嘉应学院

本科毕业论文(设计)

(2013 届)

题　　目：基于 Gecode 的析取时态推理系统研究与设计

姓　　名：林 晓 骏

学　　号：２０９１１２３１１６

学　　院：计 算 机 学 院

专　　业：计算机科学与技术（网络方向）

指导老师：刘 越 畅

嘉 应 学 院

中文摘要

对于一个智能规划系统而言，时间元素扮演着重要的角色。因而，时态规划成为规划领域重要的研究课题，并应用在越来越多的工程项目中。时态规划器的实现往往需要集成一个独立的时态推理引擎，而引擎的实现依赖于它所使用的时态模型以及它要完成的时态推理任务。析取时态问题是时态规划中一种具有强表达力的时态模型，可以表示大多数时态问题从而承担更多时态推理方面的工作。析取时态问题的表示中隐含了大量可用于改善求解速度的启发式信息，可以从中挖掘出多种启发式技术。为了更好地验证和组合各种启发式技术，以获得更高求解效率，本文探讨使用了的基于约束可满足问题模型的求解库 Gecode，设计实现了析取时态求解系统 TRSE。

关键字：时态规划，时态推理，析取时态问题，启发式技术，约束可满足问题

Abstract

Temporal information plays an important role for an intelligent planning system. Hence, temporal planning has become an important research project in the planning field and applied on more and more engineering projects. The realization of temporal planner always need to integrate an independent temporal reasoning engine, whose realization relies on the temporal model it uses and the temporal reasoning assignment it has to finish. Disjunctive Temporal Problem (DTP) is a strong-expressive temporal model in temporal planning, which is able to express most temporal problems so that it can take more responsibility for the temporal reasoning work. There is much implicit information used for improving the solutions in the expression of DTP, from which we can exploit many heuristics. In order to test and combine different kinds of heuristics to fetch higher efficiency, this paper discusses and uses a solving library based on Constraint Satisfied Problem (CSP) called Gecode to design a solving system called TRSE.

Key words: Temporal Planning, Temporal Reasoning, Disjunctive Temporal Problem (DTP), Heuristics, Constraint Satisfaction Problem (CSP)

目 录

第1章 引 言

1.1 研究背景和意义

你一定遇到过这样的一个情景：当手头有很多的工作时，我们不得不花点时间对其做个详细的计划。例如，早上 9 点钟前到达办公室，9 点 15 分左右向上司汇报上周的工作进度，9 点半到 10 点给客户发传真，12 点前完成项目计划书等等。人们为了达到某个目的(通常是较为复杂的问题)，就会去思考应该“做什么”、“怎么做”、“能完成吗”等问题。对于十分复杂的带有不确定性的又对时间要求苛刻的问题，更加需要有一个及时有效的规划。例如在实际应用中，研究人员会关注以下问题：供电故障恢复问题 PSR（故障发现)、管道输送问题 pipes world（路由选择)、钢厂调度问题 SIDMAR（生产规划）[2]等等。人们对规划问题的研究已经有六十多年，目前也提出了各种各样的解决方案，但是规划问题是一个非常难解决的问题，现在能够解决的规划问题仍然十分有限。

规划的具体定义是什么？McDermott 和 James Hendeler 认为“规划就是设计某个（组）实体（Entity）的动作序列，其结果被称之为规划解（Plan)”。Deepak Kumar 更加形象地说“Planning = How do I get from here to there?”。在人工智能领域，直观地说，一个规划解实质上就是一个动作序列，此序列能够实现某一目标，智能规划就是设计这个动作序列的过程，也就是说它主要解决怎么做，而不解决为什么[3]。

正如一开始的例子所提及的那样，我们的活动规划经常是放在一个时间段中进行考虑的，也就是说，任务是否能完成依赖于时间的多少以及我们对时间的规划，时间因素对于规划的重要性显而易见。基于时间的智能规划我们称之为时态规划。时态规划在现实应用领域已经发挥了实际作用，时态规划系统已经在欧美国家的航天项目（如哈勃太空望远镜维修项目、深空一号探测器的运作等）中占有一席之地。

一般看来，规划需要解决两个问题：一个是“干什么”，另一个是“怎么干”或者“何时干”，后者称之为调度问题，又称为时态推理。从这个角度看，时态规划既要找到一个可以实现目标的动作结构，又要保证这种动作结构在时态上是一致的（即可以执行的）。因此，时态规划可以看作是规划（狭义）和调度的统一，或者说是两者的集成问题。在规划（狭义）和调度的众多集成模型中，约束可满足问题（CSP）模型是求解规划和调度问题的良好框架[1]。本文基于约束满足模型侧重讨论调度模块，即时态推理。

时态推理的基本任务是确定一个约束集合的一致性，但在此之前，我们需要先尽可能清晰地描述要解决的时态问题，于是引入另一个时态模型——具有强表达力的析取时态问题（DTP）。析取时态问题也是一个约束可满足问题，前者相比后者，可以允许子句中存在不同变量对之间的约束的析取形式（详见第 2 章时态规划相关定义）。目前 DTP 求解算法主要分为两类：一是将其作为一个元-CSP，使用标准的约束可满足算法来求解；二是将其编码为一个命题可满足问题，使用已有的可满足算法来求解。两种方法下的 DTP 求解算法在求解速度上呈现出此消彼长的竞争态势。目前，基于命题可满足算法的 DTP 求解器—TSAT++比其他已知的基于约束可满足算法的 DTP 求解器具有更高的求解效率（更少的一致性检查次数和访问节点数以及更快的求解速度）。

要想进一步提高基于约束可满足算法的 DTP 求解算法的效率，关键在于挖掘利用更多的启发式技术，也就是说，要找到能够充分优化求解过程的方法。采用良好的启发式函数及其改良组合可以获得比现有同类技术减少一个数量级以上的访问节点数和耗费的 CPU 时间，且这种优势随着问题规模和难度的增大变得更加明显。因此，如何挖掘和组合各种启发式技术来获取更高的求解效率，是当前析取时态问题求解的一个研究方向。

本文希望探讨实现基于约束可满足模型的析取时态推理系统，以提供一个良好的测试平台来更好地验证和组合各种启发式技术，追求更高的求解效率。

1.2 主要工作

本文希望探讨实现基于约束可满足模型的析取时态推理系统，工具选择的考虑和工作任务的安排如下：

首先编写建模模块，将通过脚本随机生成的 DTP 转换为内部数据结构加以缓存。在计算机科学领域，图形结构可以描述和解决许多问题，因此这里考虑使用图来表示 DTP。由于 DTP 对应的析取时态网络（DTN）通常是一个加权多重图（允许两结点间的边数多于一条，也允许顶点通过同一条边和自己关联，后者在本文不予考虑），因此选用的图形建模工具必须支持有权图和多重图，本文选用开源工具 JGraphT 来编写 DTN 的存储结构。另外，为了更好的表示 DTN 结构，作者还选用了 JUNG 开源框架来编写系统的视图模块；最后编写引擎模块，由于 DTP 求解系统是基于 CSP 模型的，因而选用的工具软件需要支持约束系统开发，本文将使用 Gecode 求解库及其接口工具 Gecode/J 所提供的建模和求解功能来实现求解系统的引擎，并编写启发式算法进行求解测试。

为方便表达，本文将设计的推理系统称为 TRSE（时态推理求解环境）。

1.3 文章结构

第 2 章：对规划领域相关理论和定义进行阐释，主要包括约束可满足问题、简单时态问题和析取时态问题，并对时态问题的求解作了相关介绍。

第 3 章：简要地交代了系统开发和运行的环境要求，介绍了系统所使用的软件工具：图形库 JGraphT 和求解工具 Gecode 等等。

第 4 章：描述了求解系统的功能结构，以及主要软件工具的适配。

第 5 章：设计数个有代表性的测试用例，以验证系统的可行性，最后列出实验结果并得出结论。

第 6 章：对本文探讨和设计的求解系统进行总结和展望。

第2章　相关定义和理论

2.1　约束可满足问题

2.1.1　基本定义

定义 2-1.（约束可满足问题 CSP）一个（有限）约束可满足问题是一个三元组 $P=\langle X, D, C\rangle$，其中：$X=\{x_1, x_2, \cdots, x_n\}$是一个变量集合；$D=\{D_1, D_2, \cdots, D_n\}$是对应于 X 的（有限）取值域组成的集合，其中 x_i的取值域为（有限）集合 D_i；$C=\{c_1, c_2, \cdots, c_m\}$是一个约束集合，表示禁止的变量赋值元组，其中 c_i为对应于变量 x_i的约束且 $c_i=\{\langle x_i, d_j\rangle | x_i \in X, d_j \in D_i\}$。若对于任意的 c_i均只涉及两个变量，即$|c_i| \equiv 2$，则称该 CSP 为二元约束可满足问题。

