超越计算复杂性——试错、动态行动与智能[1]

Chuyu Xiong

独立研究者

*chuyux99@gmail.com

摘 要

计算复杂性是计算机科学的核心理论，它决定了计算的难度。我们必须应对许多复杂度很高的问题，人工智能领域尤其如此。这引出了一个重大问题：除了受计算复杂性限制之外，是否还有更好的方法来应对这些高度复杂的问题？我们认为，智能科学的思想和方法可以应用于这些问题，帮助我们超越计算复杂性。本文尝试厘清相关概念，并提出非特定化计算、特定化计算、计算主体和动态搜索等定义。我们还提出并探讨了一种框架，即试错+动态搜索。数划分问题是一个广为人知的NP完全问题，我们以此问题为例，阐释文中讨论的思想。

关键词： 计算复杂性、试错、动态行动、智能、非特定化计算、特定化计算、数划分问题

1 引言

计算技术与理论正在迅速发展，但最终关注的问题依然是：我们如何将实际计算能力拓展到工程和科学问题，包括人工智能领域的问题？

这些问题的一个重点是计算复杂性。计算复杂性是计算机科学的核心部分，研究计算一个问题需要多少时间和存储空间。通常，计算问题都有规模，例如矩阵的大小、整数的位数、自动驾驶的环境复杂度、视觉识别的准确度等。计算复杂性会随着问题规模增长，增长速度可能相对缓慢，通常表现为多项式增长；也可能迅速增长，通常表现为指数增长。复杂度呈指数增长的问题通常被认为极其困难。最典型的例子是大整数分解。这是一个复杂度呈指数增长、极其困难的计算问题，后来成为密码学的理论基础。

计算复杂性高的问题无疑非常困难，但我们不能忽视这些问题，必须正视它们。人工智能更是离不开这些高复杂度问题。在人工智能几乎所有领域，都不可避免地会遇到这类问题。例如，人脸识别就是一个 NP完全问题 [1]。这意味着，处理这些问题时，对计算时间和存储空间的要求都非常高。因此，我们希望采用智能方法来

[1] 感谢妻子始终如一的支持

降低对时间和存储空间的要求。因此，人工智能与计算复杂性密不可分：我们需要在人工智能领域的问题上超越计算复杂性；反过来，我们也需要运用智能方法来超越复杂度。这种纠缠会不断推动我们走向更高的智能。这种情形可以与生命的发展相类比：生命在应对困难的同时，也不断发展和提升智能。

但我们首先必须澄清其含义。计算复杂性是一套坚实而严谨的理论。这套理论所设定的理论界限无法逾越。然而，我们需要知道，理论所设定的界限针对的是一整类问题实例，而非单个实例。但我们往往只需要解决某个特定实例，而不是整个问题类。于是我们会问：对于某个特定实例，是否存在一种远胜于整个问题类通用解法的方案？又该如何为这个特定实例找到一个好的专门解法？如果我们能够做到这一点，就能针对特定实例取得远胜于遵循通用解法的结果。这就是我们所说的超越计算复杂性。

毫无疑问，超越计算复杂性非常困难。事实上，这样做是否可能，本身就是一个大问题。目前，没有坚实的理论能够支持这一观点，当然也没有确凿的理论能够彻底否定它。因此，本文尝试对此进行初步探讨。我们试图梳理出一个讨论框架，以便进一步探索。事实上，促使我们朝这个方向前进的是：在研究数划分问题 [2], 时，我们发现有些情况看起来非常困难，实际上却很容易解决；但也有些情况确实非常棘手（一道极其曲折、难以察觉的屏障 [3]）。面对这些谜题并反复思考时，我们逐渐发现，其中隐约显现着智能。

本文提出一个核心观点：对于计算复杂性高的问题，我们应进行“特定化计算”，而非“未特化计算”。具有智能和主体性的计算主体，能够针对特定实例找到特定化计算方法。我们为这类计算主体提出了一种框架，即试错+ 动态行动。我们以数划分问题为例阐释这一观点；数划分问题是卡普的25个NP完全问题之一。

2　计算的局限——计算复杂性

任何计算都需要资源，无论是存储空间还是处理器周期；这些资源需求取决于该计算的计算复杂性。这甚至可以被视为一种物理屏障：若无法获得所需资源，在物理上就不可能执行该计算。事实上，众所周知，现代密码学正是建立在计算复杂性所造成的这种物理需求之上。

