# 

# 第一章 引言

## 1.1 项目背景

软件工程是应用系统的、规范的、可量化的方法来开发、运行和维护软件。随着软件工程几十年来的飞速发展，软件开发流程——软件生命周期变得越来越成熟。软件设计在软件生命周期过程的比重越来越大，一些团队甚至将设计的时间占到了整个开发时间的百分之五十以上。因为高质量的软件依赖高质量的软件设计，同时软件中的问题如果能在设计的过程中被发现，那么修复它的代价远小于在之后的过程中被发现再去修复的代价，所以设计过程的验证就成为了关键性的问题。在设计过程中，最常见的情况之一是使用场景描述交互过程，而场景交互的描述最常用的就是使用UML顺序图和UML交互概观图，其中UML顺序图描述单个场景，UML交互概观图描述多个场景的组合，二者共同来描述系统成员间的交互，构成了UML交互模型 。UML顺序图中的时间约束和UML交互概观图中的时间约束构成了该UML交互模型的时间约束集合。

时间是交互系统的重要属性之一，本论文旨在验证UML顺序图和UML交互概观图组成的交互模型的时间性质是否满足，同时对不满足时间约束的UML顺序图和UML交互概观图进行一定程度的纠正，并在此基础上，针对某个目标进行优化，使得取得最终的最优值。通过上述过程，能够及时发现UML顺序图和UML交互概观图在时间方面的缺陷，并提出合理的改进方案，同时能够利用线性规划得到某个具体目标的最优值，对软件设计中UML顺序图和UML交互概观图的时间正确性验证起到极大的辅助作用。

## 1.2研究现状

UML交互模型自诞生至今，经过了数次迭代发展，已经成为软件设计过程中最重要的一部分，越来越多的应用于软件设计的过程中，进行系统组件的交互设计。UML交互模型中的UML顺序图和UML交互概观图在大型的复杂的系统设计方面有着自己的优势，因此使用的频率和范围都越来越广，比较常见的应用是提供系统设计、建模，模拟系统行为等。

对UML交互模型的正确性验证国内外已经有了不少的研究，但是对时间性质的分析较少。潘敏学的论文《Timing analysis of MSC specifications with asynchronous concatenation》对时间性质进行了满足性验证，但是该论文基于图遍历算法，性能有待提高。本文借鉴了《SAT---LP---IIS joint-directed path-oriented bounded reachability analysis of linear hybrid automata》论文的SAT-LP-IIS的技术，对UML交互模型进行时间满足性进行验证的效率较高，具有较好的可行性。

在进行线性时间满足性验证的过程中，经常会发现UML交互模型无法满足线性时间约束的情况。这种情况的发生意味着UML交互模型在设计过程中出现了错误，需要对错误进行更正。这种纠正往往是必要的，尤其对于一些时间关键性交互系统，比如医疗器材软件，错误的存在可能是致命的。UML交互模型的时间性质是由线性约束定义的，对于如何让线性约束从无解变的有解，业界在这方面的研究较少，没有成熟且被人公认的办法。本论文从多个角度多个目标出发，针对现有的与时间相关的交互系统，提出了如下几种优化方案：

1. 时间不敏感系统——扩大约束范围

2. 时间关键系统——缩小约束范围

3. 时间敏感系统——小幅调整约束范围

4. 对软约束、硬约束的处理

本论文借助了IIS技术，获得了最小不可解的约束，修正的范围被定位于当前最小不可解约束，使得修正效率更高。同时针对不同目标的与时间相关的交互系统，设计了不同的修正方案，保证在修正时间约束的同时，不违背交互系统的特性。有些系统的时间约束是硬性的，且不可更改的，对于这种硬约束，本论文也提出了适当的改进建议。

同时，本论文还在前面研究的基础上，针对单一目标求解最优值。线性规划的求解在业界已经十分成熟，比较普遍使用的方法有单纯形法、内点法、遗传算法、神经网络算法等等。前两种算法是比较传统的办法，已经在MATLAB， Cplex等多个科学计算软件中得到了实现，后两种算法是随着人工智能的兴起而逐渐应用于线性规划问题的求解中去的。线性规划作为一种优化问题，是完全可以使用遗传算法来高效的求得最优解的。前人对线性规划的研究，规模一般较小，对于单纯形法和遗传算法在复杂问题下的效率没有直观的体现。而UML交互模型所涉及到的时间约束一般比较复杂，约束个数可能达到数千个。本论文对比了单纯形法和遗传算法在复杂问题下的效率问题。单纯形法借助了IBM的cplex软件，遗传算法则是在GENOCOP算法的基础上进行了一定程度的修改而来。通过对比两个算法，能够发现在复杂情况下处理线性规划问题的比较高效的方式。

## 1.3 本论文的主要工作

UML顺序图和UML交互概观图是软件设计中的常用模型，本论文的主要工作就是验证这两个图的时间性质可满足性，针对不满足的情况进行优化更正，并求解单一目标情况下的最优解。对主要工作的详细描述如下:

一、UML交互模型的时间满足性验证。首先使用轻量化xml 分析工具SAX 分析Umlet工具建立的UML顺序图和UML交互概观图的xml模型，将分析获得的图模型存储在内存中，方便以后随时取用。之后，读取内存中的图模型，使用SAT 分析图模型潜在的路径。UML顺序图和UML交互概观图的时间性质由线性约束来定义，可以通过线性规划来验证其满足性，对于不满足的部分，分析线性约束的不可约简的不可解集，并转换成路径编码添加到SAT模块中，避免下次继续重复寻找该潜在路径，提高执行效率。

