arXiv:2305.00705v1 [cs.SE] 1 May 2023

高效动态模型测试 ⋆：采用贪心测试用例选择

P.H.M. van Spaendonck[0000−0002−9536−1524]

埃因霍温理工大学数学与计算机科学系
P.H.M.v.Spaendonck@tue.nl

摘要。 基于模型的测试（MBT）提供了一种自动化方法，用于发现软件模型与其实现之间的差异。如果我们要将 MBT 融入快速迭代的软件开发流程——持续集成与持续部署，那么 MBT 必须能够在尽可能短的时间内测试整个模型。

然而，目前的学术界基于模型测试工具要么随机遍历模型（我们证明这种方式对此目的并不有效），要么使用预先计算的最优路径，而对于大型工业模型，这些路径无法高效计算。我们提出了一种新的遍历策略，其错误检测率提升幅度可与使用预先计算的路径相媲美。我们证明，这种新策略能够高效应用于大型模型。基准测试在真实世界模型和伪随机生成模型的组合上进行。我们观察到这两类模型之间没有显著差异。

关键词： 基于模型的测试 · 测试用例选择 · 高效测试

1 引言

测试已成为现代软件工程的核心原则。研究一再表明，经过全面测试的软件质量更高，维护和开发成本也显著更低 [3,5]。这促成了测试驱动工程的发展：在组件或功能开始实际实现之前，先编写相应的测试；只有所有测试均通过后，才认为该实现正确。

同样，我们也看到模型驱动工程日益普及。在这种方法中，软件组件首先以更高层次的抽象形式描述为状态机／模型。随后便可运用基于模型的技术，交付正确且经过验证的软件。例如，在 [4],UMLsec 中，可利用模型验证单点登录软件应用的安全需求是否得到正确落实。

基于模型的测试（MBT），例如 Tretmans 最初在 [14]中提出的**ioco**方法，正处于这两种工程

⋆ 本出版物是 PVSR 项目（项目编号 17933）的一部分。该项目隶属于 Mas-Cot 研究计划，由荷兰研究理事会（NWO）资助。

方法的交汇处，使我们能够使用抽象模型自动测试其实现。自动生成测试的能力使基于模型的测试对软件开发非常有用；理论上，它可以免去手动编写测试所需的时间。然而在实践中，基于模型的测试采用的穷举方法很容易导致对相同行为进行多次测试。这与手工编写的测试套件形成对比，后者通常会尽可能减少已测试行为之间的重叠。因此，在使用基于模型的测试时，减少已测试行为的重叠应被视为将其应用于大型复杂系统测试的关键一步。

我们的研究重点是在应用基于**ioco**的一致性基于模型的测试技术时，减少已测试行为之间的重叠。对于这种技术，我们将研究范围限定为离散数据类型，并将系统形式化为有向图，其中的边可以标记为使用这些数据类型的输入或输出。因此，测试运行在模型中的路径应尽可能减少与自身的重叠。多次访问模型的同一部分意味着，与该模型部分相对应的行为也会被多次测试。

在 [6], van den Bos和Tretmans提出了一种通过自动计算覆盖给定模型所有符号转换的测试路径来减少这种重叠的可行方案。随后，基于模型的测试工具使用SMT 求解器，在遍历预先生成的路径时计算查询的具体值。与经典的随机测试用例选择方法相比，他们的方法在使用有界重传协议 [8]的多个变异体检测不一致性时，表现出显著的加速效果。

然而，计算这样的全局最优路径是NP</style-困难问题，因此应用于大型工业系统时，计算成本会过高。另一个问题是，van den Bos 和 Tretmans 也承认，每次都使用相同的测试路径，可能会导致测试漏掉那些不会沿着给定路径显现的错误 [6]。可能正是出于这些原因，学术界的基于模型测试工具（例如 TorXakis [15], ）选择随机遍历模型。

我们研究一种简单直接的贪心测试用例选择策略，能否在适用于大型模型的同时，达到与使用预生成路径相近的加速效果。为进行初步比较，我们使用贪心策略运行与 [6], 中全局最优策略相同的实验设置，即在有界重传协议 [8] 上进行变异检测。根据算术平均值，我们的贪心策略比随机策略快 8.2 倍。

其次，我们通过将贪心解应用于大型真实世界模型以及不同规模的随机生成状态空间，测试其可扩展性。这些随机状态空间是对运行至完成语义下运行的复杂系统的近似。当目标是覆盖给定模型至少70%的状态时，采用贪心策略相比随机测试用例选择策略能够显著提速。我们还观察到，生成模型的结果与真实使用场景的结果相近。

在第2节中，我们简要讨论基于模型的测试技术和优化方面的相关工作。在第3节中，我们给出基于**ioco**的基于模型的测试理论的基础形式化。在第4节中，我们讨论用于**ioco**的基于模型的测试的算法。在第5节中，我们讨论新的优化方法。在第6节中，我们展示并讨论基准测试结果。最后，我们在第7节中给出结论，并讨论软件开发周期中使用基于模型的测试可能涉及的其他重要研究领域。