然而，处理计算极限的方法各不相同，所需的资源也各不相同。我们可以看一个非常简单的例子，4567×2341。如果采用普通乘法规则，就需要进行16次个位数乘法和12次个位数加法。对于这个特定的例子，这种计算方法很合适，对其他任何情况也同样适用。但如果我们的问题是4567×2300呢？当然，我们也可以采用上述方法，因此需要进行16次乘法和12次加法。然而，显然我们可以采取更简洁的方法，只需进行8次乘法和4次加法即可完成计算。这是因为我们充分利用了这个问题的特殊性质——2300中的0，因此所需的资源可以少得多。

从这个简单的例子中，我们可以看出，处理计算任务有两种方法。一种是对所有情况都使用同一个确定不变的程序；另一种则是在可能的情况下，针对具体情况采用适合它的特定方法。正如这个例子所示，后一种方法所需的资源少得多。

我们来看一个更贴近实际的例子。有一个整数数组：$\Omega = \{63,48,932,266,671,47,110,82,39\}$，我们要把这个数组划分成两个数组，使它们的和相等。有一种方法适用于任何数组，那就是穷举搜索，也就是遍历所有可能的划分。这个数组的长度为 9，因此穷举搜索需要进行 2^8 次搜索，计算量相当大（至少对于手动操作来说如此）。不过，对于这个特定的例子，我们可以尝试用一种特殊的方法来解决。我们将通过试错和调整来求解。首先，随意将它划分成两个数组：$\Omega_1 = \{63,48,932,266\}$，$\Omega_2 = \{671,47,110,82,39\}$。很容易看出，第一个数组的和更大。接下来，我们做些调整，让第一个数组的和变小、第二个数组的和变大，如下所示：$\Omega_1 = \{63,48,932,110\}$，$\Omega_2 = \{671,47,82,39,266\}$。现在，第一个数组的和仍然更大。不过这一次，我们能更清楚地看出这一点，并知道如何进行最后的调整。结果如下：$\Omega_1 = \{63,48,932,47,39\}$，$\Omega_2 = \{671,82,266,110\}$。现在两个数组的和相等。这样一来，计算量就小得多了。

也就是说，当我们面对一个具体案例时，如果尝试在其中寻找各种特定关系，并尽可能利用这些关系，就能以少得多的资源进行计算。因此，一个问题自然浮现：这种方法真的能节省资源吗？这并不是一个容易回答的问题。我们首先需要仔细作出一些定义。

定义 2.1（未特化计算与特定化计算）
假设有一个计算问题，其所有实例构成一个集合 W。如果存在一个固定且明确的计算程序 C，对于任意 $w \in W$，$C(w)$ 都能得到正确结果，我们就称计算程序 C 是一个覆盖 W 的未特化程序。对于某个特定实例 w，如果存在一个计算程序 C_w，使得 $C_w(w)$ 能够得到正确结果，并且尽可能少地消耗资源，我们就称 C_w 是实例 w 的特定化计算程序；不过请注意，对于 $u \in W$，$u \neq w$，$C_w(u)$ 可能无法得到结果（程序不停止或崩溃），也可能结果不正确，或者资源消耗大得多。

根据这一定义，我们可以看出，在上述数划分问题的例子中，穷举搜索法属于非特定化计算，因为它适用于所有情形；尝试并调整的方法则属于特定化计算，因为它只适用于特定情形，而不适用于其他情形。

面对计算问题，迄今为止主流的做法一直是尝试寻找非特定化计算程序。这是合理的，因为采用非特定化计算时，思路会变得简单。对于任何实例，只需将该实例“代入”计算程序，就总能得到正确结果，无须考虑其他任何事情。思路简单且结果始终正确，正是人们所期望的。对于较简单的问题，或者资源充足的情况，这种方法完全合理。然而，处理复杂度较高的问题时，这种方法就不再合适。正如上面的例子所示，探索特定实例的具体性质，并充分利用这些性质，才更为合理。这些具体性质可能包括对称性、弱点、模式等，可以用来减少资源消耗。也就是说，对于复杂度较高的问题，我们应该追求特定化计算，而不是非特定化计算。

