arXiv:2601.14212v1 [cs.NE] 20 Jan 2026

随机局部搜索算法的泛化与完备性*

Daniel Loscos、Narciso Martí-Oliet 和 Ismael Rodríguez †

摘要

我们将随机局部搜索（SLS）启发式算法泛化为一种统一的形式化模型。该模型包含两个关键组件：一个旨在尽可能扩大的通用结构，以及一个旨在尽可能缩小的参数化结构。每种启发式算法都通过以不同方式实例化参数化部分获得。文中给出了遗传算法（GA）、蚁群优化（ACO）和粒子群优化（PSO）的具体实例。随后，我们利用该模型证明了 SLS 算法总体上的图灵完备性。证明采用我们的框架构造出一种能够模拟任意图灵机的遗传算法。这种图灵完备性意味着，对于遗传算法而言，确定任何关于输入与计算输出之间关系的非平凡性质都是不可判定的；进一步而言，对于 SLS 方法的整体集合也是如此（尽管对其中每一种具体方法未必如此）。对于 PSO 和 ACO，文中还以较为非正式的方式给出了类似证明。

关键词： 随机局部搜索、进化计算、群体智能、形式语言、操作语义、泛化、可计算性、图灵完备性。

1 引言

解决优化问题几乎是所有工业和科学领域的必然需求，但对于其中许多问题，计算上难以求得最优解。因此，我们更关注如何通过启发式算法，为典型的问题实例寻找合理的次优解。随机局部搜索（SLS）算法 [1], 以及尤为值得关注的算法家族

*本文发表于《Swarm and Evolutionary Computation》。此版本为作者接受的稿件。研究部分由项目 TIN2015-67522-C3-3-R、PID2019-108528RB-C22，以及马德里自治区 S2018/TCS-4339（BLOQUES-CM）计划资助；该计划由欧盟 EIE 基金共同资助。

†计算机系统与计算系，信息学院，马德里康普顿斯大学，西班牙马德里 28040。Narciso Martí-Oliet 和 Ismael Rodríguez 也隶属于知识技术研究所。电子邮箱：{dloscos,narciso,isrodrig}@ucm.es

例如进化和群体优化方法 [2, 3], ，都是解决许多优化问题的不错选择。在这些方法中，一组简单的实体依据简单规则彼此交互，在迭代过程中不断变化，并协同构建和改进所研究问题的解。例如，在遗传算法 [4],中，染色体（实体）相互交叉，自行构成不断演化的解；而在蚁群优化 [5, 6] 中，一些蚂蚁（智能体）协同在图上构建候选解（一条路径或一组路径），同时反复遍历该图。SLS启发式算法总体而言，尤其是进化算法和群体算法，已应用于各种各样的领域（例如，参见 [7, 8, 9, 10, 11]）。

近年来，该领域见证了大量新方法的涌现；在某些情况下，这些方法的主要动机是试图复现某种自然过程的美感，而不是展现真正不同的行为，并在搜索过程中保证获得新的、有趣的数学性质 [12]。鉴于每种方法都允许使用数量庞大的参数和备选步骤，让一个新算法在已知基准测试上取得良好解（或其中相当大的一部分），往往取决于使用强大的参数调优器 [13, 14, 15] 针对特定基准测试对其进行优化，直到达到其最佳、甚至是过拟合的结果——或者甚至仅仅取决于设计者经过长时间的反复试错，直到它运行良好（有时甚至是偶然奏效，有时只是因为所做的修改使其大致等价于其他此前已知的优秀算法） [16]。许多新提出的方法似乎只需对某些算法步骤进行一些小改动，就能与其他方法等价，这让人感觉我们谈论的与其说是一组概念上不同、值得拥有不同名称的算法，不如说是某种连续的算法空间。

为了梳理这一领域，人们提出了一些基于这些算法的形态结构、自然隐喻或基本机制的分类（例如参见 [17, 1, 18]），也提出了一些非正式定义的概括性框架，只涵盖少数特定算法（例如 [19]），并发表了许多描述和比较各种方法的综述（例如 [20, 21]）。或许，最具普适性的框架是广义局部搜索机（GLSM） [22],，它将随机局部搜索策略建模为非确定性有限状态机。该框架阐明了随机局部搜索算法的运行时行为与控制机制，并为混合型和非合作SLS策略提供了建模工具，但它并不完整：它没有表示终止条件、邻域关系，也没有定义搜索空间或解集。此外，作者提出的一些扩展（即针对合作问题和进化问题的扩展）彻底改变了这一框架，而非仅对某些特性稍作修改。

据我们所知，现有文献缺少一种完全形式化且足够通用的定义，用以描述SLS 方法的共性部分与差异部分，使得任何方法都能通过采用该框架并以某种特定方式实例化而得到。这样的框架能够以通用语言定义新算法和现有算法，从而便于对它们进行分类和直接比较。通过使用通用领域语言，新方法可以

通过对其他已知方法进行科学且标准化的变体设计来开发，其性质也能更容易地得到研究，而研究结果则可以无缝地分类并添加到知识库中。

本文提出一种具有固定结构和参数化结构的形式语言，使得据我们所知，所有 SLS 方法都可以通过按照目标方法实例化后者来实现。请注意，由于所有（标准或变体）SLS 方法都是算法，任何编程语言实际上都能让我们实现它们。然而，我们的语言并不追求编程语言那样的通用性，反而力求将通用性和自由度限制在概括随机局部搜索所必需的最低程度，具体而言，就是尽可能扩大共同结构，并尽可能缩小参数化结构。这样一来，该语言旨在使所有基本算法要素清晰可见、彼此独立且易于理解、操作和测试。

为了证明我们的一般形式化模型有助于发现新的非平凡性质，我们将用它来证明一个令人沮丧的元属性：一般而言，不可能检验任何关于 SLS 方法结果的非平凡性质。有人可能会认为，这种不可能性显然源于它们的随机性：由于它们涉及非确定性选择，我们无法确定地预测其结果。然而，我们将表明，它们的不可预测性源于更深层的原因，远非随机性所能解释；即使它们以确定性方式运行（或者，即使我们预先知道执行期间决定其所有随机选择的每一次抛硬币结果），其行为仍然不可预测。具体而言，这意味着无法一般而言 通过统计学来预测它们的行为，因为即使没有随机性，它们仍然不可预测。

揭示 SLS 方法不可预测性的关键，在于证明它们是图灵完备的。赖斯定理 [23, 24] 表明，判断用图灵完备语言编写的程序是否满足一项仅涉及其可观察行为（即其输入与输出之间的关系）的非平凡性质，是不可判定的。[1] 我们的形式化模型将使我们能够证明，SLS 方法通常是图灵完备的，因此赖斯定理同样适用于它们。[2]

有人可能会认为，这类方法具备图灵完备性，只不过是因为我们假定它们是一个过于一般且自由的概念——或者更确切地说，只不过是因为我们用一个过于一般的模型来表示它们。例如，假设我们的 SLS 方法一般模型包含一个后处理步骤，该步骤将特定算法找到的最佳解

[1]所谓非平凡，是指该性质既非所有程序都满足，也非没有任何程序满足。因此，判断程序是否满足诸如“输入 5 时返回 7”“对某个输入返回 7”“计算函数 $f(x) = 2x$”或“对所有输入都停机”等性质，都是不可判定的。[2]请注意，我们的目标与证明 SLS 方法的结果 （即它为所求解的问题找到的解）可以是用图灵完备语言编写的任意程序毫无关系（这正是遗传编程中的理想输出 [25, 26, 27]）。与此完全无关的是，我们将证明，通常 SLS 方法本身是图灵完备的，也就是说，对于任意递归可枚举函数，它们都能够计算该函数。

在算法终止时，连同问题输入一起映射为实际算法应返回给用户的解。假设这个后处理步骤可以是任意算法。那么，图灵完备性就会成为我们可以自由地让该步骤采用任意算法这一事实所带来的平凡结果。[3] 恰恰相反，我们关于图灵完备性的证明将表明，我们根本不需要如此大的自由度——无论模型实际上是否允许这样做。事实上，即使每个涉及的步骤都以极其简单且精简的方式定义，这些算法内在的基本机制仍然会产生图灵完备性。

为了强调模型的一般性并非实现图灵完备性所必需，我们将针对一般模型中代表一种特定 SLS 方法的一个特定实例证明图灵完备性：遗传算法（GA）。对于给定的待模拟图灵机及其相应输入，该 GA 将迭代地构造表示该图灵机在该输入下更长历史（即经过的配置序列）的个体。由于按照图灵机的转换正确地利用后续图灵机步骤扩展该历史会提高个体的适应度，该 GA 必然会倾向于创建此类个体——尽管适应度函数不会嵌入任何图灵机机制，而只会检查子串包含关系。

尽管我们将特别针对一般模型的遗传算法实例展示其图灵完备性，但请注意，一般模型的图灵完备性由其任一实例的图灵完备性所蕴含，因此这确实将证明 SLS总体上具有图灵完备性。这并不意味着所有特定的 SLS 方法都是图灵完备的。基于我们的一般模型，我们还将说明，如何将在遗传算法实例背景下发展出的证明，轻松地调整为适用于其他代表流行 SLS 方法的实例，即粒子群优化（PSO） [28] 和蚁群优化（ACO） [5, 6]。

