arXiv:2302.05405v2 [cs.AI] 2 Sep 2024

ACE：通用约束求解器

Christophe Lecoutre，CRIL、阿图瓦大学与法国国家科学研究中心，法国朗斯

lecoutre@cril.fr



版本 2.4 – 2024年8月28日

摘要

约束编程（CP）是一种用于建模和求解组合约束问题的实用技术。一方面，可以使用 PyCSP^3 这样的库，轻松地对各种应用领域（例如调度、规划、数据挖掘、密码学、生物信息学、有机化学等）中的问题进行建模。随后，可以根据特定模型和数据直接生成问题实例。另一方面，要解决这些实例（尤其是以 XCSP^3 格式表示的实例），可以使用本文介绍的 ACE 等约束求解器。ACE 是一款用 Java 开发的开源约束求解器，专注于整数变量（包括 0/1 布尔变量）、先进的表约束、常用的全局约束、搜索启发式以及（单准则）优化。

1 引言

组合问题在我们周围的世界中无处不在。实际上，它们存在于人类活动的所有领域。例如，可能是安排工业流程中需要执行的操作（车辆、飞机或卫星的生产线），从事务数据库中提取重复模式（数据挖掘），组织某项服务的值班表（医院、大学或工业环境），生成具有良好性质的分子结构（化学或生物信息学）等。

求解优化问题仍然是一项艰巨的任务，尤其是在待求解问题的实例规模较大和／或要求达到最优性时。实际上，困难有两个方面：一是能够为遇到的问题恰当地建立模型，二是能够有效地求解这些问题的不同实例。优化的主要范式，即数学规划、元启发式和约束编程（CP），包括

布尔可满足性 框架，提供了多种有趣的工具（语言、库、软件），并且处于
一种方式，相当互 补；每种范式都有各自的成功案例。

多年来，我们实验室（CRIL）一直致力于约束编程 [20]。我们开发了一个 Python 库 PyCSP
3 [26]，，它让我们能够以声明式方式编写组合约束问题的模型。目前，借助 PyCSP^3，，您可以编写约束满足和优化问题的模型。具体来说，您可以构建以下问题的模型：

- CSP（约束满足问题）
- COP（约束优化问题）

PyCSP^3 模型编写完成后，可以提供一些数据并进行编译，以生成一个 XCSP^3 实例（文件），然后使用兼容 XCSP^3 [10, 1, 9]的约束求解器来求解该问题实例。参加年度 XCSP^3 竞赛的求解器（参见 xcsp.org/competitions 和 [2, 3, 4]），以及 ACE，都是这种情况。ACE（AbsCon Essence）源自约束求解器 AbsCon。多年来，我们团队一直在 CRIL 将 AbsCon 用作研究平台。我们从 AbsCon 中舍弃了许多想法和算法，以获得一个规模适中、易于理解的约束求解器。

目前，我们发现 ACE 有三个弱点：

1. 由于历史原因，ACE 的主类继承自一个已不再适用的类，因此引入了无用的接口和代码。这不会影响效率，但会降低代码的清晰度。我们可能会在未来决定重构代码。

2。ACE 不擅长处理非常大的域。我们计划在不久的将来解决这个问题。

3. ACE 缺少适用于全局约束 noOverlap 的优质传播器。我们计划实现一个（例如，遵循边缘查找和/或非首项/非末项规则 [17] 的原则）。

本文的组织结构如下：介绍如何运行 ACE 并简要描述其主要组成部分后，我们将详细介绍构成 ACE Java 代码结构的 13 个主要程序包。

2 运行 ACE

ACE 采用MIT许可证授权，其代码可在GitHub上获取。配置好类路径（克隆 GitHub 上的代码后），即可通过执行以下命令，让求解器处理任意 XCSP^3 [10, 1, 9] 实例（文件）：

```
java ace  <xcsp3_文件>  [选项]
```

使用：

- <xcsp3 文件>：表示一个约束满足问题或约束优化问题实例的 XCSP^3 文件
- [选项]：运行求解器时可使用的选项

本文稍后将讨论其中一些选项，但你可以通过执行以下命令显示所有选项：

```
Java ACE
```

3 ACE 简介

ACE 是一款用 Java 开发的开源约束求解器。ACE 专注于：

- 整数变量，包括 0/1（布尔）变量
- 先进的表约束（也称为外延约束），包括普通表约束、带星号表约束和混合表约束
- 常用的全局约束（allDifferent、count、element、cardinality、cumulative 等）
- 搜索启发式
- 单准则优化

这是很重要的是要注意，ACE 的外围基本上与 XCSP^3-core [10]相同。

在本节中，我们首先简要描述 ACE 如何处理约束网络的核心要素（变量、约束和目标），然后强调搜索的重要性，以及在 ACE 中如何执行（并控制）搜索。

变量。 尽管 XCSP^3, 格式可以表示多种形式的变量，例如实数变量、集合变量和图变量，但 ACE 只处理整数变量和符号变量。布尔变量只需表示为 0/1 变量，而符号变量则会在求解器内部转换为整数变量（只需为符号关联任意整数即可）。这类整数变量的域由双向链表表示，这些链表通过数组和（部分）位向量实现，从而能够保留值的递增顺序。稀疏集合这种数据结构不用于表示域，因为根据我们的经验，它们的缺点并未超过其优点。

