通过搜索自动化修复可配置信息物理系统中的配置错误：电梯调度算法的工业案例研究

Pablo Valle蒙德拉贡大学蒙德拉贡，西班牙
pablo.valle@alumni.mondragon.edu

Aitor Arrieta蒙德拉贡大学蒙德拉贡，西班牙
aarrieta@mondragon.edu

Maite Arratibel*Orona*埃尔纳尼，西班牙
marratibel@orona-group.com

arXiv:2301.01487v2 [cs.SE] 9 Jan 2023

摘要—**真实世界中的信息物理系统通常是可配置的。通过参数，可以配置、选择或取消选择不同的系统功能。尽管这带来了很高的灵活性，但配置错误也因此成为故障的来源。此类系统拥有大量参数，而由于采用基于仿真的测试，测试执行时间又很长，因此人工修复过程十分繁琐。在这种情况下，自动化修复方法至关重要。本文提出了一种自动修复信息物理系统配置错误的方法。我们通过电梯领域的一项工业信息物理系统案例研究对该方法进行了评估。在真实建筑中开展的实验以及从运行阶段获取的数据表明，该方法优于基线算法和实践现状（即由领域专家进行的人工修复）。**

索引词—**信息物理系统、修复、调试、可配置系统。**

I. 引言

信息物理系统将数字计算与并行物理过程相结合 [1]–[3]。在此类系统中，计算单元、低级和高级软件以及通信协议等数字技术彼此交互，通过传感器和执行器控制物理过程 [1]。实际上，大多数信息物理系统都涉及参数。例如，某重型起重系统涉及 2,000 多个配置参数 [4]。信息物理系统的行为会因这些参数而发生显著变化。这常常会导致配置错误，即使所选参数处于制造商提供的范围内也是如此 [5]。一项近期研究表明，19.6% 的无人机特定漏洞由参数导致 [6]。Garcia 等人 [7] 发现，27.25% 的自动驾驶汽车错误由配置不正确导致。在我们针对电梯系统交通调度算法开展的工业案例研究中，分配给交通团队的问题约有 55% 是通过配置变更解决的。因此，利用自动化且可扩展的技术来自动修复信息物理系统中的配置错误至关重要。然而，这涉及四项核心挑战：

1)挑战 1——测试执行成本高昂：众所周知，执行信息物理系统测试非常耗时，耗时约为 [8]–[15]。这是因为测试是在系统级执行的，而信息物理系统需要使用计算密集型模型来模拟系统的物理部分（例如电动机模型、系统动力学模型）。因此，计算适应度以评估算法距离修复配置错误还有多远，成本十分高昂。例如，在我们的工业案例研究中，执行一个测试用例大约需要 5 分钟。

2) 挑战2——配置空间庞大：由于可配置的信息物理系统涉及许多参数，因此信息物理系统可能拥有的配置数量极其庞大。因此，对所有这些配置进行测试在计算上不可行 [16]–[21]。此外，通常并不清楚导致配置错误的原因（即哪些参数导致了配置错误）。

3) 挑战3——多项需求：可能存在多个未满足的需求。其中一些需求彼此独立 [15],，另一些则可能相互冲突（例如，在我们的案例研究中，降低能耗可能会导致乘客等待更久）。因此，应将修复算法视为一个多目标优化问题。

4)挑战4——优先处理严重故障：修复技术需要按照故障的严重程度依次解决故障 [15]。例如，显示乘客平均等待时间（AWT）为55秒的测试用例，比显示35秒的测试用例更为关键。因此，与其他信息物理系统修复技术 [15],类似，我们的算法应优先处理较不关键的测试用例。

一方面，有些方法旨在修复可配置软件的配置错误 [22], [23] 。然而，这些方法只应对了前文提到的第二项挑战。另一方面，Swarmbug [24]专注于修复群体机器人的配置错误，而群体机器人可视为信息物理系统。不过，Swarmbug [24] 只关注一个特定目标（例如，不发生崩溃），因此无法应对我们的工业案例研究所要求的第三和第四项挑战。

本文提出了一种专门针对信息物理系统配置错误的自动化修复方法。具体而言，我们将配置错误修复问题重新表述为多目标搜索问题，以此应对前文提到的第一项挑战。针对第二项挑战，我们提出了一种采用单种群方法的算法。遗传算法等多种群算法并不适用于此情境，因为修复过程需要与仿真器交互以执行测试用例。这类算法需要较大的种群，而测试执行时间较长会导致算法收敛耗时过多，最终可能造成信息物理系统的可扩展性问题。针对第二项挑战，我们的修复方法采用一种策略来衡量每个参数的可疑度。随着搜索过程的推进，该策略会提高选择可疑参数来生成新补丁的概率。因此，在搜索初期，我们的方法着重探索哪些参数可能是关键参数。随着搜索的推进，算法开始着重利用搜索结果，针对可疑参数进行修复。针对第三项挑战，我们的方法采用基于帕累托最优存档的策略来选择并演化潜在的错误配置补丁。这使修复配置错误时能够同时关注多项需求。针对最后一项挑战，我们会根据搜索目标的严重程度对其进行优先级排序。

我们的主要贡献如下：

1) 我们提出了一种可扩展的自动化方法，用于修复信息物理系统中的配置错误。2) 我们将该方法应用于 Orona 提供的工业案例研究。Orona 是欧洲最大的电梯公司之一。该案例研究涉及交通调度算法，这是一种高度可配置的软件系统。3) 我们通过一个真实场景对该方法进行了实证评估：在这一场景中，Orona 的工程师曾不得不手动介入配置错误修复过程。我们的修复技术不仅优于基线算法，也优于 Orona 领域专家手动编写的修复补丁。4) 我们总结了在工业案例研究中应用该方法所获得的关键经验，并提供了适用性指南，以便推广采用该方法

由其他信息物理系统开发人员完成。

本文其余部分的结构如下：第二节介绍我们的工业案例研究、测试的开展方式以及配置错误产生的原因。第三节介绍我们在工业环境中修复配置错误的方法。第四节介绍我们对该方法的评估方式。第五节提炼关键经验，并解释将该方法应用于其他信息物理系统所需进行的变更。第六节将我们的工作与相关研究进行对比。第七节总结全文并介绍未来工作。

II. 工业案例研究

我们的修复算法应用于电梯领域的一个工业案例研究。本节介绍该案例研究的各项具体细节。

信息物理系统： 图 1 展示了信息物理系统的概况。电梯系统是一种复杂的信息物理系统，其目标是在安全地将乘客从一层运送到另一层的同时，尽可能提供舒适的乘坐体验。在该系统中，乘客通过按下呼叫按钮，在某一楼层登记呼叫。该信息通过控制器局域网（CAN）总线传输至交通主控器。交通主控器收集其他信息物理系统信息（例如每部电梯的位置和电梯载客量）后，会为每个活动呼叫分配一部可用电梯。分配时可以采用不同的目标（例如减少乘客等待时间、降低能耗）。呼叫分配完成后，电梯便会接载乘客。

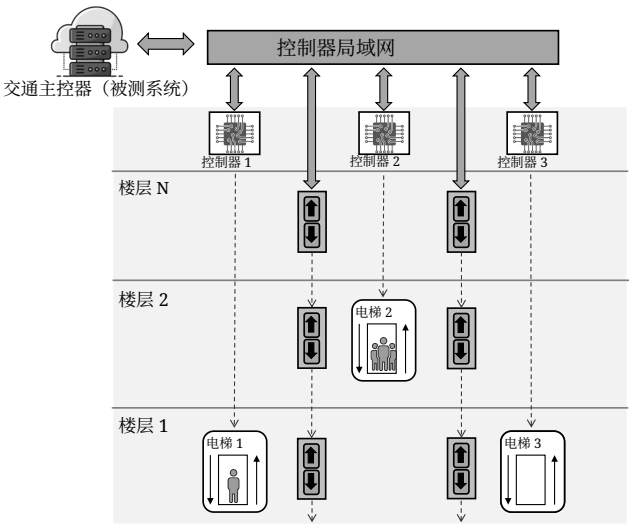


图 1：工业案例研究概况

被测系统（SUT）： 我们的被测系统是交通调度算法（即调度器），它是交通主控器中的一个重要模块。为适应不同的功能和优先级，调度器可通过参数进行高度配置。Orona 中存在多种交通调度算法，每种算法都对应一个配置文件。每种调度器的潜在配置数量超过万亿。