好，对于某个特定实例，采用特定化计算，而非非特定化计算。听起来很不错。但特定化计算程序从何而来？对于非特定化计算，我们知道该怎么做：人类程序员努力编写出一个适用于所有实例的程序。我们对此很熟悉，也非常了解。整个行业几十年来一直在做这件事。然而，我们不知道如何进行特定化计算，这对我们来说非常陌生。我们应该为每个特定实例手动开发一个程序吗？这根本不现实。或者，我们是否应该有一个查找程序，帮助我们为某个特定实例找到相应的特定化计算程序？听起来不错。但这样的查找程序本身也会消耗资源。于是问题来了：总资源消耗能否降低？因此，我们需要对资源作出一些定义。

定义 2.2（非特定化计算所需资源）假设有一个计算问题，所有实例构成集合 W，如果 C 是一个未特化

计算程序，覆盖 W，对于每个特定实例 $w \in W$，其消耗的资源 C（w）为 Z_w，那么 C 所需的资源为：$Z = max\{Z_w | w \in W\}$。如果有两个以上的未特化计算程序覆盖 W，则取这两个计算程序所需资源中较小的一个，作为覆盖 W 的未特化计算程序所需的资源。

对于特定化计算，我们有如下命题。

命题 2.3（特定化计算所需资源）假设有一个计算问题，所有实例构成集合 W，且所需资源

由覆盖 W 的未特化计算所需资源为 Z。假设我们使用一个确定且固定的查找程序 S，为某个特定实例查找特定化计算程序；也就是说，对于任意 $w \in W$，S（w）都能获得该实例 w 的特定化程序 C_w，那么 S（w）和 C_w（w）消耗资源的上界将等于 Z。

证明： S 是一个确定且固定的查找程序，因此，对于任意实例 $w \in W$，我们先执行 $S(w)$ 得到 C_w，再执行 $C_w(w)$ 来计算该实例。因此，这一过程构成了一个覆盖 W 的非特定化计算程序，我们将其记为 C。因此，C 所需的资源为 Z。■

这个命题告诉我们：使用一个确定且固定的查找程序来查找未特化计算并不可取，因为这样做无法节省资源。根据这个命题，非特定化计算所需的资源是一条稳固的基准线，不容易逾越。因此，我们将计算所需资源定义如下。

定义 2.4（计算所需资源）假设有一个计算问题，其所有实例构成集合 W，该问题的计算所需资

源是覆盖 W 的非特定化计算所需资源。

也就是说，对于实例 $w \in W$，计算需要资源 Z，这是由非特定化计算定义的。Z 是计算复杂性所规定的上限。但如果我们知道适用于 w 的特定化计算程序，就可以用远少于 Z 的资源进行计算。这就是超越计算复杂性。如果能做到这一点，那当然很好。然而，这需要我们知道适用于 w 的特定化程序。如果我们事先不知道这样的程序，还能超越计算复杂性吗？如果能，该如何做到？当然，正如命题 2.3 所指出的，不能依靠一个确定且固定的查找程序。智能至关重要。

3　计算主体与试错 + 动态行动

对于给定的特定实例，问题在于如何找到所需资源少得多的特定化计算程序。我们无法手动找到这样的程序，也不能使用确定且固定的查找程序。以下是获得特定化程序的几种可能方式。A）我们直接拥有它。B）我们记忆了大量这样的程序，并拥有一张查找表来定位所需程序。C）我们与特定实例交互，然后从交互中获得特定化程序，而且这一过程消耗的资源阶数更低。

A）类似于预言机。这非常有趣，表明特定化计算与非确定性图灵机之间存在紧密联系。但我们现在暂不讨论这一点。B）效果也不太理想，因为它需要记住所有实例。通常实例的数量非常庞大，记住它们需要耗费大量资源。C）这种可能性意味着，执行特定化计算的计算实体具备某种能力。事实上，这是一种非常强大的能力：它能够探索情况、作出判断，并利用刚刚显现的可能性。

我们想把这样的计算实体称为计算主体，而这个主体内部具有智能和主体性。Wang Pei 是一位通用人工智能研究者和倡导者。根据他所倡导的智能定义：智能是在资源有限的情况下作出最佳适应的能力[4]。显然，他对智能的定义与我们所说的计算主体的能力即智能，方向一致。事实上，Wang Pei 提出的这一智能定义对我们有所启发。基于这些考虑，我们给出以下定义。