实际遇到的 CSP 多数都是二元的，比如 N 皇后问题，下文将要介绍的析取时态问题多数也是二元的。在约束表示和推理中，图（网络）模型是一个理想的建模工具。二元的 CSP 可以用约束图来表示[6]，网络节点表示约束变量，节点之间的边表示两个变量之间存在的约束。例 2-1 给出一个二元 CSP 实例。

例 2-1. 着色问题：L1, L2, L3, L4 是地图上四个需要着色的区域，每个区域有三种可供选择的颜色{red, green, blue}，要求给四个区域分别涂上颜色，使得相邻区域的颜色各不相同。

定义 2-2.（约束可满足问题的解）一个约束可满足问题 P 的解是一个满足所有约束的集合 $s=\{\langle x_i, d_i\rangle | d_i \in D_i, i=1,2,\cdots,n\}$，也可以写作 $s=\{x_i=d_i | d_i \in D_i, i=1,2,\cdots,n\}$。

定义 2-3.（一致性）一个约束可满足问题 P 是一致的，当且仅当 P 至少存在一个解。

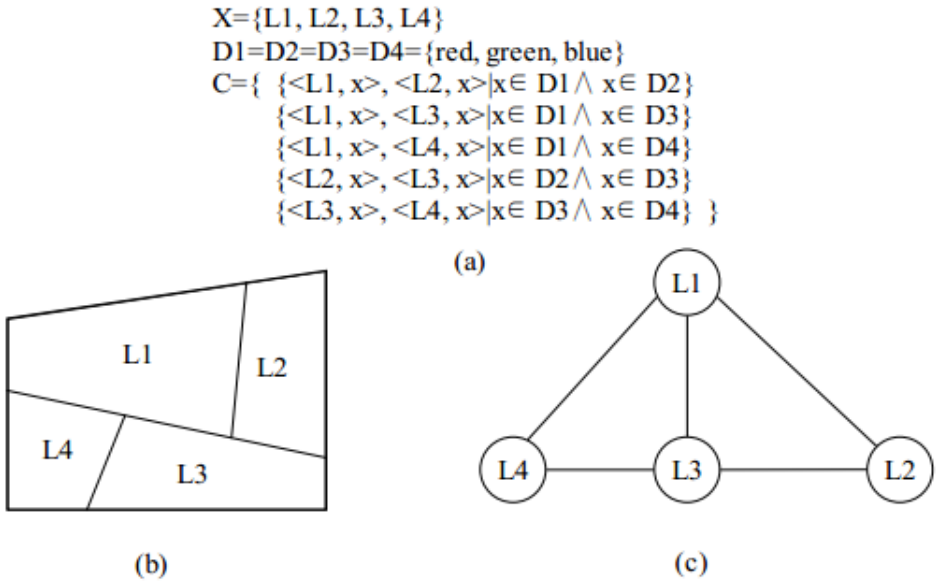


图 2-1 着色问题(b)的 CSP 定义(a)及其约束图(c)

一致性是约束可满足推理的一个重要任务。在一些解决实际问题的系统中，往往不需要计算得出问题的（一个或者所有）解，而仅仅需要计算其一致性即可。虽然 CSP 的定义是简单的，但是其求解却是困难的（计算复杂度为 NPC[1]）[6]，即便只是计算其一致性也是如此。

定义 2-4.（等价性）给定约束可满足问题 P=<X, C>和 P'=<X, C'>,如果 P 和 P'有相同的解集合，则称它们是等价的，记作 P≡P'。

等价性的定义给 CSP 的求解提供了理论支持。在求解的过程中，通过不断地添加约束来减小问题求解的复杂性，使求解目标更为明确。

2.1.2 求解方法

目前 CSP 的求解主要依靠搜索技术和推理技术的结合[1]。CSP 的计算复杂度是 NPC，同其他 NPC 问题一样，搜索是最为基本的求解算法。由于一个 CSP

[1] P(Polynomial)问题是可以在多项式时间内被确定机解决的问题；NP(Non-Deterministic Polynomial)问题是指可以在多项式时间内被非确定机解决的问题；NPC(NP-Complete)问题是指这样一类 NP 问题：所有的 NP 问题都可以用多项式时间划归到他们中的一个[7]。

并不只有定义里给出的约束，可能存在更多的隐含约束，若将其尽可能地显式表示出来，可以加速搜索过程来获取结果（有解或无解）。这意味着可以在搜索过程中增加推理（约束传播）功能来获得启发式信息，用于指导搜索的进行。推理是一个十分难解决的问题，也是本文重点关注的问题。

2.2 简单时态问题

2.2.1 基本定义

定义 2-5.（简单时态问题 STP）一个简单时态问题是一个二元组<X, C>，其中：$X=\{x_1, x_2, \cdots, x_n\}$称为时态变量集合；$C=\{c_1, c_2, \cdots, c_m\}$是简单时态约束集合，且每个 c_i 是形如 $x_i-x_j\leqslant c$（$x_i, x_j\in X$，$c\in R$）的二元约束[2]。

由定义可知，一个 STP 与一个所有变量定义在实数域上的 CSP 是等价的，也就是说，STP 的解、一致性、等价性的概念与 CSP 相似，这里不再赘述。

需要注意的是，若给定 STP 中任意两点 x_i, x_j 之间存在两个以上的简单时态约束，即存在 $x_i-x_j\leqslant c\in C, x_i-x_j\leqslant c'\in C$ 时，可以将其合并为 $x_i-x_j\leqslant \min\{c, c'\}$，这并不改变原 STP 的一致性。

正如 CSP 可以使用约束图来表示那样，STP 也可以使用时态约束网络来表示，下面给出具体定义。

定义 2-6.（简单时态网络 STN）给定一个 STP<X, C>，其简单时态网络是一个有向边标号图<V, E, W>，其中：$V=\{v_i \mid x_i\in X\}$，$E=\{\langle v_i, v_j\rangle \mid x_i-x_j\leqslant c\in C\}$[3]，W 是一个函数 W: E→R，使得$\forall x_i-x_j\leqslant c\in C$ 有 $W(\langle v_i, v_j\rangle)=c$。

定义 2-7.（负环）给定一个 STN<V, E>，它的一条负环是一条简单回路 $v_1, \cdots, v_k, v_1$，且 $W(\langle v_1, v_2\rangle)+W(\langle v_2, v_3\rangle)+\cdots+W(\langle v_k, v_1\rangle)<0$。

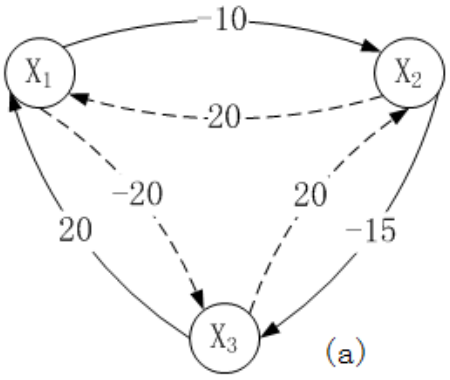
[2] 有些文献将约束定义在诸如 $c'\leqslant x_i-x_j\leqslant c$ $(c'\leqslant c)$的区间上，由于这种形式也可以用 $x_i-x_j\leqslant c$ 和 $x_j-x_i\leqslant -c'$来等价表示，因而本文选用更为简单的定义形式。

[3] 在某些文献中的 STN 边定义可能为 $E=\{\langle v_j, v_i\rangle \mid x_i-x_j\leqslant c\in C\}$，但这并不影响原 STP 的一致性。