测试执行： 测试调度算法 [25], [26] 时分为三个不同阶段：软件在环 (SiL)、硬件在环 (HiL) 和运行阶段。我们的算法针对第一阶段，即软件在环测试级别。在此阶段，领域专用仿真器 Elevate[1]，接收以下输入：(1) 调度算法的可执行文件，(2) 建筑安装配置，(3) 配置文件，以及 (4) 乘客文件。乘客文件被视为测试输入，其中包含一组

[1]https://peters-research.com/index.php/elevate/

在建筑内不同楼层间通行的乘客。每位乘客都有不同的属性，例如到达时间（即乘客抵达楼层并按下按钮的时间）、到达楼层（即乘客抵达的楼层）、目的楼层（即乘客前往的楼层）、乘客体重等。执行测试时，Elevate 会返回一个包含仿真结果的文件（例如每位乘客所需的等待时间、乘客的行程时间、能耗以及每部电梯行驶的距离）。系统会解析这些信息，并使用必要的测试预言机来评估测试执行的质量。

功能性能需求：执行测试用例时，除了考虑某些功能需求外，我们还关注“功能性能需求”。功能性能定义为“间接从系统输出中得出的属性，而非系统对计算资源的高效使用”[26]。这些属性直接用于评估 Orona 调度算法的功能性能需求。这些属性涉及电梯交通领域的度量指标，例如乘客的平均等待时间（AWT）、乘客的平均乘行时间（ATT）、最长等待时间（LWT）、最长乘行时间（LTT）、电机启动次数、每部电梯的行驶距离或能耗。需要注意的是，配置变更会影响功能性能需求，而功能需求（例如确保不会发生反向行程）原则上不会受到此类变更的影响。

配置错误为何发生以及如何处理：调度器具有不同的参数，以适应不同的功能，这些功能会直接影响信息物理系统的性能。然而，值得注意的是，某种配置在一套电梯安装系统中可能表现良好，在另一套系统中却表现不佳，从而导致配置错误。这是因为电梯系统的性能在很大程度上取决于（1）建筑的类型及其组成，以及（2）其交通流量。就前者而言，性能可能因建筑内电梯的数量、建筑的楼层数、电梯是否服务于所有楼层等因素而异。对于某些类型的建筑，有些配置比其他配置更合适。至于后者，不同类型建筑的交通流量也各不相同。例如，医院和住宅建筑的交通流量截然不同。在医院，楼层间行程很常见；而在住宅建筑中，大多数呼叫都是从基准层到公寓所在楼层，反之亦然。当电梯系统表现不佳时，会在软件在环测试级别重现其交通流量，以便调试并尝试通过更改参数来提升其性能。如果一组新参数提升了系统性能，那么原始配置就被视为配置错误。需要注意的是，在我们的工业案例研究中，由于信息物理系统暴露于不确定性之中 [27], [28]，配置错误可能在系统运行阶段开始之前既未被检测到，也无法预见。

III. 信息物理系统配置错误修复方法

算法 1 概述了我们的修复算法。该算法的输入包括：（1）一个错误的配置文件 C，由 N 个参数组成，即 $C = \{p_1, p_2, ..., p_N\}$；以及（2）一个测试套件，由 M 个失败测试用例组成，即 $TS = \{tc_1, tc_2, ..., tc_M\}$。算法的第一步是评估错误的配置文件，解析所有参数值（第 1 行），并执行所有测试用例（第 2 行）。执行失败测试套件后，使用预言机返回的值初始化存档（第 3 行），并初始化参数的可疑度分数（第 4 行）。之后，算法进入 while 循环（第 5 至 11 行），直到满足终止条件。终止条件包括：（1）修复配置错误，或（2）超过运行时间。

算法 1： 基于搜索的修复概述算法

输入： C //故障配置文件
测试套件 //测试套件
输出： 存档 //包含改进配置的存档

1 $Patch_0 \leftarrow$ getValues(C);
2 初始得分$\leftarrow$ 执行测试套件($Patch_0$, 测试套件);
3 存档 $\leftarrow$ 保存到档案($Patch_0$, 初始得分);
4 可疑度 $\leftarrow$ initSusp();
5 **当***terminationCriteriaNotMet* **执行**
6 父级 $\leftarrow$selectAParentArchive(Archive);
7 $Patch_1 \leftarrow$ 生成补丁(父级,可疑度);
8 得分 $\leftarrow$ executeTestSuite($Patch_1$, TS);
9 可疑度 $\leftarrow$ updateSusp($Patch_1$, Parent, Score, 父级得分);
10 档案 $\leftarrow$ 保存到档案($Patch_1$, 得分);
11 **结束**
12 **返回**档案;

在这个 while 循环中，第一步是从档案中选择一个解（第6行），该解将作为父级。解是完全随机选定的。基于选定的解，提出一个潜在补丁（第7行），包括更改父级解中的一个或多个参数（第三节-A）。通过执行失败测试套件来评估此补丁（第8行），并获取测试执行结果，将其存储为得分（第三节-B）。第四步，重新计算每个参数的可疑度分数（第9行，第三节-C）。最后，更新档案（第10行，第三节-D）。

A. 补丁生成

在我们的语境中，补丁是指至少一个参数的变异。算法 2 展示了用于生成潜在补丁的算法。其输入包括：(1) 一个父级配置，对应于算法存档中的一个配置；(2) 所有参数的可疑度排名。首先，根据

每个参数的可疑度，选定一个待变异参数（第 4 行；有关如何计算可疑度分数的更多详情，请参见第 III-C 节）。可疑度越高，被选中的概率就越大。待变异参数通过算法3确定。通过在其取值范围内随机赋值，对选定参数进行变异（第5行）。之后，决定是否对另一个参数进行变异（第8行）。随着新补丁中已变异参数数量的增加，对新参数进行变异的概率会降低。我们确保每个参数最多只变异一次。

算法 2： 补丁生成算法

输入： 父级 //故障配置
可疑度排名 //可疑度排名
输出： 补丁 //变异配置
1 变异参数数量 ← 0;
2 补丁 ← 父级;
3 **do**
4 varToMutate ← 选择参数(可疑度排名);
5 补丁 ← 变异(补丁,varToMutate);
6 变异参数数量 ←变异参数数量 +1;
7 p ← 随机数生成(); //返回 *0* 到 *1* 的随机值
8 **当** $p < 0.5^{numOfMutatedParams}$;
9 **返回**补丁;

算法3： 基于可疑度的参数选定算法

输入： 可疑度分数 = $\{ss_1, ss_2, ..., ss_N\}$
输出： 选定项 //选定参数的索引
1 总计 ← $\sum_{i=1}^{N}(ss_i)$;
2 迭代和←0;
3 概率 ← [];
4 **对于***i* ← *1* **到***nPop* **执行**
5 概率[i] ← 迭代和 + 可疑度分数[i]/总数;
6 迭代和←概率[i];
7 **结束**
8 概率←升序排列(概率);
9 r←随机数生成();//返回 *0* 到 *1* 之间的随机数
10 j←0;
11 选定项=N;
12 **当***j*<*N* 且选定项==*N* **执行**
13 **如果** *r*<概率*[j]***则**
14 选定项←j;
15 j←j+1;
16 **结束**
17 **返回**parameter(selected); //转换索引
选定项到参数 *ID*

B. 测试套件执行

生成补丁后，需要对其进行评估。我们通过重新执行测试套件中所有失败的测试用例来评估每个补丁。我们不执行通过的测试用例，因为执行这类测试用例会显著增加我们方法的计算时间。此外，为了提高修复算法的效率，测试用例的执行过程采用并行化处理。执行测试套件时，测试预言机会评估系统的性能。在我们的语境中，与其他方法类似 [8]–[11], [14], [15],[29],，测试预言机不仅会给出布尔判定（即通过或失败），还会给出置信度。置信度越低，信息物理系统在此类测试预言机所评估属性方面的性能就越低。

这些预言机的置信度用作搜索目标，以引导修复算法找到有效补丁。修复信息物理系统时，可能存在总计 k 个测试预言机。每个 k 预言机在修复算法中都充当一个独立的目标函数。对于失败测试套件（TS）中的每个测试用例（tc），每个 k 预言机都会返回其置信度，即 $Conf(tc, o_i) \in [-1, 0]$，其中 o_i 是第 i个预言机。-1 表示故障严重程度达到预设的最高级别，而 0 表示预言机已通过。修复算法旨在最大化该置信度。因此，执行完 TS中的所有测试用例后，与 Abdessalemm 等人类似， [15], 我们会获得每个测试预言机的最小值（即最严重值），从而将修复问题转化为一个多目标优化问题，并优先处理最严重的故障，使得:

$$\begin{cases} \max Oracle_1(Patch) = \min\limits_{tc \in TS}\{Conf(tc, o_1)\} \\ ... \\ \max Oracle_k(Patch) = \min\limits_{tc \in TS}\{Conf(tc, o_k)\} \end{cases} \quad (1)$$