定义 3.1（计算主体的智能）假设存在一个计算问题，其所有实例构成

一个集合 W，并假设覆盖 W 所需的非特定化计算资源为 Z。现在有一个计算主体 A 负责处理 W；如果对于实例 $w \in W$，A 可以拥有一个特定化程序 C，并且可以用比 $Z(O(log(Z)))$ 少一个数量级的资源完成 $C(w)$，那么我们就说 A 能够智能地计算 w。如果 $V \subset W$ 是 A 能够智能计算的所有元素构成的子集，那么 A 的智能程度由以下数量衡量：$q(A) = |V|/|W|$。

也就是说，智能程度大于零的计算主体能够突破计算复杂性的障碍。然而，一个根本性问题是：这样的计算主体真的存在吗？据我们所知，目前还没有理论探讨这类计算主体是否存在。此外，也没有理论告诉我们如何构建智能计算主体。这些问题不仅是计算理论中的重大问题，也是人工智能领域的重大理论问题，仍有待进一步研究。在此，我们尝试提出这样一个构想：如果采用适当的试错过程和动态行动，就有望形成一个智能计算主体。

在我们的常识中，试错是一种非常合理的方法；事实上，我们经常会在不知不觉中使用它。在计算理论中，Gold、Putnam、Kugel 等人坚持采用试错方法来处理可计算性问题 [5]。Kugel 将这种试错计算程序称为 Gold-Putnam 机器。由此，他们开发出一种试错过程，用于处理更难的可计算性问题，并尝试处理不可计算的问题。

试错究竟在做什么？为什么它能成功？试错实际上基于以下事实：1）承认我们存在一个未知，但这个未知可以通过努力获得；2）可以通过某种变换，将未知转移到一个确定的搜索空间中；3）试错努力构成一个

循环：从尝试中获得反馈，根据反馈寻求更好的结果，而寻求就是在搜索空间中从一个点移动到下一个点；4）可以在搜索空间中到达（或接近）未知。试错是一种非常有效的机制，通常也是解决问题不可或缺的工具。

回到我们寻找特定化计算的问题。假设该问题的所有实例构成集合W，给定一个实例$w \in W$，我们想要找到适用于w的特定化计算程序C_w。因此，这里的未知量是C_w。正如前面所讨论的，我们想把未知量转换到某个搜索空间中。搜索空间可以有很多种。这里我们要施加一些限制：将搜索空间限制为布尔向量空间，也就是说，搜索空间是B^N。当然，这种限制有其局限性。不过，如果N足够大，布尔向量空间实际上可以覆盖任意参数空间，而且使用参数空间来规范计算十分常见（非确定性图灵机就是一个例子，参见 Cook [6]）。因此，将搜索空间选为N维布尔向量空间是合理的。如有必要，我们之后可以改用其他搜索空间。这样，搜索的目标就是获得正确的参数向量$p \in B^N$，然后参数向量p就会为我们带来适用于w的特定化程序C_w。由于C_w取决于参数向量，我们可以将其写作$C(w,p)$。

有了搜索空间，我们来考虑一个试错过程。这样的过程需要具备以下组成部分。首先，一个试错程序$T(w,p),\ w \in W, p \in B^N$：如果给定的参数向量$p$正确，$T(w,p) = 1$；否则，$T(w,p) = 0$。这是主要的反馈。不过，$T$也会反馈其他信息$t$。其次，一个搜索程序$S(w,p,t), w \in W, p \in B^N, S$会给出$p_n$，即试错过程在搜索空间中的下一个点。$S$也会为试错提供其他信息。第三，一个计算程序$C(w,p),\ w \in W, p \in B^N$：如果参数向量$p$正确，该程序就是特定化程序。具备这些组成部分后，试错过程如下：

1) 首先设置参数$p = p_0$并开始试错循环。 2) 试错循环：运行试错（c、t）$\leftarrow T$（w、p），其中c是测试结果，t是其他所有反馈信息。如果测试结果为 1，则参数向量正确，退出循环。如果测试结果为 0，则继续。 3) 搜索S（w、p、t），S生成p_n，即用于下一次试错的参数向量。 4) 试错循环持续进行，直到获得正确的参数p，或报告错误。 5) 如果获得了正确的参数向量，也就获得了特定化程序C（w、p）。

请注意，在试错过程中，最重要的组成部分是S（w、p、t），它将生成下一个用于试验的参数向量。S可以采用最简单的方法——穷举搜索，搜索每一个可能的点。然而，这并非我们所期望的。穷举搜索必须使用2^N资源。我们期望使用少得多的资源。因此，我们需要一种更好的S，即动态搜索，它能够智能地利用试错过程中的反馈信息。要智能地利用反馈信息，动态搜索需要具备主体性。我们在 [7]中详细讨论了机器的主体性和动态行动。现在我们可以定义智能搜索了。