具体来说，域的表示所采用的底层技术对应于舞蹈链，如 [18, 19] 中所述。 [27]，中也使用了舞蹈链，但撰写本文时，我们尚未注意到 Knuth 2000 年论文中的相关引用。

约束。 ACE 可以自然地处理通用形式的约束，即以内涵或外延形式表示的约束。ACE 采用原语和重ification，以高效处理内涵约束。较少见的内涵约束形式则通过通用过滤方案（广义）弧一致性 3^{rm} [31, 22, 28] 进行处理。扩展约束（也称为表约束）提供了多种过滤算法，称为传播器：Valid-Allowed（VA） [29, 27], 、简单表约简（STR） [34, 21, 23] 和 Compact-Table（CT） [12, 35]。表可以是普通表、带星号的表，甚至是混合（智能）表 [32]。ACE 处理的约束主要是 XCSP^3-core 中的约束：

- 通用约束：内涵和外延约束
- 基于语言的约束：正则约束和多值决策图
- 基于比较的约束：allDifferent、列表全异约束、全等约束、递增约束、递减约束、字典序递增约束、字典序递减约束和优先约束
- 计数约束：求和、count、不同值数量约束和 cardinality
- 连接约束：最大值、最大值索引约束、最小值、最小值索引约束、element 和通道约束
- 装箱与调度约束：noOverlap、cumulative、装箱约束和背包约束

目标。 目标是必须进行优化（即最小化或最大化）的对象。该对象可以是一般表达式（例如单个变量），也可以是与计算和、最小值、最大值或不同值的数量相对应的值。这符合 XCSP^3-core 中对单准则优化的预期。

搜索。 ACE 是一个完备求解器，通过回溯对搜索空间进行深度优先探索。在每一步（搜索树的一个节点）中，都会做出一个决策（变量赋值或值否定），并执行过滤过程（称为约束传播）。为应对重尾运行时间分布的问题， [15], 搜索会定期重新启动，遵循几何级数（或 Luby 序列）。在对搜索空间进行深度优先遍历时，变量的选择顺序由变量排序启发式决定；一种经典的通用启发式是 `dom/wdeg` [8], ，并结合一种模拟特定形式智能回跳的机制 [25]。给变量赋值时，值的选择顺序由值排序启发式决定；对于约束优化问题实例，强烈建议优先选择上一次找到的解中所包含的值，这种技术称为基于解的阶段保存 [36, 13]。

针对约束优化问题的回溯搜索依赖一种优化策略：每当找到一个解，就递减地更新最大界（假设是最小化问题）。这是一种逐步下降策略（与分支定界有关），其原理等同于向约束网络中添加一个特殊的目标约束 obj $<\infty$ （尽管该约束最初显然满足），并在每次找到新解时更新此约束的界限。这意味着，每当找到一个成本为 B =obj(S) 的解 S 时，目标约束就变为 obj $< B$。因此，这种逐步下降策略会产生一系列越来越好的解，直到不再有解为止，并保证最后找到的解是最优解。

撰写本文档时，我们认为 ACE 是一款具有竞争力的求解器，原因在于以下几点：

- 频繁重启搜索
- 记录重启过程中产生的不可行集 [24]
- 使用约束加权来选择变量 [8, 16, 37, 30, 5]
- 基于最近冲突的推理 [25]
- 使用基于解的阶段保存 [36, 13]
- 在某些问题中使用 BIVS [14]