如前所述，在信息物理系统的语境下执行测试非常耗时。此前针对计算密集型信息物理系统的研究利用代理模型加速测试用例的生成 [8], [10], [13], [14]。也就是说，在完成一组测试执行后，使用测试结果训练模型，并以该模型替代基于仿真的测试执行，从而加快测试用例的生成。尽管我们曾考虑使用代理模型加速修复过程，但发现要获得可靠的代理模型需要进行过多仿真。不同于仅使用测试输入维度的先前方法 [8], [10], [13], [14], , 可配置信息物理系统还需要考虑参数维度，这使得代理模型更难训练。我们在工业案例研究中进行了初步评估，发现获取用于构建可靠代理模型的数据所需的时间，与修复算法收敛所需的时间相当，甚至更长。因此，我们放弃了使用代理模型加速修复过程的方案。

C. 衡量参数可疑

通过分析工业案例研究中的行为并采访领域专家，我们注意到，相较于其他参数，某些参数对系统性能的影响更大。因此，我们在 C 中实现了一种机制，用于衡量每个参数的可疑度。

可疑度以 0 到 1 之间的得分表示；可疑度越高，该参数影响系统性能的可能性就越大。该策略的最终目标是提高对那些会影响系统性能的参数进行变异的概率。

所有可配置参数的初始可疑度得分均为 0.5。在参数被补丁生成算法变异 N_{susp} 次之前（我们在实验中采用了 N_{susp} = 5 ），该可疑度保持不变。这样，算法便能在搜索过程开始时专注于探索阶段，并随着搜索过程的推进转而专注于利用。每当补丁生成算法变异某个参数后，我们都会在评估补丁之后判断该参数是否（1）对系统性能产生了正面影响，（2）对系统性能产生了负面影响，或（3）完全没有影响。若根据测试结果，该参数对应的补丁未被系统中的任何其他补丁支配，则认为该参数 p_i 产生了正面影响。若该补丁被存档中的解（即包括其原始父代）支配，则认为该参数 p_i 产生了负面影响。当测试结果表明性能与原始父代相同时，则认为该参数 p_i 没有产生任何影响。在某个参数 p_i 被选定 N_{susp} 次后，便开始按如下方式计算其可疑度：

$$susp(p_i) = \frac{P_{p_i} + N_{p_i}}{P_{p_i} + N_{p_i} + S_{p_i}} \tag{2}$$

其中，P_{p_i} 是参数 p_i 产生正向影响的次数，N_{p_i} 是参数 p_i 产生负向影响的次数，而 S_{p_i} 是参数 p_i 完全没有产生影响的次数。

请注意，正向或负向影响都会提高特定参数的可疑度。这是因为补丁是通过在参数的取值范围内，将其值变异为另一个随机数生成值而提出的。因此，先前产生负向影响的参数的另一个取值，可能会对信息物理系统（CPS）的性能产生正向影响。

根据我们的分析，即使具备领域专业知识，信息物理系统中参数的可疑度原则上仍是未知的。这在很大程度上是因为信息物理系统高度依赖其运行环境。例如，在我们的工业案例研究中，参数对信息物理系统性能的影响可能因建筑类型而异（例如，在有 2 部电梯的住宅建筑中，参数的表现可能与在有 4 部电梯的医院建筑中不同）。因此，我们假设事先并不了解某个参数在信息物理系统环境中的影响。不过，我们用于衡量参数可疑度的方法可以轻松扩展至其他策略（例如，为算法提供配置中每个参数的初始可疑度分数）。

D. 更新档案

我们的算法使用一个存档，其中包含通过纳入补丁而生成的非支配解。存档中第一个被更新的配置文件是导致故障的错误配置。在通过执行失败测试套件评估补丁（$Patch_1$）后，需要更新存档。该补丁将与存档中的其余解进行比较。比较基于支配关系的概念，与其他研究 [15], 类似，存档按以下方式更新：

1) 如果 $Patch_1$ 支配存档中的至少一个解，则将$Patch_1$ 纳入存档，并移除被支配的解。2) 如果存档中没有任何元素支配 $Patch_1$，且$Patch_1$ 也未被存档中的任何解支配，则将 $Patch_1$ 纳入存档。3) 如果 $Patch_1$ 被存档中的至少一个解支配，则存档保持不变。

遵循这一策略存在一定风险：存档的规模可能会增大。这会导致算法需要更多时间才能收敛。为解决这一问题，如果存档超过一定规模，我们就会从中移除解。与 Abdessalem 等人一样， [15], 我们的存档最大规模限制为 $2 \times k$, k 其中 k 为预言机的数量。不过，与 [15], 不同，我们不是随机移除存档中的解，而是移除平均等待时间（AWT）最长的解。之所以这样决定，是因为在电梯领域，平均等待时间是评估调度算法性能的主要度量指标 [30]。如果两个或多个解的 AWT 同为最高，则在这些解中随机选择一个。

E. 停止准则

修复过程在满足以下两个准则之一时停止：(1) TS 中的所有测试用例均通过，即所有测试用例中的预言机都返回值 0；或 (2) 超出搜索预算（即修复时间超出限制）。如果发生后一种情况，可能是测试用例要求过高。因此，修复过程将转变为一个参数优化问题。例如，在分析我们针对电梯调度算法开展的工业案例研究时，我们发现，一些测试输入可能会在较短的时间窗口内包含过多的乘客呼叫。在这种情况下，信息物理系统可能进入饱和状态，唯一的解决办法是增加额外电梯，以便更好地响应呼叫，而这超出了调度算法的能力范围。

F. 决策者

当修复算法因超出搜索预算而停止时，存档中很可能存在多个解。在这种情况下，决策者（DM）需要依据一定规则从中选出一个解，并将其作为补丁提出。在本案例中，该决策者是领域特定的。这个决策者是一个基于规则的算法，其设计过程中纳入了领域专家的意见。指定的阈值

是可配置的，因为某些阈值在某些建筑中可能有效，但在其他建筑中则不然。该算法按照以下流程决定提出哪个补丁：

1) 首先，选出所有 AWT 小于 25 秒的补丁。这是国际标准认定的电梯系统良好性能阈值 [31]。由于 AWT 是评估电梯系统质量时最广泛采用的度量指标 [30]，，我们优先考虑这一指标。如果没有解满足该需求，我们就选择 AWT 最低的补丁。

2) 如果还有多个补丁，决策者会优先考虑测试执行中等待时间超过 55 秒的乘客人数较少的补丁。规定该比例应低于 10%，这是一个可接受的数值。领域专家认为，等待近一分钟是一种反模式，因此他们决定优先选择等待超过 55 秒的乘客人数较少的解。3) 第三阶段，如果候选解决方案集合中仍有多个补丁，则优先考虑 ATT 指标。决策者选择 ATT 低于 45 秒的解。如果没有解满足该需求，我们就选择 ATT 最低的补丁。4) 如果仍有多个补丁候选项，决策者会选择测试执行中乘行时间超过 70 秒的乘客人数较少的解。规定该比例应低于 10%，这是一个可接受的数值。5) 此后，如果仍有多个候选补丁，决策者会选择 LWT 最低的补丁，因为 LWT 被认为比 LTT 更重要。如果仍有多个补丁，则选择 LTT 最低的补丁。尽管这种情况极少发生，但仍有可能出现多个解。在这种情况下，会使用众所周知的汉明距离度量指标，将候选补丁的配置文件与原始配置文件进行比较，选择相似性最高的补丁。原因有二。一方面，工程师通常不愿意更改原始配置文件中的过多参数，因为适用于某些客流的配置，可能并不适用于其他客流。另一方面，我们推测，更改的参数越多，解就越有可能过拟合失败测试套件。因此，我们通过这一机制，力求降低合理补丁过拟合的概率。