定义 2-8.（距离图）给定一个 STP：P=<X, C>，其距离图 DG 是一个|X|×|X|的矩阵，使得 DG[i][j]=c_{ij}。

为了说明定义 2-6~8，下面给出具体例子。

例 2-2. 考虑这样的一个简单时态问题<X, C>:X={x_1, x_2, x_3},C={$x_1-x_2\leqslant-10$, $x_2-x_1\leqslant20$, $x_2-x_3\leqslant-15$, $x_3-x_2\leqslant20$, $x_1-x_3\leqslant-15$, $x_3-x_1\leqslant20$}，其简单时态网络和距离图数组如图 2-2。其中(a)里的虚线是冗余的，即任意两个顶点之间的距离由实线决定。从 DG 数组可以观察到对角线上均为负值，即存在负环。



DG=

	x1	x2	x3
x1	-5	-10	-25
x2	5	-5	-15
x3	20	10	-5

(b)

图 2-2 STN 和 DG 示例

2.2.2 求解算法

定理 2-1. 给定一个 STP：P=<X, C>，P 是一致的当且仅当∀i，有 $c_{i,i}\geq0$,即对应的 STN 不存在负环[6]。

上述定理给出了求解 STP 的一个思路：通过判断其对应的 STN 中是否存在负环便可。用于计算每一对顶点之间的最短路径的 Floyd-Warshall 算法是较好的一个解决算法，这是由于该算法允许图中有带负权值的边，但不允许有带负权值的回路，因此适用于 STP 的求解。很明显，在计算复杂度方面，STP 是多项式时间可解的。图 2-3 给出的 Floyd-Warshall 算法的伪代码[4]。

```
For i←1 to n do
    For j←1 to n do
        dist(i,j) = weight(i,j)
For k←1 to n do
    For i←1 to n do
        For j←1 to n do
            if(dist(i,k) + dist(k,j) < dist(i,j)) then
                dist(i,j) = dist(i,k) + dist(k,j)
```

图 2-3 Floyd-Warshall 算法

虽然 Floyd-Warshall 算法的时间复杂度为 $O(|V|^3)$，但由于其代码很紧凑，且并不包含复杂的数据结构，因此隐含的常系数是很小的，即使对于中等规模的输入来说，它仍然是相当有效的。

2.3 析取时态问题

2.3.1 基本定义

为了便于理解析取时态问题的相关定义，先引入另外一个时态模型：时态约束可满足问题（TCSP）[6]。

定义 2-9.（时态约束可满足问题 TCSP）一个时态约束可满足问题是一个二元组<X, C>，其中：$X=\{x_1, x_2, \cdots, x_n\}$称为变量集合；$C=\{C_1, C_2, \cdots, C_m\}$是一个约束集合，每个 C_i 是一个形如 $c_{i1} \vee c_{i1} \vee \cdots \vee c_{ik}$ 的析取式，其中的 c_{ij} 是一个形如 $x_u - x_v \leqslant c$ 的简单时态约束（$x_u, x_v \in X$；$x_u, x_v, c \in R$），且在每个析取式中，所有简单时态约束对应的约束变量（即 x_u 和 x_v）是相同的。

规划系统中的时态模型往往是使用基于时间点上约束可满足的形式来表达时态关系。下面通过一个基于时间点的例子来说明 TCSP。

例 2-3. 李雷上班或者开车（30-40 分钟）或者搭公交（至少 60 分钟），韩梅梅上班或者打的（20-30 分钟）或者搭公交（40-50 分钟）。今天李雷出发的

时间是 7:10-7:20，韩梅梅到达的时间是 8:00-8:10。我们还知道李雷在韩梅梅出发 10-20 分钟后到达。我们希望回答如下等等问题：以上故事里的信息是一致的吗？有没有可能李雷和韩梅梅都搭公交？韩梅梅出发的可能时间段是什么？

如果用 P_1 表示“李雷去上班”，P_2 表示“韩梅梅去上班”，X_1 和 X_2 分别表示“李雷出发和到达的时刻”，X_3 和 X_4 分别表示“韩梅梅出发和到达的时刻”，则 $P_1=[X_1, X_2]$ 且 $P_2=[X_3, X_4]$，根据已知条件有 $30\leqslant X_2-X_1\leqslant 40 \vee X_2-X_1\geqslant 60$ 且 $20\leqslant X_4-X_3\leqslant 30 \vee 40\leqslant X_4-X_3\leqslant 50$，“李雷在韩梅梅出发 10-20 分钟后达到”可以表示为 $10\leqslant X_2-X_3\leqslant 20$，若引入一个时间点 X_0=7:00，则“李雷出发时间”可以表示为 $10\leqslant X_1-X_0\leqslant 20$，“韩梅梅到达时间”可以表示为 $60\leqslant X_4-X_0\leqslant 70$。根据这些信息就可以回答接下去的问题。

由上述 TCSP 例子可知，TCSP 相比 STP 而言增加了对析取逻辑的支持，也就是说允许存在不确定性的约束，这将导致 TCSP 求解复杂度为 NP-hard[4][6]。虽然 TCSP 相比 STP 已经有了更为丰富的表达力，但是实际应用可能更为复杂，这就涉及到本节主要要介绍的模型——析取时态问题 DTP。

定义 2-10.（析取时态问题 DTP）一个析取时态问题是一个二元组 $P=\langle X, C\rangle$，其中：$X=\{x_1, x_2, \cdots, x_n\}$ 是一个变量集合；$C=\{C_1, C_2, \cdots, C_m\}$ 是一个约束集合，每个 C_i 是一个形如 $c_{i1}\vee c_{i1}\vee\cdots\vee c_{ik}$ 的析取式（也称为子句），可以表示为集合 $\{c_{i1}, c_{i1}, \cdots, c_{ik}\}$，其中的 c_{ij} 是一个形如 $x_u-x_v\leqslant c$ 的简单时态约束（$x_u, x_v\in X$；$x_u, x_v, c\in Z$[5]）。

本文假设 DTP 中两个相同变量之间不存在简单时态约束，也就是说不存在类似 $x-x\leqslant c$ 的简单时态约束，这和第 5 章探讨使用的 DTP 生成脚本所做的定义是一致的。另外，当 $|C_i|=1 (i=1,2,\cdots,m)$ 时，DTP 退化为一个 STP。

[4] NP-hard 问题是指至少和 NP 问题中最难问题一样的难解决的一类问题[7]。
[5] 跟其他研究文献类似，本文只讨论取值为整数的情况，然而它可以很容易拓展到任意精度的有理数的情况中去。

与 STP 类似，DTP 也是一个典型的 CSP（变量域为整数域，约束集合是二元不等式组成的），根据 CSP 的相关定义可以得到 DTP 的解、一致性、等价性的概念。由一致性相关定义可得到下述性质。

性质 2-1. 给定一个 DTP：$P=\langle X, C\rangle(X=\{x_1, x_2, \cdots, x_n\}, C=\{C_1, C_2, \cdots, C_m\})$，P 是一致的，当且仅当存在一个一致的 STP：$P=\langle X, C'\rangle(C'=\{c_1, c_2, \cdots, c_m\}, c_i \in C_i$ 是一个简单时态约束）。这样的 STP 通常称为析取时态问题的解 STP。

与 STN 类似，DTP 也有其对应的时态约束网络，下面给出定义。

定义 2-11.（析取时态网络 DTN）给定一个析取时态问题 $P=\langle X, C\rangle$，它相应的析取时态网络是一个五元组 $N=\langle V(N), E(N), L, W, PRJ\rangle$，其中：$V(N)=\{v_1, v_2, \cdots, v_n\}$ 是顶点集合；$E(N)=\{\langle v_i, v_j, k, w\rangle | v_i, v_j \in V(N), k \in L, w \in W\}$ 是一个四元组边集合；$L=\{k | C_k \in C\}$ 是一个标号集合；W 是一个权重符号集合；PRJ 是个映射函数，使得：$PRJ(x_i)=v_i(x_i \in X)$，$PRJ(c_k=x_i-x_j \leqslant c)=\langle v_i, v_j, k, c\rangle$。

大量的研究发现，从约束问题的图形结构中可以发现许多有用的启发式信息，常常可以用于在搜索中减少访问节点，甚至可以发现不需要任何回溯的求解算法[1]。因而研究 DTN 这一重要的数据结构十分必要。一个 DTN 是一致的，当且仅当其对应的 DTP 是一致的，两者常常不加区别的使用。