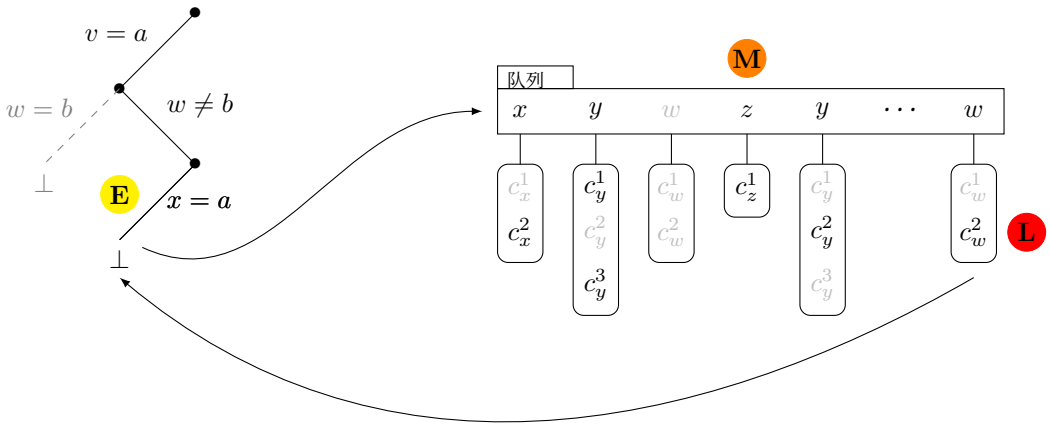

图1：用于收集冲突信息的关键时刻示意图：对应于冲突处理的早期（E）、中期（M）和后期（L）阶段

重要的是，在 [5], 中，我们表明，利用冲突指导搜索的三种不同方式表现出某种互补性。这可以解释为：信息是在不同时间点提取的——在导致冲突的过程刚开始时（即决策时）、约束传播期间，或最后一个传播器（过滤算法）被调用时。因此，可以将这些方法称为冲突的早期（E）、中期（M）和后期（L）操作处理。图1对此作了说明：在求解约束网络 P时，沿着先前作出的两个决策 $v = a$ 和$w \neq b$，作出新的决策 $x = a$ 。在我们的场景中，将值 a 赋给 x后，对 P （当前状态）运行约束传播 ϕ ，即 $\phi(P|x = a)$ ，会发现一个新的冲突（死胡同）情形（记为 $\bot$）。对这一新冲突进行早期处理，就是将参与决策的变量 x 视为罪魁祸首。这正是启发式 frba/dom [30] 的原理。对这一冲突进行中期处理，就是将传播过程中发挥过作用（即被选取过）的所有变量都视为导致失败的因素。这是启发式 pick/dom [5] 的基本原理。对这一冲突进行后期处理，就是将导致域擦除（即移除某个域中的最后一个值）的最后一个约束（此处为 c_w^2）视为关注对象。这正是约束加权的原理，例如 wdeg/dom [8, 16, 37]。

这三种主要启发式方法的互补优势，体现在2024 XCSP^{3}竞赛 [4],中；ACE的两个版本参加了该竞赛：

- ACE采用默认行为（使用单一变量排序启发式，即 wdeg^{cacd} [37]），
- ACE-mix对应于使用选项 -rr，即采用上述三种主要变量排序启发式。

显然，ACE-mix受益于这些基于冲突的启发式方法的互补优势，因为在竞赛的各个赛道中，这一版本的成绩都优于ACE。

另一方面，我们确实认为，当前约束规划社区广泛采用SAT技术（即将约束和/或基于子句的推理转换为子句）的趋势很有意思（并且已证明在不少情况下行之有效），但在某种程度上削弱了约束规划的价值。关于优化问题，在展示2024 XCSP^3 竞赛结果的网站上（参见 www.cril.univ-artois.fr/XCSP24），可以看到两张仙人掌图：

- 一种仙人掌图，从证明的角度呈现结果：基于SAT的方法在证明最优性方面非常高效，
- 一种仙人掌图，从搜索的角度呈现结果（即舍弃最优性证明）：基于约束规划的方法在找到高质量的界方面非常高效。

事实上，我们的观点是，搜索重获新生，这是在受邀演讲中提到的 [33] ，该演讲由 Peter Stuckey 在 CP'13 上发表，他声称"搜索已死，证明万岁"。这是因为求解优化问题时，我们会经历三个阶段：

- 寻找第一个解的钩子阶段，
- 寻找最优解的下降阶段，这需要从第一个解出发，沿着一条可能很长的路径前进，
- 进行最优性证明的证明阶段，

在我们看来，从一开始就只关注证明未必是正确的做法，尤其是搜索方面已有了新的进展（特别是解保存和三种互补的基于冲突的启发式）。在下降阶段，经典约束规划求解器所采用的搜索仍然极具竞争力（ACE-mix 已证明了这一点）。

4 通用包

ACE 的代码由 13个包构成。

4.1 main包

main包包含两个类:

- Head 是主对象（head）的类，负责求解一个 CSP 或 COP 实例。此类对象会请求求解器来求解问题。
- HeadExtraction 是负责提取约束的不可满足核的主对象的类。

要在问题实例 airTraffic.xml 上运行求解器，我们可以执行:

java main.Head airTraffic.xml

由于默认包中存在一个类，因此有一个快捷方式:

java ace airTraffic.xml

要从（不可满足的）实例 Rlfap-scen-11- f08.xml 中提取约束的不可满足核，我们执行:

java main.HeadExtraction Rlfap-scen-11-f08.xml

4.2 包 `dashboard`

在 dashboard包 中，我们可以看到以下三个类:

- Control，用于管理与问题（通常是其表示方式）和求解器（通常是执行搜索的方式）相关的所有选项。
- Input，用于处理用户通过命令行提供的参数。这些参数可能与要解决的问题有关，也可能更广泛地涉及求解过程（即要选择的选项，例如要使用哪种搜索启发式）。
- 输出，其作用是输出与问题实例求解过程有关的数据/信息。这些数据通过 XML 文档收集，并可保存。部分数据也会直接显示在标准输出上。

Control 类相当核心，因为其中包含许多重要选项。用户可以通过命令行指定这些选项的值。例如，在 Control 类的一个内部类（表示一个选项组）中，我们可以看到:

长整数 timeout = addL("timeout", "t", PLUS_INFINITY, "停止前的毫秒数上限；也可以像 -t=10s 这样以秒为单位指定");

方法 addL 的第二个参数是命令行中使用的快捷方式。因此，要在最多 10 秒内运行求解器求解问题实例 airTraffic.xml，我们需要执行:

java ace airTraffic.xml -t=10s

执行 java ace 时，会简要介绍所有选项。

4.3 包接口 `interfaces`

了解 Java 接口在 ACE 代码中的作用非常重要。

4.3.1 标签

有些接口用作标签（也就是说，它们不包含任何代码）：

- TagSymmetric，用于表示约束完全对称的标签
- TagNotSymmetric，用于表示约束完全不对称的标签
- TagCallCompleteFiltering，用于表示约束在每次调用时都能执行完整过滤的标签（不仅限于围绕最后接触的变量）
- TagAC，用于表示约束保证实施（广义）弧一致性的标签
- TagNotAC，用于表示约束不保证实施（广义）弧一致性的标签
- TagNegative，用于表示表约束为负约束（即包含冲突）的标签
- TagPositive，用于表示表约束为正约束（即包含支持）的标签
- TagStarredCompatible，用于表示约束可能包含带星号元素（即可能包含 *）的标签
- TagMaximize，用于表示某个对象（例如启发式）旨在最大化一个表达式（变量、求和、最大值等）的标签

4.3.2 观察器

有些接口用于观察。

- 构造观察器，用于观察主要对象（问题和求解器）的构造。方法如下：– 任意构造前，在开始构建主要对象（问题和求解器）之前调用– 问题构造后，在问题构造完成时调用– 求解器构造后，在求解器构造完成时调用
- 求解过程观察器，用于观察求解的主要步骤（预处理和搜索）。方法如下：

  – 求解前，在开始求解（先进行预处理，再进行搜索）之前调用– 预处理前，在开始预处理之前调用– 预处理后，在预处理完成后调用– 搜索前，在开始（回溯）搜索之前调用– 搜索后，在（回溯）搜索完成后调用– 求解后，在求解（先进行预处理，再进行搜索）完成后调用
- 运行观察器，用于观察求解器执行的连续运行（可能只有一次运行，若已停用重新启动）。方法如下：

  – 运行前，在下一次运行开始前调用– 运行后，在当前运行结束后调用
- 回溯观察器，用于观察求解器执行的回溯。方法和子接口包括：

– restoreBefore(回溯前的整数深度)，当因从指定深度发起的回溯而需要恢复时调用– 接口 ObserverOnBacktracksSystematic，用于观察求解器执行的回溯。适用于系统性地需要恢复的观察器– 接口 ObserverOnBacktracksUnsystematic，用于观察求解器执行的回溯。适用于不系统地需要恢复的观察器。

- ObserverOnDecisions，用于观察求解器所做的决策。方法如下：

  – beforePositiveDecision(Variable x, int a)，在即将做出正决策（变量赋值 $x = a$）时调用– beforeNegativeDecision(Variable x, int a)，在即将做出负决策（值否定 $x \neq$）时调用

- ObserverOnAssignments，用于观察求解器所做的变量赋值。方法如下 ：

  – afterAssignment(Variable x, int a)，在完成变量赋值 $x = a$后调用– afterUnassignment(Variable x)，在变量 x 被取消赋值后调用（由于回溯）

- ObserverOnRemovals，用于观察求解器对域的缩减（通常在过滤时进行）。方法如下：

  – afterRemoval(Variable x, int a)，当域 x 的索引 a （对应某个值）被移除时调用– afterRemovals(Variable x, int nRemovals)，当变量 x的域被缩减时调用；删除值的数量会被指定

- 冲突观察器，用于观察求解器在搜索过程中遇到的冲突。其方法包括：– whenWipeout(Constraint c, Variable x)：当使用指定约束进行过滤时，指定变量的域变为空（即所谓的域清空）时调用– whenBacktrack：求解器即将回溯时调用

4.3.3 特定过滤

有一个重要的接口，用于指定某个约束拥有自己的过滤算法（传播器）。

```
public interface SpecificPropagator {

   /**
* Runs the propagator (specific filtering algorithm) attached
* to the constraint implementing this interface, and returns
* false if an inconsistency is detected.
* We know that the specified variable has been picked from the
* propagation queue, and has been subject to a recent reduction
* of its domain.*
* @param evt  a variable whose domain has been reduced
* @return false if an inconsistency is detected*/
boolean runPropagator(Variable evt);}
```

4.4 包集合 sets和 utility

ACE 中有三种主要的集合实现（集合是重要的数据结构），可在 sets 包中找到：

- SetDense（稠密集合）基本由一个数组（存放整数）和一个上限组成：任意时刻，它都包含数组中索引从 0 到该上限（含）位置上的元素（通常是值索引）。

- SetSparse（稀疏集合） [11]基本由两个数组（存放整数）和一个上限组成：任意时刻，它都包含数组“dense”中索引从 0 到该上限（含）位置上的元素（通常是值索引）。可以通过数组“sparse”检查元素是否存在。稀疏集合始终是简单的，也就是说，“dense”中的值必定是索引 0、1、2.....此外，我们还有稠密[稀疏[值]] = 值。