G. 补丁确认

由于我们只使用失败测试套件来修复错误配置，因此需要重新测试补丁。这样可以确保补丁没有针对失败测试套件过拟合，这是自动化程序修复的核心问题之一 [32]–[35]。可以采用任何一种先进技术来完成此项工作。在我们的案例中，我们使用回归测试预言机 [26]，并执行合成测试输入（即基于全天理论乘客分布模板的测试输入 [36] ，以及国际电梯标准建议的上行和下行高峰分布 [31]）。我们确保新补丁的表现不逊于原补丁。此外，我们按照 Ayerdi 等人 [37], [38]的建议，使用较短的测试用例进行变形测试，以验证其功能。

IV. 评估

在评估中，我们旨在回答以下两个研究问题（RQ）：

- ***RQ1*** – 合理性检查： 我们的方法与基线相比如何？为评估待解决的问题是否微不足道，第一个研究问题是一次合理性检查。为此，我们实现了修复算法的无引导版本。
- ***RQ2*** – 与实践现状的比较： 我们的方法与领域专家执行的人工修复相比如何？ Orona 当前的做法是人工修复配置错误。该研究问题旨在比较我们的算法是否能够胜任与公司领域专家执行的人工修复过程相同的工作。

A. 实验设置

1) 被测系统与建筑： 我们使用了 Orona 的常规群控（CGC）交通调度算法 [30]，，该算法也曾用于其他研究 [27], [28], [37]–[39]。此外，我们使用实际安装环境来评估所提出的方法。该安装系统共涉及三台电梯和 12 个楼层。我们选择这一安装系统，是因为这是一个实际案例：Orona 曾不得不手动介入以解决配置错误。此外，工程师采取的手动配置错误处理过程有详尽记录。我们还能够获取来自冲突安装系统的运行数据，并将其用作失败测试输入。我们总共使用了三个会导致故障的测试输入，每个输入包含 16 小时的客流，共涉及 3,105 至 3,769 名乘客。

我们使用的算法版本共涉及 43 个参数。潜在配置总数超过 9.3×10^{92}，因此搜索空间过大，无法采用暴力搜索。

2）测试预言机：通过仔细分析 Orona 用于解决安装冲突的内部文档，我们根据其旨在优化的度量指标定义了六个预言机。下面我们将介绍选定的功能性能指标：

- 平均等待时间（AWT）：衡量所有乘客的平均等待时间。等待时间是指从乘客登记呼叫到电梯抵达并响应呼叫所经历的时间。

- 最长等待时间（LWT）：衡量乘客经历的最长等待时间。

- 等待时间（WT）超过55秒的乘客比例：用于衡量等待时间超过55秒的乘客所占的百分比。

- 平均乘行时间（ATT）：用于衡量所有乘客的平均乘行时间。乘行时间是指从乘客进入电梯到电梯抵达其目的地所经历的时间。

- 最长乘行时间（LTT）：用于衡量所有乘客中的最长乘行时间。

- 乘行时间（TT）超过70秒的乘客比例：用于衡量乘行时间超过70秒的乘客所占的百分比。

在修复这一配置错误时，领域专家希望尽可能提高上述功能性度量指标。因此，在本研究中，我们选择对阈值采取较为激进的设定，将所有阈值均设为0。我们承认，这些数值实际上无法达到。不过，这样可以使其与人工方法的比较更加公平。此外，我们还希望评估决策者（DM）选定的补丁。

3）执行平台：使用 Elevate 8.19 作为仿真器来执行测试。实验在一台装有 Windows 10 操作系统的 PC 上进行，该 PC 配备第 7 代 Intel Core i5 CPU 和 16 Gb RAM。

4）基线算法与实践现状对比：我们开发了所提修复算法的无引导版本，作为基线算法。无引导版本与本文提出的修复算法有两项核心区别：（1）无引导版本会将所有配置保存到存档中；（2）它认为所有待变异参数的可疑度分数都相同（即此版本不测量可疑度）。值得注意的是，与纯随机搜索（RS）相比，这一基线更为强大；纯随机搜索（RS）通常是评估基于搜索的软件工程问题时使用的基线算法 [17], [19], [40]–[43]。这是因为，随机搜索会以初始失败配置为基础，采用我们的补丁生成方法（算法 2 ）提出一些补丁。然而，在随机搜索中，这些生成的补丁之后不会继续演化。相反，在我们的无引导方法中，我们允许存档中的补丁继续演化，从而提高找到补丁的概率。

关于与实践现状的比较，对于所使用的建筑安装配置，我们获得了 Orona 工程师提供的数据。具体来说，问题提出后，Orona 的工程师提出了不同的潜在补丁（即调度器的不同配置）。我们将算法得出的结果与领域专家提出的补丁进行了比较。Orona 的工程师共提供了六种不同的补丁。

5) 评估指标：由于我们的算法符合帕累托原则，因此必须对存档中的所有解进行整体评估。基于这一点，并参考相关指南 [44],[45],，我们采用了超体积（*HV*）质量指标。超体积是评估基于帕累托的搜索算法时最广泛使用的度量指标之一，基于帕累托的搜索算法 [44]–[46]。超体积用于衡量搜索算法在目标空间中覆盖的体积，并具有诸多优势 [46],，例如：（1）符合帕累托原则；（2）能够同时评估解集的收敛性和多样性；（3）只需要一个参考点。

除了 超体积 质量指标外，由于我们设计了一个决策者，我们还比较了搜索预算耗尽后，决策者为解所提出的方案中，作为测试预言机性能指标的六个目标函数。

6）统计检验：由于所采用的算法具有非确定性，我们对每种算法运行了 10 次。鉴于搜索预算设定为 12 小时，我们无法承担更多次运行。因此，实验总共耗时 10（次运行）$\times$ 12（小时）$\times$ 2 （基线算法和修复算法） = 240 小时。

为评估统计显著性，我们采用了 Wilcoxon 秩和检验。当 p值低于 0.05 时，我们认为两种算法之间存在统计显著性。此外，我们采用了 Vargha 和 Delaney A_{12} 值，该值衡量一种技术优于另一种技术的概率。

7）算法配置：我们为本文的算法和基线算法都设定了 12 小时的时间预算。与 [15],类似，存档中的最大解数设为 12（即，6 个目标函数 $\times$ 2）。我们还设置了参数 $N_{susp} = 5$，这意味着在某个参数被变异 5 次之前，其可疑度保持中性（即，可疑度分数为 0.5）。

B. 结果分析与讨论

1）*RQ1* – 合理性检查：图 2 展示了本文提出的修复算法与基线算法（即修复算法的无引导版本）各自 10 次运行的平均超体积得分。如图所示，经过第二个执行小时后，修复方法的平均超体积高于基线。搜索预算耗尽时，修复算法在超体积质量指标方面比基线平均提高了 29%。值得注意的是，超体积值相当低。这是因为超体积偏重帕累托前沿 [45]上解集的膝点。如前所述，在我们的案例中，指定的阈值为 0（即，修复算法旨在尽可能优化所有功能性能指标）。达到这一数值并不现实，因此我们没有膝点。此外，我们采用了 6 种不同的预言机（即适应度函数）来引导搜索生成补丁。尽管超体积值较低，但这并不会使两种技术之间的比较失去公平性，而这正是第一个研究问题的目标。

这些结果还通过统计检验得到了进一步验证。表 I 列出了在将修复算法与基线进行比较时，12 个小时中每个小时的 $\hat{A}_{12}$ 以及 p 值（采用 Wilcoxon 秩和检验计算）。$\hat{A}_{12}$ 显示修复算法优于其无引导版本的概率。根据 Romano 等人

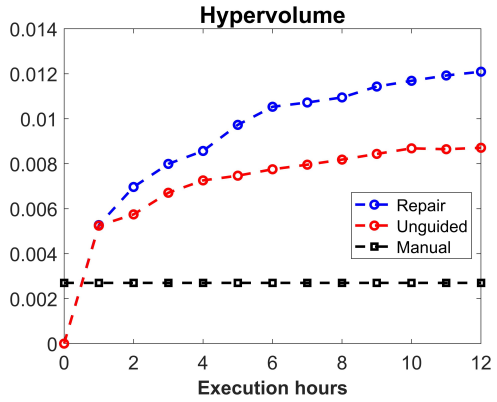


图 2：比较修复算法与无引导算法时，10 次运行的超体积质量指标平均值

等人的建议， [47],我们将修复算法与其基线之间的差异分类如下：若 $d < 0.147$，则为可忽略；若 $d < 0.33$，则为小；若 $d < 0.474$，则为中；若 $d >= 0.474$，则为大，其中 $d = 2|\hat{A}_{12} - 0.5|$。根据这一分类，执行的第一个小时内差异可忽略，第二和第三个执行小时之间差异较小，第四个执行小时内差异中等。在最初的四个执行小时内，修复算法与基线之间不存在统计显著性。相反，从第五个小时起，差异具有统计显著性（即，p 值 < 0.05），且根据相关分类，效应量较大 [47],，这些结果均有利于我们的方法。