本文的贡献如下：

(a) 构建首个具有充分一般性的完全形式化模型：通过实例化其参数化元素，该模型能够表示我们所知的任何随机局部搜索方法，同时尽可能扩大模型的通用部分；(b) 使用该形式化模型证明，随机局部搜索方法普遍具有内在的不可预测性（尤其是超越随机性的不可预测性），其根源在于这些方法的机制在最基本层面上具有图灵完备性；(c) 对遗传算法的图灵完备性给出形式化证明，并概述蚁群优化和粒子群优化的非形式化证明。

本文其余部分安排如下。下一节将通过操作语义描述 SLS启发式的一般模型。第 3 节

[3]同样，通过滥用定义任意适应度函数的自由度，适应度函数本身也可以是任意可计算函数。

介绍遗传算法、粒子群优化和蚁群优化的具体模型，以展示一般形式的实例。第 4 节给出遗传算法（进而也是 SLS）图灵完备性的证明；第 5 节则非正式地讨论如何证明 SLS 其他实例的图灵完备性。第 6 节总结全文并介绍未来的研究方向。

2 随机局部搜索的一般形式

为了抽象出随机局部搜索策略的共同结构，本节设计了一种语言和一组操作语义，构成 SLS 的一般形式 。我们认为，遗传算法、进化策略、蚁群优化、迭代局部搜索、随机梯度下降、模拟退火、粒子群优化，以及我们所知的所有其他随机局部搜索算法，都是这种一般操作语义的实例。

关于如何形式化语义的决策，将主要借鉴 Nielson & Nielson [29, 第 12–14 页和第 32-36] 页中**While**的操作语义。稍后将针对不同的具体算法提供更具体的语义，但我们相信，以下语义处于能够概括整个 SLS 计算的最低抽象层次。下文假设，给定的算法A 接收问题p 的一个具体实例，并探索不同的解。计算找到的最佳解存储在名为$Best$的变量中。

2.1 状态与句法类别

计算的**状态** 是一个从变量到值的函数。计算中的相关变量构成以下元组：[4]

$$(Prob, Sols, Best, FPW, Extra) \in \textbf{问题} \times \textbf{解[]} \times \textbf{解} \times \textbf{F/P/W} \times \textbf{Extra}$$

由于我们的目标是推广 SLS 计算模型，因此定义该元组的类别将足够抽象，以适用于任何 SLS 策略。不过，我们将说明每个类别所代表的含义，并简要介绍它们在不同算法中的结构。

状态 不是一个常函数，因为与每个变量相关联的值可能会在计算过程中发生变化。字母 s 表示**状态**的一个实例。我们用 $s[A \mapsto b]$ 表示一个新状态，其中与 A 相关联的值为 b，而其他每个变量的值都与状态 s中的相同。

接下来，我们将介绍 SLS 计算所需的句法类别：

问题 必须能够封装我们的算法需要解决的优化问题。不仅要封装抽象问题（例如非线性优化，

[4]这里没有体现数组索引和循环计数器等其他辅助变量，尽管实现 SLS 算法时也需要这些变量。

旅行商问题[5], 等），还要封装问题的具体实例和参数（例如待优化函数、旅行商问题中带有边权重的图等）。在计算开始时初始化$Prob\in$ **问题** 后，它将确定计算该问题所需的函数如何运行。

解 是表示计算过程中问题可能解的类别。例如，在遗传算法（GA）中，**解** 表示染色体的编码；而在蚁群优化算法中，**解**是用于存储表示蚂蚁所遵循路径的子图的数据类型。**解[]**表示一个实例集合。因此，$Best \in$**解**将是表示计算所找到的最佳解的变量，而解集 $\in$ **解[]** 则表示当前迭代中正在考虑的解集。

最一般的类别是**F**/**P**/**W**，代表适应度／信息素／权重。**F**/**P**/**W** 必须足够通用，以表示引导 SLS 算法进行搜索所需的每个变量；例如，在基于群体的启发式算法中，$FPW \in$ **F**/**P**/**W** 将由每个个体的权重、其线性动量等组成；对于遗传算法（GA），FPW 将存储每个个体的适应度，而在蚁群优化算法中，则存储每条路径的信息素。

最后，我们介绍**Extra** 类别。它用于封装定义计算所需辅助函数所需的参数，以及特定算法运行时所需的任何其他变量。例如，遗传算法的交叉率或变异率就属于这一类别。**Extra** 对于实现复杂启发式算法至关重要，因为它是传递计算步骤所产生横向效应的默认方式。在混合方法中，**Extra** 将携带在不同启发式算法之间转换所需的所有数据。函数会读取这些信息并据此运行。

此外，令**T** [29, p.14] 由真值**tt**（表示真）和**ff**（表示假）组成；令 **Pexp** 为用于将问题输入计算的句法类别（一个**Pexp** 实例将被转换为相应的 **问题** 实例以供处理）；令**语句** 为构成 SLS 计算模型的语句（语义块）集合，并令**算法** 为 SLS 算法集合。

我们还将定义一些元变量，用于遍历句法类别中的结构：

p 将遍历输入问题表达式 **Pexp**。
A 将遍历 SLS 算法集合 **Algorithm**。请注意，我们考虑的是抽象算法，它们独立于所运行的实例或当前问题。

最后，S 将遍历语句 **Stm**。此外，S'、S_1 和 S_2 均表示语句。

我们假设**Pexp** 构造式的结构已在其他地方给出，因为它与计算无关。不过，**Stm** 构造式的结构确实相关，现列举如下：

[5]旅行商问题：在一个正权图中，寻找一条恰好经过每个节点一次且总权值最小的路径。

$S := S_1; S_2$ | setProb(p) | 生成 | nextGen | 评估 | stop |compute(p)

我们使用 compute(p) 启动给定算法A 的执行，首先运行 setProb(p)，将问题从**Pexp**转换为**Prob-lem** ，然后调用 generate 生成初始候选解集合，调用 evaluate 对其进行评估，并调用 stop 判断算法是否已完成，或者是否必须通过调用 nextGen 继续迭代；nextGen 会利用获得的 SLS知识生成一组新的解。

以下函数和规则进一步详细说明了这些语句的含义。

2.2 辅助函数

在介绍用于定义计算过程中状态如何变化的辅助函数之前，我们先引入以下符号：令**Space** 为与计算相关的任意空间，变量或句法构造可以在其中取值（例如$\mathbf{F/P/W}$），并令**Space** 不等于**Extra**；那么，我们将**Space*** 定义为 **Space** $\times$**Extra**。

由于对于至少某些随机局部搜索算法，计算的任意步骤以及状态的任意变化都可能涉及**Extra** 的变化，因此我们注意到，以下每个函数也都可以改变**Extra**的值。下面介绍这些函数:

- $\mathbb{C}[\![A]\!]$: **算法** $\longrightarrow$ **Extra**。此函数接收算法A的一个实例，并通过初始化启发式算法所需的每个参数和变量来启动计算。

- $\mathcal{P}[\![p]\!]_s$:**Pexp** $\times$ **状态** $\longrightarrow$ **问题*** 。这是 **Pexp** 的语义函数，将问题从输入形式转换为计算形式，以便将其作为变量值存储。计算的前一个状态用于调整问题，使其适用于正在运行的算法。

- $\mathcal{G}[\![]\!]_s$: **状态** $\longrightarrow$ **解[]***。此函数使用状态 s 给出的 $Prob$值，为解集 生成一个与所存储问题相符的新随机值。

- $\mathcal{N}ext[\![]\!]_s$: **状态** $\longrightarrow$ **解[]*** 。它使用状态 s 给出的 $Prob$、解集和 FPW 的值，随机地计算 解集的新值，并利用算法前几次迭代获得的信息，使所得解与问题相符。

- $\mathcal{A}eval[\![]\!]_s$:**状态** $\longrightarrow$ $\mathbf{F/P/W}$* 。它使用状态 s 给出的 $Prob$、解集 和 FPW 的值，根据存储的问题以及算法先前迭代中获得的信息，计算FPW的新值。

- $\mathcal{B}eval[\![]\!]_s$:**状态** $\longrightarrow$ **解***。它使用状态 s 给出的 $Prob$、解集、$Best$ 和 FPW 的值，计算$Best$的新值；它是迄今为止找到的最佳解。

- $\mathcal{SC}[\![]\!]_s$:**状态** $\longrightarrow$ $\mathbf{T}$ 。这是停止判据函数。它会分析整个状态，并在满足 **tt** 停止判据时返回该值；若未满足，则返回**ff** 。

由于这是对 SLS 计算模型的抽象，不同的 SLS 算法会以不同方式实现这些函数。举例来说，我们来看两个遗传算法：第一个返回最后一代中最优个体的值，第二个则返回所有代中最优的个体。前者的 $\mathcal{B}eval[\![]\!]_s$ 函数会忽略$Best$，而后者则会在每一代都考虑它。同样，算法的参数会直接影响其中大多数函数。

读者还应注意，许多此类函数，尤其是函数$\mathcal{A}eval[\![]\!]_s$，$\mathcal{B}eval[\![]\!]_s$ 和 $\mathcal{SC}[\![]\!]_s$，可以通过守护动作进行重载，例如局部搜索阶段、自动参数修改，以及其他用于改进解的机制。为保持通用性并简化记法，我们的设计中没有使用专门的术语或函数来明确表示守护动作，因为它们可以轻松整合到语言的其他元素中。

