后问题 P 的规模会自动减少一项；termination 标识符后的字符是算法递推关系的结束条件；Recur 标识符后的

字符是算法的核心部分即递推语句关系组，表示问题的分化递推关系结果。

3.4.3 最优化问题算法设计理论的应用实例

下面将通过一个简单实例来说最优化问题算法设计理论的可行性与有效性。

单源最短路径问题描述如下：给定一个图$G(V,E)$，确定从源点s到其它节点t的最短路径长度，其中$s \in V$。

此问题的 Radl 规约如下：

```
algorithm:singleshortpath
|[in n:integer;s:NODE;G(V,E):Graph;out mindis:array[0..n-1:real];aux p:real;
t:NODE]|
{Q: n≥0}
{R: mindis[t]=MINt:t∈V:( ∀p:p∈path(s,t):length(p))}
```

常见的解决此问题的方法为贪心法，先计算与当前相邻各个节点之间的路径距离，然后从中选取最小者。采用贪心解决此问题的具体 Radl 算法如下：

```
algorithm:singleshortpath
|[in n:integer;s:NODE;G(V,E):Graph;out mindis:array[0..n-1:real];aux i,j,u,v:
integer; min,p:real; book: array[0..n-1:boolean];t: NODE]|
begin:i=1++1;
termination:i<n;
a_i:
recur:
    min= 0x3f3f3f3f;
    begin:j=1++1;
    termination:j≤n;
    recur:
        if：book[j]=0∧mindis[j]<min→min=mindis[j];u=j; fi
    end
    book[u]=1;
    begin:v=1++1;
    termination:v≤n;
    recur:
        p=length(path(u,v));
        if p<min→if mindis[v]>mindis[u]+p→mindis[v]=mindis[u]+p; fi
    end
end
```

不难看出上述算法的计算复杂度为$O(n^2)$，所需要花费的运算时间较高，算法复杂度较高。为了降低算法的复杂度得到更高效的算法，在此运用基于 PAR 方法的最优化问题的算法设计理论对该问题进行求解，与原有方法进行比较。基于该理论对上述问题进行求解过程如图 3-1 所示，图中展示了对问题的分划过程，以及结合 PAR 方法与强化学习思想的图最优化问题算法设计理论的求解思想。

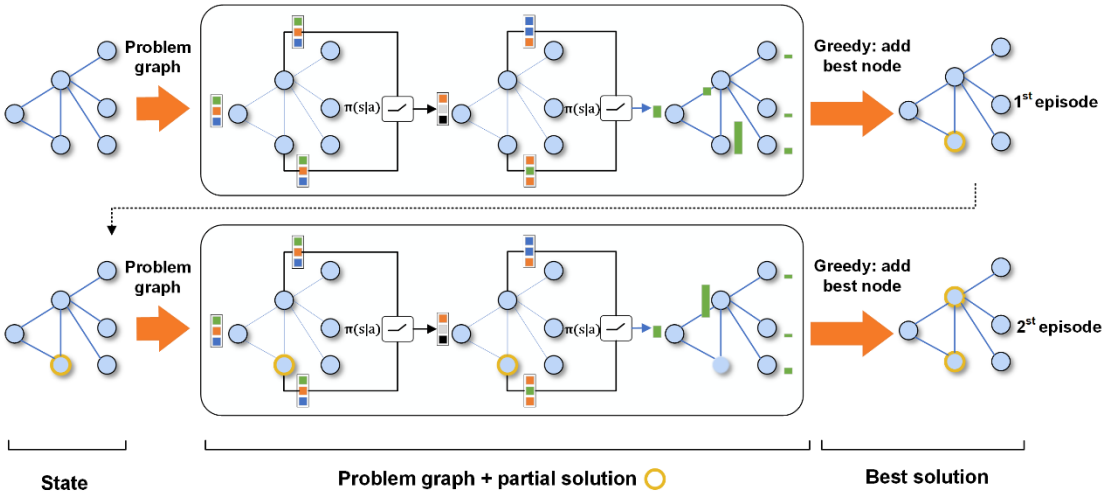

图 3-1　图最优化问题算法设计理论求解最短路径问题

采用上述提出的算法设计理论解决此问题的具体 Radl 算法如下：

algorithm: singleshortpath
recurfunc solution1(k): integer;
recurfunc solution2(k): integer;
function decompose(S,V):node, node;
|[in n:integer; S:NODE; G(V,E):Graph; k:integer; out result :integer;]|
begin: k=n--1; solution1(0)=0; solution2(0)=0;
termination: k=0;
recur:
　　solution2 (k)= decompose (solution2 (V_{k-1}), V_k);
　　solution1 (k)= decompose (solution1 (V_{k-1})+ V_{k-1}, solution2 (V_k));
end

具体解题步骤如下：

1.给定一个有权图$G(V,E,w)$，V 为顶点集，E 为边集，w 为边的权重，其可代表距离，长度等属性。现假定节点候选集合 S 的大小为 n，设前 n-1 个节点组成的集合及其子集合的与初始节点之间的最短路径为已知。

2.将第 n 个节点添加到解中，比较前 n-1 个节点组成的图及其子节点组成的图的最短路径和只包含 n 个节点集合的最短路径。

3.只包含 *n* 个节点集合的图最短路径和可由前 *n-1* 个节点组成的集合加上S_n得到，又因前 *n-1* 个节点组成的集合的最短路径之和和S_n的值都是已知的，所以只包含 *n* 个节点集合的最短路径之和的解便解决了。

4.此时问题便转化为求解前 *n-1* 个节点组成的图及其子节点组成的图的最短路径。再重复前面几个步骤便可以将问题转化为求解前 *n-2* 个节点组成的集合及其子节点集合之间的最短路径。迭代若干次之后，问题最终转化为求解前 1 个节点组成的集合及其剩余子节点集合的最短路径，此时可以求得节点S_0的权值即为所求的最短路径之和，通过之前迭代建立好的递推关系，再反向回溯回去，就能得到整个图中各个节点之间的最短路径之和。

从上述算法开发步骤不难看出，采用图类型最优化问题的算法设计理论，在实践过程中比原有的算法设计方法更具有统一性，避免了要选择何种算法设计方法的困恼，以此为基础设计出来的算法程序在计算复杂性上相比原解法也具有一定的优势。

采用传统贪心算法设计的最短路径问题求解算法与基于 PAR 方法的算法设计理论设计的高效算法相比较可以发现，该算法的计算复杂度小于传统算法。并且算法是经由形式化算法描述语言 Radl 开发的，其可靠性和正确性都可以得到保障。