二、对于不满足的线性时间约束的纠正。本论文根据不同的交互系统，从多个角度多个目标出发，提出了三种系统以及一种特殊需求的处理方式，分别是：

1. 时间不敏感系统——对于这种系统，时间的限制往往不是关键，往往最优先考虑的是可满足性。因此通过尽可能少的修改某个或者某些约束，使之范围扩大，从而满足时间约束。

2. 时间关键系统——比如某些医疗设备的系统，火箭发射系统等等，时间要求往往是关键的甚至致命的。因此要尽量缩小时间范围，使之不能超过某个阈值，从而满足时间约束。

3. 时间敏感系统——对于这种系统，时间的剧烈波动可能导致系统不稳定，例子还是火箭发射系统，在进行轨迹纠正时，时间不能过长，也不能过短，避免过度纠正。虽然现在都是智能控制系统，但是控制系统也需要和轨迹纠正系统进行交互。因此通过对多个约束进行小幅度的修改，使约束满足。

三、针对某个目标函数，求解出该目标函数的最优解。本论文采取了单纯形法和遗传算法分别求解，单纯形法使用了IBM的CPLEX工具；遗传算法则借鉴了GENOCOP算法，针对本论文研究的问题，选择合适的初始种群和杂交变异算子，使得种群朝着最优解进化。并对比了两种算法的效率。

## 1.4 本论文的组织结构

本论文的组织结构为：

第二章将简单介绍UML交互模型中的UML交互概观图和UML顺序图，使用UMLet绘制交互模型并解析。

第三章介绍基于SAT-LP-IIS的验证算法的设计和实现，先分别单独介绍基于SAT的模块，基于LP的模块和基于IIS的模块的设计和实现，再将这些模块整合起来，最终实现对基本UML顺序图和UML交互概观图的时间性质满足性的验证。

第四章将对不可约不可解（IIS）约束集进行分目标优化，使得IIS约束集变得有解。这里将和现实中的例子相联系，将与时间有关的交互系统分为三大类，根据这三类系统的目标，分为不同的策略对IIS约束集进行优化。

第五章是针对某个单一的目标函数，分别使用CPLEX工具所集成的单纯形法和遗传算法来进行求解最优值，比较二者的效率。

# 第二章 UML交互模型

UML [[12](#_ENREF_12)] 交互模型包含UML顺序图 [[3](#_ENREF_3)] 和UML交互概观图 [[4](#_ENREF_4)]，UML顺序图描述单个场景，UML交互概观图描述多个场景的组合，由此可知UML交互概观图由UML顺序图构成，二者是用来描述系统成员之间交互的图文模型，而系统成员可以是一组对象，也可以是一组进程，它们是现实事物在软件系统中的抽象。本章主要介绍UML顺序图和UML交互概观图的定义 。

## 2.1 UML顺序图和时间约束

UML顺序图 [[3](#_ENREF_3)] 是包含有限个实例、有限个事件、有限个实例间信息交互流、有限个时间约束的图，它描绘了一个没有选择和迭代的基本场景。

图2.1 UML顺序图

图2.1是一个UML顺序图，UML顺序图描述单个场景中系统实例之间的信息流。在一个UML顺序图中，垂直的轴线表示实例，实例的名称在顶端。信息交互通过实例之间的箭头来表示，箭头的发出端和接收端分别表示消息发送事件和消息接收事件。比如图2.1中，含有三个实例Controller、Elevator和Door，箭头SpeedUp表示加速的消息，e1表示SpeedUp消息的发送事件， e2表示SpeedUp消息的接收事件。

在现实世界中，每个事件都会有发生的时间，用事件的名称来表示事件发生的时间，向图中添加事件间关于时间的线性约束条件，比如图2.1中的5 ≤ e15 - e14 ≤ 10，就是要求事件e15发生的时间减去e14 发生的时间的差不能小于5，并且不能大于10。5和10都是抽象的时间概念，表示5个单位或者10个单位的时间，根据现实情况，单位可以是秒，也可以是分钟。在这个例子中，e14和e15之间的差可以看做是电梯开门的持续时间，根据实际情况，要求开门持续时间在5-10秒之内。在实际实践中，经常会遇到更加复杂的情况，因此会有更加复杂的约束条件。这些线性约束条件被称作基本时间约束，其公式的定义为公式1 [[5](#_ENREF_5)]：

**a≤∑ci（es-et）≤b (0≤i, s, t≤n) （公式1）**

公式1中，a, b, ci表示实数，a，b是要约束的时间，ci是参数，事件es和事件et分别表示各自的发生时间，n是事件的个数。在图2.1的例子中，电梯的上升和下降会有一个加速的过程，在停止的时候也会有一个减速的过程，这个过程就需要时间来约束，以避免突然的加速和减速。因此图2.1包含0 ≤4(e4−e2)−(e12−e2) ≤ ∞ 和0 ≤ 4(e12−e8)−(e12−e2) ≤∞ 两个约束，这两个约束表示加速的时间或减速的时间在整个上升或者下降的过程中占得比例不能小于百分之二十五。

本质上来说，UML顺序图是由实例以及实例之间的消息流构成的，实例可以是现实世界对象的抽象，也可以是系统进程等，消息流包含事件的发出者和事件的接受者。两个事件e和e＇相邻，且事件e发生在事件e＇之前，它俩构成一组事件对(e，e＇)，事件对中两个事件的关系有三种情况：