2.3 规则

最后，我们以一般形式介绍 SLS 计算模型的规则。执行从初始状态 s 开始，通过输入问题 p 作为参数，执行 [compute] 规则：$\langle$compute(p) $s\rangle$。

初始状态 s 的取值无关紧要，因为它们会被输入问题改变，而且先前的信息不会以任何方式影响计算。

我们采用与定义函数时相同的方式引入一种特殊记法。令 **Var**为 **State** 中除 **Extra**以外的任意变量。我们将 **Var***定义为 **Variable** $\times$**Extra**。

按照操作语义中的惯例，语句的含义由一个包含两类配置的转换系统指定：

- $\langle S, s\rangle$ 表示要从状态 s执行语句 S 。
- s 表示一个终止状态。这些终态通常采用 $s[A \mapsto b]$记法。

从配置 α 到配置 β 的有效转换表示为$\alpha \Rightarrow \beta$。转换规则 $\dfrac{\gamma \Rightarrow \delta}{\alpha \Rightarrow \beta}$ 表示，只有当可以从 γ 有效转换到 δ 时，我们才能从配置 α 转换到配置 β 。

一般形式的规则定义于图 1 中，其作用如下： [comp] 规则允许连接携带状态的指令

[分量^1]	$\dfrac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1; S_2,\ s\rangle \Rightarrow \langle S_1'; S_2,\ s'\rangle}$
[分量^2]	$\dfrac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$
[设置问题]	$\langle \text{setProb(}\ p\ \text{)}\ s\rangle \Rightarrow s[Prob^* \mapsto \mathcal{P}[\![p]\!]_s]$
[生成]	$\langle \text{生成}\ s\rangle \Rightarrow s[Sols^* \mapsto \mathcal{G}[\![]\!]_s]$
[下一代]	$\langle \text{nextGen}\ s\rangle \Rightarrow s[Sols^* \mapsto \mathcal{N}ext[\![]\!]_s]$
[评估]	$\langle \text{评估}\ s\rangle \Rightarrow s[FPW^* \mapsto \mathcal{A}eval[\![]\!]_s, Best^* \mapsto \mathcal{B}eval[\![]\!]_s]$
[stop$^{\text{tt}}$]	$\langle \text{stop}\ s\rangle \Rightarrow s$ 如果 $\mathcal{SC}[\![]\!]_s = \mathbf{tt}$
[stop$^{\text{ff}}$]	$\langle \text{stop}\ s\rangle \Rightarrow \langle \text{nextGen; 评估; stop}\ s\rangle$ 如果 $\mathcal{SC}[\![]\!]_s = \mathbf{ff}$
[compute]	$\dfrac{\langle \text{setProb(}\ p\ \text{)}\ s[Extra \mapsto \mathbb{C}[\![A]\!]]\rangle \Rightarrow s'}{\langle \text{compute(}\ p\ \text{)}\ s\rangle \Rightarrow \langle \text{生成; 评估; stop}\ s'\rangle}$

图 1：一般形式的操作语义。

修改； [set problem] 将要运行的实例加载到计算状态中； [generate], [next generation], 和 [evaluate] 分别运行相应的语句，以生成初始解、下一代，并评估其代的适应度；而 [stop] 规则则根据当前状态下停止函数的评估结果，或者结束计算并返回最终状态，或者触发该过程的下一次迭代。最后， [compute]是我们用于启动计算的第一条规则，其余规则则在展开计算时依次应用。

3 遗传算法、粒子群优化和蚁群优化作为一般形式的实例

本节介绍特定算法（遗传算法、蚁群优化、粒子群优化）的一般形式实例。

3.1 遗传算法

既然已经介绍了一般模型，接下来将基于该模型，为遗传算法提供更详细的语义。现在的目标是展示遗传算法计算的结构，同时使语义具有足够的通用性，以适用于遗传算法的具体实例

（即遗传算法的不同变体）。染色体的表示方式、选择方法、变异率、停止判据以及遗传算法实例的其他具体特性，都不应影响其语义。这种语义具有足够的通用性，能够涵盖遗传算法的退化实例，例如进化策略，其中选择和交叉步骤都是简单的。

添加到该语义中的语句和函数旨在展示，依次修改一组个体对于引导搜索的重要性。遗传算法的记忆存于其种群中，因为在生成下一代后，上一代的适应度值便不再有意义。

定义**状态**的方式与一般形式相同。不过，某些语法类别可以进一步具体化。在此情形下，**解**表示单个染色体的编码，而**解[]** 表示**解**的实例集合。因此，$Best \in$ **解**表示计算过程中发现的适应度最高的染色体，而解集 $\in$ **解[]** 表示当前代，以及计算交叉和变异所需的其他所有解。此外，$FPW \in \mathbf{F}/\mathbf{P}/\mathbf{W}$现在由每个个体的适应度组成（每一代都必须重新计算）。

我们将使用与一般形式相同的元变量，唯一的区别是**语句**构造式的结构：

$S :=$ $S_1; S_2$ | setProb(p) | 生成 | 选择 | 交叉 | 变异 | nextGen | 评估 | `stop` | compute (p)

新增了三个语句（选择、交叉和变异），用于描述此前由 nextGen 用一个步骤抽象表示的过程。这展示了遗传算法中的进化过程：根据适应度偏向选择个体进行交叉，生成新个体，并对种群进行变异。这些语句的含义由以下函数和规则进一步说明。

- $\mathcal{S}[\![]\!]_s$:**状态** $\longrightarrow$ **解[]*** 。此操作使用状态 s 给出的$Prob$、解集和FPW 的值，计算解集 的新值，其中包含为交叉阶段选出的上一代个体。

- $\mathcal{C}[\![]\!]_s$: **状态** $\longrightarrow$ **解[]***。它使用状态 s 给出的$Prob$、解集和FPW 的值，计算解集的新值；该值由对选中个体执行交叉操作的结果与部分原始个体组合而成。

- $\mathcal{M}[\![]\!]_s$:**状态** $\longrightarrow$ **解[]*** 。它使用状态 s 给出的 $Prob$、解集 和FPW 的值，来计算新的 解集 值，其中包含修改（变异）部分个体后仍存在于 解集 中的个体。

由于这是对遗传算法计算模型的抽象，不同的遗传算法会以不同方式实现这些函数。例如，考虑一种采用精英保留策略的遗传算法：在每一代中， $\mathcal{S}[\![]\!]_s$ 会将部分个体选为精英； $\mathcal{C}[\![]\!]_s$ 操作会强制将这些个体放入下一代 解集 中，而 $\mathcal{M}[\![]\!]_s$ 不会修改它们。相反，

[选择]	$\langle$选择 $s\rangle \Rightarrow s[Sols^* \mapsto \mathcal{S}[\![]\!]_s]$
[交叉]	$\langle$交叉 $s\rangle \Rightarrow s[Sols^* \mapsto \mathcal{C}[\![]\!]_s]$
[变异]	$\langle$变异 $s\rangle \Rightarrow s[Sols^* \mapsto \mathcal{M}[\![]\!]_s]$
[下一代]	$\langle$nextGen $s\rangle \Rightarrow \langle$选择；交叉；变异 $s'\rangle$

图 2：遗传算法操作语义的实例化。

在无精英遗传算法中，每个个体都会受到 $\mathcal{C}[\![]\!]_s$ 或$\mathcal{M}[\![]\!]_s$的修改，并且可能不会被 $\mathcal{S}[\![]\!]_s$选中。以下规则中使用的其他函数，其定义与一般形式保持不变。

为了补全语义，我们从一般形式中取出规则并加以修改：重新定义一条规则（[下一代]），并新增三条规则：引入 [选择], [交叉], 和 [变异], 分别表示遗传算法在每一代中执行种群转换的对应同名阶段。一般形式的操作语义所做的修改见图 2。

3.2 蚁群优化

本节将一般形式的操作语义具体化，以表示蚁群优化算法。蚁群优化算法在每一代都会生成一个全新的种群。不过，前几次迭代会影响新一代的生成方式。因此，我们可以认为，蚁群优化的记忆存储在信息素中，而非个体中。

对于此模型，**状态**的定义与一般形式中的定义相同，不过部分句法类别进一步具体规定如下。

问题基本上如一般形式中所述。唯一的不同在于，我们现在处理的是图中的某种最优路由搜索，因此至少需要表示问题的图以及计算所需的其他相关数据。

解是表示所考虑的路径查找问题的可能解的类别，因此它是一种用于存储有序子图的数据类型。这些子图由蚂蚁走过的路径构成。请注意，有时最佳解可能由多只不同蚂蚁走过的路径组合而成。**解[]** 表示**解**的实例集合。因此，$Best \in$**解** 是计算过程中找到的最高效子图，而解集 $\in$ **解[]**则是算法当前迭代中由蚂蚁生成并模拟的子图集合。

$\mathbf{F}/\mathbf{P}/\mathbf{W}$ 的核心是表示先前迭代中蚂蚁留下的信息素。因此，$FPW \in \mathbf{F}/\mathbf{P}/\mathbf{W}$ 将存储最优性