2 相关工作

将系统形式化为抽象模型时，我们通常倾向于对数据进行抽象，并将其形式化为离散数据类型，因为这样可以显著降低系统的复杂性。然而，在对低层信息物理组件进行形式化时，我们发现数据的确切值（例如进气口通风口的角度）无法被抽象掉，因为它们对于此类组件的正确性至关重要。在这种情况下，系统可以形式化为连续或混合自动机 [10]。我们介绍一些用于连续数据系统的最先进的测试优化技术。

对于具有连续数据的系统，基于模型的测试通过基于搜索的测试来完成，其中使用遗传算法或模拟退火等元启发式算法自动生成测试数据。在这类情况下，减少已测试行为的重叠可以重新表述为一个优化问题。例如，在 [2], 中，元启发式算法用于寻找最优输入，以最大化预期最优输入与先前输入之间的距离，从而提高故障发生的可能性，同时减少与先前已测试行为的潜在重叠。在 [12], 中，结合使用基于监督学习的降维和代理建模技术，将类似技术扩展到能够应用于汽车行业信息物理领域中遇到的极其庞大的搜索空间。

我们在第3节中描述并在全文中使用的系统，是采用离散数据类型的模型，这些数据类型之间通常没有直接关联。因此，不存在可供优化的连续搜索空间，测试用例选择归结为优化对规范模型所表示图的遍历。不过，这两种技术并非互斥，完全可以结合使用，例如 [11], 中所示，使用基于搜索的技术为函数的数据参数挑选可行的候选项。

最后，我们讨论在使用在线（即有限测试套件预先生成的）基于模型的一致性测试技术时，减少已测试行为重叠的相关工作。在 [7], Cartaxo 等人通过度量所生成测试套件中不同测试之间的相似度，并选取一个测试子集，使该子集中测试之间的相似度度量尽可能低，从而提供了一种减少已测试行为重叠的解决方案。通过这种方式，他们能够将运行的测试数量减少 80%，同时仍保持与完整运行测试套件相近的故障检测率。在 [1], Aichernig 等人使用基于模型的

变异测试会从正确的规范模型中生成变异模型（即包含故障行为的模型），再利用这些模型生成测试，以检测其中包含的故障。Aichernig 等人使用有界等价检查器找出并移除相同的变异体，避免生成的测试包含重叠的已测试行为。离线基于模型的测试技术（例如我们采用的技术）通常更适合测试只有在经过特定的长输入和输出序列后才会出现的行为。

3 基于模型的测试理论

本节介绍理解如何将给定系统的行为形式化，以及如何在动态基于模型的测试中使用这些形式模型所需的理论。此处介绍的基于模型的测试理论与 [14]中描述的原始理论相同，仅在符号表示上有所不同。

系统的预期行为用输入输出标记转换系统（IOLTS）表示。IOLTS 是标记转换系统的一种变体，其中动作标签被划分为互不相交的输入动作集和输出动作集。我们将 IOLTS 定义如下：

定义 1. *IOLTS* L 定义为五元组 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$，其中：

- Q 是有限状态集，- $q_0 \in Q$ 是初始状态，- A_I 和 A_O 分别是输入动作集和输出动作集，使得$A_I \cap A_O = \emptyset$ 且 $\tau, \delta \notin A_I \cup A_O$。- $\rightarrow \subseteq Q \times (A_I \cup A_O \cup \{\tau, \delta\}) \times Q$ 是转换关系，使得当且仅当 $q = q'$ 时有 $\langle q, \delta, q' \rangle \in \rightarrow$，并且状态 q 没有标记为输出动作 $a_{out} \in A_O$ 或 τ动作的出向转换。

给定状态 p，q 和某个动作标签 a，我们使用简写符号 $p \xrightarrow{a} q$ 来代替 $\langle p, a, q \rangle \in \rightarrow$。

在动作集合 $A_I \cup A_O$ 中，每个元素都对应实现中的一个函数。我们区分输入动作 A_I（即我们作为输入调用系统的函数）和输出动作 A_O（即其他组件的函数；这些函数既可以响应提供给系统的输入而触发，也可以自发触发，例如内部计时器超时）。

我们进一步用特殊动作 τ 和 δ 扩展动作集合。τ 动作表示外部不可观测的动作，例如调用函数后发生的某些内部计算。δ 动作称为静默，表示系统保持空闲。带有 δ 动作的转换恰好发生在无法进行输出转换或内部动作的所有状态中，并且必须是自环，即起点和终点相同。

为便于阅读，我们用 A_δ 作为 $A_I \cup A_O \cup \{\delta\}$ 的简写，并用 A作为 $A_I \cup A_O$的简写。

在基于模型的测试过程中，使用一系列动作来记录迄今为止执行过的动作，这一系列动作称为悬停轨迹。我们通过定义 2 和定义 3 定义悬停轨迹集合。

定义 2. 设 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$ 为一个输入输出标记转换系统。我们将可观测路径关系 $\Rightarrow \subseteq Q \times A_\delta^* \times Q$ 定义为满足以下条件的最小关系：

– 对于任意状态 $q \in Q$，其中 ϵ 表示空序列，– 对于状态 q、q'、$q'' \in Q$ 以及某个词 $w \in A_\delta^*$，若 $q \xrightarrow{\tau} q'$ 且 $q' \stackrel{w}{\Rightarrow} q''$，则 $q \stackrel{w}{\Rightarrow} q''$；– 对于状态 q、q'、$q'' \in Q$、某个可观测动作 $a \in A_\delta$以及一个词 $w \in A_\delta^*$，若 $q \xrightarrow{a} q'$ 且 $q' \stackrel{w}{\Rightarrow} q''$，则 $q \stackrel{aw}{\Rightarrow} q''$。