我们再次引入例 2-3 并对某些条件进行修改，得到一个 DTP 实例 2-4。

例 2-4. 李雷上班或者开车（30-40 分钟）或者搭公交（至少 60 分钟），韩梅梅上班或者打的（20-30 分钟）或者搭公交（40-50 分钟）。今天李雷出发的时间是 7:10-7:20，韩梅梅到达的时间是 8:00-8:10。我们还知道李雷在韩梅梅出发 10-20 分钟后到达，**或者在韩梅梅到达 15-20 分钟前到达**。我们希望回答如下等等问题：以上故事里的信息是一致的吗？有没有可能李雷和韩梅梅都搭公交？韩梅梅出发的可能时间段是什么？

上述例子修改一个约束条件，即 $10 \leqslant X_2-X_3 \leqslant 20 \vee 15 \leqslant X_4-X_2 \leqslant 20$。这样的一

个改变使得其表达力可以满足大多数智能规划和时态规划中的时态问题[1]，当然，其复杂度也是 NP-hard。

下面提供例子解答。首先列出所有约束不等式：

$30 \leqslant X_2-X_1 \leqslant 40 \vee 60 \leqslant X_2-X_1 \leqslant \infty$，

$20 \leqslant X_4-X_3 \leqslant 30 \vee 40 \leqslant X_4-X_3 \leqslant 50$，

$10 \leqslant X_2-X_3 \leqslant 20 \vee 15 \leqslant X_4-X_2 \leqslant 20$，

$10 \leqslant X_1-X_0 \leqslant 20$，

$60 \leqslant X_4-X_0 \leqslant 70$

根据基本的命题逻辑理论和算术理论，可以将其编码为一个析取时态问题 $P=\langle X, C\rangle$（对应的 DTN 见图 2-4）：

$X=\{X_0, X_1, X_2, X_3, X_4\}$,

$C=\{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}\}$.

$C_1=\{X_1-X_2 \leqslant -30\}$,

$C_2=\{X_2-X_1 \leqslant \infty\}$,

$C_3=\{X_2-X_1 \leqslant 40,\ X_1-X_2 \leqslant -60\}$,

$C_4=\{X_3-X_4 \leqslant -20\}$,

$C_5=\{X_4-X_3 \leqslant 50\}$,

$C_6=\{X_4-X_3 \leqslant 30,\ X_3-X_4 \leqslant -40\}$,

$C_7=\{X_2-X_3 \leqslant 20,\ X_4-X_2 \leqslant 20\}$,

$C_8=\{X_3-X_2 \leqslant -10,\ X_4-X_2 \leqslant 20\}$,

$C_9=\{X_2-X_3 \leqslant 20,\ X_2-X_4 \leqslant -15\}$,

$C_{10}=\{X_3-X_2 \leqslant -10,\ X_2-X_4 \leqslant -15\}$,

$C_{11}=\{X_1-X_0 \leqslant 20\}$,

$C_{12}=\{X_0-X_1 \leqslant -10\}$,

$C_{13}=\{X_4-X_0 \leqslant 70\}$,

$C_{14}=\{X_0-X_4 \leqslant -60\}$.

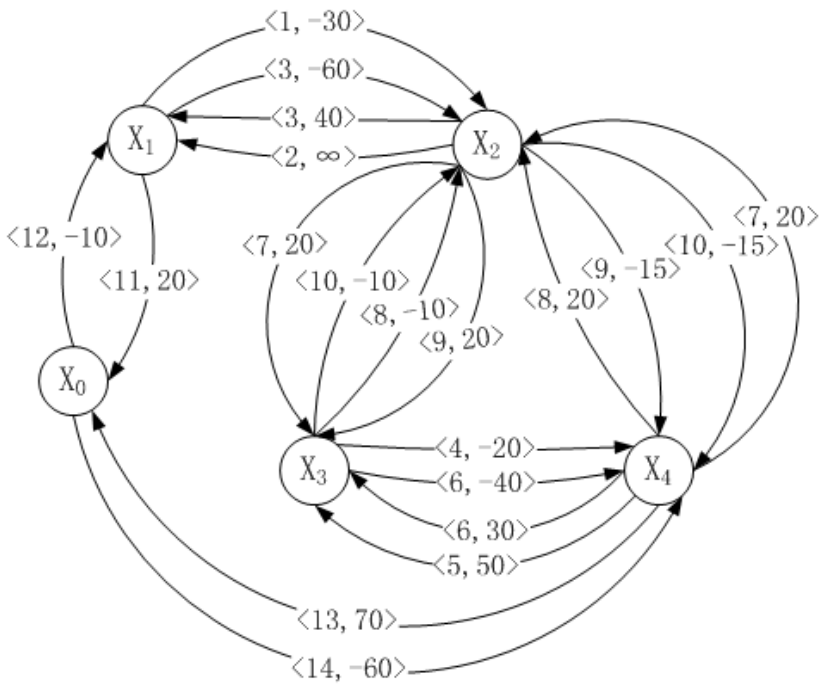


图 2-4 例 2-4 析取时态网络 DTN

2.3.2 求解技术

根据 CSP 的定义，一个 DTP(P=⟨X, C⟩)也是一个 CSP(P'=⟨Var, Dom, Cons⟩)：Var 即是 X，Dom 即隐含的整数域 D，Cons 即是 C。从求解的角度来看，可以把一个 DTP 看作是一个元约束可满足问题（meta-CSP)：变量表示每个析取式，而每个变量的取值范围则是变量对应的析取式中的析取肢（简单时态约束）组成的集合；对应 meta-CSP 问题的解就是给每个变量取一个值，使得所有变量的赋值组成一个一致的 STP。meta-CSP 与 CSP 的对比见表 2-1。本文探讨的析取时态求解系统就是基于 meta-CSP 模型实现的。

对于 meta-CSP 的求解，有两个问题需要特别注意：一是如何判定问题的一致性（约束方面)，一是如何解决状态爆炸的问题（启发式方面)。解决这两个问题的方法决定着实际的计算效率，而解决后者对提高效率的共享更大。关于

一致性的判定前面已介绍过方法，这里主要讨论的是启发式技术。

表 2-1 CSP 与 meta-CSP 的对比

元素	CSP	meta-CSP
变量	x_1, x_2, …, x_n	每个 DTP 约束 C_i 对应一个变量 C_i
取值域	$x_i \in (-\infty, +\infty)$	$Dom(C_i)=\{c_{i1}, c_{i2}, \cdots, c_{ik(i)}\}$
约束	C 中的析取约束	所有变量的值（隐含地）组成一个一致的 STP

引入启发式技术的目的，是为了将搜索首先集中于那些更可能引起冲突的变量集合，从而在最大程度上提高算法效率。目前用于求解 DTP 的启发式技术主要有最少剩余值变量选择策略（MRV）、前向检查技术（FC）和被吸收变量移除技术（RSV）等，其中 FC 和 MRV 被证明是最有效的启发式技术组合[8,9]。下面分别对这些启发式进行介绍。

1. 最少剩余值变量选择策略（MRV）

在当前节点选择变量进行赋值的时候，优先选择那些具有最少候选元值的元变量。这是一种最常使用的基于“失败优先”原则[5]的变量选择策略。与前向检查技术配合使用时，由于元变量的取值域随着搜索的进行不断地发生变化，因此 MRV 实现了动态的变量选择功能。MRV 算法可参见图 2-5。大量实验证明，FC 与 MRV 的组合具有显著的剪枝能力。

```
ArrayList[] x;
int minSize = MAX_VALUE;
for (int i = 0; i < x.size(); i++) {
    var v = x.get(i);
    if (!v.assigned() && v.size() < minSize) {
        minSize = v.size();
        index = i;
    }
}
```

图 2-5 MRV 启发式算法

2. 前向检查技术（FC）

在尝试赋值之前，首先从未赋值元变量的取值域中删除那些与当前部分解不一致的元值（条件见定理 2-2）。这种技术的优点在于可以在进行下一轮迭代之前首先发现搜索的死点（即发现某个元变量的取值域为空），从而可以提前回溯，避免进行下一轮迭代。具体算法参见图 2-6。

```
1  // initialize distance graph DG
2      ...
3  // check consistency
4  for every unassigned v {
5      for every value w to v {
6          // RSV
7          if (isRSV() && DG[x][y] <= w) {
8              v = w;
9              break;
10         }
11         // FC
12         if (isFC() && DG[y][x] + w < 0) {
13             v != w;
14         }
15     }
16 }
```