表 I：RQ1——比较修复算法与其无引导版本在运行 12 小时期间的超体积质量指标统计检验汇总。$\hat{A}_{12}$ 值高于 0.5 表示结果有利于修复算法。统计显著性设定为 *p值<0.05*

Hour	$\hat{A}_{12}$	p值
1	0.51	0.9698
2	0.61	0.4273
3	0.65	0.2730
4	0.71	0.1212
5	0.80	0.0256
6	0.86	0.0081
7	0.89	0.0040
8	0.82	0.0172
9	0.85	0.0090
10	0.85	0.0090
11	0.90	0.0028
12	0.92	0.0017

除了 超体积，我们还分析了决策者（DM）提供的单个补丁。在这种情况下，算法的目标是降低这些度量指标。因此，$\hat{A}_{12}$ 低于 0.5 意味着修复算法表现更好。表 II 总结了十次运行中各个目标函数的统计检验。半数目标函数（即 LWT、ATT 和 LTT）的结果具有统计显著性。对于这些情况，效应量较大（即，$\hat{A}_{12}$ 介于 0.18 到 0.2 之间）。对于其余未达到统计显著性的目标，AWT 和 %WT>55 的效应量显示差异可忽略不计，而 %TT>70 的差异较小。

表 II：决策者采用修复算法时，将其提供的补丁与基线和人工修复方法进行比较的统计检验结果汇总

	与基线相比		与人工修复相比	
	$\hat{A}_{12}$	p值	$\hat{A}_{12}$	p值
AWT	0.52	0.9097	0.10	0.0014
LWT	0.18	0.0165	0.20	0.0161
%WT>55s	0.47	0.8788	0.00	<0.0001
ATT	0.20	0.0312	0.40	0.4429
LTT	0.20	0.010	0.00	<0.0001
%TT>70s	0.37	0.3438	0.00	<0.0001

表 III 展示了预言机在 10 次运行中使用的各项功能性能指标的平均值，以及决策管理器提供的补丁。这些结果与表 II 中的结果在某种程度上保持一致。显而易见，最显著的差异涉及 LWT 和 LTT 功能性能指标。相比之下，AWT、%WT>55、ATT 和 %TT>70 的差异并不大。这可能与决策者的性质有关。请注意，对于这些度量指标，决策者接受低于某些阈值的值（例如，AWT < 25 秒），而对于 LWT 和 LTT，决策者会选择数值最低的补丁。不过，除 AWT 外，我们的算法在所有度量指标上都显示出较低的平均值。

表 III：配置错误的配置文件、决策者通过人工修复提供的补丁、决策者针对基线算法返回的补丁平均值，以及决策者针对修复算法返回的补丁平均值之间的比较

	错误配置	手动	基线决策者	修复决策者
AWT	25.99	23.10	22.66	22.77
LWT	435.70	223.00	241.55	213.72
%WT >55秒	12.78	11.99	9.93	9.92
ATT	42.01	41.60	41.77	41.58
LTT	209.80	220.60	206.24	195.56
%TT>70秒	10.24	10.02	9.64	9.45

In c综上所述，第一个研究问题可以回答如下：：

对第一个研究问题的回答：修复算法的表现优于基线算法。以超体积质量指标衡量时，修复算法相较于基线算法的平均改进幅度约为 *29%*。此外，在一半的目标函数上，对决策者提出的单个补丁进行比较时，结果均具有统计显著性且效应量较大，且全部有利于修复算法。所有这些都表明，修复信息物理系统配置错误并非易事，因此需要自动化且可扩展的修复技术。

2) 研究问题 *2*——与人工修复的比较：在研究问题 2 中，我们旨在将所提出的修复算法

与领域专家手动修复配置错误的流程进行比较。具体而言，领域专家总共提供了 6 个补丁。利用这些补丁，并应用我们算法中的六个预言机，我们得出了超体积度量指标。如图 2 所示，超体积相当低。这是因为只有四个补丁属于非支配补丁，而我们的存档最多可处理十二个补丁。因此，这四个补丁无法覆盖目标空间中的较大范围。此外，值得注意的是，此处未考虑时间，因为我们没有相关信息。就超体积而言，与人工生成的补丁相比，我们的修复算法平均最多提升了 77.5%。

对于此案例，我们也使用决策者从非支配补丁中选出一个。表 II 展示了统计检验结果：将执行修复算法后由决策者提供的补丁，与决策者处理四个非支配解后提出的补丁进行比较。可以看出，在六项度量指标中，有五项存在统计显著性；按照 Romano 等人提出的分类方法，效应量显示差异较大 [47]。这些效应量均有利于修复算法。另一方面，对于不存在统计显著性的情况，即 ATTˆ 度量指标，A_{12} 值方面的差异较小，但仍有利于修复算法。

表 III 展示了决策者提供的补丁（运行 10 次）相较于人工方法，在各项功能性能指标上取得的改进幅度。这些结果与统计检验结果一致：可以看出，ATT 的平均值相近。在此情况下，与基线算法相比，AWT、% WT $>$ 55、LTT 和 %WT>70 的改进幅度更大。然而，就 LWT 而言，改进平均仅为 10 秒；相比之下，基线算法的改进平均接近 29 秒。

总之，第二个研究问题可以回答如下：

第二个研究问题的答案：修复算法优于人工修复过程。以超体积质量指标衡量时，修复算法相较于领域专家提供的补丁，平均改进幅度约为 *77.5%*。此外，在六个目标函数中，有五个目标函数在比较决策者提出的单个补丁时，结果具有统计显著性，且效应量较大。另外，我们的方法实现了全自动化，因此在处理交通调度算法的错误配置时，可以提高 *Orona* 工程师的工作效率。

C. 效度威胁

现在，我们总结本研究的效度威胁以及为减轻这些威胁所采取的措施。

我们评估中的一种内部效度威胁 可能与算法中使用的参数有关，这些参数并未更改。需要配置三个主要参数：(1) 时间预算，设为 12 小时；(2) 在开始计算某个参数的可疑度分数之前，该参数需要被选定的次数（即，N_{susp}），设为 5；以及 (3) 存档中的解的数量。前两个参数是根据初步评估选定的。相反，存档中解的最大数量与其他面向信息物理系统的修复方法相同 [15]。

与任何基于搜索的软件工程问题一样，结论效度威胁涉及所用算法的随机性。为减轻这一问题，我们将每种算法运行了 10 次。值得注意的是，由于用于评估潜在补丁的仿真是穷尽式的，我们的技术需要较长时间才能收敛，因此无法承担大量运行次数。此外，我们按照 Arcuri 和 Briand [48] 的建议，应用了统计检验来分析结果。

与任何涉及人类的研究一样，我们的评估也面临外部效度威胁。其中一种威胁与 Orona 工程师提出的补丁有关。不过值得注意的是，这些工程师拥有丰富的经验和领域专业知识，而且他们提出的补丁后来确实部署到了真实信息物理系统中。研究结果的普遍适用性也是本研究面临的另一项外部效度威胁；但请注意，我们采用了工业案例研究，使用了实际安装环境以及从运行阶段获取的数据。未来，我们计划通过以下方式降低此类威胁：(1) 使用来自不同领域的其他案例研究；(2) 使用发生过配置错误的其他实际安装环境。

最后，当所用测量指标在不同算法之间不可比较时，就会产生构念效度威胁 。我们通过为两种算法（即修复算法和无引导算法）提供相同的搜索预算，降低了这一威胁。

V. 经验总结与适用性

本节将介绍我们在开发和评估修复算法的整个过程中获得的经验。此外，我们还将说明，将该方法应用于其他信息物理系统领域时需要进行的主要调整。

A. 经验教训

经验教训 1——降低人员成本：当前修复配置错误的实践现状完全依赖人工。这需要投入大量人员成本，因为整个过程中都需要领域专家参与。我们提出的全自动修复方法不仅在提供更优补丁以修复配置错误方面优于实践现状，还显著降低了人工修复过程所需的人员成本。