定义 3.2（智能搜索）假设试错中的搜索空间

误差过程为$P = B^N$，动态搜索为$S(w,p,t)$。如果对于给定的w，任意初始参数p_0，S都能仅使用$O(N)$资源，为w找到正确的参数向量p，我们就称S能对w进行智能搜索。

具备智能搜索能力的计算主体才是真正智能的。

命题 3.3（试错+ 动态行动）假设有一个规模为 N 的计算问题，并且

所有实例构成集合 W，覆盖 W 所需的未特化计算资源为 $O(2^N)$。假设计算主体 A 由一个带有参数的程序 $C(w,p)$ 和试错过程 + 动态搜索 S 组成。对于一个实例 $w \in W$，如果 S 能够针对 w 进行智能搜索，并且程序 $C(w,p)$ 只需要 $O(N)$ 资源，那么计算主体 A 就具有定义 *3.1* 中所述的智能。

很容易看出这个命题成立：如果 S 和 $C(w,p)$ 都只需要 $O(N)$ 资源，那么定义 3.1 中指定的集合 $V \subset W$ 就非空，因此计算主体 A 具有智能。所以，A 能够超越计算复杂性。

智能搜索和计算主体都很了不起。但是，我们怎样才能得到它们呢？这是一个重大问题，需要开展大量进一步研究。在下一节中，我们将以数划分问题为例，对此进行一些探讨。

4 数划分问题

数划分问题是一个非常著名且重要的问题。现在，我们以这个问题为例，运用前面讨论过的思路，希望能帮助我们更好地理解。数划分问题可以用一句话来解释：给定一个自然数集合 Ω，问能否将 Ω 划分为两个子集 Ω_1 和 Ω_2，使得 Ω_1 中的数之和等于 Ω_2中的数之和？这个问题是 P 与 NP 问题的典型示例：验证解很容易，但找到解很困难。

让我们更详细地描述这个问题。现在设数组长度为 N，我们考虑长度为N的集合 Ω，其中的元素都是自然数，也就是说，$\Omega \in I^N, I$ 是自然数集合。接着，我们考虑一个 N维布尔向量 $p = (p_1, p_2, \dots, p_N)$，$p_j \in B, p \in B^N$；对于集合 $\Omega = \{\omega_1, \omega_2, \dots, \omega_N\}$ 和布尔向量 p，我们定义一个量：

$$< p, \Omega >= \sum_{j=1}^{N} q_j \omega_j \quad \text{if } p_j{=}1, q_j{=}1; \text{if } p_j{=}0, q_j{=}\text{-}1 \tag{1}$$

很容易看出，这 m 个子集的数量是两部分之和的差值：显然，参数向量 p 说明了如何将 Ω 划分为两个子集，因此我们称 p 为划分向量。也就是说，数量 $< p$、$\Omega >$ 实际上是在检验划分向量 p 能否将 Ω等分划分；这个数量还反馈了该划分距离等分划分有多远。我们定义另一个功能：$\varphi(\Omega, p)$：

$$\varphi(\Omega, p) = \begin{cases} 1 & if < p, \Omega > = 0 \\ 0 & if < p, \Omega > \neq 0 \end{cases} \tag{2}$$

$\varphi(\Omega,\ p)$ 是一个布尔函数：如果划分向量 p 将 Ω 等划分，则函数值为 1，否则为 0。因此，该函数是一个试错函数，使用 p 作为参数向量。我们在 [2], 中定义了算子 $\odot$，表示“尝试遍历”。利用函数 $\varphi(\Omega,\ p)$和算子 $\odot$，，我们可以定义划分函数：

$$Par_N(\Omega)\colon I^N \to B,\ \ Par_N(\Omega) = \varphi(\Omega, p) \odot P \tag{3}$$

伪代码：试错 + 穷举搜索

设置： p_0，　Ω
初始值： $\mathrm{N} = 0, \mathrm{p} = p_0$
当 $\mathrm{N} < 2^N$
　继续试错： $t \leftarrow < p, \ \Omega >$
　如果 $\mathrm{t} = 0$ **则**
　　停止，输出"计算成功， $Par_N(\Omega) = 1$**，划分向量："** p
　Else
　　继续搜索： $p_n \leftarrow S, \quad p \leftarrow p_n, \quad N \leftarrow N + 1$
　结束如果
结束循环
停止，输出"计算成功， $Par_N(\Omega) = 0$**"**