图 2-6 FC 和 RSV 启发式算法

定理 2-2.（FC 条件定理）一个元值 c_{ij}:x-y≤w_{xy} 与当前 STP 不一致当且仅当在当前距离图 DG 中，有 DG[y][x]+w_{xy}<0（FC 条件）成立。

3. 被吸收变量移除技术（RSV）

假设在当前部分解 S 的距离图中，有 DG[x][y]=c，如果对于某个未赋值的元变量 C_k 其取值域中具有一个候选元值 c_{kh}:x-y≤c'且有 c≤c'，则称 C_k 被 S 所吸收，此时可将 c_{kh} 直接赋值给 C_k 而不需要进行一致性判定和尝试其它的值。算法见图 2-6。之所以将 RSV 算法与 FC 放在一起，是因为两者可以放在同一循环体中运行，这样处理可以提高效率，避免在同时使用两者时添加冗余代码。

针对以上三种启发式技术，下文将以一个例子来展示各自的求解过程。

例 2-5. 已知一个 DTP：P=<X, C>， X={x_1, x_2, x_3, x_4, x_5}， C={C_1, C_2, C_3}.

C_1={c_{11}(=x_1-x_2≤2)， c_{12}(=x_3-x_2≤-1},

$C_2=\{c_{21}(=x_2-x_3\leqslant 2),\ c_{22}(=x_1-x_2\leqslant 4\}$,

$C_3=\{c_{31}(=x_2-x_1\leqslant -3),\ c_{32}(=x_3-x_1\leqslant -5\}$.

首先，作为对比，在无启发式的情况下观察该 DTP 的求解搜索过程，见图 2-7 和图 2-8。其元值的选择按照从上到下、从左到右的顺序进行。

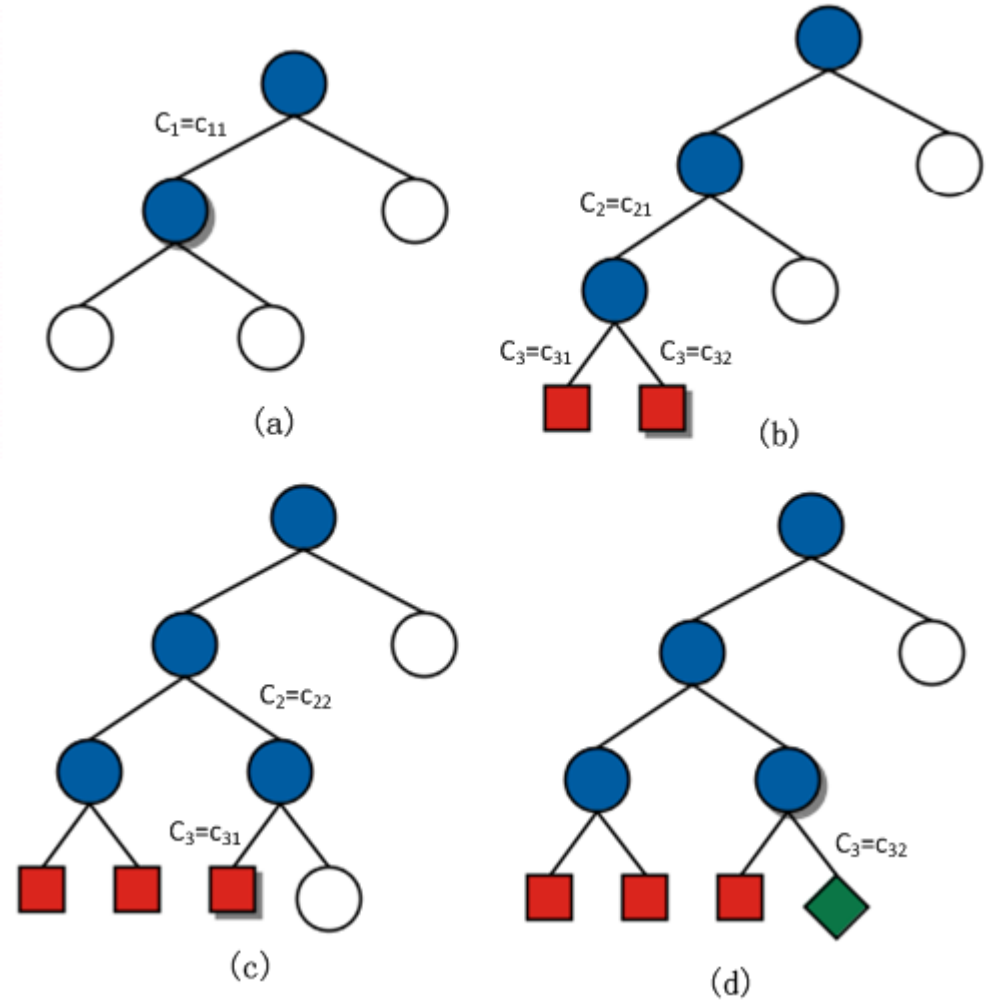


图 2-7 无启发式搜索（选项路径）

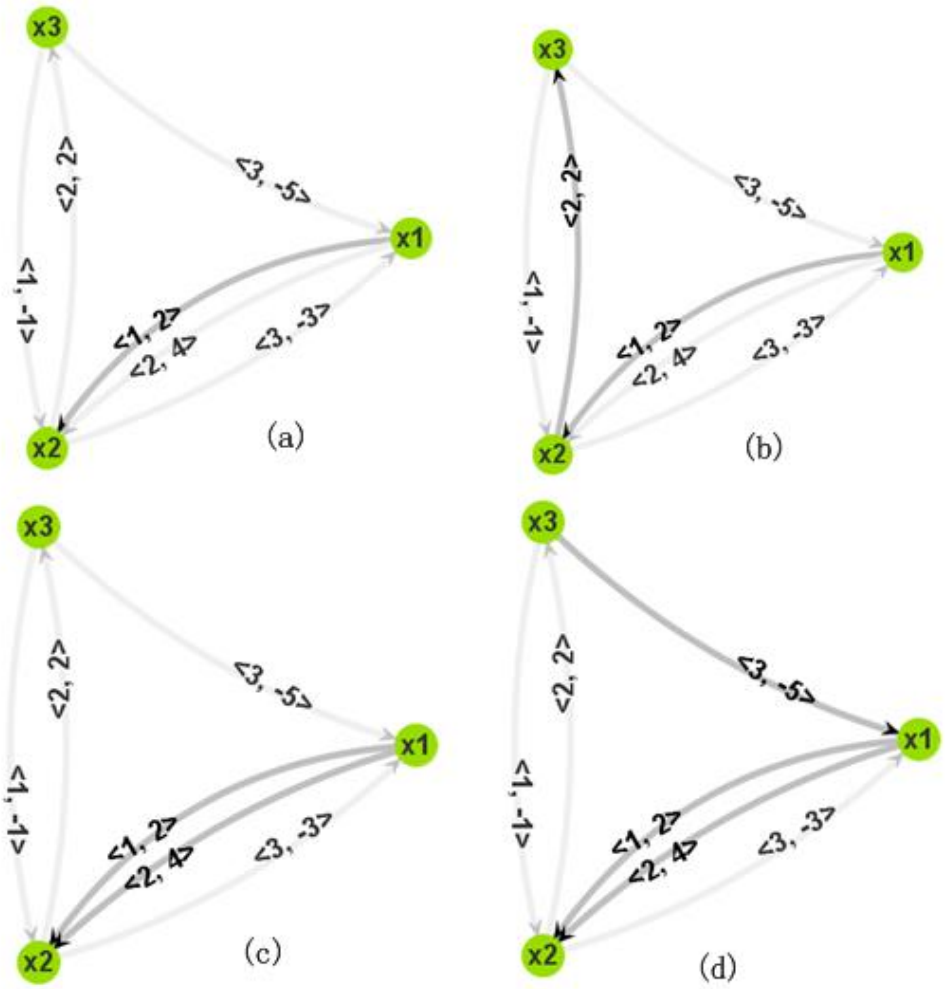


图 2-8 无启发式搜索（边选择）

接下来分析各种启发式技术的求解步骤。

1）MRV 技术求解

根据该启发式的主要思想，首先要选择具有最少候选原值的元变量。由于本例子的各个元变量的候选元值个数均为 2，其搜索过程与无启发式情况一致，因而单纯使用该启发式技术并不能体现剪枝效果。在以下 FC 技术内容中，将会结合 MRV 技术进行阐释。