课程 2 – 可扩展技术：可扩展性是测试和调试信息物理系统时的主要关注点之一，主要是因为需要考虑涉及具有连续动态的物理设备，以及系统与其环境（例如，人）之间复杂并发交互的属性 [49]。我们看到，我们基于搜索的修复算法

大约 10 小时后收敛，这对我们的工业合作伙伴来说是可以接受的，因为工程师可以在夜间启动自动配置错误修复工具。

经验 3——原则上，代理模型并不适用：尽管我们并未对此进行仔细评估，但在开发算法时，我们原本打算集成代理模型，以加速修复过程。然而，我们发现，这项技术需要耗费过多时间来构建可靠的代理模型，而这段时间与修复算法收敛所需的时间预算相近。尽管我们评估了不同类型的代理模型，但仍需对此进行更仔细的分析，这将留待未来工作。

经验 4——应对具有挑战性的冲突安装系统：在完成实验并向 Orona 的工程师展示结果后，我们注意到所选的冲突安装系统颇具挑战性。事实上，其交通流量异常，存在许多意料之外的情况（例如，在较短的时间窗口内出现过多呼叫），因此，据领域专家所述，修复此类安装系统中的配置错误，比修复其他安装系统中的配置错误更加困难。

B. 适用性

我们应用修复方法的场景是 Orona 的电梯调度算法。不过，我们认为，我们所应对的三项关键挑战（即测试执行成本高、配置空间庞大以及功能性能需求多样）是所有可配置信息物理系统的共同挑战。由于我们在开发修复方法时纳入了领域专家的意见，因此考虑了若干特定于领域的设计选择；若要将该方法应用于其他领域，则需要进行相应调整。下面我们将介绍不同的替代方案，以及在其他领域采用本方法所需的更改。

测试执行过程：我们的方法需要进行的首要调整之一是测试执行。如第三节-B所述，我们使用领域专用仿真器执行测试用例，并衡量算法距离修复配置错误还有多远。在其他信息物理系统中，这一过程需要替换为用于执行测试的仿真器。此外，我们采用了并行测试执行，这在我们的场景中是可行的。然而，其他仿真器（例如，用于自动驾驶车辆的仿真器）可能需要更多计算资源。例如，测试自动驾驶车辆通常需要使用高保真模拟器在虚拟场景中渲染驾驶场景 [13], ，这可能要求按顺序执行测试用例。最后，还需要定义测试预言机。使用 Simulink 模型执行测试时，作为一种主流的信息物理系统测试工具 [50], ，可以选择使用 SOCRaTEs [11],，这是一种面向 Simulink 的基于领域特定语言的测试预言机规范与生成工具。具体而言，SOCRaTEs [11] 能够定量衡量需求的违反程度，这与我们的算法引导配置错误修复过程所需的度量类似。

从存档中移除解：如第三节-D所述，存档的规模可能会增大，这可能会直接影响修复算法的收敛性。因此，当存档中的解超过预先设定的数量时，就需要移除其中一个解。我们的算法会移除 AWT 最大的解，因为在测试调度算法时，这是使用最广泛的度量指标 [30]。在其他领域，可以考虑两种替代方案。第一种是采用最重要的度量指标之一。如果所有度量指标的重要性相近，第二种方案则是从存档中随机移除一个解，或者使用拥挤距离移除彼此过于接近的解。

决策者：决策者是我们根据领域专家的建议，专门为交通调度算法开发的另一个组件。我们建议分析特定信息物理系统的优先级，以便做出决策。如果没有明确的优先级，一种解决方案是采用加权方法，让所有目标具有同等重要性。

补丁确认：我们仅使用失败测试套件来指导修复过程。核心原因是测试执行时间长。最终，可能会出现这样的情况：提出的补丁导致通过测试套件中的某个测试用例失败。因此，我们按照传统的回归测试方法 [26]，结合 Orona 新引入的变形测试方法 [37], [38]，实现了补丁确认流程。补丁确认模块应遵循内部标准化的测试方法，而这种方法可能因公司而异。

VI. 相关工作

自动化程序修复领域的相关工作浩如烟海。Monperrus 持续维护着一份关于此类技术的综述 [51]。表 IV 总结了相关工作的分类，并分析了我们的方法所涵盖的四项关键特征。第一项特征（**C1**）分析了修复计算开销高昂的系统的可能性。第二项特征（**C2**）考察该方法是否旨在修复配置错误。第三项特征（**C3**）分析该方法能否处理众多需求（即超过 3 项）。最后一项特征（**C4**）考察该方法是否会优先处理关键故障，而非不那么关键的故障。

表 IV：相关工作与我们的修复技术所需不同特性的对比

	C1	**C2**	**C3**	**C4**
[15]	+	-	+	+
[22], [23]	-	+	-	-
[24]	+	+	-	-
[52]–[56]	-	-	-	-
[57]–[64]	+	-	-	-

我们发现，在信息物理系统领域，修复方法仍处于起步阶段。事实上，据我们所知，只有两种方法解决了信息物理系统的修复问题。一方面，Swarmbug [24] 专注于修复群体机器人的错误配置。具体而言，他们采用一种名为“因果贡献度”的机制，通过行为因果分析来抽象配置对群体无人机的影响

。该方法在四种群体算法上进行了评估，并为每种算法分别设定了修复目标。这些目标涉及避免障碍物或危险区域等方面，以免无人机发生碰撞。然而，该方法未涵盖**C3**和**C4**。另一方面，Ariel [15]专注于修复自动驾驶系统中的特征交互故障。与我们的方法类似，ARIEL [15] 采用多目标、单种群方法，并使用存档跟踪部分修复的解。然而，与本文侧重于修复错误配置不同，ARIEL [15] 通过对集成规则应用修改 和快速 变异算子来修复特征交互缺陷，从而解决自动驾驶系统特征之间的冲突。因此，ARIEL未涵盖**C2**。

CADET [22] 确实涵盖了**C2** ，因为它旨在调试并修复导致非功能性故障的错误配置。Xiong 等人 [23] 通过生成范围修复列表来帮助满足约束，重点修复软件产品线中的错误配置。然而，这两种方法都未考虑执行测试时需要消耗大量计算资源的系统。此外，CADET [22] 仅涵盖两种非功能属性（即延迟和能耗），而 Xiong 等人 [23] 则侧重于满足单个约束。最后，这些方法没有优先修复较不关键的故障。因此，这两种技术都未涵盖**C1**、**C3**和**C4**。

除了与我们的方法最为密切相关的这三项研究外，自动化程序修复 [52]–[64]领域还有其他研究。与本文方法类似，其中一些研究采用了搜索技术，例如遗传编程 [55], [56]。GenProg [55]是最早提出使用元启发式搜索来修复软件程序的方法之一。具体而言，该方法利用遗传编程修复 C 程序。然而，所有这些方法都侧重于修复代码中的缺陷。相反，我们的方法侧重于修复可配置信息物理系统领域中的错误配置。

另一项与我们的方法相关的研究是统一调试 [65], [66]。该技术利用补丁执行结果来改进故障定位 [65], [66]。因此，即使修复过程无法修复缺陷，统一调试也有助于改进故障定位，以便后续进行人工修复。我们的方法采用了类似的策略，旨在定位可疑参数，从而最终帮助修复配置错误。然而，除了统一调试 [65], [66] 并非旨在调试配置错误，而是针对代码层级的缺陷之外，它还假设已经存在初始可疑度分数（即语句层级的可疑度）。相比之下，我们的方法将所有参数的可疑度设为相同，因为目前没有关于哪些参数会影响系统性能的信息。

VII. 结论与未来工作

现实世界中的信息物理系统（CPS），例如电梯，涉及许多参数。信息物理系统的性能与这些参数密切相关，因此可能会出现配置错误。一方面，人工处理此类配置错误未必总是可行；另一方面，自动化解决方案需要应对某些挑战，例如执行测试用例所需的仿真成本高昂。本文提出了一种基于元启发式搜索的自动化、可扩展解决方案，用于修复信息物理系统中的配置错误。我们将该方法集成到 Orona 提供的一项工业案例研究中。Orona 是欧洲最大的电梯制造商之一。评估在一个实际安装环境中进行，Orona 的领域专家曾在其中人工介入，以修复配置错误。结果表明，除了将此前完全依赖人工的流程自动化之外，我们的算法所提供的补丁也优于领域专家提供的补丁。具体而言，在领域专家用于评估补丁质量的六项质量指标中，有五项显示，我们的算法提供的补丁在统计意义上显著优于领域专家提供的补丁。