因果关系：e是发送事件，e＇是接收事件，e事件发送的消息由e＇接收。发送事件e的发生是接收事件e＇发生的原因。

控制关系：e和e＇在同一个实例的轴线上，e在e＇的上方，e是消息接收序列，并且e＇是一个消息发送序列。e接收到消息之后，控制e＇发送消息

先进先出序列关系：相邻接收事件e和e＇在同一个实例的轴线上，同时，他们相关的发送方也在同一个轴线上。

根据这三个关系，UML顺序图还描绘了事件发生的轨迹，图2.1中，事件的轨迹为e1 🡪 e2 🡪 e3 ……e14 🡪 e15 🡪 e16。其中(e1，e2)为因果关系，(e2，e3)为先进先出序列关系，(e4，e5)为控制关系，这条事件轨迹要满足图中定义的时间约束。

## 2.2 UML交互概观图和时间约束

多个UML顺序图 [[3](#_ENREF_3)]，按照一定的顺序组织，并用箭头表示它们的执行顺序，加上开始节点和结束节点，以及各个节点之间的高层时间约束，就构成了UML交互概观图 [[4](#_ENREF_4)]。开始节点和结束节点只是表示开始状态和结束状态，并不是UML顺序图。

图2.2是UML交互概观图，这张图中，左上角的实心圆为开始节点，右下角的圆为结束节点。B1、B3、B2是UML顺序图，属于UML交互概观图的每个基本UML顺序图都至少有1个入口和至少1个出口。比如图2.2中，B1 有三个入口，分别来自开始节点、B3、B2；有一个出口，指向B3。

图2.2 UML交互概观图

A和B都是来自同一个UML交互概观图的非开始节点和非结束节点的不同节点，表示不同的UML顺序图。变量ai是来自UML顺序图A的事件，变量bj是来自UML顺序图B的事件，那么m<= bj - ai<=n是关于节点A和节点B的高层时间约束，m和n分别表示不同的抽象时间，且m<n，i和j分别是a和b的下标，且事件B发生在事件A之后。

假设图2.2中节点B1中的事件都用小写字母a和数字下标构成，节点B2中的事件都用小写字母b和数字下标构成，节点B3中的事件都用小写字母c和数字下标构成，那么可能存在的节点间的高层时间约束的例子见图2.3。

图2.3 节点间的高层时间约束

由于图2.3约束是软件设计过程中UML交互概观图的设计者给出的，因此这里只列了部分可能存在的约束，以方便解释什么是高层时间约束。根据定义，高层时间约束是形如图2.3中的约束，它必然包含两个变量，两个变量分别属于不同的UML顺序图。这两个变量的差运算描述的含义是：减数所在的UML顺序图发生在被减数所在的UML顺序图之后，只有这样，减法才会有意义。比如0 <= b1 - a2 <= 2，表示B2发生在B1之后，而且B2中的事件b1发生的时间和B1中事件a2发生的时间差大于0个单位时间，小于2个单位时间。

从开始节点出发，沿着箭头所指的方向执行，经过有限个若干节点直到到达结束节点，所构成的一条路径就是UML交互概观图中的潜在路径。这条潜在路径不考虑UML顺序图中的基本时间约束与UML交互概观图中的高层时间约束。

在UML交互概观图2.2中，对于一条执行顺序B1到B2，采用同步连接语义，即B2要执行，必须首先确保B1已经执行完毕。图2.2中，设定开始节点为B0，结束节点B4，那么明显的一条潜在路径是B0🡪B1🡪B3🡪B4。这条图存在环，其中一条环是 B1🡪B3🡪B1，图2.2是个简单的图，但是实际生活中的图往往都很复杂，包含十几个甚至几十个UML顺序列图，并且可能会包含许多个环。比如B0🡪B1🡪B3🡪B2🡪B1🡪B3🡪B4，这条路径是否可行，仅仅从这条路径是无法判断的，还需要分析UML交互概观图的高层时间约束以及路径中每个UML顺序图的基本时间约束。

## 2.3 UML交互模型的同步和异步路径

## 2.4 使用UMLet绘制并解析UML交互模型

### 2.4.1 使用UMLet绘制UML交互模型的基本格式

### 2.4.2 解析UML交互模型

## 2.5 本章小结

这一章介绍UML顺序图 [[3](#_ENREF_3)] 和UML交互概观图 [[4](#_ENREF_4)]，并给出了它们的定义。UML交互概观图由UML顺序图构成，表示UML顺序图间的组合和执行顺序。UML顺序图表示不同实例之间的信息交互。无论是UML顺序图还是UML交互概观图，都是与时间线性相关的，时间约束的存在要求验证UML顺序图和UML交互概观图的时间性质可满足性。

# 第三章 基于SAT-LP-IIS的可达性验证算法

本章是基于SAT-LP-IIS的关键算法的设计和实现，基于SAT [[13](#_ENREF_13)] 的模块是使用SAT4J [[14](#_ENREF_14)]工具设计和实现的，它通过将邻接表编码为SAT4J工具支持的公式格式，调用SAT4J工具中的SAT解析器去找到这个公式的的满足解，这个满足解就是潜在的路径。找到潜在路径后，使用IBM的cplex工具中的LP模块 [[10](#_ENREF_10)] 来分析潜在路径的约束是否有解。若无解，cplex工具中的IIS模块 [[11](#_ENREF_11)] 则可以找到对应不可约简的不可解集，避免再次遇到时重复计算降低执行效率。