定义 3. 设 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$ 为某个输入输出标记转换系统。给定一个状态 $q \in Q$，悬停轨迹集合 $\mathit{straces}(q) \subseteq A_\delta^*$ 定义如下：

$$straces(q) = \{w \in A_\delta^* \mid \exists_{q' \in Q}[q \stackrel{w}{\Rightarrow} q']\}.$$

基于模型的测试用于检验实现的行为是否符合规约的行为，即实现绝不会产生规约不允许的输出。这种一致性通过输入输出一致性关系 ioconformance relation ioco 在输入输出标记转换系统上描述，该关系最初由 [14], 定义，并见定义 5。

定义 4. 设 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$ 为一个输入输出标记转换系统。我们定义映射 $\mathit{out}_L : \mathcal{P}(Q) \rightarrow \mathcal{P}(A_O)$ 以及 $L\ \mathbf{after}: A_\delta^* \rightarrow \mathcal{P}(Q)$ 如下：

– 给定某个状态集合 $\mathit{qs} \subseteq Q$，有

$$out_L(qs) = \{a_{out} \in A_O | \exists_{q \in qs, q' \in Q}[q \xrightarrow{a_{out}} q']\},\ and$$

– 给定某个悬停轨迹 $\sigma \in \mathit{straces}(q_0)$，我们有

$$L\ \mathbf{after}\ \sigma = \{q \in Q | q_0 \stackrel{\sigma}{\Rightarrow} q\}.$$

定义 5. 给定一个实现的输入输出标记转换系统 L_{impl} 和一个规约的输入输出标记转换系统$L_{spec} = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$，一致性 $L_{impl}\ \mathbf{ioco}\ L_{spec}$成立，当且仅当对于任意轨迹 $\sigma \in \mathit{traces}(q_0)$，我们有

$$out_{L_{impl}}(L_{impl}\ \mathbf{after}\ \sigma) \subseteq out_{L_{spec}}(L_{spec}\ \mathbf{after}\ \sigma).$$

4 基于模型的测试算法

现在我们讨论来自 [14], 的基于模型的测试算法，其伪代码描述见算法 1。基于模型的测试算法使用集合 qs，该集合最初包含初始状态 q_0 以及仅通过 τ- 动作从该状态可达的所有状态，用于跟踪每次迭代开始时算法可能处于哪些状态。

算法 1： 基于模型的测试算法

数据： 一个规约输入输出标记转换系统 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$ 和超时时间 t。

1 $qs \leftarrow \{q \in Q | q_0 \stackrel{\epsilon}{\Rightarrow} q\}$
2 **当** 下一个$(L,\ qs)\ \neq \emptyset$ **时：**
3 **执行以下任一操作**:
4 **如果** 下一个$(L,\ qs)\ \cap A_I \neq \emptyset$ **则**:
5 **选择** a_{in} **自** 下一项$(L, qs)\ \cap A_I$
6 send(a_{in})
7 $qs \leftarrow \{q \in Q | \exists q' \in qs :\ q' \stackrel{a_{in}}{\Longrightarrow} q\}$
8 **or** :
9 **try** $a_{out} \leftarrow$rcv(t)
10 **如果** $a_{out} \in$ 下一个$(L,\ qs)$ **那么** :
11 $q \leftarrow \{q \in Q | \exists q' \in qs :\ q' \stackrel{a_{out}}{\Longrightarrow} q\}$
12 **否则**:
13 **返回** $false$
14 **当** $timeout$ ：
15 **如果** $\delta \in$ 下一步$(L,\ qs)$ **则**:
16 $qs \leftarrow \{q \in qs | q \stackrel{\delta}{\rightarrow} q\}$
17 **否则**:
18 **返回** $false$

我们将 next(L, qs) 定义为可以从 qs 中至少一个状态采取的可能动作集合，即 next(L, qs) $= \{a \in A_\delta | \exists_{q \in qs, q' \in Q}[q \stackrel{a}{\rightarrow} q']\}$。

第 6 行的 send(a) 函数会使实现执行与输入动作 a 对应的函数。每当实现调用与某个输出动作 a_{out} 对应的函数时，标签 a_{out} 就会发送回基于模型的测试算法，并添加到响应队列中。队列中的消息按先进先出的顺序，通过第 9 行的 rcv(t) 函数读取；读取消息的同时会将其从队列中移除。如果调用 rcv(t) 时队列为空，该函数会等待，直到收到新消息。如果在超时时间 t 内未收到消息，该函数会抛出 超时。

该算法会反复执行以下两项操作之一:

– 从任意状态都可以执行的某个输入动作 a_{in} ，在 qs 中被选中，并要求实现执行相应的函数，使用 send(a_{in})。随后，工具相应地更新可能状态集合 qs ，并继续执行。– 基于模型的测试工具使用 rcv(t) 监听可能的输出动作 a_{out} 。如果收到动作，我们会验证该动作是否为允许的动作。如果允许，则相应地更新状态集合 qs 并继续执行。如果 a_{out} 不是 qs 中任何状态都可能产生的动作，则可以断定实现不满足相对于规范模型的 **ioco** ，测试失败。如果在超时时间 t 内未收到响应，则假定实现处于空闲／静默状态。该算法将集合 qs