这里，P 是由所有划分向量构成的空间（这里 $\mathrm{P} = \mathrm{B}^{\mathrm{N}}$）。该函数的含义是：对 φ的所有 $p \in P$ 应用 $\Omega(p)$ 进行尝试。如果测试结果中有任何一个值等于 1，则 $Par_N(\Omega)$ 的值为 1，否则 $Par_N(\Omega)$ 的值为 0。也就是说，Par_N 的函数值是通过试错定义的。这清楚地表明，数划分问题的定义是"通过试错 + 穷举搜索"。因此，我们很自然地可以直接将这个定义转化为计算。请参见伪代码"试错 +穷举搜索"。

显然，先前的试错过程 + 穷举搜索只是表达$Par_N(\Omega) = \varphi(\Omega, p) \odot P$的程序。事实上，程序和方程完全是一回事。因此，使用试错-

伪代码：试错 + 动态搜索

设置： p_0，$\mathrm{N}_{\max}$，　Ω
初始： $N = 0, \quad p = p_0$
当 $\mathrm{N} < \mathrm{N}_{\max}$
　继续试错： $t \leftarrow < p, \ \Omega >$
　如果 $\mathrm{t} = 0$ **则**
　　停止，输出"计算成功， $Par_N(\Omega) = 1$**，划分向量："** p
　Else
　　继续搜索： $(p_n, \mathrm{i}) \leftarrow S(t, p, \Omega)$
　　如果 $i = 0$**，则**
　　　停止，输出"计算成功， $Par_N(\Omega) = 0$**"**
　　否则 $i = 1$
　　　$p \leftarrow p_n, \ N \leftarrow N + 1$
　　Else
　　　停止，输出"计算失败，搜索停止"
　　结束如果
　结束如果
结束循环
停止，输出"计算失败，超出范围"

和错误 + 搜索来计算 Par_N 是很自然的。在程序中，搜索 S 是穷举搜索，即S 遍历整个 P。实现穷举搜索有很多种方法，只要 S 能够遍历整个 P即可。

我们要强调，上述计算是未特化计算，适用于任何实例，并且总能得到正确结果。值得注意的是，这种计算所需的资源为$O(2^N)$。现在我们已经有了数划分问题的未特化计算。那么，如何实现特定化计算呢？正如我们在第 3 节所讨论的，我们需要将其转换到搜索空间。对于数划分问题来说，这相对容易，因为划分函数的定义已经包含了搜索空间。但我们需要将穷举搜索改为动态搜索。随后，计算程序以伪代码“试错+ 动态搜索”的形式给出。

注意，在试错+ 动态搜索中，程序有几处不同于试错+ 穷举搜索。首先，动态搜索不保证搜索成功。程序有 4 个出口，其中两个表示成功，对应于$Par_N(\Omega)$等于 1 或 0。另外两个表示失败：一种是搜索次数超过预设值而被迫停止，另一种是动态搜索认为搜索已无法继续。因此，程序并非总能成功。程序成功时，会给出 $Par_N(\Omega)$的值；否则，无法给出$Par_N(\Omega)$的值。还应注意，当$Par_N(\Omega)$的值为 1 时，也会给出划分向量，并且可以保证该值和划分向量均正确。然而，当$Par_N(\Omega)$的值为 0 时，结果可能正确，也可能错误，因为这并非穷举搜索，而是具有智能和主体性的动态搜索，因此可能出错。总之，计算主体针对一个实例（给定的数列）执行特定化计算。结果有 3 种：1）计算完成，且结果正确；2）计算完成，但结果错误；3）计算失败，失败有 2 种：一种是搜索停止，另一种是超出范围。

程序中最重要的部分是动态搜索 $S(t,p,\Omega)$。它是一个具有智能和主体性的计算主体。它会查看其输入：t、 p、 Ω，其中，t 是测试程序提供的反馈信息，p 是当前使用的划分向量，Ω 是当前正在处理的数列。S 会智能地利用这些信息，决定下一次尝试使用哪个划分向量。S 不会按照预设方式行动，而是能够判断当前情况，探索各种可能性，并尝试为下一个划分向量给出最佳猜测，同时节省将要使用的资源。S 还应当能够从当前情况和自身的错误中学习。因此，S 显然不是传统的搜索程序。但问题是：我们能否构造出这样一个内部具有智能和主体性的动态搜索?