## 3.1 基于SAT路径遍历算法设计和实现

### 3.1.1 SAT原理

布尔可满足性问题（Boolean Satisfiability Problem, SAT）[[13](#_ENREF_13)] 是一个非常强大且有用的问题，不仅仅局限于本文的潜在路径寻找方面，还可以用来做各种各样的工作。布尔可满足性问题是判断一个布尔公式是否有满足解的问题，序列中每个变量仅有2个值，true或者false。举个例子，有两个布尔变量a和b，构成一个公式a and b，那么要想这个式子为真，就必须让a和b同时为真，这是一个有布尔可满足性解的式子。布尔可满足性问题解的情况有三种：有且仅有一个解，比如前文介绍的a and b；无解，比如b and ¬b是一个没有解的布尔逻辑表达式，有解且解的个数大于2，比如a or b是一个拥有多解的表达式。

首先介绍一些关于布尔可满足性问题的基本概念，本文用∧， ∨， ¬分别表示布尔操作逻辑与、或、非；用→来表示充分条件。

1. 布尔序列：由若干布尔变量，逻辑操作与、或、非和→构成的序列， 布尔序列由分句和逻辑与操作构成。
2. 不可行解：对于同一个变量a，a和¬a不能用∧单独连接，这是矛盾的，这种情况不可能存在解。
3. 分句：在布尔序列中，不同的分句用∧连接，分句当中不允许∧的存在。哪怕分句只有一个变量，那么也是一个分句。比如（a∨¬e∨d）∧ (¬a∨e∨c∨d) ∧ (¬c∨¬d)就含有三个分句，用两个∧隔开。
4. cnf [[15](#_ENREF_15)] ：clausal normal form 或者conjunctive normal form，中文名为合取范式 ，是一种布尔逻辑标准格式，表示第3点介绍的分句之间连接的格式，即表示用and操作连接的分句。
5. 本文使用SAT4J 工具处理布尔可满足性问题，如果从外部文件读取，那么文件的格式如图4.1所示：

图4.1 SAT4J工具的外部文件输入格式

图4.1中，第一行c表示该行为注释，SAT解析器不会处理这行的内容。第二行p cnf行告诉SAT这是一个合取范式格式的sat文件，5表示最多有5个布尔变量，3表示最多有3个分句。接下来三行中，绝对值大于零的数字表示一个变量，正值表示它自身，负值表示对其本身施加一个 ¬ 操作，0表示一行的结束，不同行之间用 ∧ 连接。所以，以上三行如果转化为布尔序列来表示的话，就是（a∨¬e∨d）∧ (¬a∨e∨c∨d) ∧ (¬c∨¬d)。图4.1中的1,2,3,4,5分别代指上述公式中的a、b、c、d、e。

1. → 逻辑操作：是一个二元操作符，表示左边的布尔序列如果是true，那么右边的布尔序列也一定是true。因为SAT不包含→操作，需要把→转化为与、或、非逻辑操作。假如有一个布尔序列, (a∧b→¬c) ∧ (¬c→a∧b) 。转化→为逻辑操作与、或、非的公式为公式2:

**(a∧b→¬c****)∧(¬c→a∧b)≡(¬a∨¬b∨¬c)∧(a∨c)∧(b∨c) （公式2）**

### 3.1.2 UML交互概观图的编码

为了解决这个寻找路径的问题，本文首先给出一些定义：

1. 编码：将SAX工具分析得到的图模型转为工具SAT4J支持的公式格式的过程称为编码。
2. 解码：SAT4J分析布尔序列的可行性，如果可行，将得到的满足解转化为相应的路径称为解码。
3. E表示节点集合。
4. Ei表示初始节点。
5. Et表示目标节点。
6. Bound：表示路径的最大限制长度。
7. INIT序列：初始节点序列∧其他节点的逻辑非值，是一个合取范式布尔序列。
8. NEXT序列：∧（当前节点（属于E）→∨下一个节点）。这是由多个分句构成的合取范式布尔序列，对于每一个节点都可以生成一个分句，若干分句由∧连接。当前节点和下一个节点必须构成一个UML交互概观图中的边。
9. EXCLUDE序列：∧（当前节点→∧其他所有节点的逻辑非），该合取范式布尔序列通常含有多个分句，它表示当前节点如果是true，那么其他节点一定是false，用来确保当前位置节点的唯一性。
10. TARGET序列：∨目标节点，是一个合取范式布尔序列。Target含有多少个目标节点由bound决定。

本文给一个简单的例子来描述如何将获得的图模型编码为INIT，NEXT，EXCLUDE，TARGET布尔序列，假设有一个简单的UML交互概观图 （见图4.2）：

图4.2 简单的UML交互概观图

图4.2中的UML交互概观图由一个开始节点v0（黑色的实心圆）和一个结束节点v5（另一个带外环的实心圆），四个UML顺序图v1、v2、v3、v4，以及6条边组成，这是一个带环的图

那么为了找到一条从v0到v5的潜在路径，利用布尔可满足性问题的方式，本文需要构造INIT，NEXT，EXCLUDE，TARGET四个合取范式布尔序列 ，为了方便描述设定bound为1，结果如表4.1所示：