缩减为 qs 中仅包含静默状态，即带有 δ-自环的状态。如果 qs 中没有任何状态处于静默状态，则实现同样不满足相对于规范模型的 **ioco** ，测试失败。

请注意，如果始终未检测到行为差异，基于模型的测试算法将不会终止。测试人员必须决定何时终止。全文始终采用的终止准则是状态覆盖率，即在整个运行过程中已到达或测试过的状态所占的百分比。

5 贪心测试用例选择策略

为了减少已测试行为之间的重叠，我们希望避免查询会导致到达已测试状态的输入。然而，计算全局最优路径，即寻找一条至少访问每个状态一次的最短路径，对于具有大量状态的复杂系统而言，计算开销过大，难以实现。因此，我们提出的策略旨在通过选择局部最优解来近似全局最优路径。具体而言，我们计算从当前状态集合 qs 出发、长度为n的所有可能路径，然后选择与已访问状态数最少的路径相对应的输入动作。如果有多个这样的动作可供选择，则随机选取一个候选项。

我们采用两项优化来减少寻找这些局部最优路径所需的工作量。为了说明采用这些优化的必要性，考虑一个输入输出标记转换系统，其中每个状态平均有 λ 条出向转换。若只是计算所有长度为 n 的可能路径，并选出最优的一条，所需计算量仍将达到 $O(|qs| \cdot \lambda^n)$ 的量级。

对于第一项优化，我们注意到，一旦找到一条长度为 n的最优路径并执行了一个动作，就没有必要重新考虑所有长度为 n 的可能路径，因为我们已经有一条长度为 $n-1$ 的近似最优路径。相反，我们只考虑剩余路径的扩展。在定义6中，我们概述了用于跟踪先前计算的路径树数据结构。

定义6。给定一个输入输出标记转换系统 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$，路径树是一个六元组 $pt = \langle a, q, next\ d, v, v_{max} \rangle$，其中

– 我们在 pt 上定义了投影函数 a、q、$next$、d、v 和 $vmax$，使得 $a(pt) = a$、$q(pt) = q$、$next(pt) = next$、$d(pt) = d$、$v(pt) = v$，且 $vmax(pt) = v_{max}$，– a 是一个指向状态 q 的动作标签，即 $\exists_{q' \in Q}[q' \xrightarrow{a} q]$，– $next$ 是一个路径树集合，满足 $\forall_{pt' \in next}[q \xrightarrow{a(pt')} q']$，– d 是树的深度，即 $next$ 中路径树的最高深度加 1，– v，即路径树的值，等于从转换 $q' \xrightarrow{a} q$开始，通过一系列 d 转换可到达的未访问状态的最大数量，– v_{max} 用于存储 $next$ 中路径树的最大值。

算法 2： 用于路径树的 grow(L，pt n) 算法

数据： 输入输出标记转换系统 $L = \langle Q, q_0, A_I, A_O, \rightarrow \rangle$，一棵路径树 $pt = \langle a, q, next, d, v, v_{max} \rangle$，以及目标深度为 $n \geq d$。

1 **如果** $d = n$：
2 **return;**
3 **否则，如果** $next = \emptyset$:
4 **对于** a: A，q': Q **s.t.** $q \xrightarrow{a} q'$ **执行：**
5 **将** $\langle a, q', \emptyset, 1, covered(q')\ 0 \rangle$ **添加到** $next$
6 **对于** $pt' \in next$ **执行：**
7 **如果** v(pt') + (($n-1$) − d(pt')) $\geq v_{max}$ **：**
8 增长($L, pt', n-1$)
9 $v_{max} \leftarrow$ max(v_{max},v(pt'))
10 $v \leftarrow v_{max} + covered(q)$
11 $d \leftarrow n$

现在我们讨论算法 2 中概述的增长函数，该函数将给定的路径树 $pt = \langle a, q, next\ d,\ v,\ v_{max} \rangle$扩展到深度 n。第 5 行和第 10 行使用的 $covered$ 函数会在状态 q 尚未被访问时返回 1，否则返回 0。如果目标深度 n 与路径树深度 d相等，则无需进一步计算，我们会立即终止（第 1 行和第 2 行）。否则，我们首先检查 $next$ 是否为空，也就是说，是否尚未在状态 q之后执行属于该路径树的计算。如果 $next$ 为空，则会为每个源自状态 q 的迁移插入一棵深度为 1 的新路径树（第 3 至第 5 行）。随后，针对 $next$ 中的每棵路径树递归调用增长函数，并将目标深度设为 $n-1$，同时相应地更新变量 d、v 和v_{max}（第 6 行以及第 8 至第 11 行）。