未来，我们希望从不同角度拓展我们的方法。在适用性方面，我们希望将该算法整合到其他配置已被发现存在问题的信息物理系统中（例如，无人驾驶飞行器 [5]）。此外，我们希望探索在提出可行补丁之前防止潜在过拟合问题的方法。这是自动化程序修复领域已发现的核心挑战之一 [32]–[35], ，因此我们应对此保持警惕。在 Orona 内部适用性方面，我们希望在其他发生过配置错误的安装系统中评估我们的方法。此外，我们还希望将修复算法推广至交通团队以外的其他部门。最后，我们希望进一步研究是否存在其他策略，能够更好地训练代理模型并将其整合到修复过程中。

致谢

本项目由 BBVA基金会2021年莱昂纳多研究人员与文化创作者资助项目支持。BBVA基金会不对项目中包含的观点、评论和内容，及其产生的结果承担责任；这些内容和结果完全由作者负责。Aitor Arrieta 是 Mondragon Unibertsitatea 软件与系统工程研究组的成员（IT1519-22），该研究组由巴斯克地区教育、大学与研究部资助。

参考文献

[1] P. Derler、E. A. Lee 和 A. S. Vincentelli，“信息物理系统建模”，*Proceedings of the IEEE*，第100卷第1期，第13–28页，2011年。[2] R. Baheti 和 H. Gill，“信息物理系统”，*The impact of control technology*，第12卷第1期，第161–166页，2011年。[3] R. Alur，《信息物理系统原理》。MIT出版社，2015年。

[4] S. Fischer、R. Ramler、C. Klammer 和 R. Rabiser，“高度可配置的信息物理系统测试——多案例研究”，载于第*15*届软件密集型系统变异性建模国际工作会议，2021年，第1–10页。[5] R. Han、C. Yang、S. Ma、J. Ma、C. Sun、J. Li 和 E. Bertino，“有害的控制参数：通过学习引导搜索检测无人机配置模块中的范围规范错误”，载于第*44*届国际软件工程会议论文集，2022年，第462–473页。[6] D. Wang、S. Li、G. Xiao、Y. Liu 和 Y. Sui，“无人驾驶飞行器中自动驾驶仪软件缺陷的探索性研究”，载于第*29*届*ACM*欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2021年，第20–31页。[7] Garcia, Joshua、Feng, Yang、Shen, Junjie、Almanee, Sumaya、Xia, Yuan、Chen 和 Qi Alfred，“自动驾驶汽车缺陷的全面研究”，载于第*42*届*ACM/IEEE*国际软件工程会议论文集，2020年，第385–396页。[8] R. B. Abdessalem、A. Panichella、S. Nejati、L. C. Briand 和 T. Stifter，“使用多目标搜索测试自动驾驶汽车的特征交互故障”，载于*2018*年第*33*届*IEEE/ACM*自动化软件工程国际会议（*ASE*）。IEEE，2018年，第143–154页。[9] R. B. Abdessalem、S. Nejati、L. C. Briand 和 T. Stifter，“使用可学习进化算法测试基于视觉的控制系统”，载于*2018*年第*40*届*IEEE/ACM*国际软件工程会议（*ICSE*）。IEEE，2018年，第1016–1026页。[10] C. Menghi、S. Nejati、L. Briand 和 Y. I. Parache，“计算密集型信息物理模型的近似细化测试：一种基于系统辨识的方法”，载于*2020*年第*42*届*IEEE/ACM*国际软件工程会议（*ICSE*）。IEEE，2020年，第372–384页。[11] C. Menghi、S. Nejati、K. Gaaloul 和 L. C. Briand，“为具有连续和不确定行为的Simulink模型生成自动化在线测试预言机”，载于*2019*年第*27*届*ACM*欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2019年，第27–38页。[12] S. Nejati、K. Gaaloul、C. Menghi、L. C. Briand、S. Foster 和 D. Wolfe，“评估模型测试和模型检验以发现Simulink模型中的需求违规”，载于*2019*年第*27*届*ACM*欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2019年，第1015–1025页。[13] F. U. Haq、D. Shin 和 L. Briand，“使用代理辅助多目标优化对DNN使能系统进行高效在线测试”，载于第*44*届国际软件工程会议论文集，2022年，第811–822页。[14] D. Humeniuk、F. Khomh 和 G. Antoniol，“一种用于自动生成信息物理系统测试环境的基于搜索的框架”，《信息与软件技术》，第106936页，2022年。[15] R. B. Abdessalem、A. Panichella、S. Nejati、L. C. Briand 和 T. Stifter，“自动驾驶系统中特征交互故障的自动化修复”，载于第*29*届*ACM SIGSOFT*国际软件测试与分析研讨会论文集，2020年，第88–100页。[16] G. Perrouin、S. Sen、J. Klein、B. Baudry 和 Y. Le Traon，“面向软件产品线的自动化可扩展t-wise测试用例生成策略”，载于*2010*年第三届软件测试、验证与确认国际会议。IEEE，2010年，第459–468页。[17] A. Arrieta、S. Wang、G. Sagardui 和 L. Etxeberria，“面向信息物理系统产品线基于仿真的测试的基于搜索的测试用例优先级排序”，《系统与软件杂志》，第149卷，第34页，2019年，第1页。[18] C. Henard、M. Papadakis、G. Perrouin、J. Klein、P. Heymans 和 Y. Le Traon，“绕过组合爆炸：使用相似性生成软件产品线的t-wise测试配置并确定其优先级”，*IEEE Transactions on Software Engineering*，第40卷第7期，第650–670页，2014年。[19] S. Wang、S. Ali 和 A. Gotlieb，“使用搜索技术实现产品线中经济高效的测试套件最小化”，《系统与软件杂志》，第103卷，第370–391页，2015年。[20] D. Marijan、A. Gotlieb、S. Sen 和 A. Hervieu，“软件产品线的实用成对测试”，载于第*17*届国际软件产品线会议论文集，2013年，第227–235页。[21] A. Hervieu、D. Marijan、A. Gotlieb 和 B. Baudry，“使用约束程序对成对覆盖测试配置进行实用最小化”编程”，《信息与软件技术》，第71卷，146，2016年，129页–。[22] R. Krishna、M. S. Iqbal、M. A. Javidian、B. Ray 和 P. Jamshidi，“CADET：利用反事实推理调试并修复错误配置”，arXiv预印本 arXiv:2010.06061，2020年。[23] Y. Xiong、H. Zhang、A. Hubaux、S. She、J. Wang 和 K. Czarnecki，“范围修复：软件配置的交互式错误解决”，IEEE Transactions on Software Engineering，第41卷，619，2014年6月，第603页–。[24] C. Jung、A. Ahad、J. Jung、S. Elbaum 和 Y. Kwon，“Swarmbug：调试群体机器人的配置错误”，载于第29届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2021年，第868–880页。[25] J. Ayerdi、A. Garciandia、A. Arrieta、W. Afzal、E. Enoiu、A. Agirre、G. Sagardui、M. Arratibel 和 O. Sellin，“面向信息物理系统设计—运行连续体需求获取的分类体系”，载于2020年 IEEE 第28届国际需求工程会议（RE）。IEEE，2020年，第280–290页。[26] A. Gartziandia、A. Arrieta、J. Ayerdi、M. Illarramendi、A. Agirre、G. Sagardui 和 M. Arratibel，“基于机器学习的测试预言机：用于信息物理系统性能测试的电梯调度算法工业案例研究”，《软件杂志：演化与过程》，第e2465页，2022年。[27] L. Han、S. Ali、T. Yue、A. Arrieta 和 M. Arratibel，“工业电梯系统的不确定性感知鲁棒性评估”，《ACM软件工程与方法学汇刊》，2022年。[28] L. Han、T. Yue、S. Ali、A. Arrieta 和 M. Arratibel，“电梯软件能否抵御不确定性？来自工业案例研究的结果与经验”，载于第30届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2022年，第1331–1342页。[29] A. Arrieta、M. Otaegi、L. Han、G. Sagardui、S. Ali 和 M. Arratibel，“为工业电梯在 DevOps 中自动生成测试预言机”，载于2022年 IEEE 国际软件分析、演化与再工程会议（SANER）。IEEE，2022年，第284–288页。[30] G. Barney 和 L. Al-Sharif，《电梯交通手册：理论与实践》。劳特里奇，2015年。[31] G. Barney，《建筑物中的交通系统：CIBSE D指南：2010》。伦敦：特许建筑服务工程师学会，2010年。 [在线]。可访问：https://www.worldcat.org/title/transportation-systems-in-buildings-cibse-guide-d-2010/oclc/880899711[32] C. L. Goues、M. Pradel 和 A. Roychoudhury，“自动化程序修复”，《ACM通讯》，第62卷，第12期，第56–65页，2019年。[33] M. Martinez、T. Durieux、R. Sommerard、J. Xuan 和 M. Monperrus，“Java中真实缺陷的自动修复：基于Defects4J数据集的大规模实验”，《实证软件工程》，第22卷，第4期，第1936–1964页，2017年。[34] A. Nilizadeh、G. T. Leavens、X.-B. D. Le、C. S. Pˇasˇareanu 和 D. R. Cok，“利用形式化方法探索动态自动程序修复中的真实测试过拟合”，载于2021年第14届 IEEE 软件测试、验证与确认会议（ICST）。IEEE，2021年，第229–240页。[35] E. K. Smith、E. T. Barr、C. Le Goues 和 Y. Brun，“治愈是否比疾病更糟？自动化程序修复中的过拟合”，载于2015年第10届软件工程基础联合会议论文集，2015年，第532–543页。[36] M.-L. Siikonen、T. Susi 和 H. Hakonen，“高层建筑中的乘客交通流仿真”，《电梯世界》，第49卷，123，2001年8月，第117页–。[37] J. Ayerdi、S. Segura、A. Arrieta、G. Sagardui 和 M. Arratibel，“QoS感知蜕变测试：电梯案例研究”，载于2020年 IEEE 第31届国际软件可靠性工程研讨会（ISSRE）。IEEE，2020年，第104–114页。[38] J. Ayerdi、P. Valle、S. Segura、A. Arrieta、G. Sagardui 和 M. Arratibel，“信息物理系统的性能驱动蜕变测试”，《IEEE可靠性汇刊》，2022年。[39] J. Ayerdi、V. Terragni、A. Arrieta、P. Tonella、G. Sagardui 和 M. Arrat- ibel，“利用遗传编程为信息物理系统生成蜕变关系：工业案例研究”，载于第29届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2021年，第1264–1274页。