[评估]	$\dfrac{\langle \text{simulate } s\rangle \Rightarrow s'}{\langle \text{评估 } s\rangle \Rightarrow s'[Best^* \mapsto \mathcal{B}eval[\![]\!]_s]}$
[simulate]	$\langle \text{simulate } s\rangle \Rightarrow s[FPW^* \mapsto \mathcal{A}eval[\![]\!]_s]$

图3：蚁群优化操作语义的实例化

值，且存储时间不止一个迭代。此外，它还将表示计算辅助函数所需的任何基于性能的参数。当然，信息素值不必是标量，向量是实现多信息素蚁群优化启发式的自然方式 [30]。

我们将使用与一般形式相同的元变量，唯一的区别在于**语句** 构造式的结构，现在定义如下：

$S := S_1; S_2$ | setProb(p) | 生成 | nextGen | simulate | 评估 |停止 | compute(p)

在这种情况下，唯一新增的语句是 simulate。这样设计是为了凸显适应度评估对于这些方法的重要性。尽管新个体的生成受到信息素偏向，且相对简单，但在对选定路径进行适应度评估（模拟）时，才会释放信息素，为下一代做准备。为了使用这一语句，我们引入 [simulate]规则，并修改 [evaluate] 以纳入模拟步骤。

函数列表与一般形式相比保持不变，但其规格值得注意：

- $\mathcal{N}ext[\![]\!]_s$: **状态** $\longrightarrow$ **解[]*** 。该函数使用状态 s给出的 $Prob$ 和FPW 的值，计算由已存储的问题以及算法此前迭代（首次迭代除外）获取的信息所确定的新值解集 。请注意，解集 对于此函数已不再重要。

- $\mathcal{A}eval[\![]\!]_s$: **状态** $\longrightarrow$ **F**/**P**/**W***。如一般形式中所见，不过在此特定情况下，该函数执行两项不同的任务：计算解集 中每只蚂蚁的适应度，并存储它所释放的信息素。

同样，不同的蚁群算法可能会以不同方式实现这些函数；例如： $\mathcal{B}eval[\![]\!]_s$ 可以将不在 解集中的解（如信息素最密集路径）视为潜在的最佳解。最后，蚁群优化的规则与一般形式的规则相同，只是加入了前文提到的修改以纳入 simulate，如图3所示。

3.3 粒子群优化

本文介绍的 SLS 计算操作语义的最后一种特化形式，是粒子群优化算法的操作语义。

在粒子群优化中，解群体中的个体在搜索空间中移动，试图找到最优解。这个过程与遗传算法类似，因为我们可以将每次移动视为在前一代种群之后、按照一组规则生成新一代解。不过，粒子新位置的计算方式在结构上与遗传算法不同。同样，粒子群优化的记忆主要存在于个体中，而不太依赖于每一代都必须重新计算的权重。因此，这是一个有趣的例子，说明遗传算法的相同理念如何在不同的计算模型中实现，同时仍将其操作语义纳入 SLS计算的一般形式。

状态的定义与一般形式中的定义相同，不过部分句法类别进一步规定如下。

在此情形下，解表示粒子在搜索空间中的位置、方向和速度，而解的集合表示一组粒子。因此，Best 解是计算迄今为止找到的最优点（忽略其动量），而解集将给出群体中每个粒子的当前位置和动量。

$\mathbf{F}/\mathbf{P}/\mathbf{W}$ 类别表示在 SLS 方案中引导搜索所需的所有变量。$FPW \in \mathbf{F}/\mathbf{P}/\mathbf{W}$ 现在由每个粒子的基于适应度的权重组成（这些权重必须在每一代重新计算）。

我们将使用一般形式中的相同元变量，唯一的区别在于$\mathbf{Stm}$ 构造式的结构，现在定义如下:

$S := S_1; S_2$ | setProb(p) | 生成 | divert | aim | move |
nextGen | 评估 | stop | compute(p)

$\mathbf{Stm}$ 的修改方式与遗传算法中的类似：引入 divert、aim 和 move，以详细说明 nextGen 的工作方式。首先，divert 会在后续移动中引入随机影响；然后，aim 会使移动朝向目标；最后，move 会结合这两种作用力，确定粒子的新位置。

对于计算而言，divert 和 aim 语句的顺序实际上可以互换，但对于接下来的函数和规则，我们假设每次迭代中 divert 都先于 aim。为了使用这些语句，我们将添加与其同名的规则。与遗传算法类似，还需要额外的函数:

- $\mathcal{D}[\![]\!]_s$:**状态** $\longrightarrow$ **解[]*** 。此函数使用状态 s 提供的 $Prob$ 值，为 解集计算新的随机动量值，同时不改变任何位置。这些动量变化称为偏转。

- $\mathcal{A}[\![]\!]_s$: **状态** $\longrightarrow$ **解[]***。它使用状态 s 给出的 $Prob$、解集 和FPW 值，来计算粒子之间的吸引力

[divert]	$\langle \text{divert } s \rangle \Rightarrow s[Sols^* \mapsto \mathcal{D}[\![]\!]_s]$
[aim]	$\langle \text{aim } s \rangle \Rightarrow s[Sols^* \mapsto \mathcal{A}[\![]\!]_s]$
[move]	$\langle \text{move } s \rangle \Rightarrow s[Sols^* \mapsto \mathcal{M}[\![]\!]_s]$
[下一代]	$\langle \text{nextGen } s \rangle \Rightarrow \langle \text{divert; aim; move } s' \rangle$

图 4：粒子群优化操作语义的实例化

解集，然后将这种吸引力与通过$\mathcal{D}[\![]\!]_s$ 获得的动量值合并，并为解集中的每个粒子设置一个新的动量值。

- $\mathcal{M}[\![]\!]_s$ ： **状态** $\longrightarrow$ **解[]*** 。它使用状态 s 提供的解集 值，根据每个粒子在解集 中的先前位置和动量（方向与速度），计算其新位置。

- $\mathcal{A}eval[\![]\!]_s$ ： **状态** $\longrightarrow \mathbf{F}/\mathbf{P}/\mathbf{W}*$ 。这与一般形式中的情况相同，不过在此特定情况下，该函数会根据个体在搜索空间中的位置计算并更新其适应度值。每个粒子的适应度决定了其在吸引阶段的权重。

粒子群算法仍然可以用不同方式实现这些函数。例如，不同的邻域拓扑[31]会产生不同的 $\mathcal{D}[\![]\!]_s$ 和 $\mathcal{A}[\![]\!]_s$ 函数。与前述实例类似，PSO 的规则列表增加了新的规则，以展示每次迭代中的计算阶段，同时修改 [下一代]以引入这些阶段，如图 4 所示。

前面三个例子（GA、蚁群优化、粒子群优化）展示了一般形式语义在建模不同随机局部搜索算法的语义行为方面的强大能力。我们将利用这些例子证明一项重要的语义属性：一般而言，随机局部搜索策略具有图灵完备性。下一节中，我们将通过随机局部搜索模型的一个实例来证明这一属性。

请注意，鉴于图灵完备性，任何类型的算法（而不仅仅是 SLS）都可以由我们的 SLS 模型模拟——只要将其恰当地编码为已证明具有图灵完备性的随机局部搜索实例的输入。由此显然可知，特别是，任何随机局部搜索算法都可以编码进我们的 SLS 模型并由其模拟。然而，我们的 SLS 模型的目的不只是表示任意随机局部搜索方法，还要通过自然地使用模型参数来实现这一点。前文各节为遗传算法、蚁群优化和粒子群优化定义的模型实例展现了这种能力；同样，也可以为这些方法的变体及其他方法创建类似的自然实例。

我们已经提到，该框架如何处理具有不同邻域关系的粒子群优化算法，以及多信息素蚁群优化算法。显然，最初作为其他算法的改进版本而设计的算法，预计也可以用类似的方式定义（例如，河流形成动力学（RFD） [32] 可以用与蚁群优化算法非常相似的方式定义）。

许多常见的算法变体也很容易定义。例如，对于多种群算法 [33], ，可以将种群标识符作为个体的参数添加进去，并由不同的函数进行处理。对于多模态或多目标优化， [34, 35, 36] $Best$ 可以轻松变为向量，而 $\mathcal{B}eval[\![]\!]_s$ 则可以管理不同目标的标识。在协同进化算法中 [37], ，我们可以借用前述技术来处理多种群，而 $Extra$ 可以包含管理模型全局适应度所需的结构。类似的技术也可以用于实例化基于分解的策略 [38]。一些基于轨迹的启发式算法，例如模拟退火 [39] 或迭代局部搜索，需要单个个体种群。我们的一般形式也支持处理约束优化问题 [40], ，尤其是通过将惩罚或黑名单参数纳入 $\mathcal{A}eval[\![]\!]_s$ （与所实例化的启发式算法无关）。

当然，针对上述启发式算法的这些技术并非实例化我们框架的唯一方式——它们只是展示其灵活性的例子。最后，作为一个例子，考虑 [36] 中介绍的 SEMO、FEMO 和 GEMO 多目标优化算法：它们都维护一个由帕累托最优个体组成的种群，因此在每次迭代结束时执行 $Best = Sols$ 。这里可以看到，将 $Best$设为向量，便能轻松适配多目标优化。我们还可以将这些方法的“下一代”阶段分为两个步骤：选择和变异。由于这三种算法仅在变异哪些个体方面有所不同，因此它们的实例化方式相同，只有 $\mathcal{M}[\![]\!]_s$ 函数略有不同。这样，我们的模型既能捕捉它们共同的整体结构，又能适应各自的特点。