表4.1 图模型编码为bound为1布尔序列

首先，初始节点是固定的，一定是表4.1中所示的V0。因为初始节点一定是路径位置为0的节点，且要保证本文的路径是从V0开始，不能从其他的节点开始，因此取其他值都为逻辑非。然后，next节点描述了当前节点之后的下一个节点，next节点序列包含若干分句，以表4.1中的NEXT序列为例，只存在从位置0节点到位置1节点的一个相邻链路对，因此是V0 0 → V1 1。如果将bound修改为2，那么只需要将当前节点的上标和下一个节点的上标加1，并 ∧ 到原序列即可。比如 (V0 0 → V1 1) ∧ (V0 1 → V1 2 ∨ V1 5) ∧ (V0 2 → V1 3) ∧ (V0 3 → V1 4)∧(V0 4 → V1 1) ∧(V1 0 → V2 1) ∧ (V1 1 → V2 2 ∨ V2 5) ∧ (V1 2 → V2 3) ∧ (V1 3 → V2 4)∧(V1 4 → V2 1) ∧ (V1 5 → V2 5)。EXCLUDE和TARGET也是如此。EXCLUDE表示，如果当前节点是V0，那么下一个节点就不能是相同位置的其他节点，由于在INIT序列中，位置为0的初始节点被确定为V0，所以只需要考虑位置从1开始到位置是bound值为止的序列。TARGET序列表示本文的目标是V5，因为限定长度为bound，也就是长度不能超过bound，因此位置0，位置1… 一直到bound的位置都是可以是V5的，这里面必定有一个值是真的，以保证本文的序列是以V5结束的，所以用∨连接V5。

由此，如果需要找到路径，只需要构造表4.1的序列，并将这些序列用∧连接，使用SAT4J工具即可得到潜在的路径，本文将这个图模型转化为SAT4J工具支持的格式的过程称为编码。由此，将编码的过程汇总为一个公式（公式3）：

公式3中，BGk表示潜在路径长度限定为K的合取范式格式的布尔序列。INIT0表示初始节点序列，初始节点序列只能在位置0。∧ NEXTi表示从位置0开始一直到位置k-1的NEXT序列，这之间用∧操作连接。∧ EXCLUDEj表示从位置1一直到位置k的EXCLUDE序列的逻辑与操作连接。∨ TARGETl表示从位置0到位置k的TARGET序列的逻辑或操作连接。

通过调用SAT4J工具，本文能够找到一个满足解，这个满足解是一个真值分配表，也就是得到的结果中为正数的那些，将这些正数按照各自的上标排序，即可得到本文想要的路径。当然这些路径可能存在多条，比如图4.1中，从V0到V5，存在很多条路径，可以是V0→V1→V5，也可以是V0→V1→V2→V3→V4→V1→V5，这个过程就是译码。

### 3.1.3 基于SAT4J的路径遍历算法实现

SAT模块是比较重要的模块，因为使用了SAT4J这款工具，因此如何构造布尔序列就成为了关键的部分，对SAT4J的使用可以通过查阅SAT4J提供的参考接口文档 [[16](#_ENREF_16)]。图4.3是基于SAT的模块类图

图4.3 基于SAT的模块类图

类SatSolver使用了SAT4J工具的接口，getSolver() 方法实现了对UML顺序图和UML交互概观图的xml模型的编码，即编码为INIT，NEXT，EXCLUDE，NEXT布尔序列，并将它们添加到对应的SAT分析器的对象实例中，这个分析器就是ISolver类，最后返回这个分析器ISolver的实例。

|  |
| --- |
|  |

图4.4 SAT4J的部分实现代码

图4.4描述了如何使用SAT4J工具的部分代码，编号1，2的行设置了两个常量，MAXVAR表示布尔序列中变量的最大个数，NBCLAUSES表示预期可能包含的最大分句个数，并在编号第4，5的行设置为ISolver对象的属性。第三行创建一个ISolver对象，这个对象就是SAT4J工具的SAT分析器了，编号第6的行for循环依次添加分句，分句内的变量用逻辑或操作符连接，每向ISolver中添加一个分句，这个分句就用逻辑与操作与之前的分句连接。注意，每个分句之中不能包含0。编号第8的行判断是否有解，如果有解，提取解到model数组中去，model数组是一个包含正数和负数的整数型数组，其中所有的正数构成了一个解。

本文通过SAX获得的只是一张用邻接表表示的图，需要将其转化为SAT4J工具支持的公式格式。首先要考虑到，上一节中提到的转换公式3是一个包含两个符号的节点构成的序列，比如INIT序列：V0 0 ∧ ¬V0 1 ∧ ¬V0 2 ∧ ¬V0 3 ∧ ¬V0 4 ∧ ¬V0 5，这个序列中的节点是用上标表示在潜在路径的位置，同时还有下标表示某个元素，而SAT4J只能接受非零的整数。

本文考虑使用n位整数的方式来编码邻接表表示的图模型为SAT4J工具支持的公式格式。以图4.1给的简单例子为样本，来描述如何转换为布尔序列。首先，需要给定一个bound，这个bound约定了路径的长度，它决定了INIT，NEXT等四组序列的上标。然后需要获得元素的个数，以便确定这串数字高位的位数。那么这个n位整数的n就等于bound的位数+元素个数的位数。在这个例子中，给定bound为1，元素个数为6。那么本文用一个两位数来表示每个元素Vi j。之所以选择两位数，是因为bound占1位，元素个数也占1位，二者相加就是两位。用高位表示某个元素，且高位从1开始，因为SAT不能接受正数0。低位表示位置，低位从0开始计数，。那么INIT序列就可以表示为如图4.5所示的情况：

图4.5 INIT序列编码的部分代码

在图4.5中，代码的第1行获取了用邻接表表示的图，第2,3行分别获取了元素个数的位数和bound的位数，这里为了方便描述都设置为1。第4行开始遍历这个邻接表，对每个元素，获取他们的位置并加上后置位bound个0构成他们在SAT布尔序列的整数（图4.5中编号b-d）。另外，对于每个元素，要判断他们是不是初始节点，如果是初始节点，就是正整数，如果不是初始节点，就修改为负数(图4.5中编号e-i)。每个节点的位置都单独构成一个分句，每个分句只含有1个整数，因此每个for循环都要调用一次addClause()方法。