[40] A. Arrieta、S. Wang、U. Markiegi、A. Arruabarrena、L. Etxeberria 和 G. Sagardui，“面向基于仿真的测试的帕累托高效多目标黑盒测试用例选择”，《信息与软件技术》，第114卷，第137–154页，2019年。[41] A. Arrieta、S. Wang、U. Markiegi、G. Sagardui 和 L. Etxeberria，“运用多目标搜索增强工业信息物理系统测试中的反应式测试用例生成与优先级排序”，*IEEE Transactions on Industrial Informatics*，第14卷，1066，2017年3月，第1055页–。[42] P. McMinn，“基于搜索的软件测试：过去、现在与未来”，载于*2011*年 *IEEE* 第四届软件测试、验证与确认国际会议研讨会。IEEE，2011年，第153–163页。[43] M. Di Penta、G. Canfora、G. Esposito、V. Mazza 和 M. Bruno，“服务级别协议的基于搜索的测试”，载于第*9*届遗传与进化计算年度会议论文集，2007年，第1090–1097页。[44] S. Wang、S. Ali、T. Yue、Y. Li 和 M. Liaaen，“评估基于帕累托的搜索算法时选择质量指标的实用指南”，载于第*38*届国际软件工程会议论文集，2016年，第631–642页。[45] M. Li、T. Chen 和 X. Yao，“如何评估基于帕累托的基于搜索的软件工程中的解：批判性综述与方法指导”，*IEEE Transactions on Software Engineering*，第48卷，第05期，第1771–1799页，2022年。[46] K. Shang、H. Ishibuchi、L. He 和 L. M. Pang，“进化多目标优化中超体积指标的调查”，*IEEE Transactions on Evolutionary Computation*，第25卷，第1期，第1–20页，2020年。[47] J. Romano、J. D. Kromrey、J. Coraggio、J. Skowronek 和 L. Devine，“探索评估NSSE及其他调查中组间差异的方法：t检验和Cohen’s d指数是否是最合适的选择”，载于南方高等教育机构研究协会年会。Citeseer，2006年，第1–51页。[48] A. Arcuri 和 L. Briand，“在软件工程中使用统计检验评估随机化算法的实用指南”，载于*2011*年第*33*届国际软件工程会议（*ICSE*）。IEEE，2011年，第1–10页。[49] L. Briand、S. Nejati、M. Sabetzadeh 和 D. Bianculli，“测试不可测试对象：复杂软件密集型系统的模型测试”，载于第*38*届国际软件工程会议论文集（附属会议），2016年，第789–792页。[50] R. Matinnejad、S. Nejati、L. C. Briand 和 T. Bruckmann，“具有动态行为的Simulink模型的测试生成与测试优先级排序”，*IEEE Transactions on Software Engineering*，第45卷，第9期，第919–944页，2018年。[51] M. Monperrus，“自动化程序修复综述”，博士学位论文，HAL Archives Ouvertes，2018年。[59] F. DeMarco、J. Xuan、D. Le Berre 和 M. Monperrus，“利用可满足性模理论自动修复错误的if条件和缺失的前置条件”，载于第*6*届软件测试、验证与分析中的约束国际研讨会论文集，2014年，第30–39页。

[52] T. Ackling、B. Alexander 和 I. Grunert，《用于软件修复的进化补丁》，载于第*13*届遗传与进化计算年会论文集，2011年，第1427–1434页。[53] D. Kim、J. Nam、J. Song 和 S. Kim，《从人工编写的补丁中学习自动补丁生成》，载于*2013*年第*35*届国际软件工程大会（*ICSE*）。IEEE，2013年，第802–811页。[54] J. D. Knowles、R. A. Watson 和 D. W. Corne，《通过多目标化减少单目标问题中的局部最优》，载于进化多准则优化国际会议。Springer，2001年，第269–283页。[55] C. Le Goues、T. Nguyen、S. Forrest 和 W. Weimer，《Genprog：一种通用的自动软件修复方法》，*IEEE Transactions on SoftwareEngineering*，第38卷，第1期，第54–72页，2011年。[56] M. P. Gissurarson、L. Applis、A. Panichella、A. van Deursen 和 D. Sands，《Propr：基于属性的自动程序修复》，载于第*44*届国际软件工程会议论文集，2022年，第1768–1780页。[57] A. Arcuri，《论软件缺陷修复的自动化》，载于第*30*届国际软件工程大会伴随会议，2008年，第1003–1006页。[58] V. Dallmeier、A. Zeller 和 B. Meyer，《从对象行为异常中生成修复方案》，载于*2009*年*IEEE/ACM*自动化软件工程国际会议。IEEE，2009年，第550–554页。[60] T. Ji、L. Chen、X. Mao 和 X. Yi，《利用包含修复要素的相似代码进行自动化程序修复》，载于*2016*年*IEEE*第*40*届年度计算机软件与应用会议（*COMPSAC*），第1卷。IEEE，2016年，第197–202页。[61] H. D. T. Nguyen、D. Qi、A. Roychoudhury 和 S. Chandra，《Semfix：通过语义分析进行程序修复》，载于*2013*年第*35*届国际软件工程大会（*ICSE*）。IEEE，2013年，第772–781页。[62] Y. Qi、X. Mao、Y. Lei、Z. Dai 和 C. Wang，《随机搜索在自动化程序修复中的优势》，载于第*36*届国际软件工程大会论文集，2014年，第254–265页。[63] Z. Qi、F. Long、S. Achour 和 M. Rinard，《对生成并验证式补丁生成系统的补丁合理性与正确性的分析》，载于*2015*年国际软件测试与分析研讨会论文集，2015年，第24–36页。[64] W. Weimer、Z. P. Fry 和 S. Forrest，《利用程序等价性实现自适应程序修复：模型与初步结果》，载于*2013*年第*28*届*IEEE/ACM*自动化软件工程国际会议（*ASE*）。IEEE，2013年，第356–366页。[65] Y. Lou、A. Ghanbari、X. Li、L. Zhang、H. Zhang、D. Hao 和 L. Zhang，《自动化程序修复能否改进故障定位?一种统一调试方法》，载于第*29*届*ACM SIGSOFT*国际软件测试与分析研讨会论文集，2020年，第75–87页。[66] S. Benton、X. Li、Y. Lou 和 L. Zhang，《评估并改进统一调试》，*IEEE Transactions on SoftwareEngineering*，2021年。