对于数划分问题，对于任何给定的、能够被等分的数列 Ω，总是存在一种能够非常快速地得到 Ω 的划分向量的搜索方法，因为我们可以将初始向量 p_0 设置为该划分向量。然而，这类搜索既平凡又琐碎，并不是我们真正想要的动态搜索。我们所期望的动态搜索应当是这样的：从任意初始向量 p_0出发，仅使用少得多的资源，就能够得到正确结果。我们提出如下猜想。

猜想 4.1（数划分问题存在动态搜索）对于数划分问题，存在一个具有以下性质的动态搜索 S（t，p，Ω）：

对于大多数整数数组 $\Omega \in I^N$，以及任意初始划分向量 p_0，S 都能够仅使用 O（N）资源达到正确结果。达到正确

结果的含义是：试错 + 动态搜索能够完成计算（不会异常退出），并正确给出值Par_N（Ω）。

如果我们确实能够展示这样的动态搜索，它将是计算理论的一项重大突破，可用于许多工程应用领域。不仅如此，找到这样的动态搜索也将是智能科学的一项坚实进展。不过，我们要指出：即使我们找到了这样的动态搜索，数划分问题是 NP完全问题这一事实也不会改变，也就是说，存在一些 $\Omega \in I^N$ ，使得试错 + S 仅使用 $O(N)$ 资源时，将无法计算 $Par_N(\Omega)$。

5 备注

我们想再次强调几个要点。计算复杂性设定了一种限制：进行计算必须有足够的资源；如果资源不足，就无法完成计算。但这一限制适用于问题的所有实例。对于某个特定实例，可以使用特定化程序来完成计算，而且只需少得多的资源。这就是超越计算复杂性。然而，要获得特定化程序，我们需要具备智能和主体性的计算主体。

不过，我们必须承认，这一观点极具争议。迄今为止，还没有人构造出这样的计算主体，也尚无理论能够为这样的主体提供充分支撑。但我们坚信，这样的计算主体确实存在，而且人们对它有着巨大的需求，因为许多棘手问题都在等待这样的主体来解决。因此，我们试图拨开这一问题上的迷雾，为进一步讨论奠定坚实基础。我们厘清了一些关键概念，例如非特定化计算与特定化计算、试错、动态搜索等。数划分问题因其自身特性，可以作为这些概念的一个恰当示例。对于这一问题，我们推测这样的计算主体确实存在。请注意，数划分问题是一个 NP完全问题。如果我们能为它找到一个计算主体，那么该主体也能用于解决许多其他棘手问题。现在，方向已经明确。我们将沿着这条路径继续研究，并希望最终找到一个具体的计算主体。

在此，我们想提及特定化计算与非确定性图灵机 [6] 之间的密切关系。我们认为，这种关系是计算理论和人工智能领域中的一个重大问题。人工智能与超越计算复杂度的研究深度交织在一起。事实上，这是一件好事。这种交织将有力地推动人工智能的发展。

我们也可以从另一个角度看待特定化计算与智能之间的关系。如果没有难题，所有计算都可以通过非特定化计算完成，也就不需要智能。面对高度复杂的计算问题，只有特定化计算才有希望解决。然而，只有智能才能使特定化计算成为可能。这就是智能存在且必然存在的原因。我们可以通过智能创造特定化计算的能力来衡量智能。如果我们看到某个计算主体能在困难情境下创造出出色的特定化计算，就能看出它具有智能。

参考文献

[1] A. Yang、J. Wright、Y. Ma 和 S. Sastry，《人脸识别中的特征选择：稀疏表示视角》，2007年。

[2] C. Xiong, 《配分函数的采样与复杂性》, *arxiv.org*, 2021年。[3] S. Aaronson, 《P =? NP》, 2011年。[4] P. Wang, 《人工智能的三个根本性误解》, 第19卷, 第3期, 第249—268页, 2007年。[5] P. Kugel, 《思考可能不只是计算》。[6] S. Cook, 《P与NP问题》, 2000年。[7] C. Xiong, 《关于机器主体性及其功能的若干讨论——ICIS 2020论文集》, 载于*2020*年智能科学国际会议, 印度西孟加拉邦, 2020年。