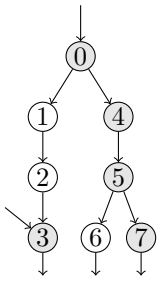


图 1.抢先终止示例

第 7 行的 **if** 语句对应于我们的第二项优化，即在确定给定路径不再可能优于当前最优路径候选的值时，抢先终止该路径的计算。例如，参见图 1，其中展示了部分探索过的输入输出标记转换系统的一部分。为便于阅读，省略了转换标签，并将先前访问过的状态标为灰色。基于模型的测试算法当前处于状态 0，并希望找到一条长度为 3 的最优路径。它首先沿状态 1 计算一条长度为 3 的可能路径，并找到一条包含 2 个未访问状态的可能路径。随后，它沿状态 4 计算一条路径，但发现前两个状态（即 4 和 5）已经访问过后，便终止计算，因为我们可以确定，沿状态 4 的路径不可能包含 2 个或更多未访问状态。

为了使用贪心测试用例选择策略，我们使用一个变量$paths$ 来存储当前从 qs 出发的一组路径树，并将算法 1 所示的基于模型的测试算法第 5 行中的语句替换为

算法 3： 贪心测试用例选择策略。

数据： 输入输出标记转换系统 L、当前维护的路径树集合 $paths$，以及集合当前状态 qs、 n 的目标深度，以及一组可能的动作 $options$。

```
1  选择输入(L, paths, qs, n, options) :=
2      paths ← {pt ∈ paths | a(pt) ∈ options}
3      如果 paths = ∅ :
4          对于 q ∈ qs , 执行:
5              对于 a ∈ options, q' ∈ Q , 使得 q -a-> q' , 执行:
6                  将 ⟨a, q', ∅, 1, covered（q'） 0⟩ 插入 paths
7      v_best ← 0
8      pref ← ∅
9      对于 pt ∈ paths , 执行:
10         增长(L, pt, n)
11         如果 v(pt) > v_best :
12             v_best ← path.v
13             pref ← {a(pt)}
14         否则如果 v(pt) = v_best :
15             插入a(pt) 位于 pref
16         返回 pref
```

以下陈述：

$$\text{从 } a_{in} \text{ 中 选取输入}(L, _ paths, qs\ n, \text{next}(L, qs) \cap A_I)。$$

选取输入函数见算法 3，该函数负责将所有当前路径树扩展至目标深度 n，并将可能的输入动作选项缩减为属于长度为 n 的局部最优路径的选项。每当实现执行某个动作 a 时，也就是说，动作 a 被发送至实现或从实现接收时，路径集合路径就会更新为所有动作等于 a 的路径树中 $next$ 的并集。这样我们便能重用已经完成的计算。

6 基准测试结果

我们将基准测试分为两组。在第6.1节中，我们测试贪心策略是否能像 [6] 中提供的全局最优解那样，相较于随机测试用例选择策略有所改进。为此，我们使用与全局最优解相同的一组基准测试来评估我们的策略。在第6.2节中，我们分别采用贪心策略和随机策略，在各种大型、复杂的生成模型和真实世界模型上进行基准测试，以评估我们方案的可扩展性。

对于这两组基准测试，我们关注的是每种遍历策略达到特定目标所需的迁移次数。对于第一组，目标是检测不一致性；对于第二组，目标是达到特定百分比的状态覆盖率。我们测量的是迁移次数，而不是

所需时间，因为吞吐量（即每秒迁移次数）在很大程度上取决于静默时间，而静默时间因系统而异。

两个章节中考察的系统都是确定性的，并且采用内部选择；也就是说，在每个状态下，都可以向实现查询输入，或从实现接收输出。

6.1 故障检测速度

本文使用的基准测试 [15] 来自自动机维基 [13], ，这是一个在线资料库，收录了各种可用于基准测试的形式化模型。其中一组模型涉及飞利浦的有界重传协议 [8] 。对于该通信协议，维基既提供了其预期行为的形式化描述，也提供了 6 个变异体。该协议是交替位协议的一种变体，可通过可能不可靠的通信信道，为拆分成三条消息的序列提供有序且部分可靠的通信。因此，消息序列总会按明确标记的顺序到达；但是，如果某条消息传输失败的次数达到固定上限，整个序列的传输就会终止。这六个变异体分别代表该通信协议的不正确实现。关于变异体与正确实现之间差异的详细说明，请参阅维基 [13] 。

如 [6], 中所述，我们使用基于模型的测试（MBT），分别采用随机策略和我们的贪心策略，测量发现由每个变异引起的不一致性所需的迁移平均数量。对于每种策略，平均值均基于每个变异体共 100 次运行计算得出。这些运行的平均值，以及 van den Bos 和 Tretmans [6], 的结果，列于表1中。作为参考，最后两列包含原始基准测试结果，其中转换覆盖是其全局最优解，而 TorXakis 采用的是符号化随机测试用例选择策略。由于我们的工具不支持无限数据类型，因此代表三条消息的变量被常量替代。因此，我们无法正确捕获变异体 6，因为它对所使用的三条消息作出了断言，所以在基准测试中省略了该变异体。对于每种测试用例选择策略，我们计算了全部 5 个实验所得平均值的算术平均值。

我们观察到，在测试变异体 2 或 6 时，平均检测速度存在显著差异。在变异体 2 中，通信终止前允许的传输失败次数从 5 次减少到 4 次。在变异体 5 中，如果仅前 2 条消息被正确传输，则会发送错误的最终响应消息。这两种变异都需要非常特定的输入序列才能被观测到；由于随机选中特定序列的概率非常低，随机测试用例选择策略需要明显更长的时间才能观测到这些变异体。