其他的分句像NEXT，EXCLUDE和TARGET也是用这样的思路去构成SAT4J工具的输入格式，并依次将这些分句添加到对应的solver中去即可。

通过编码为SAT4J接受的序列，当找到潜在的路径的时候，剩下的就是要译码了。译码的过程，就是把布尔序列求得的解转化为路径。需要找到所有的正整数，取一定的高位求得某个元素，剩下的低位作为某个路径的具体位置，排好即可。具体的代码见图4.6：

.6 满足解译码为路径部分代码

在存在解的情况下，图4.6的内容可以分为两个部分，第1部分，获得solver中的解，并将其中的正整数取出。第2部分，分析正整数的结构，取低位排序，取高位代表元素位置，根据位置取得对应的元素，并将元素添加到pathlist中去，最后返回pathlist。如果解不存在，图4.6的第一行中的条件判断语句为false，直接返回null。

## 3.2 基于LP的时间性质分析算法设计和实现

### 3.2.1 基于LP的时间性质分析算法设计

上一节中，讨论如何利用SAT工具 [[9](#_ENREF_9)]，并以此实现了基于SAT工具的模块，去找到潜在的路径。但是仅仅找到路径是不够的，因为路径中约束条件的存在，要求必须去验证约束条件是否满足。而约束条件全部是线性约束，所以本文可以借助线性规划问题（Linear Programming, LP）求解算法来分析约束的可满足性，使用IBM的cplex工具的LP模块 [[10](#_ENREF_10)] 来实现约束条件的验证。

基本的想法是，先使用SAT工具找到潜在的路径，然后取得路径中的约束条件，调用LP模块去判断路径的可行性。如果可行，那么算法就结束执行，输出相应的结果。如果不可行，那么这条路径就是不可行路径，本文继续去找下一条路径。当遍历完所有的路径的时候，本文如果还没有找到满足约束条件的路径，那么说明路径是不存在的。

潜在路径中的时间约束C由路径中每个UML顺序图中的约束集C1，UML交互概观图中的相关约束集C2，并经过改名操作(R)之后构成。

C1：潜在路径中，所有UML顺序图的约束集之和。

C2：对于一个UML交互概观图中的约束，它是潜在路径的相关约束当且仅当约束中的两个变量，分别属于潜在路径中的两个UML顺序图。相关约束集由这些约束构成。

R：改名操作，潜在路径中存在重复的UML顺序图，由于发生的时间不同，需要更改重复发生的UML顺序图中约束变量的名称，同时更改UML交互概观图中相关高层时间约束的名称。改名的原则是，对于UML顺序图中的约束集C1，第一次出现在潜在路径中的UML顺序图中的约束变量名不需要更改，之后若干次出现的均需要更改，且名称不能和任意其他重复UML顺序图中的约束名称相同。对于UML交互概观图中的相关约束C2，比如5 ≤ e15 - f14 ≤ 10，在潜在路径中，含有e15的UML顺序图发生在含有f14的UML顺序图之前，且变量名字要分别和改名之后的UML顺序图中的名字对应。这两个UML顺序图构成一个序列对，对于潜在路径中的所有序列对，都需要该操作，比如路径V0→V1→V2→V3→V1→V2→V5，有两对V1→V2，如果这两对的相关约束若存在，则名字都需要更改。

C：经过改名操作后的C1和C2之和。

由于合取范式布尔序列可能会存在多个解，即潜在路径可能有多条，而如果每条可能的路径都要遍历一遍，这是十分复杂的。因为不同的路径之间可能有相同的路径片段，而恰恰这段路径片段正好是不满足约束条件的部分，那么在下一条路径进行分析的时候，就可以直接跳过这条路径。这样就可以避免重复的调用LP进行验证。这条路径本文将它叫做不可行路径(Infeasible path)。

公式4是不可行路径公式，p表示某条路径，如果以图4.1为例，假设其中V0→V1→V2→V3→V4→V1路径是不可行的，那么*Infeasiblek*(p)：= ∧(vi 0 ∧ vi+1 1 ∧ vi+2 2 ∧ vi+3 3 ∧ vi+4 4 ∧ vi+5 1 → ¬vi+6 5)其中，0≤i≤k-len+1(len为路径的长度)。

公式5表示该布尔序列要并上不可行路径译码后的路径。而不可行路径在本文在上一次求解可能的路径的时候已经获得了，需要注意的是，这个不可行路径并不仅仅只是一条从位置0开始的路径。这条路径可以是BG的一条子路径，也就是说，只要BG里面有一条路径有一段恰为该不可行路径，那么这条路径就是不可行的。

可以结合SAT和LP来写一个伪代码，伪代码描述了基本的算法实现过程。由于潜在的路径是有限的，因此，这份伪代码是一定可以终结的。具体的伪代码图4.7 SAT-LP算法伪代码

图4.7中，第2行和第3行，将UML顺序图图和UML交互概观图编码为SAT格式的布尔序列BGk，设置Infeasible为空。第4行，进入一个while循环，循环终止的条件是return语句。在while循环中，第5，6，7行判断布尔序列BGk是否有解，如果没有解，那么说明没有潜在路径，返回unsat。如果有解，即第9，10，11,行，将潜在的路径找出，并找到路径中的约束条件。在第12行，调用LP判断约束条件是否可行，如果可行，那么返回sat。如果LP判断不可行，即第14-16行，将不可行的路径编码为infeasible，用逻辑与操作连接到BGk中进行下一个循环。直到找到一条路径，或者找不到合适的路径。