- SetLinked（链式集合）是一种接口，用于表示一个被视为索引的元素列表，也就是说，列表中的元素值范围为 0 到指定容量减 1。例如，如果对象的初始大小（容量）为 10，那么索引/元素列表就是 0、1、2.....、9。可以从列表中移除索引，可以按前向或后向顺序遍历当前存在的索引，也可以遍历已删除的索引。每个已删除的索引都关联一个层级。这类接口主要用于管理变量域的值索引。粗略地说，SetLinked 可以看作是$dancink\ links$的一种实现，如 [18, 19]中所述。

有几个子类：

- SetDenseReversible，一种可在不同搜索层级处理的稠密集合
- SetSparseReversible，一种可在不同搜索层级处理的稀疏集合
- SetLinkedBinary 和 SetLinkedFinite，分别用于仅包含两个元素（索引为 0 和 1）的集合，以及包含有限个元素（索引）的有序集合。

在 utility 包中，有几个包含实用方法的类：

- Bit，包含用于位向量的方法
- Kit，包含各种实用方法
- Reflector，包含用于执行反射操作的方法
- Stopwatch，包含用于测量求解问题时某些操作所耗时间的方法

5 问题表示相关包

5.1 包 problem

problem 包含主类 Problem，以及一些辅助类。该包的主要类包括：

- Problem，用于表示问题实例（约束网络）的类
- Features，用于存储有关问题实例的各种信息（特征，例如域大小、约束类型等）以及显示方式的类
- 强化器，这个类包含一些内部类，可通过添加冗余约束或对称性破缺约束来强化约束网络。

• XCSP3，这个类允许我们加载 XCSP[3] 格式的实例。该类属于接口部分，而 Problem 类属于实现部分。这种分离是出于历史原因（源自 API JvCSP[3]），但未来可能会取消这种分离，以简化代码。

5.2 Package `variables`

variables 包含用于定义变量和域的类。它还包含一个类，可用于遍历指定域的笛卡尔积中的元组。该包的主要类有：

• Variable 是所有参与问题的变量的抽象根类，目前有两个内部子类：VariableInteger 和 VariableSymbolic。域附属于变量，对应于可赋给该变量的（有限）值集合。为变量赋值时，该变量的域会缩减为该值。求解器尝试为变量赋值时，会使用值排序启发式来确定首先尝试哪个值。当然，一个变量可以出现在其所属问题的不同约束中。

• Domain 是所有变量的域的抽象根类。一个域由一组整数值组成。域最初是满的，但在进行逻辑推理（使用约束）时通常会缩减。处理域时，为了简化编程，通常会遍历值索引；如果域包含 d 个值，那么索引范围为 0 到 d-1。例如，如果域是值集合 $\{1, 4, 5\}$，它们对应的索引分别为 $\{0, 1, 2\}$。值索引与值之间的对应关系由 toIdx 和 toVal 方法给出。Domain 的子类有：

– DomainBinary，表示二元域的类，通常只包含 0 和 1。– DomainRange，表示由两个（整数）界之间的整数列表组成的域的类。– DomainValues，表示由不一定连续的整数列表组成的域的类。请注意，这些值已排序。

请注意，DomainRange 和 DomainValues 是 DomainFinite 的两个内部子类 .

• TupleIterator 是一个类，它允许我们遍历变量序列的各个域的笛卡尔积。例如，它允许我们找到第一个有效元组，或找到指定元组（或上次记录的元组）之后的下一个元组。每个约束都配有这样一个对象。请注意，迭代使用的是值索引，而不是直接使用值。

5.3 constraints 包 `constraints`

constraints 包包含许多表示各种约束类型的类。约束可以通过以下方式定义：

• 外延地通过表或图来定义

• 通过内涵方式，借助通用布尔树表达式，这些表达式通常对应于称为原语的经典形式

• 来自一个知名模板（所谓的全局约束）。

因此，该软件包中更抽象的类如下：

• Constraint，任何约束的根类。回顾一下，约束附属于一个问题，涉及该问题的一部分变量，并帮助我们进行推理，从而筛选搜索空间（即变量的域）。

• ConstraintExtension，用于表示外延约束（也称为表约束）的根类。两个直接子类是 ExtensionGeneric（用于实现弧一致性过滤 `，采用 AC3^{rm}方法）和 ExtensionSpecific（用于实现特定的传播器）。用于过滤外延约束的算法位于子包 extension 中：– STR1、STR2 和 STR3– CT– CMDD

• ConstraintIntension，用于表示内涵约束的根类；内涵约束是指其语义由涉及变量的布尔表达式树给出的约束。大多数情况下，人们会使用原语，而不是这种通用形式。用于过滤原语和重化约束的算法位于子包 intension 中。