4 随机局部搜索启发式算法的图灵完备性

本节证明 SLS 计算的图灵完备性，这意味着任何能够在图灵机（TM）上运行的程序，也可以通过启动一种 SLS 策略来运行。我们通过证明这类策略中的一个实例——遗传算法——具有图灵完备性来实现这一点。据我们所知，此前尚未证明这一性质，而它对语义性质的研究有着重要影响。需要指出的是，文献中已在特定条件和配置设置下研究过某些 SLS 方法的收敛性（例如，参见 [41, 42, 43, 44, 45, 46]）。鉴于 SLS 方法的图灵完备性和赖斯定理，我们得出结论：一般而言，要获得这类

语义知识——就像一般而言无法判定程序是否会产生预期输出，但如果对程序的形式施加足够严格的限制，就可以对某些程序子集证明这一点（例如，循环结构受到限制或完全不含循环的程序）。

引入严格约束，可以让我们在某些 SLS 场景下推理理想的语义性质；例如，对最大化次模函数进行多项式时间近似 [47, 48]，或在强遍历性条件下使 SA 收敛到全局最优解 [42]。不过，这也必然会限制方法（视为计算模型时）的表达能力。图灵完备性通常也会使统计学无法用于预测 SLS 方法的行为。即使我们定义一个模拟通用图灵机的遗传算法，在许多实例上运行它，并计算其平均输出或执行时间；这也无法告诉我们，如果在新的、未经测试的实例上运行同一个遗传算法，一般而言会有什么预期结果。这就相当于仅仅观察 C++ 程序在某些输入下的运行情况，并计算其输出或执行时间的平均值，就试图预测它对所有输入的行为。然而，当额外约束确实限制了 SLS 方法的适用范围时，统计学对于分析这些方法确实非常有用。

请注意，证明遗传算法的图灵完备性，与某些遗传算法所构造的解具有图灵完备性毫无关系（例如，某些处理遗传编程 [25, 26, 27]的遗传算法）。证明遗传算法的图灵完备性，旨在表明存在一种遗传算法，能够对它可能接收到的任意输入进行模拟，执行任意程序（或者更准确地说，图灵机）。也就是说，当该遗传算法的初始配置以某种形式编码了那台图灵机及其输入时，该遗传算法的最终解就会给出那台图灵机对该输入产生的输出。

要实现图灵完备性，就必须表示程序的全部内部存储器，无论其增长到多大。因此，在我们证明中构造的遗传算法里，染色体必须能够任意增长，就像在遗传编程等遗传算法领域中未设置大小限制时那样。鉴于在我们的证明中遗传算法的行为与程序完全相同，限制染色体的大小并不会让遗传算法变得容易预测；这就像使用多项式量级的内存运行程序，或仅执行指数数量的步骤时，虽然可以判定其输出，但相应问题仍分别是 PSPACE 难或 EXPTIME难（因此，在 P≠PSPACE 的情况下，以及无论任何尚未证明的结果如何，分别都是难以处理的）。

请注意，如果一个计算模型包含某个允许为任意程序的组件，那么该模型极有可能是图灵完备的。我们可以认为，遗传算法足够通用，能够让某些组件（例如其适应度函数）成为任意程序。如果要实现图灵完备性，就必须利用定义某个组件时的任意自由度，那么人们可能会怀疑，这种图灵完备性是否仅存在于具有如此高复杂度的遗传算法中。如果确实如此，那么许多更简单但广泛使用的遗传算法类别可能都是非图灵完备且易于预测的，从而使遗传算法总体而言的图灵完备性在实践中的影响变得微乎其微。恰恰相反，我们的证明将表明，即使所有组件都以极其简单的方式定义，遗传算法仍然是图灵完备的。具体而言，适应度函数将只会

检查子字符串是否包含。这种简单性将表明，如果我们希望通过限制遗传算法组件的定义方式，来定义一个非图灵完备且可高效预测的遗传算法类别，那么这些限制将会非常严格，以至于几乎使所得类别失去意义。

4.1 证明遗传算法的图灵完备性

本节通过构造一个遗传算法来证明遗传算法具有图灵完备性。该遗传算法能够模拟任意给定输入下的任意给定图灵机，即它对图灵机具有通用性。具体而言，我们将给出一个遗传算法，用于求解这样的问题：对于任意给定输入下的任意给定图灵机，找出其完整的计算历史（即计算过程中迭代到达的所有配置组成的序列）。在介绍这个遗传算法之前，我们先给出一些定义和结果，以便构造该算法。

设 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ 是 [49, 第 327], 页中定义的图灵机，也就是说： Q 是状态集合； Σ 是输入字母表； Γ 是纸带字母表； δ 是转移函数； $q_0 \in Q$ 是初始状态； $B \in \Gamma$ 是空白符号； $F \subseteq Q$ 是终态（接受状态）集合。波斯特对应问题（PCP）是指：给定一个有限集合 W，其中包含有限字符串对 (a, b)，判断是否存在一个由这些字符串对构成的有限序列（同一对可以出现多次），使得依次读取各对的第一个组件所得的字符串与依次读取第二个组件所得的字符串相同。修正波斯特对应问题（MPCP）增加了一项约束：序列中的第一对固定。为了证明 MPCP 的不可判定性，在 [49, 第 401-412] 页中，任意一对 (M, w)（其中 M 是任意图灵机， $w \in \Sigma^*$ 是其输入）都会被映射为一个修正波斯特对应问题中的配对集合 T，使得 M 在 w 上停机，当且仅当 T 的 MPCP 答案为“是”。接下来，我们将在构造中使用这个通过确定性方式生成的有限配对集合 T。

只有当图灵机从初始位置开始从不向左移动且从不写入空白符时，才能生成这组对 T。幸运的是，对于每台图灵机，都存在一台具有这些限制的等价图灵机 [49]。根据 T 的构造方式，我们将有一个用于表示（M, w）的初始状态和输入的对，一个用于表示计算结束的结束对，用于表示 M 的接受状态q_f 并模拟纸带上每个符号的消耗的若干对，一个针对每个 $x \in \Gamma$的（x, x）对，一个用于分隔符号 # 的 $(\#, \#)$ 对（这些符号将每个完整图灵机配置的表示与下一计算步骤中到达的配置分隔开来），以及一个用于表示 δ中每次转换的对。此外，如果 MPCP 部分解的第一个对是表示机器初始状态的对，那么将有且仅有一个对$t \in T$ 能够扩展该部分解。这使得部分解的扩展能够有效模拟（M, w）的计算。

为了证明邮递员问题的不可判定性，我们将图灵机的停机问题归约到 MPCP，再将 MPCP 归约到邮递员问题。因此，如果邮递员问题可判定，那么停机问题也可判定（但停机问题是不可判定的）。如前所述，这个证明利用 T 中的配对来证明：MPCP 有解，当且仅当 M 在以 w 为输入时停机。此外，最短的

MPCP 解将表示 M 在输入 w 时的完整计算历史。

有了这些要素，我们可以设计一种遗传算法，尝试通过依次找出下一个配对（计算的下一步），来求解输入为 T 的 MPCP，直到 MPCP 得到解。我们将证明，对于所表示的图灵机的每个计算步骤，我们的遗传算法都会收敛到计算的下一步。这意味着遗传算法能够成功模拟每一种可能的（M，w）元组的行为和纸带状态。

其思路是，这种遗传算法将模仿遗传编程构建结构化个体的方式（在本例中，个体仅具有线性结构，即由不同数量元素组成的序列），以设计出一个个体，将 T 中的配对 t_i 排列成递增的 MPCP 部分解，直到找到完整的解。我们将采用遗传算法的操作语义来描述该算法。其种群仅包含一个个体。该个体表示为一个由若干元素组成的序列，我们称这些元素为瓦片；并且应将 $tile_i$ 视为 $t_i \in T$ 的计算表示（即 t_i 在遗传算法领域中的表示）。

适应度函数始终返回排列正确的瓦片数量，并保存迄今为止找到的最佳部分解（只有在计算结束时才需要返回该解）。选择和交叉阶段都是不必要的，也不会改变个体中的任何内容。变异阶段所做的更改只影响个体瓦片序列的末尾，也就是遗传算法正在构建的程序执行历史的末尾：如果最后一块瓦片不正确，就用另一块随机瓦片替换；如果正确，则在其后添加一块新的随机瓦片，以扩展序列。

借助这些要素，我们可以保证，对于每一个执行步骤，遗传算法都依概率收敛到添加下一个正确的执行步骤。然而，通过引入瓦片黑名单，并借助 $Extra$ 过滤掉在当前 MPCP 步骤中已被舍弃的瓦片，我们可以保证正则收敛。为了加快该实现的收敛，表示序列开头的特殊对 T 所对应的瓦片，可以从可随机添加到个体中的瓦片中移除。最后，只有在闭合对对应的瓦片被正确放置后，遗传算法才会停止。