我们观察到，贪心策略的表现比随机（非符号）策略好 ≈ 8.2 倍。排除变异体 6 后，全局操作-

最优解的表现实际上比随机（符号）策略好 ≈ 20.6 倍，而不是纳入变异体 6 时出现的 ≈ 7.7 倍提升。原作者指出，将全局最优策略应用于变异体 6 时性能下降，是因为缺少随机性来决定消息变量使用哪些值。因此，我们认为，贪心策略所具有的轻微随机性应能带来与随机测试用例选择策略相似的性能结果；即使将变异体 6 加入基准测试集合，性能提升也应与目前保持一致。

	显式数据（2.9k 个状态）		符号数据（6 个状态）	
	贪心 含 $N = 5$	随机	先验	TorXakis
变异体 1	20	22	44	12
变异体 2	95	412	16	234
变异体 3	16	21	8	12
变异体 4	27	29	6	18
变异体 5	180	2280	18	1620
变异体 6	-	-	164	76
mean	67.6	552.8	42.7	328.7

表1。 通过基于模型的测试使用贪心测试用例选择、先验生成路径 [6],和随机测试用例选择，检测有界传输协议的不同变异所需的平均迁移数

6.2 在大型模型上的适用性

为了研究我们的方法在大型系统上的应用效果，我们测量了在此类大型模型上，分别采用我们的贪心策略和随机策略，为覆盖给定百分比的状态所需的平均迁移数量。这些测量既在真实世界模型上进行，也在生成模型上进行。我们还选择生成具有代表性的模型，因为获取大量不同类型的复杂大型工业模型相当困难。生成模型基于 Groote 等人 [9],先前的研究，该研究生成了随机标号迁移系统（LTSs），并证明这些系统能够代表真实世界模型。具体而言，这是通过将故障检测结果与真实模型进行比较，并进行统计分析来实现的。

在 [9], 中生成的模型具体如下：首先生成一个具有 N 个状态的标号迁移系统（LTS），并为每个状态赋予 λ 个出向转换，目标状态均匀分布。重复此过程 p 次，随后将所有 p 个随机标号迁移系统并行组合。最终得到的模型将用作一个代表性标号迁移系统。

我们生成的模型旨在代表遵循运行至完成语义的系统。该语义规定，一旦系统开始处理某个输入，在

第一个输入被完全处理之前，即使两个输入可以同时处理，系统也不会开始处理另一个输入。这种语义非常实用，因为它能显著降低模型的复杂度，包括状态空间和认知复杂度。

由于我们可以将给定输入的处理视为一个单一的整体动作，因此最终得到的模型与上述生成的标号迁移系统相似。于是，我们决定按如下方式生成模型：按照之前的方法生成 p 标号迁移系统，每个迁移都标记有一个标签，该标签表示任意输入动作，后面跟随构成该输入动作处理过程的任意序列 r 输出动作。当所有 p 进程并行组合后，将单个迁移替换为一系列迁移，并分别标记对应的输入动作和 r 输出动作序列。

我们要求所得的输入输出标记转换系统具有确定性且完全连通，即始终存在一条从任意状态到其他任意状态的路径。使用确定性输入输出标记转换系统可以简化问题情境，减少测试过程中的变化，并使我们能够专注于给定模型的高效遍历。提出完全连通这一要求，是为了确保最终始终能够测试所有状态。直观而言，不满足这一要求的系统会更有利于采用一种智能策略：避免执行错误的迁移，即避免执行会导致尚未测试的状态变得不可达的迁移，而不是采用随机方法。

我们实现了一个 LTS 模拟器，用于模拟模型的行为，并将其用作所有基准测试的实现，因为我们只关注对 LTS 的探索。这大幅减少了运行实现以及实现与工具之间通信所耗费的时间。

对于我们的第一组基准测试，我们在飞利浦的一个工业组件模型上应用基于模型的测试（MBT）。该组件是飞利浦 Azurion 所使用的大型软件集合的一部分。飞利浦 Azurion 是一台大型且复杂的 X 射线设备，可在心脏外科手术等关键操作过程中提供实时成像。图 2 展示了这些基准测试的结果。每种策略均进行了 10 次运行。虚线（– –）表示这些运行的平均值。点线（· · ·）表示平均值加减这些运行的标准差。我们发现，与随机策略相比，贪心策略只需更少的步骤（减少 ≈ 6 倍），即可覆盖飞利浦模型中 90% 的状态。值得注意的是，该模型包含一些转换，一旦经过这些转换，就会导致某些状态不再可达。因此，随机策略无法始终如一地测试超过 $\approx 90\%$ 的状态。

对于第二组基准测试，我们使用一个大型 Dezyne 分拣机器人的规约中的一个子系统。该模型是 Dezyne 工具附带的示例模型之一。我们认为，尽管它只是一个示例模型，但它能够代表现实世界中可能遇到的机器。该模型由大量经过建模的子组件构成，因此包含约 220000 个状态。由于探索如此庞大的模型需要相当长的时间，我们使用状态空间