### 3.2.2 基于LP的时间性质分析算法实现

本文使用IBM公司提供的cplex工具，该工具既包含LP，也包含对应的IIS，所有的工作都可以在一个类里面完成。本小节先介绍LP的使用方式。

首先是对LP的使用。LP是一种线性约束分析技术，它能够接受含参的不等式组，等式组，不等式和等式混合组。本文处理的都是含参的不等式组，LP能够分析该含参的不等式组是否有解，如果有解，那么可行，如果没解，说明不等式组之间由矛盾，即路径不可行。图4.8是LP模块添加相应的约束的代码。

在进行图4.8的处理之前，需要对约束进行处理，包括定义六进行的改名操作，提取不等式组中每个不等式的变量、参数和不等式左端的下限和右端的上限。

在图4.8中，将所有约束中的变量提取出来放在了varList里面。对每个不等式约束，同时分别将左侧变量提取放到了varArrLeft数组里面，右侧变量提取放到了varArrRight数组里面；并分别将对应的左端的下限放到了lbArr数组里，右端的上限放到了ubArr数组里，。一个更加合理的做法是为这些变量构建一个新类，每个类的对象都对应一个约束，通过持有若干成员对象来保存约束的格式。这里特别要注意行1-7的操作，这里的作用是设置所有的变量值均大于0，小于MAXVALUE。这些操作必须在第7行一次完成，这是为了将变量放在同一LP model中，不然就会导致同名变量被识别为不同的变量，后面的工作肯定也会是错误的了。第8行开始将每个不等式的结构添加到model中，model是LP分析器的一个对象。addLe() 方法是小于等于操作，它含有三个参数，图4.8中的含义是第一个参数减去第二个参数的值小于等于第三个参数。addGe() 方法是大于等于操作，图4.8中三个参数的含义是第一个参数减去第二个参数的值大于等于第三个参数。这个方法会返回一个LP的model，可以利用这个model来判断最后的结果是否可行。

由于本文会在4.3节中使用IIS来加速程序的执行效率，这里仅仅实现了LP对线性约束的处理，并没有将LP判断为不可行的路径编码为布尔序列并添加到BGk当中，就像公式中那样。

## 3.3 基于IIS的优化算法设计和实现

### 3.3.1 基于IIS的优化算法的设计

前面的部分结合使用SAT和LP，通过列举所有的潜在路径，并检查路径可行性，来找到可行的路径。但是，如果bound非常大，那么路径会很长，这样不仅仅在找每条路径的时候会效率低下，在验证每条路径可行性的时候也会有很多约束条件，会消耗比较多的处理时间。更何况一旦LP在验证潜在路径时间满足性时，发现路径不满足时间约束的时候，会将整个路径编码为布尔序列，还要用逻辑与操作连接到之前使用SAT获得的布尔序列中，使这个布尔序列变得更加复杂。因此，这种情况会消耗大量的计算时间，时间复杂度会很高。但是，如果我们使用IIS获得最小的不可行路径，将它编码为布尔序列，并用逻辑与操作连接到之前用SAT获得的布尔序列中，那么，就不会产生一个包含该不可行路径的潜在路径，更不会再去对它进行线性约束的满足性验证，这样，就能明显的提升程序的执行效率。

一个不可约简的不可解集（IIS）是一个关于不可行约束的最小集。一般来讲，一个不可约简的不可解集是约束集C的一个子集C＇，C＇⊆C， C和C＇是不可行的，但任何C＇的真子集都是可行的。

从上述描述中可以知道，一个不可约简的不可解集首先是线性约束的集合，这个集合一定是无解的，也就是说不可行。然后任意去掉其中的一条或若干条约束，那么，剩下的约束一定会有解。因此，如果给一条不可行的路径p，这样就能通过分析约束集C，产生一个C的子集C＇，这个子集C＇是满足不可约不可行的。根据前面的描述，知道这个不可约不可行约束集是从前面的每个消息顺序子图和消息顺序图中获得的，接下来只要挨个对比前面的消息顺序子图和消息顺序图中的约束，那么就能知道不可行路径到底是哪些。

举个例子，以图3.1为例，假设有一条路径V0→V1→V2→V3→V4→V1→V5，其中V1的内部约束为0≤x1-x2≤10，5≤x2-x3≤10，21≤x1-x3≤30,这三个式子联立是不成立的，也就是说，V1本身就是一个不可约简的不可解集。在使用IIS求解不可约简的不可解集的时候，能够得到的不可约不可行约束条件就是这三个不等式，分别用这三个约束去找相对应的UML顺序图，那么就能判定V1就是可以确定的不可约简的不可解集。

很明显，如果一条路径p包含一个不可行的路径片段，那么这个p就没有验证的必要。在4.2节中放弃将LP工具分析得到的不可行路径添加到BGk中的原因就是因为路径长度会大于IIS分析得到的路径长度。因此更简单高效的做法是，当找到了一条不可约不可行路径，那么只需要把这条路径编码到SAT，并用交操作和BGk连接，有点类似于前面的公式5，不过这里的路径更短了，因此效率更高。这样，就能避免产生包含已经产生过的不可约不可行的路径。

因此，本文添加IIS到SAP和LP中去，构成了一个SAT-LP-IIS联合解决路径问题的新的工具。由此得公式6和公式7的两个公式 :

**IIS ≡ ∧ IISk（p＇） （公式6）**

**BGk ≡ BGk ∧ IIS （公式7）**

在公式6和公式7中，p＇表示不可约不可行路径，将IIS编码为布尔序列，用 ∧ 连接到BGk即可得到新的布尔序列，再次调用SAT即可找到一条不含IIS的路径，然后验证该路径的LP满足性。