• ConstraintGlobal，用于表示全局约束的根类；全局约束本质上是具有特定过滤形式（传播器）的约束。用于过滤全局约束的算法位于子包 global 中。

6 关于求解过程的软件包

6.1 Package `solver`

软件包 solver 包含主类 Solver，以及一些用于管理未来变量（即未赋值变量）、决策、解、重启、最近冲突和统计信息的辅助类。该软件包的主要类包括：

• Solver，用于解决问题的主对象所属的类。

• FutureVariables，用于管理过去变量和未来变量的对象所属的类。

• Decisions，用于存储求解器在搜索过程中所做决策集合的对象所属的类。目前，每个决策都以整数的形式编码。不过，未来计划使用长整数而不是整数来表示每个决策。

• Solutions，用于记录和显示解（以及界等）的对象所属的类。

• Restarter，用于管理重启的对象的根类（从根节点开始的连续运行并重新启动）。

• `LastConflicts`，实现最近冲突推理（lc）的对象所属的类；请参见 [25]。

• Statistics 类是一个对象的类，可用于收集求解器的所有统计信息（例如回溯次数）。

6.2 Package `propagation`

在包传播中，我们可以找到一些类，它们实现了不同的过滤过程（或约束传播）方式，并且都继承自根类 Propagation。例如，其中包括分别对应弧一致性（广义弧一致性）、SAC（单例弧一致性）和 GIC （全局逆一致性）的类。为简便起见，

传播 a nd 一致性概念不作区分。这就是为什么有些子类被赋予
c 的名称 致性。
子类包括:

• 后向传播是后向传播的根类。这种传播形式对应于一种回溯方法，处理已赋值变量，也就是说，未赋值变量的域永远不会被修改。其子类包括 BT（基本回溯算法）和 GT（“生成并测试”算法）。

• 前向传播是前向传播的根类。这种传播形式对应于一种前瞻方法，处理未赋值变量，也就是说，未赋值变量的域可以被筛选。其子类包括:

– FC，前向检查（FC）的类– AC，（广义）弧一致性（AC）的类。这种传播对象会调用每个约束传播器（即附加到约束上的过滤算法），直到达到不动点（与 FC 相反）。请注意，只有当每个传播器都确保 AC 时，完整约束网络上的 AC 才算真正得到完全强制执行。需要指出的是，当约束被独立处理时，AC 是可能达到的最大过滤层级。还请注意，为简便起见，无论约束的元数是多少，我们都使用缩写 AC（而不是 GAC）。– GIC，全局逆一致性（GIC）的类。这种一致性非常强，因为一旦得到强制执行，它就能保证每个文字 (x,a) 至少属于一个解。它只能在某些特殊问题上执行。例如，请参见 [7]。– SAC，单例弧一致性（AC）的类。有关 SAC 以及实现该一致性的算法的信息，例如可以在 [6] 中找到。

6.3 包学习 learning

在包 学习中，我们可以找到与约束学习相关的类。其中一些与
不可行集有关，另一些 则与所谓的不一致部分状态（IPS）有关。

备注 1 目前，出于历史和维护方面的原因，*IPS* 学习已停用。我们计划在不久的将来重新启用它。

搜索过程中可以学习的实体来自以下类:

• Nogood；严格来说，这个类的对象被当作禁忌约束使用，也就是说，它是必须强制成立的否定决策析取式。不过，在约束传播过程中，不可行集会单独处理。

• Ips；IPS 是不一致部分状态。它由成员资格决策组成，等价于广义禁忌。为了识别和推理这类实体，我们有以下类:

• NogoodReasoner，它允许我们记录禁忌并对其进行推理。

• IPsReasoner，它允许我们记录不一致部分状态（IPS），并基于这些状态进行推理。推理可以基于等价或支配。目前，学习仅限于通过重启进行禁忌记录。

6.4 包 heuristics

6.4.1 变量排序启发式

变量排序启发式位于启发式包中，由继承自 HeuristicVariables 的 Java 类实现。假设我们要实现经典的动态变量排序启发式 dom，它会选择域大小最小的变量。我们需要编写一个继承自 HeuristicVariablesDynamic 的类，并为 scoreOf 方法指定方法体，如下所示：

```
public class Dom extends HeuristicVariablesDynamic {
  public Dom(Solver solver, boolean anti) {
    super (solver, anti);
  }

  @Override
  public double scoreOf (Variable x) {
    return x.dom.size();
  }
}
```

如果必须选择得分最高的项（而不是最小的项，如 dom），则该类必须标记为：implements TagMaximize。添加新的启发式后，我们可以使用选项 -varh 来选择它。例如，要使用 dom 作为变量排序启发式，在问题实例 airTraffic.xml 上运行求解器，只需执行：

java ACE airTraffic.xml -varh=Dom

如果出于某些原因，我们想使用反启发式，则执行：

java ace airTraffic.xml -varh=Dom -anti_varh

已实现的启发式包括：

- Rand
- Dom
- DDegOnDom
- Wdeg
- WdegOnDom
- FrbaOnDom
- PickOnDom
- 活动度与影响度，含基本实现