请注意，这是一个极其简单的遗传算法，尤其是它只有一个个体的种群，且仅通过变异来改变。这使它类似于模拟退火（SA）或迭代局部搜索（ILS）等简单的轨迹式方法。

现在，我们定义数据结构（不要将其与作为这些数据结构实例的实际遗传算法变量混淆；我们分别使用粗体和斜体来区分它们）以及操作语义中使用的函数：

数据结构

- **瓦片** 是由一对字符串组成的。它用于表示成对的 $t_i \in T$。

- **问题**是一个由**瓦片** 类型的元素组成的数组和一个整数，分别表示输入 **(**M, w**)** 的 MPCP 编码中的成对集合 T 以及 $|T|$ 。

- **解** 由一个**瓦片**元素链表组成，表示构成 MPCP 实例的各对元素的有序排列。

- $\mathbf{F}/\mathbf{P}/\mathbf{W}$ 只是一个整数。它统计解集中唯一元素截至目前串联的正确瓦片数量。

- **Extra** 是一个布尔值数组。它表示一个瓦片黑名单：每次向部分解中添加新的正确瓦片时都会重置，并筛除当前迭代中截至目前尝试过的错误瓦片。

函数定义

- $\mathbb{C}[\![A]\!]$ 返回一个所有值均设置为 true 的**Extra** 实例。

- $\mathcal{P}[\![p]\!]_s$ 处理（M， w）以创建 T ，如 [49, 第 401-412], 页所述；然后将 T 中的每一对元素编码为**瓦片** 实例，并生成一个**问题** 实例：数组第一位置存放初始牌片，第二位置存放结束牌片，其余瓦片存放在接下来的 $|T|-2$ 个位置。**问题** 的整数值设置为 $|T|$。

- $\mathcal{G}[\![]\!]_s$ 返回一个包含 2 个元素的链表：第一个是数组$Prob$[6]的第一个元素，第二个则从$Prob$ 中所有索引小于所存整数的其他位置里随机选取。它还会返回一个**Extra**实例，该实例等于$Extra$ ，其中对应位置被设为 false。

- $\mathcal{S}[\![]\!]_s$ 始终返回$Sols$ 和$Extra$中的唯一元素。

- $\mathcal{C}[\![]\!]_s$ 始终返回$Sols$ 和$Extra$中的唯一元素。

- $\mathcal{M}[\![]\!]_s$ 定义如下：如果FPW 等于$Best$中的瓦片数量，则从$Prob$ 中追加一个随机瓦片（索引为 $i \in [1, N-1]$，其中 N 是存储的整数，数组的第一个索引为 0），然后返回扩充后的瓦片列表，以及一个**Extra** 实例，其中除索引为 i的值外，所有值均设为 true（这会重置黑名单）。否则，从 $Prob$ 中随机选择一个新瓦片，索引为 $i \in [1, N-1]$ ，其中 $Extra[i]$ 为 true。返回值是一个瓦片链表，它由$Best$ 的修改版本组成，末尾附有 $Prob[i]$ ；此外还返回一个**Extra** 实例，该实例等于$Extra$ ，但索引为 i的值设为 false。

[6]待解决问题的表示，在本例中即所有 MPCP 瓦片组成的列表。

- $\mathcal{A}eval[\![]\!]_s$ 将$Sols$ 中各瓦片的所有字符串拼接为两个字符串a 和 b，其中 a 按$Sols$ 给出的顺序拼接每个瓦片的第一个元素，而 b则以同样方式拼接第二个元素。如果FPW 尚未初始化，且 b 不能表示为 a 后跟其他符号的形式，则函数返回 1 和$Extra$。如果FPW 尚未初始化，且 b 可以表示为 a 后跟其他符号的形式，则函数返回 2，以及一个所有元素均设为 true 的 **Extra** 实例。否则，如果 b 不能表示为 a后跟其他字符的形式，函数则返回FPW 和$Extra$；如果可以，则返回 $FPW + 1$和一个所有元素均设为 true 的**Extra** 实例。

- $\mathcal{B}eval[\![]\!]_s$ 按相同顺序返回链接的解集中的前 FPW 块。
- $\mathcal{SC}[\![]\!]_s$ 仅当 Best 中瓦片的数量等于 FPW，且 Best 中的最后一块等于 $Prob[1]$ （如 $\mathcal{P}[\![p]\!]_s$ 中所定义的结束牌片）时才停止。

图灵完备性的证明

定理 1. 存在一种遗传算法，对于任意图灵机 M和任意输入 w，都能够模拟 M 针对 w的计算过程中的每一步。因此，遗传算法是图灵完备的。

证明。 为证明这一结果，我们将考虑上述遗传算法，并对已计算步数进行归纳。令 a 为按照顺序连接 $Best$ 中每个瓦片的第一个元素所得的字符串， b 为第二个元素的连接结果，并令 a_i, b_i 分别为 a 和 b 中第 $(i-1)$ 个与第 (i) 个分隔符 # 之间的字符串。

归纳假设（IH）如下：如果（M， w）的一次计算经过 n 次转换，那么在某个时刻， b 将拥有 $n+2$ 个分隔符 #，第（$n+2$）个分隔符之后不再有其他符号，并且每个 $b_i \subset b$ 都将正确表示 M 经过 i个计算步骤后的状态、指针和纸带。

归纳基础： 对于 $n = 0$，我们必须证明，在某个时刻存在 $\exists\, b_0 \subset b$，使得 b_0 表示（M， w）的原始配置。第一次求值后即可保证这一点，因为根据算法，染色体的第一个瓦片被确定为 MPCP 的第一个配对。

归纳步骤： 对于 $n = k+1$，我们使用（IH）并考察算法在该时刻的行为。b_k 是一个具有有限符号集合的字符串。 [49, 第 401-412] 页中所述的配对集合 T 的构造意味着，只有一个瓦片可以扩展 MPCP 的部分解。此外，不存在其第一个或第二个元素包含 0 个符号的瓦片。

因此，如果所述遗传算法总能找到下一个瓦片，以扩充存储在 $Best$ 中的当前部分解，那么添加 $|b_k|+1$ 个瓦片后（其中 $|b_k|$ 是字符串 b_k 中符号的数量），我们至少会向 a 添加一个分隔符（以匹配 b 中的分隔符），这意味着将其添加到 b，因为只有 $(\#, \#)$ 和 $(q_f\#\#, \#)$ 这两个瓦片含有分隔符。

剩下要证明的只有两点：添加的瓦片已正确计算出 b_k，以及所述遗传算法总能找到下一个瓦片来扩充 $Best$。第一条性质见 [49, 第 401-412], 页，第二条则由下述引理证明。

引理1。 上述遗传算法总能找到下一个瓦片，以扩展其存储在 $Best$中的当前部分解。

证明。 T 的构造意味着，对于MPCP 实例的每个部分解 c ，存在且仅存在一个 $t_i \in T$ ，使得 c 后接 t_i同样是一个部分解 [49, pp。第401-412]页。将这一性质应用于初始化后的 $Best$，可知存在且仅存在一个$Prob[i] \in Prob$，使得 $Best$ 与$Prob[i]$ 的串联同样是一个部分解。

变异步骤从$Prob$中随机选择一个非初始且未被列入黑名单的瓦片。在修改 $Best$后的第一次迭代中，选择正确瓦片的概率为 $\frac{1}{|Prob|-1}$。每有一个被列入黑名单的瓦片，除数就减少一个单位，而每次错误选择都会将一个瓦片列入黑名单。在最坏情况下，经过 $|Prob| - 2$ 次迭代后，选择正确瓦片的概率为1。

□

至此，归纳步骤的证明完成。因此，我们已经证明，所描述的遗传算法能够正确模拟任意给定（M， w）的每一步计算。

□

类似地，可以证明，当且仅当（M， w）停机时，我们的遗传算法才会停机；这使得一般情况下 SLS 计算的停机问题不可判定。不过，直接应用 T 的性质可以更简单地证明这一点，如下所示。如果（M， w）停机，那么一个表示其第一个字符串中接受状态的瓦片会被加入我们的部分解。如果这是一个（q_f##， #）瓦片，那么 $\mathcal{SC}[\![]\!]_s$ 将返回 **tt** ，我们的遗传算法也会停机。如果不是这种瓦片，那么遗传算法会继续添加消除纸带符号的瓦片，例如（Xq_f， q_f）、（Xq_fY， q_f）或（q_fY， q_f），直到唯一可行的选项是（q_f##， #）瓦片。关于这一消除过程的更详细说明见 [49, 第 401-412] 页。

值得注意的是，所提出的通用遗传算法尤其简单。其核心仅包含两个关键思想：(a) 变异算子改变染色体（种群中唯一的染色体）时，只修改其末尾部分，并用从给定的有限可能序列集合（瓦片）中随机选取的某个序列替换或扩展这部分；(b) 适应度函数以及用于决定在 (a) 中进行替换还是扩展的条件，只需检查某个序列是否为另一个序列的前缀，以及前缀长度。该遗传算法的机制尤其简单，绝不包含图灵机行为的内部细节（纸带、指针移动、状态等）。遗传算法要模拟的图灵机，实际上是以 (a) 中提到的有限集合形式提供的。请注意，使用任何通用图灵机 (UTM)[7] 都需要接收