当找到这条路径p＇时，如何编码成了最关键的事情。在LP中，这一整条路径都不可行，因此不需要考虑路径位置编码的事情。其实很简单，举个例子。假设p = V0→V1→V2→V3→V4→V1→V5是验证的LP不可行路径，p＇= V3→V4→V1是验证的IIS不可行路径，那么：

**IISk(p＇)≡ ∧(vi 3 ∧ vi+1 4 ∧ vi+2 1 → ¬vi+3 5) （公式8）**

**(0≤ i ≤k-len+1，len是路径p＇的长度)**

基于前面的分析，整合SAT-LP-IIS，取得一个验证UML顺序图和UML交互概观图时间性质满足性的算法，优化验证性能。这个算法见图4.9：

图4.9 SAT-LP-IIS整合算法伪代码

在图4.9的算法中，第2行和第3行，将MSC图编码为需要的SAT格式的布尔序列BGk，设置IIS为空。第4行，进入一个while循环，循环终止的条件是return语句。在while循环中，第5，6，7行判断布尔序列BGk是否有解，如果没有解，那么说明没有潜在路径，返回unsat。如果有解，即第9，10，11,行，将潜在的路径找出，并找到路径中的约束条件。在第12行，调用LP判断约束条件是否可行，如果可行，那么返回sat。第14-16行，是不可行的情况，找到不可约不可行路径p＇，将其编码为IIS，用逻辑与操作连接到BGk中进行下一个循环，直到找到一条路径，或者找不到合适的路径。每次遍历SAT找到的路径，都不会包含前面几次所求得的IIS，直到找到一条满足要求的路径，返回这条路径。

### 3.3.2 基于SAT-LP- IIS的优化算法的实现

基于IIS的模块的实现需要结合LP，因为IIS和LP同属于IBM的cplex的两个组件，IIS取得的不可约简的不可解集是从LP建立的不等式组model中获得的。只有当时间约束组无解的时候，不可约简的不可解集才存在。结合使用LP和IIS的代码示例如图4.10：

**图4.10 IIS和LP的使用**

图4.10在第一行构造了一个简单的不可行的约束集合，利用这个约束集合创建了一个LP的解析类LpModel的实例model，调用这个model的getIIS() 方法，可以获得一个IloConstraint的数组，然后这个数组的每个元素都是一个约束，数组中所有的约束构成了不可约不可行解。对这些约束的处理，我们只需要提取出每个约束的参数，与Element中的相应元素作对比，即可获得最终的不可约不可行路径。图4.10的例子中，得到的不可约简的不可行集为：3.0 <= (1.0\*x1 - 1.0\*x2) <= infinity和3.0 <= (-1.0\*x1 + 1.0\*x2) <= infinity。

本文将SAT-LP-IIS整合的算法的部分代码实现整理为图4.11。

图4.11 实现SAT，LP和IIS的整合部分代码表

图4.11的代码中，编号1的行创建了一个SAT的solver对象，getSatSolver()方法的五个参数为邻接表表示的图elementList，初始节点start，目标节点target，路径约束bound，经过的节点链表gothroughList，该方法将图模型转化为合取范式布尔序列添加到SAT4J的分析器中，并返回该分析器。在编号2的行中，while(true)语句使得该while循环内的语句循环执行直到遇到return语句。编号3的行判断是否存在潜在路径，如果不存在，直接返回路径不存在。如果存在，找到潜在路径。在找到潜在路径后，编号4分析并找到相关高层时间约束。编号5,6的行对约束进行了名字的更改，获得了更改名字后的潜在路径时间约束，潜在路径时间约束包括路径中每个UML顺序图中的约束和UML交互概观图中的相关约束。编号7的行创建了LP的一个model，分析约束的可行性，如果可行，则返回相关信息。如果不可行，即编号8到编号9所示的内容，使用IIS找到不可约简的不可解集，根据不可约简的不可解集调用getIISElement()方法，该方法会返回一个不可行的最小路径，然后调用addIISClause()方法添加到SAT分析器中，并回到while处从循环开头继续执行。

## 3.4 实验结果

## 3.5 本章小结

这一章主要介绍了使用到的3种技术SAT、LP、IIS，详细介绍了各种技术在本论文论证的工具中的算法和实现。4.1节首先介绍了如何将SAX获得的邻接表转化为SAT4J所需要的整数序列，根据该整数序列，SAT4J能够快速的找到潜在的路径。4.2节介绍了基于LP的时间性质分析算法设计和实现，cplex的LP模块是强大的线性约束分析工具，对于UML顺序图和UML交互概观图中的约束集合能快速的判断可行性。4.3节介绍了基于IIS的优化算法设计和实现，LP分析的约束集合如果不可行，那么就可以使用IIS方法获得一个不可约简的不可解集，根据这个不可约简的不可解集找到对应的最小路径并编码为布尔序列添加到对应的SAT4J分析器中，可以提高程序的执行效率。

# 第四章 对IIS约束的多目标优化

## 4.1 时间不敏感系统-扩大范围算法

## 4.2 时间关键系统-缩小范围算法

## 4.3 时间敏感系统-小幅调整算法

## 4.4 对软约束、硬约束的处理

## 4.4 实验结果

## 4.5 本章小结

# 第五章 基于Cplex和遗传算法的目标求解

## 5.1 基于LP求解优化目标

## 5.2 基于遗传算法求解优化目标

### 5.2.1 初始种群的产生

### 5.2.2 杂交和变异算子的设计

### 5.2.3 遗传算法总流程

## 5.3 实验结果

## 5.4 本章小结

# 第六章 总结与展望

## 6.1 论文总结

## 6.2 未来研究和展望

# 参考文献