2）FC 技术求解

根据该启发式的主要思想，首先要从未赋值元变量的取值域中删除那些与当前部分解不一致的元值。按照 MRV 启发式，首先将 c_{11} 赋给 C_1，然后再根据 FC 查找 C_2 和 C_3 中不一致的元值，可以发现 c_{31} 不符合要求，将其排除，此时可以直接确定 $C_3=c_{32}$（图 2-9(a)和图 2-10(a)）。接着再执行一遍 MRV 进行分支，将 c_{21} 赋给 C_2，进行一致性检查后发现与其他赋值元变量不一致，故舍弃并使 $C_2=c_{22}$。至此，获得一个解答（图 2-9(b)、图 2-10(b)和图 2-11）。

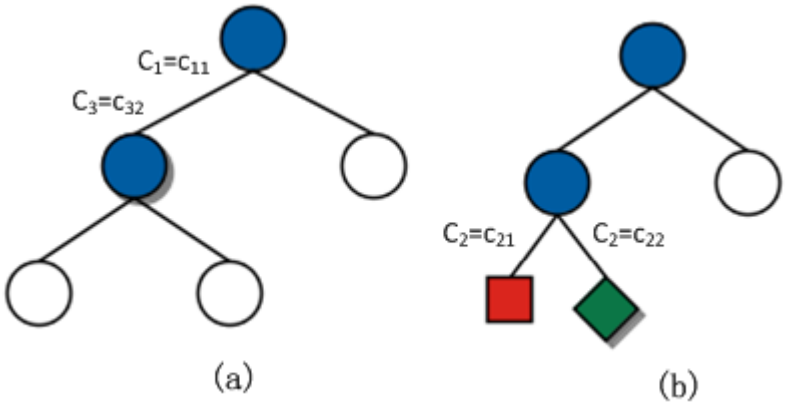


图 2-9 FC 和 MRV 启发式搜索（选项路径）

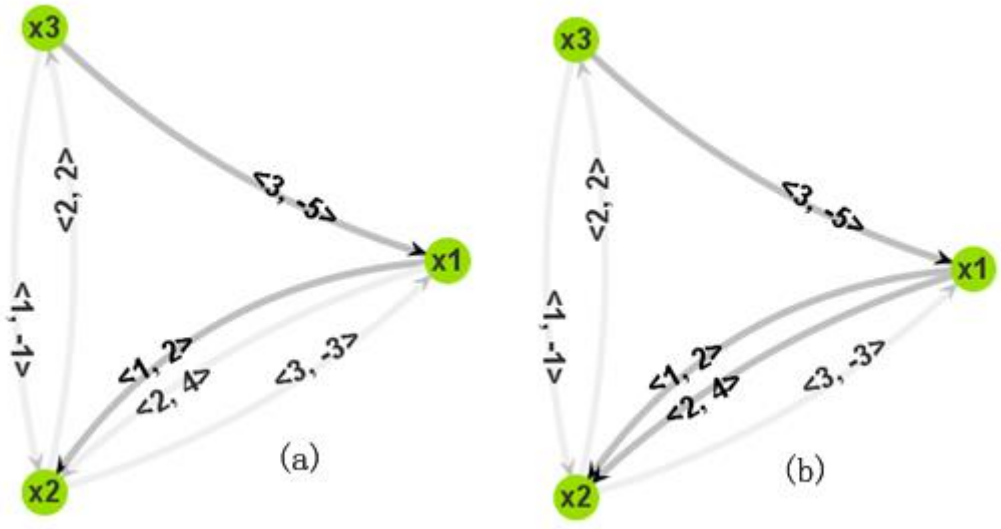


图 2-10 FC 和 MRV 启发式搜索（边选择）

```
Solution(L:label, S:source, T:target, W:weight):
    L=1     S=x1    T=x2    W=2
    L=2     S=x1    T=x2    W=4
    L=3     S=x3    T=x1    W=-5
```

图 2-11 FC 和 MRV 启发式搜索结果

若是搜索所有分支，可以得到多个解（图 2-12）。

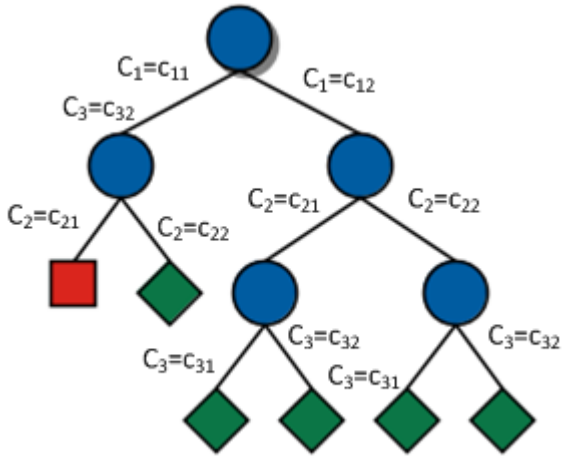


图 2-12 FC 和 MRV 启发式所有解答

3) RSV 技术求解

根据该启发式的主要思想，首先要查找未赋值元变量是否存在一个元值 c_{kh}:x-y≤c'，使得 DG[x][y]≤c'成立。先对 C_1 赋值 $C_1=c_{11}$，然后搜索未赋值元变量的值域发现，c_{22} 符合 RSV 启发式思想，即 DG[x_1][x_2]=2<4=w_{22}，因此将 c_{22} 赋给 C_2（图 2-13(a)和图 2-14(a)）。然后再进行分支，将 c_{31} 赋给 C_3，进行一致性检查后发现与其他赋值元变量不一致，故舍弃并使 $C_3=c_{32}$。至此，获得一个解答（图 2-13(b)和图 2-14(b)）。