对于约束加权启发式 Wdeg 和 WdegOnDom，存在四种变体：

- VAR，一种基本变体
- UNIT，经典加权方法，如 [8]中所述
- CACD（精化加权），如 [37]中所述
- CHS，如 [16]中所述

默认情况下，选择 Wdeg 的变体 CACD。要指定变体，必须使用选项 -wt。例如，要使用 WdegOnDom 的变体 CHS 在 airTraffic.xml 上运行求解器，我们需要执行：

java ace airTraffic.xml -varh=WdegOnDom -wt=chs

若要在 2024 XCSP[3] 竞赛中使用 ACE-mix 的搜索策略，请添加选项 -rr，如下所示：

java ace airTraffic.xml -rr

6.4.2 值排序启发式

值排序启发式可在启发式包中找到，它们由继承自 HeuristicValues 的 Java 类实现。假设我们要实现 rand 值排序启发式，以随机方式从所选变量的域中选取一个值。我们需要编写一个继承自 HeuristicValuesDirect 的类，并为方法 computeBestValueIndex 指定方法体，如下所示：

```
public class Rand extends HeuristicValuesDirect {
  public Rand (Variable x, boolean dummy) {
    super (x, dummy);
  }

  @Override
  public int computeBestValueIndex() {
    return x.dom.any();
  }
}
```

有了新的启发式后，我们可以使用选项 -valh 来选择它。例如，若要使用 rand 作为值排序启发式来运行求解器并求解问题实例 airTraffic.xml，只需执行：

java ace airTraffic.xml -valh=Rand

在已实现的启发式中，我们可以找到：

- Rand
- First and Last
- Robin and RunRobin
- BIVS，定义见 [14]

6.5 优化包 `optimization`

优化包中包含用于管理优化的类。该包的主要类包括：

- Optimizer 是用于处理（单目标）优化的引导对象的抽象根类。其子类定义了多种面向最优性的搜索策略：– OptimizerDecreasing：一种优化策略，在找到解时递减更新最大界（假设是最小化问题）；这与分支定界（以及逐步下降策略）有关。– OptimizerIncreasing：一种优化策略，递增更新最小界（假设是最小化问题）；有时称为迭代优化（或逐步上升策略）。

– OptimizerDichotomic：一种基于界定区间二分缩减的优化策略

- 可优化接口，任何约束都必须实现该接口，以表示一个目标
- 目标变量，在问题实例（约束网络）的目标由一个变量给出，且该变量的值必须被最小化或最大化时所使用对象的抽象根类。内部子类为 ObjVarLE 和 ObjVarGE。

一些有用的链接

- PyCSP^{3}网站： pycsp.org
- ACE GitHub： github.com/xcsp3team/ace
- XCSP^{3}网站： xcsp.org

致谢

本研究获得了“上法兰西大区”CPER CornelIA 项目的支持，以及法国国家研究署在法国2030框架下的 MAIA 项目 ANR-22-EXES-0009 的支持。

参考资料

[1] G. Audemard、F. Boussemart、C. Lecoutre、C. Piette 和 O. Roussel。XCSP3 及其生态系统。《Constraints》，25(1-2):47–69，2020 年 2 月。Springer。[2] G. Audemard、C. Lecoutre 和 E. Lonca。2022 年 XCSP3 竞赛论文集。技术报告 arXiv:2209.00917，CoRR，2022 年。https://arxiv.org/abs/2209.00917。[3] G. Audemard、C. Lecoutre 和 E. Lonca。2023 年 XCSP3 竞赛论文集。技术报告 arXiv:2312.05877，CoRR，2023 年。https://arxiv.org/abs/2312.05877。[4] G. Audemard、C. Lecoutre 和 E. Lonca。2024 年 XCSP3 竞赛论文集。技术报告，即将发表，2024 年。[5] G. Audemard、C. Lecoutre 和 C. Prud’Homme。通过在约束传播过程中跟踪变量来引导回溯搜索。载于 CP’23 论文集，第 9:1–9–7 页，2023 年。[6] C. Bessiere、S. Cardon、R. Debruyne 和 C. Lecoutre。单例弧一致性的高效算法。《Constraints》，16(1):25–53，2011 年。[7] C. Bessiere、H. Fargier 和 C. Lecoutre。用于交互式约束满足的全局逆一致性。载于 CP’13 论文集，第 159–174 页，2013 年。[8] F. Boussemart、F. Hemery、C. Lecoutre 和 L. Sais。通过约束加权提升系统搜索效率。载于 ECAI’04 论文集，第 146–150 页，2004 年。[9] F. Boussemart、C. Lecoutre、G. Audemard 和 C. Piette。XCSP3：用于组合约束问题基准测试的集成格式。技术报告 arXiv:1611.03398，CoRR，2016 年。https://arxiv.org/abs/1611.03398。[10] F. Boussemart、C. Lecoutre、G. Audemard 和 C. Piette。XCSP3-core：用于表示约束满足/优化问题的格式。技术报告 arXiv:2009.00514，CoRR，2020 年。https://arxiv.org/abs/2009.00514。