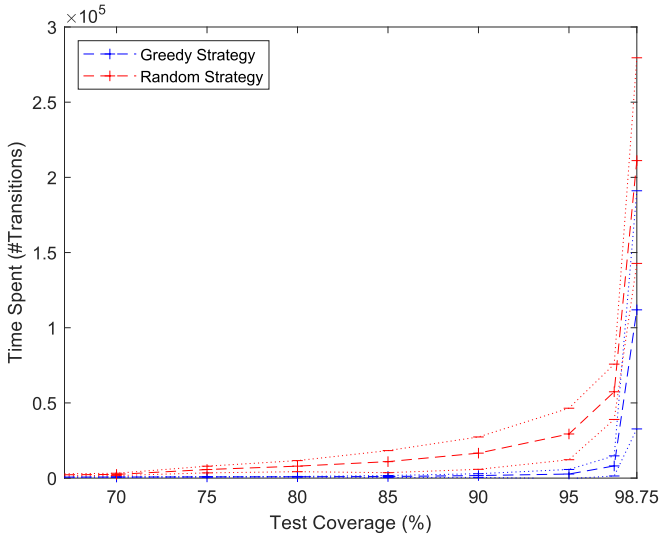


图 2. 飞利浦模型通过贪心策略和随机策略达到给定状态覆盖率所需的迁移次数

采用约简技术缩减其规模，同时仍保留所有行为。为此，我们将所有静默标签显式化，然后对标记迁移系统按分支双模拟等价进行约简。这样一来，模型的状态数从 220000 个减少到仅 617 个。

图 3 展示了在 Dezyne 分拣机上进行的基准测试结果，显示为覆盖分拣机模型的 95% 状态，贪心策略所需的迁移次数比随机策略少 ≈ 5 倍。基准测试包括每种策略各进行 10 次运行。这些运行的平均值以虚线（−−）表示。点线（...）表示这些运行的平均值加减标准差。

对于最后一组基准测试，我们将两种策略应用于若干随机生成的状态空间。所有随机状态空间均使用前述技术生成。我们使用了三组不同的生成参数。每组参数用于生成 3 个随机状态空间。随后，每个随机生成的状态空间均用于按每种策略分别进行 10 次独立运行。因此，每组生成参数对应的基准测试包括每种策略各 30 次运行。左上方基准测试所使用的状态空间通过以下参数生成：$N = 10, \lambda = 6$、$r = 1$ 和 $p = 2$。右上方基准测试所使用的状态空间通过以下参数生成：$N = 10, \lambda = 6$、$r = 1$ 和 $p = 3$。下方基准测试所使用的状态空间通过以下参数生成：$N = 800, \lambda = 6, r = 1$ 和 $p = 1$。

图 4 显示了在生成的状态空间上进行基准测试的结果。贪心策略平均所需的迁移少了 $\approx 5.5\times$ 次

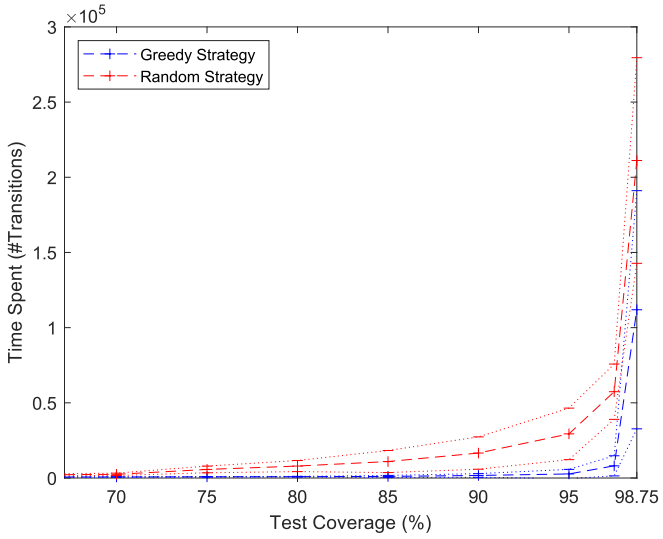


图3。 贪心策略和随机策略达到排序机模型给定状态覆盖率百分比所需的迁移数量

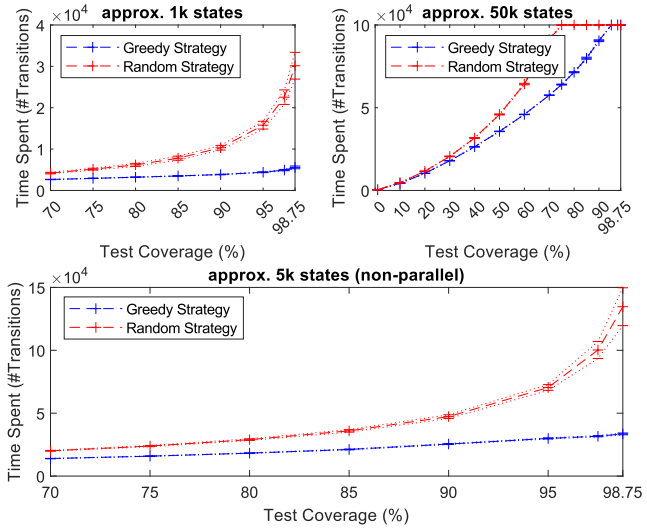


图4. 实现多个生成状态空间的指定状态覆盖率所需的迁移数量

与随机策略相比，在到达所有状态的 98。75% 时，所需的迁移显著更少；对于右上方基准测试中使用的规模大得多的状态空间，所需迁移更少，而对于底部基准测试中使用的非并行状态空间，所需迁移少了 $\approx 4.2\times$ 。所有基准测试都显示出相较于旧随机策略的类似改进。