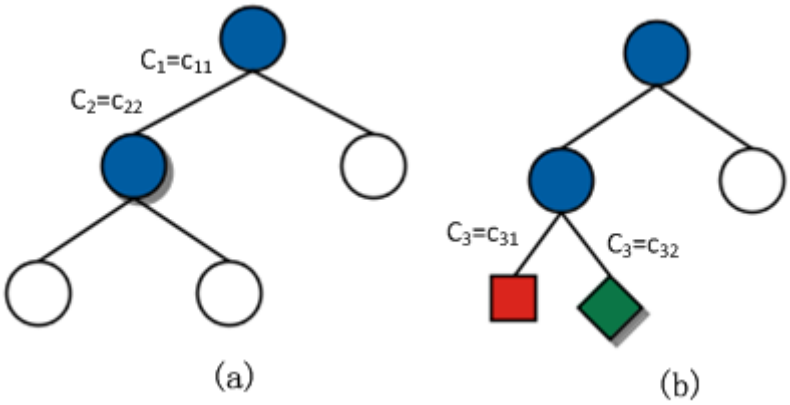


图 2-13 RSV 启发式搜索（选项路径）

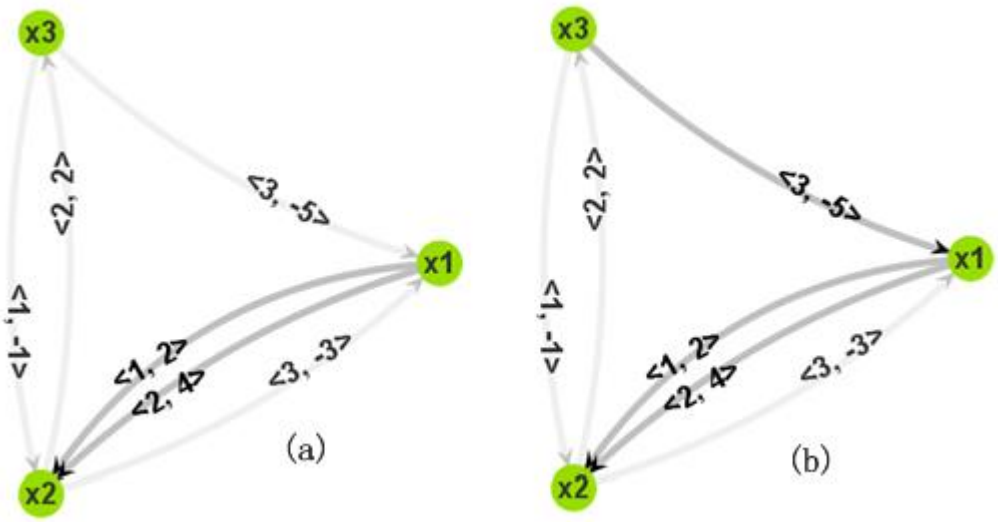


图 2-14 RSV 启发式搜索（边选择）

第3章　技术路线和实现方法

3.1　实现思路

系统实现的大致思路如下：首先编写建模模块，使用开源工具 JGraphT 来编写 DTN 的存储结构；然后编写视图模块，使用开源框架 JUNG 来实现 DTN 结构可视化；最后编写引擎模块，使用求解工具 Gecode 所提供的功能来实现推理系统的引擎核心。另外，为了方便代码的切换，实现启发式与引擎的解耦，在系统中加入 Spring 的 IoC 功能模块加以控制。

在介绍各种软件工具之前，首先对其版本进行说明：

- JGraphT 0.8.3 [11]
- Gecode 2.2.0，Gecode/J 2.2.0 [12,13,14]
- JUNG 2.0.1 [10]
- Spring 3.3.1 [15]

作者开发测试的环境如下：

- 编译环境支持[6]：Window OS, Cygwin_1.7.18-1, VS2005_cl 或 Linux OS，gcc/g++_3.4.6
- 开发运行支持：JDK1.6.0_45, Eclipse Juno

3.2　图形库 JGraphT

3.2.1　基本介绍

JGraphT 是一个提供数学图论对象和算法的简单易用的开源图形库。JGraphT 使用了 Java 泛型的特性，例如图形中的顶点可以是任何对象类型（String, URL, XML document, etc.），因而它是类型安全的（即不会因为使

[6] 这里编译对象是指 Gecode，具体所需工具可参照[12]，此处只是重点指出作者测试时使用的 c++编译器。

用强制类型转换而发生错误[16])。JGraphT 支持多种类型图：有向图和无向图；有带权边、无权边、标签或用户自定义边的图；简单图、多重图、伪图；不能修改的图；可监听图；自动更新的子图等等。JGraphT 还提供一个适配类 JGraphModelAdapter 使得程序可以通过调用 JGraph 来实现绘图功能。

3.2.2 基本构图过程

要构造一个图形，需要经历以下步骤：声明图形，设定节点和边类型；添加数个基本节点；添加数个边，以定义节点为参数，必要时要设置边的权值。

3.3 求解工具 Gecode

3.3.1 基本介绍

Gecode 是一个开放、免费、可移植、易使用、有效率的用于开发基于约束系统及应用的环境，该工具的编码严格按照 C++标准实现。Gecode 提供一个模块化和可拓展的约束求解器。Gecode/J 是 Gecode 库提供的一个 Java 接口，可以供 Java 应用程序调用。

3.3.2 编译运行

本文讨论和使用的 Gecode/J 版本是 2.2.0，根据其文档说明，需要 Gecode-2.2.0 的支持。无论是 Gecode 还是 Gecode/J，在使用之前都要先进行编译，而且前者要先于后者进行编译。由于 Gecode 的配置运行较为复杂，本文在“软件说明文档”的基础上做一个简要补充。

1. 编译器：Windows 平台只支持 VS2005 下的 cl.exe（Microsoft C/C++编译器），即编译需要使用 VS2005 提供的命令行工具（需要以超级管理员的方式启动）;Linux 平台应选用 gcc/g++的 3.4 版本才能编译成功，4.4 版本会存在兼容性问题。
2. 编译：Gecode/J 编译前，Windows 平台：需要为环境变量 include 添加

新路径，对应 Gecode 安装完成时所生成 include 文件夹；Linux 平台：在 Gecode/J 编译配置（执行 configure）前，需要为环境变量 PKG_CONFIG_PATH 添加新路径,对应于 Gecode 安装完成时所生成的 lib 文件夹下的 pkgconfig 文件夹路径。

3. 异常处理：编译过程中如果出现“未知命令”问题，只需要添加相应工具就可以解决，如 ant 命令需要配置 Apache Ant 工具；如果出现“权限不允许”问题，应该修改文件权限为用户可访问；如果出现“文件未找到”问题，则查看是否添加了对应路径。
4. 运行例子：
   1) 若使用命令行窗口运行 Gecode 的例子（验证安装），有两种方法：在 Gecode 的安装目录下，直接输入 examples/<program>；在编译后生成 bin 文件夹中运行例子。
   2) 若使用命令行窗口运行 Gecode/J 的例子（/examples 文件夹中），在运行前，Windows 平台：需要为环境变量 path 添加新路径，对应于 Gecode 安装完成时所生成的 bin 文件夹路径，在运行时在安装目录下输入 examples/<program>（Gecode/J 是把 examples 当作包名）。特别的，如果要在系统任何位置或使用 eclipse 运行例子，还需要在 path 中添加新路径，对应于 Gecode/J 安装完成时所生成的 bin 文件夹路径；Linux 平台：如果提示缺少库文件，则需要修改/etc/ld.so.conf 文件，添加 Gecode 安装完成时所生成的 lib 文件夹路径。
   3) Gecode/J 运行参数：

      -gui, -nogui: 图形界面，文本界面（默认前者为 true）;

      -mode [solution|time|timenogc]: 打印模式 [求解结果全信息（默认）|运行时间|带有垃圾回收操作的运行时间]（使用-nogui）;

      -naive, -smart: 使用简单、智能模型（默认后者为 true，求解较

快);

-solution: 解决方案最大值（默认为 1，0 表示全部);

-fails: 搜索失败最大值（默认为-1 无限制);

-time: 最大可搜索时间（默认为-1 无限制);

-print: 是否打印解决方案（默认为 true);

-bab: 是否使用最佳搜索（默认为 false);

-icl [dom|bnd|val|def]: 整型变量约束的一致性等级（默认为 bnd);

-c_d, -a_d: 拷贝和适配搜索引擎的重新计算距离（默认分别为 2,8);

-samples, -iterations: 在时间模式下的计算按样例数和迭代次数每样例（默认均为 1);

-size: 问题大小;

-help: 打印帮助信息。

3.3.3 基本术语

主要的相关术语解释如下：

1. 分支（Branching)：确定了搜索树的形状，创建了一系列选择点，它在搜索的过程中决定启发式（对不确定因素做出选择）的使用。
2. 分支描述（Branching description)：表示一个选择点（分支）的选择信息，即确定一个变量的取值。
3. 传播（Propagator)：一个传播实现一个约束（实际上，一个约束可以由一个传播集合实现)，传播的执行依赖于与传播条件共存的视图（引用一些变量)。
4. 变量（Variable)：用于建模问题中的直接或通过某些接口建模，提供有用的建模操作并排除可直接修改变量域的特定操作。

5. 视图（View）：提供和变量实现本质上一样的接口，允许值域的访问和修改。通常，视图只在分支和传播中定义和使用。
6. 角色（Actor）：或者是一个分支，或者是一个传播。它提供分支和传播共有的功能，如成员拷贝、内存分配等等。
7. 计算空间(Computation space)：包含所有需要解决的约束问题实体(所有的角色和变量)，它组织了约束传播、分支处理、路径探索和内存管理。

3.3.4 一般编程框架

使用 Gecode/J 建立模型时，需要考虑以下内容：待赋值的变量数组（代表需求解选项)、模型定义（包括分支和约束)，而后者是最为核心的部分。在定义问题求解类的时候，都需要继承 Space 类（求解的空间)，具体求解框架见图 3-1。以下重点介绍模型的定义。