[11] P. Briggs 和 L. Torczon。稀疏集合的高效表示。《ACM Letters on Programming Languages and Systems》，2(1-4):59–69，1993 年。[12] J. Demeulenaere、R. Hartert、C. Lecoutre、G. Perez、L. Perron、J.-C. R´egin 和 P. Schaus。Compact-Table：使用可逆稀疏位集合高效过滤表约束。载于 CP'16 论文集，第 207–223 页，2016 年。[13] E. Demirovic、G. Chu 和 P. Stuckey。面向 CP 的基于解的阶段保存：一种在完备求解器中模拟局部搜索行为的值选择启发式。载于 CP'18 论文集，第 99–108 页，2018 年。[14] J.-G. Fages 和 C. Prud'homme。让第一个解更好！载于 ICTAI'17 论文集，第 1073–1077 页，2017 年。[15] C. Gomes、B. Selman、N. Crato 和 H. Kautz。可满足性问题和约束满足问题中的重尾现象。《Journal of Automated Reasoning》，24:67–100，2000 年。[16] D. Habet 和 C. Terrioux。基于冲突历史的约束满足问题搜索。载于 SAC'19 论文集，第 1117–1122 页，2019 年。[17] J.N. Hooker。《优化集成方法》。Springer，2012 年。[18]D. E. Knuth。舞蹈链。载于《计算机科学的千年展望：1999 年牛津大学—微软研讨会论文集（纪念 Tony Hoare 爵士）》，第 187–214 页，2000 年。https://arxiv.org/pdf/cs/0011047。[19] D. E. Knuth。《舞蹈链》，《计算机程序设计艺术》第 4 B 卷第 7.2.2.1 节。Addison-Wesley，2022 年 10 月。[20] C. Lecoutre。《约束网络：技术与算法》。International Scientific and Technical Encyclopedia (ISTE Ltd) - John Wiley Inc.，2009 年 6 月。共 592 页。ISBN：9781848211063。[21] C. Lecoutre。STR2：针对表约束的优化简单表约简。《Constraints》，16(4):341–371，2011 年。[22] C. Lecoutre 和 F. Hemery。弧一致性中的残余支持研究。载于 IJCAI'07 论文集，第 125–130 页，2007 年。[23] C. Lecoutre、C. Likitvivatanavong 和 R. Yap。STR3：针对表约束的路径最优过滤算法。《Artificial I ntelligence》，220:1–27，2015 年。[24] C. Lecoutre、L. Sais、S. Tabary 和 V. Vidal。记录并最小化重启产生的不可行集。《Journal on Satisfiability, Boolean Modeling and Computation (JSAT)》，1:147–167，2007 年。[25] C. Lecoutre、L. Sais、S. Tabary 和 V. Vidal。约束编程中基于最近冲突的推理。《Artificial Intelligence》，173(18):1592–1614，2009 年。[26] C. Lecoutre 和 N. Szczepanski。PyCSP^3：使用 Python 对组合约束问题建模。技术报告 arXiv:2009.00326，CoRR，2020 年。https://arxiv.org/abs/2009.00326。[27] C. Lecoutre 和 R. Szymanek。正表约束的广义弧一致性。载于 CP'06 论文集，第 284–298 页，2006 年。[28] C. Lecoutre 和 J. Vion。使用位运算实现弧一致性。《Constraint Programming Letters》，2:21–35，2008 年。[29] O. Lhomme 和 J.-C. R´egin。n 元约束的快速弧一致性算法。载于 AAAI'05 论文集，第 405–410 页，2005 年。

[30] Hongbo Li、Minghao Yin 和 Zhanshan Li。用于求解约束满足问题的基于失败的变量排序启发式。载于 *CP'21* 论文集，第 9:1–9:10 页，2021 年。[31] A.K. Mackworth。关系网络中的一致性。人工智能，8(1):99–118，1977 年。[32]J.-B. Mairy、Y. Deville 和 C. Lecoutre。智能表约束。载于 *CPAIOR'15* 论文集，第 271–287 页，2015 年。[33] P. Stuckey。不能铭记过去的人注定会重蹈覆辙。载于 *CP'13* 论文集，第 5–6 页，2013 年。https://www.youtube.com/watch?v=lxiCHRFNgno。[34] J.R. Ullmann。非二元约束满足的划分搜索。信息科学，177:3639–3678，2007 年。[35] H. Verhaeghe、C. Lecoutre 和 P. Schaus。将 Compact-Table 扩展至负表和短表。载于 *AAAI'17* 论文集，第 3951–3957 页，2017 年。[36] J. Vion 和 S. Piechowiak。一种将 DFS 与大邻域搜索用于约束优化问题相结合的简单启发式。载于 *JFPC'17* 论文集，第 39–45 页，2017 年。[37] H. Wattez、C. Lecoutre、A. Paparrizou 和 S. Tabary。改进约束加权。载于 *ICTAI'19* 论文集，第 71–77 页，2019 年。