[7]一个图灵机，给定任意图灵机和一个输入，就会模拟后者在该输入上的计算，参见例如 [50,第 20] 页。

要由其模拟的图灵机，并将其编码为某种任意表示法；同样，使用我们的遗传算法模拟任何图灵机，也需要接收以适当表示法编码的图灵机，尤其是 (a) 中提到的有限集合。尽管可以针对每个模拟案例手动定义这个集合，但实际上也可以通过自动方式，从采用最常见表示法定义的图灵机中获得该集合：将其转换为一组 MPCP 瓦片，如 [49] 中所述。

注意，我们可以通过移除遗传算法中的随机性，轻而易举地将其改造成确定性的通用遗传算法。具体来说，只需让遗传算法按照任意确定性的顺序依次尝试添加到染色体中的不同瓦片，而不是随机尝试即可（黑名单可以轻易用于确定接下来必须尝试哪个瓦片）。

最后，我们证明，通过从解集中删除那些信息已不再与解的演化相关的瓦片，可以实现一种内存效率更高（但不那么简单）的实现方式。如果瓦片前半部分中的某些符号已在后半部分的拼接结果中全部匹配，那么这些符号便不再提供用于确定下一个要追加到解集中的瓦片的信息。此外，如果前半部分的拼接结果中的某个符号右侧至少有两个分隔符号 #（它们不一定相邻或紧挨该符号），那么删除该符号仍能保留上一个计算步骤中的完整图灵机配置（请记住，每个完整的图灵机配置两侧都由一个 # 符号包围）。因此，我们可以放心地从解集中删除所有这样的瓦片：其前半部分中的符号已全部匹配，且在前半部分的拼接结果中，这些符号右侧至少有两个 # 符号。

这种实现需要仔细修改$Extra$ 和先前定义的函数，特别是要跟踪解集 的第二部分拼接后，有多少个符号需要被$\mathcal{A}eval[\![]\!]_s$ 跳过（因为这些符号已与第一部分拼接后被移除的符号匹配），并且要考虑到我们不再保留完整的计算历史，而只保留最后一步的图灵机配置以及当前步骤配置的一部分。请注意，通过这种方式移除瓦片，解集 中将始终只包含至多两个图灵机配置表示中的符号。现在的停止条件要求最后一块瓦片为 $Prob[1]$ ，并且跳过第二部分拼接后已匹配的符号后，第一部分和第二部分的拼接结果相同。其余函数只需直接调整即可。

通过以这种方式裁剪不必要的瓦片，解集 的表示大小（因而遗传算法中唯一的个体的表示大小）不会像我们最初的构造那样随着模拟图灵机步数的增加而增长，而仅与模拟图灵机在执行期间使用的最大内存量成正比。染色体大小与模拟图灵机步数之间的这种独立性，对于实现遗传算法的图灵完备性而言完全不是必需的。然而，对于其他与复杂性相关的性质却是必要的，例如，染色体具有多项式大小的遗传算法能够模拟任何在多项式空间内运行的图灵机这一性质。请注意，如果将完整的历史存储在染色体中，这是不可能实现的，因为

使用多项式空间的图灵机可能需要指数时间（即具有指数大小的 历史）。相反，具有多项式大小染色体的瓦片裁剪遗传算法能够模拟所有在多项式空间内运行的图灵机，因此，即使是具有多项式大小染色体的遗传算法的非常基本的语义性质，其判定也是 PSPACE 难的。

5 证明其他一般形式实例的图灵完备性

上一节给出的图灵完备性证明可以调整，以适用于其他 SLS 计算实例。当然，并非所有 SLS 算法实例都具有图灵完备性；不过，我们认为，除遗传算法外，大多数其他流行方法也具有图灵完备性。接下来，我们将非正式地说明，如何为本文实例化的另外两种方法证明这一点。

我们来考察蚁群优化。根据一台图灵机和输入 (M, w)，我们可以像之前一样构造一组 T MPCP 瓦片。我们可以构造一种蚁群优化实例，使其应用于该 MPCP 实例时必然倾向于求解它，从而动态地执行 M for w。由于图灵完备性要求无限的内存，因此蚁群优化中必须有一张供蚂蚁遍历的无限图；更准确地说，需要一张有限图（特别是树），并且能够在蚁群优化执行过程中按需迭代扩展至任意所需的有限大小，不受限制。每条边代表选取一个 MPCP 实例中的瓦片，而从根节点到任意叶节点的每条路径代表一个部分解（瓦片序列）。蚁群优化执行开始时，这棵树只有一个节点。当一只蚂蚁到达叶节点时，它便结束行程；只有当该路径的瓦片序列仅包含合法移动时（也就是说，像遗传算法中的做法一样，沿瓦片序列读取的两个字符串之一是另一个的前缀），才会通过信息素强化该路径。在这种情况下，移走蚂蚁后，立即从该叶节点（此时它不再是叶节点）创建新的边，每种可用瓦片对应一条边，最终使后续蚂蚁能够再前进一步。

通过这种方式，蚁群优化会根据瓦片引导蚂蚁沿合法路径前进并将其延伸。信息素轨迹只会在计算正确的瓦片（边）上得到强化。我们假设每条新创建的边在执行开始时的信息素轨迹为 0，信息素在经过任意次迭代后都不会消失，并且选择每条边的概率与所有可用边上的信息素数量成正比。可以看出，在这种设置下，最终到达图灵机计算的任意步骤是依概率收敛的。通过采用与图灵完备的遗传算法相同的停止准则，我们再次实现了图灵完备性。

证明粒子群优化的图灵完备性，与遗传算法的证明更为相似，而不是与蚁群优化的证明相似。除了对粒子群优化进行标准改造以处理离散域之外，我们还需要让解向量能够按需任意扩展，以表示任意长度的

完整图灵机内存状态表示，这与遗传算法中的做法相同。此外，与遗传算法的变异算子仅限于影响解的最后一部分（即它扩展出来的部分）类似，粒子群优化中控制粒子移动的两种作用力也被定义为仅影响特定范围，具体如下：使粒子向最优粒子靠近的作用力，会使所有粒子都必须复制最佳解的所有组件（瓦片）；而使粒子随机移动的作用力，则会使粒子在其向量的一个新组件（瓦片）中随机添加内容，以便扩展 MPCP 瓦片序列。这样，复制先前最佳解（即成为该解）的粒子，随后在其向量中随机添加合适内容（该内容必须是合法瓦片），便会成为新的最优粒子，供后续跟随。同样，这一过程依概率收敛，最终生成代表相应图灵机执行过程的瓦片序列。

与遗传算法的情况类似，可以在前文的蚁群优化和粒子群优化构造中引入黑名单，将依概率收敛转变为常规收敛。这样，终止前的最大步数就可以由一个关于图灵机为运行其输入实际所需步数的函数来界定。此外，正如我们在上一节最后几段所述的遗传算法替代构造中所做的那样，也可以裁剪蚁群优化和粒子群优化的构造树和解向量中不必要的元素。

6 结论与未来工作

据我们所知，我们提出了一种完整的形式语言，借助其通用结构和参数化构造，可以定义任意随机局部搜索方法。为展示其灵活性，本文介绍了它在三个机制迥异的算法族（遗传算法、粒子群优化、蚁群优化）中的实例。通过使用这种通用且标准化的语言来定义新算法和现有算法，可以更轻松地研究算法选择与问题特征之间的函数关系。不同研究者使用不同算法得出的结果，也可以无缝地归类并纳入共同的知识库。

我们还正式证明了该遗传算法实例具有图灵完备性，这意味着它所实例化的一般模型也具有图灵完备性，因而随机局部搜索方法总体上也具有图灵完备性。此外，我们概述了如何轻松地将这一图灵完备性证明改编应用于所介绍的另外两个实例，这也支持了这样一种观点：许多其他重要的类别同样具有图灵完备性。

根据赖斯定理，方法具备图灵完备性，就意味着其所有非平凡语义性质（即涉及其对输入与输出之间关系的计算，且并非总是成立也并非从不成立的性质）均不可判定。这种不可判定性意味着，对于给定的性质，不存在能够判定该性质是否成立的算法。这一结果对任何试图预测

这些方法的行为都构成了重要限制。例如，这意味着通常无法判定据称旨在解决某种优化问题的遗传算法、蚁群优化或粒子群优化实例最终是否会产生某类期望的解（例如最优解、始终达到某个性能比的解、平均而言达到某个性能比的解、满足某个给定结构条件的解等）。对于特定情形（例如某个随机局部搜索方法求解某个特定问题或问题实例），以及通过限制方法或问题的一般自由度而不具备图灵完备性的某些情形子集，这些语义性质仍然可以得到证明。然而，正如我们在论文中指出的那样，消除图灵完备性（例如，不让遗传算法染色体无限增长）本身并不足以实现对性质的高效 检测，因为这可能只是将不可判定性转变为难以处理性（例如 PSPACE 困难性）。

关于未来工作的方向，我们希望利用我们的模型，为随机局部搜索方法建立一套完全形式化的分类法。通过将一般形式实例化到更高或更低的抽象层次，可以得到该分类法的不同层级。此外，我们还希望研究在染色体完全无法增长的情况下，语义属性识别的一般复杂性。我们猜想，这种情况下仍然会存在 PSPACE 困难性，并指出，这种难以处理性与随机局部搜索领域已知的收敛结果是一致的。例如，只要我们能够以足够缓慢的速度推进，模拟退火就能保证达到最优解；在某些糟糕情况下，尤其需要指数时间 [51]。这意味着，从实际角度看，启发式搜索所需的时间与穷举搜索相应的（指数规模）搜索空间所需的时间相当。