7 结论与未来工作

我们得出结论：当我们希望使用动态基于模型的测试彻底测试复杂系统时，与旧的随机探索策略相比，我们的贪心解决方案实现了显著改进。在应用于工业模型、分拣机以及大量生成的状态空间时，我们的策略均显示出相较于随机策略的显著加速效果。

然而，在给定时间范围内，我们能够查询被测实现的频率仍然是基于模型的测试的一大瓶颈。如果希望在软件开发中高效使用基于模型的测试，我们认为这是未来研究的重要方向。未来的另一项工作是研究非完全连通模型，在这类模型中，一次错误决策可能导致相当一部分状态不再可访问。有人可能会提出通过允许测试套件重置系统来解决这一问题。然而，重置现实世界的系统非常耗时，因此，确定何时触发重置对于加快测试速度可能至关重要。这类系统的另一个有趣方面，是寻找并比较旨在最大化可能状态覆盖率的策略，同时避免作出此类错误决策。

参考文献

1. Aichernig, B.K., J¨obstl, E., Tiran, S.: 通过符号细化检查进行基于模型的变异测试。《计算机编程科学》**97**, 383–404 (2015). https://doi.org/https://doi.org/10.1016/j.scico.2014.05.004, 专题：第12届国际软件质量会议（QSIC 2012）精选论文 2. Araujo, H., Carvalho, G., Mousavi, M.R., Sampaio, A.: 面向信息物理系统有效测试的多目标搜索。载于： ¨Olveczky, P.C., Sala¨un, G.（编）《软件工程与形式化方法》。第183–202页。Springer International Publishing，沙姆（2019年） 3. Beck, K.: 测试驱动开发：示例讲解。Addison-Wesley Professional（2003年） 4. Best, B., Jurjens, J., Nuseibeh, B.: 使用umlsec对分布式信息系统进行基于模型的安全工程。载于：第29届国际软件工程会议（ICSE'07 55 5 581 590 2007 10 1109 2007 ）。第–页（）。https://doi.org/ 。 / ICSE. . . Bhat, T., Nagappan, N.: 评估测试驱动开发的有效性：工业案例研究。载于：2006年ACM/IEEE实证软件工程国际研讨会论文集。第356–363页。ISESE '06，计算机协会，纽约，NY，美国（2006年）。https://doi.org/10.1145/1159733.1159787

6. van den Bos, P., Tretmans, J.: 使用符号迁移系统进行基于覆盖率的测试。载于: Beyer, D., Keller, C. (编)《测试与证明》。第64–82页。Springer International Publishing，沙姆（2019年） 7. Cartaxo, E., Machado, P., de Oliveira Neto, F.: 在基于模型的测试情境中使用相似度函数进行测试用例选择。《软件测试、验证与可靠性》**21**，75 413 8 100 06 2011 10 1002 – ( )。https://doi.org/ . /stvr. . Groote, J.F., van de Pol, J.: 面向大型数据包的有界重传协议。载于：《代数方法论与软件技术》。第536–550页。Springer Berlin Heidelberg（1996年） 9. Groote, J.F., van der Hofstad, R., Raffelsieper, M.: 行为迁移系统的随机结构。《计算机编程科学》**128**，51–67（2016年）。https://doi.org/https://doi.org/10.1016/j.scico.2016.02.006，https://www.sciencedirect.com/science/article/pii/S0167642316000599，关键系统自动验证（AVoCS'14）特刊 10. Henzinger, T.A.:《混合自动机理论》，第265–292页。Springer Berlin Heidelberg，柏林，海德堡（2000 59615 11 10 1007 978 3 642）。https://doi.org/ . / - - - . Lefticaru, R., Ipate, F.: 基于状态机的函数式搜索测试。载于: 2008年第一届软件测试、验证与确认国际会议。第525 32 12 528页（2008年）。https://doi.org/ . /ICST. . . Matinnejad, R., Nejati, S., Briand, L., Brcukmann, T.: 高度可配置连续控制器的模型在环测试：使用代理模型进行可扩展搜索。载于：第29届ACM/IEEE自动化软件工程国际会议论文集。第163–174页。ASE '14，计算机协会，纽约，纽约州，美国（2014 2642978 13 10 1145 2642937）。https://doi.org/ . / . . Neider, D., Smetsers, R., Vaandrager, F., Kuppens, H.: 自动机学习与一致性测试的基准测试（2019年）。https://doi.org/10.1007/978-3-030-22348-9-23 14. Tretmans, J.: 使用带标签迁移系统进行基于模型的测试，第1–38页。Springer Berlin Heidelberg，柏林，海德堡（2008年）。https://doi.org/10 8 15 1007 978 3 540 78917 . / - - - - . Tretmans, J., van de Laar, P.: 使用TorXakis进行基于模型的测试。《中欧信息与智能系统会议》，第247–258页（2019年），https://www.proquest.com/conference-papers-proceedings/model-based-testing-with-torxakis/docview/2366658121/se-2