最后，我们希望系统地研究，除算法实体的表示大小之外，启发式算法元素的定义中有哪些限制会决定该方法是否具备图灵完备性；鉴于非群体型启发式算法较为简单，我们将特别关注这类算法。

参考文献

参考文献

[1] H. H. Hoos，T. Stützle，《随机局部搜索算法：综述》，Springer Berlin Heidelberg，德国柏林、海德堡，2015年，第1085–1105页。doi: 10.1007/978-3-662-43505-2_54。URL https://doi.org/10.1007/978-3-662-43505-2_54

[2] T. Back，D. B. Fogel，Z. Michalewicz（编），《进化计算手册》，IOP Publishing，英国布里斯托尔，1997年。

[3] J. Kennedy、R. C. Eberhart、Y. Shi，《群体智能》，Morgan Kaufmann，2001。

[4] D. E. Goldberg，《遗传算法》，培生教育，2006。[5] M. Dorigo、G. Di Caro，蚁群优化：一种新的元启发式算法，载于：1999年进化计算大会论文集，CEC'99，第2卷，IEEE，1999，第1470–1477页。[6] M. Dorigo、M. Birattari、T. Stutzle，蚁群优化，《IEEE计算智能杂志》1（4）（2006）28–39。[7] F. Ducatelle、G. Di Caro、L. M. Gambardella，利用蚂蚁智能体结合反应式和主动式策略，为移动自组网中的路由提供支持，《国际计算智能与应用杂志》5（2）（2005）169–184。[8] D. Karaboga、B. Akay，一种用于约束优化问题的改进人工蜂群（ABC）算法，《应用软计算》11（3）（2011）3021–3031。[9] C. Qiu、C. Wang、X. Zuo，一种新颖的基于k均值全局最优选择策略的多目标粒子群优化算法，《国际计算智能系统杂志》6（5）（2013）822–835。[10] P. Pinto、A. Nägele、M. Dejori、T. Runkler、J. Sousa，利用局部发现蚁算法进行贝叶斯网络结构学习，《进化计算汇刊》13（4）（2009）767–779。[11] I. Rodríguez、P. Rabanal、F. Rubio，如何打造畅销产品：最优产品设计问题，《应用软计算》55（2017）178–196。[12] K. Sörensen，元启发式——揭开隐喻的面纱，《国际运筹学汇刊》22（1）（2015）3–18。[13] F. Hutter、H. H. Hoos、K. Leyton-Brown、T. Stützle，Paramils：自动算法配置框架，《人工智能研究杂志》36（2009）267–306。[14] M. López-Ibánez、J. Dubois-Lacoste、L. P. Cáceres、M. Birattari、T. Stützle，irace软件包：用于自动算法配置的迭代竞速，《运筹学展望》3（2016）43–58。[15] F. Hutter、H. H. Hoos、K. Leyton-Brown，面向通用算法配置的序贯基于模型优化，载于：国际学习与智能优化会议，施普林格，2011，第507–523页。[16] P. Rabanal、I. Rodríguez、F. Rubio，通过随机基准测试评估元启发式算法，载于：2016年国际计算科学会议（ICCS 2016），《计算机科学学报》，第80卷，2016，第289–300页。[17] K. A. De Jong，《进化计算：统一方法》，麻省理工学院出版社，2006。

[18] D. Molina, J. Poyatos, J. D. Ser, S. García, A. Hussain, F. Herrera，自然启发与生物启发优化的综合分类体系：灵感与算法行为、批判性分析及建议（2020）。arXiv:2002.08136。[19] Q. Kang, J. An, L. Wang, Q. Wu，广义群体智能计算模型的统一性与多样性，《国际人工智能工具杂志》21 (03) (2012) 1240012。[20] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, J.-J. Li，分布式进化算法及其模型：最新进展综述，《应用软计算》34 (2015) 286–300。[21] J. L. Olmo, J. R. Romero, S. Ventura，自动编程中的基于群体的元启发式算法：综述，《威利跨学科评论：数据挖掘与知识发现》4 (6) (2014) 445–469。[22] H. H. Hoos, T. Stützle，随机局部搜索：基础与应用，爱思唯尔，2004。[23] H. G. Rice，递归可枚举集的类别及其判定问题，《美国数学会汇刊》74 (1953) 358–366。[24] N. Cutland，可计算性：递归函数理论导论，剑桥大学出版社，1980。[25] J. R. Koza，作用于计算机程序种群的层次遗传算法，载于：N. S. Sridharan（编），第十一届国际人工智能联合会议论文集 IJCAI-89，第1卷，Morgan Kaufmann，底特律，密歇根州，美国，1989，第768–774页。[26] A. Teller，带索引内存的遗传编程语言中的图灵完备性，载于：第一届IEEE进化计算会议论文集。IEEE计算智能世界大会，IEEE出版社，1994，第136–141页。[27] A. Naidoo, N. Pillay，使用遗传编程进行图灵机归纳，载于：EuroGP'08：第11届欧洲遗传编程会议论文集，施普林格出版社，2008，第350–361页。[28] J. Kennedy，粒子群优化，载于：《机器学习百科全书》，施普林格，2011，第760–766页。[29] H. R. Nielson, F. Nielson，带应用的语义：形式化导论，Wiley Professional Computing，1999。[30] M. Sekara, M. Kowalski, A. Byrski, B. Indurkhya, M. Kisiel-Dorohinicki, D. Samson, T. Lenaerts，用于社会认知模拟的多信息素蚁群优化，载于：国际计算科学会议论文集，ICCS 2015，2015，第954–963页。

[31] Q. Liu, W. W. H. Yuan, Z. Zhan, Y. Li，粒子群优化的拓扑选择，Information Sciences 363 (2016) 154–173。[32] P. Rabanal, I. Rodríguez, F. Rubio，使用河流形成动力学设计启发式算法，载于：非传统计算国际会议，施普林格，2007，第163–177页。[33] E. Cantú-Paz，拓扑结构、迁移率与多种群并行遗传算法，载于：GECCO'99，《第一届遗传与进化计算年度会议论文集》第1卷，Morgan Kaufmann 出版社，1999，第91–98页。[34] S. Das, S. Maity, B. Qu, P. Suganthan，实参数进化多模态优化——最新进展综述，Swarm and Evolutionary Computation 1 (2) (2011) 71–88。[35] R. Marler, J. Arora，面向工程的多目标优化方法综述，Structural and Multidisciplinary Optimization 26 (6) (2004) 369–395。[36] M. Laumanns, L. Thiele, E. Zitzler，伪布尔函数上多目标进化算法的运行时间分析，IEEE Transactions on Evolutionary Computation 8 (2) (2004) 170–182。[37] L. Jiao, H. Wang, R. Shang, F. Liu，一种基于方向向量的协同进化多目标优化算法，Information Sciences 228 (2013) 90–112。[38] Q. Zhang, H. Li，MOEA/D：一种基于分解的多目标进化算法，IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731。[39] S. Kirkpatrick, C. G. Jr., M. Vecchi，模拟退火优化，Science 220 (4598) (1983) 671。[40] D. Karaboga, B. Basturk，用于求解约束优化问题的人工蜂群（ABC）优化算法，载于：模糊逻辑与软计算基础，第十二届国际模糊系统协会世界大会 IFSA 2007，《计算机科学讲义》第4529卷，施普林格，2007，第789–798页。[41] G. Rudolph，进化计算中的有限马尔可夫链结果：概览，Fundamenta informaticae 35 (1-4) (1998) 67–89。[42] D. Mitra, F. Romeo, A. Sangiovanni-Vincentelli，模拟退火的收敛性与有限时间行为，Advances in Applied Probability 18 (3) (1986) 747–771。[43] A. W. Johnson, S. H. Jacobson，广义爬山算法的收敛性，Discrete Applied Mathematics 119 (1-2) (2002) 37–57。

[44] I. Trelea，粒子群优化算法：收敛分析与参数选择，《信息处理快报》85 (6) (2003) 317–325。[45] C. Lin、Q. Feng，标准粒子群优化算法的收敛分析与参数选择，载于：第三届国际自然计算会议，ICNC 2007，第3卷，2007年，第823–826页。[46] M. Jiang、Y. Luo、S. Yang，标准粒子群优化算法的随机收敛分析与参数选择，《信息处理快报》102 (1) (2007) 8–16。[47] C. Qian、Y. Yu、K. Tang、X. Yao、Z.-H. Zhou，通过多目标进化算法最大化次模函数或单调近似次模函数，《人工智能》275 (2019) 279–294。[48] T. Friedrich、F. Neumann，通过进化算法在拟阵约束下最大化次模函数，《进化计算》23 (4) (2015) 543–558。[49] J. E. Hopcroft、R. Motwani、J. D. Ullman，《自动机理论、语言与计算导论》，第3版，培生教育，2001。[50] S. Arora、B. Barak，《计算复杂性：一种现代方法》，剑桥大学出版社，2009。[51]A. Nolte、R. Schrader，模拟退火及其在图着色中的问题，载于：欧洲算法研讨会，施普林格，1996年，第138–151页。