模型的特征是在求解类的构造函数中定义的，主要包括类成员初始化、约束定义和所使用的分支方案。初始化即是对待赋值的变量数组进行值域定义，既可以统一规定所有变量的值域，也可以单独针对每个变量进行规定。约束也叫传播，在一个模型中可以定义多个传播，搜索过程中会根据定义的次序顺序执行各个传播。分支是模型中较为基础的部分，一般定义一个，其定义次序与传播无关。图 3-2 给出了 N 皇后问题模型的定义例子。

```
public class Problem extends Space {
    private VarArray<IntVar> q;
    public Problem(Options opt) {
        super();
        <model>
    }
    public Problem(Boolean share, Problem p) {
        super(share, p);
        q = new VarArray<IntVar>(this, share, p.q);
    }
    public static void main(String[] args) {
        Options opt = new Options();
        opt.parse(args);
        Problem p = new Problem(opt);
        opt.doSearch(p);
    }
}
```

图 3-1 Gecode 一般的求解框架

```
<model> += <initialization> <constraints> <branching>

<initialization> +=
    n = opt.size;
    q = new VarArray<IntVar>(this, n, IntVar.class, 0, n-1);
<constraints> +=
    int c[] = new int[n];
    for (int i=0; i<n; i++)
        c[i] = i;
    distinct(this, c, q, opt.icl);
    for (int i=0; i<n; i++)
        c[i] = -i;
    distinct(this, c, q, opt.icl);
    distinct(this, q, opt.icl);
<branching> +=
    branch(this, q, BVAR_SIZE_MIN, BVAL_MIN);
```

图 3-2 N 皇后问题在 Gecode 中的模型定义

求解模型中的分支和传播是启发式技术得以应用需要关注的部分，其中的传播更是挖掘应用更多启发式的理想位置。因而，了解 Gecode 传播的规划和执行机制也十分必要，见图 3-3。

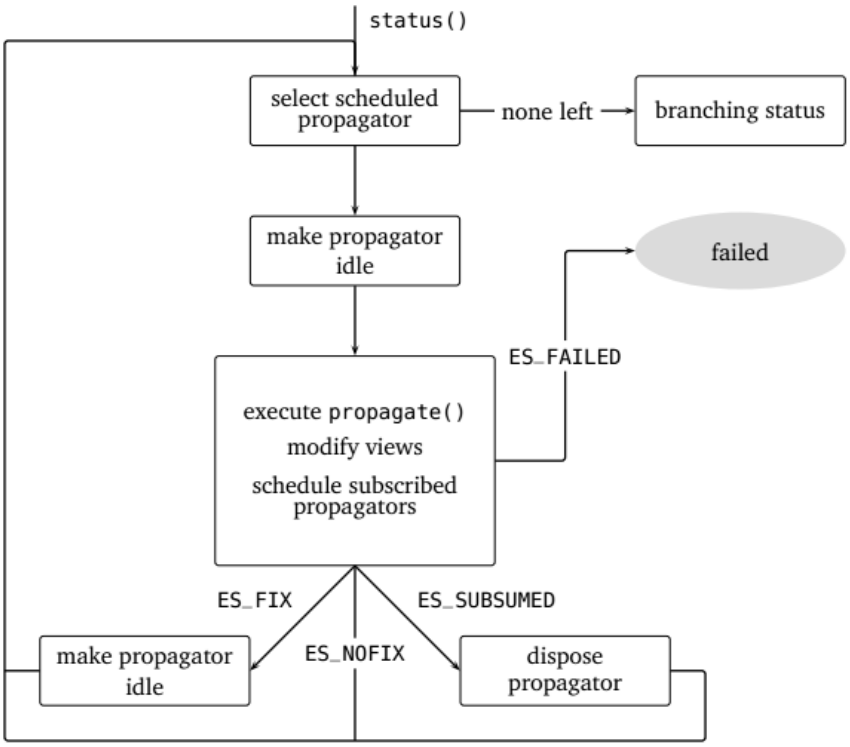


图 3-3 Gecode 中传播的规划和执行

这里结合分支简要介绍传播的执行过程。首先，在分支中判断变量数组当前状态，若尚有未赋值变量，则根据分支策略选择一个变量并为其赋值；然后，判断当前是否有可用传播，若有则选择其中一个并让其空转（idle），再执行传播添加约束（移除与约束实现相矛盾的变量取值），结束后重新规划一次传播，直到没有可用传播返回分支状态判断。

在执行分支和传播时有几个执行状态（Execution Status）需要注意。ES_FAILED 是指执行失败；ES_FIX 是指变量值没有被修改，传播没有被规划，处于定点状态，因此需要将传播置为空转状态；ES_NOFIX 是指变量值已经被修改，传播将要被规划，不处于定点状态；ES_SUBSUMED 是指传播不能再执行任何约束需要被处理。

图 3-4 展示了使用 Gecode 的图形交互搜索工具 Gist 来展示 6 皇后问题的

一条搜索路径（圆形表示分支节点，正方形表示失败节点，菱形表示成功节点）。

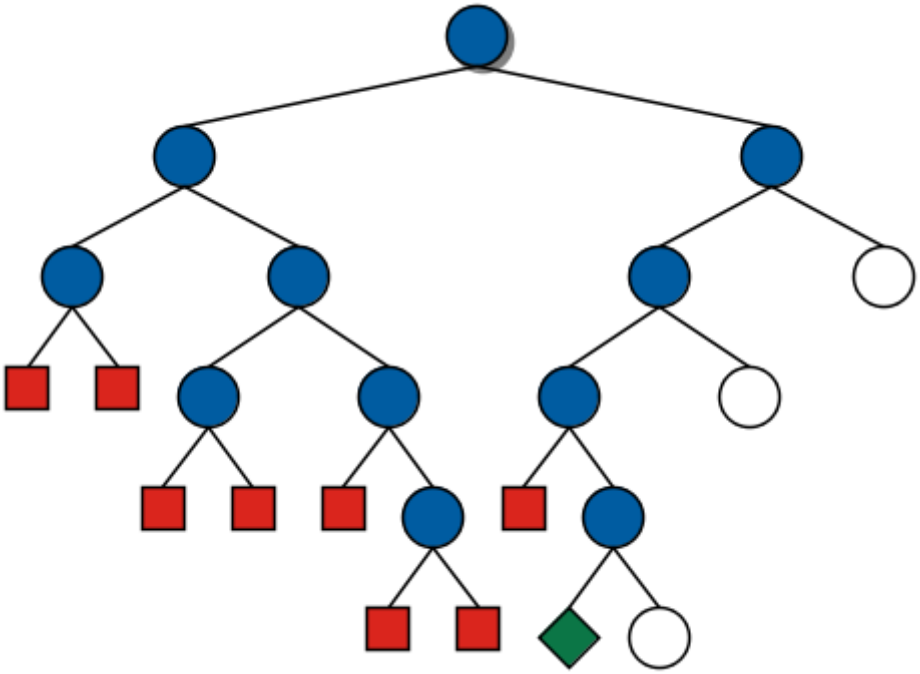


图 3-4 使用 Gist 得到 6 皇后问题的一条搜索路径

3.4 可视化框架 JUNG

JUNG（Java Universal Network/Graph Framework）是一个对建模、分析和可视化数据提供通用可扩展语言的软件库，它使用 Java 编写，允许基于 JUNG 的应用程序利用 Java API 以及其它第三方 Java 库。JUNG 支持有向图、多重图等结构的表示，支持图的多种布局，并且可以通过自定义渲染器来美化各种图形表示，使其能够表示各种各样的图形，有利于人们进行分析理解。相比其它的开源框架，JUNG 的易用性值得称道。另外，作为一款优秀的图形框架，JUNG 还实现了来自图论、数据挖掘、社交网络分析等许多算法。

3.5 模块化 Spring IoC

IoC（Inversion of Control）控制反转是 Spring 框架中 Core and Beans 模块的一个特性，也可以叫 DI（Dependency Injection）依赖注入。IoC 容器可以理解为用于管理 Bean、创建 Bean 的一个内存区，IoC 就是将接口和实现分离。利用 Spring IoC 的特性，即使程序正在运行之中，也可以很方便地切换运行模块。这种解耦的特性对于那些具备多个模块并需要随时切换的系统十分有利。作者希望利用 IoC 模块的这种特性，提供方便算法测试的运行环境。

第4章　系统设计

4.1　系统结构

4.1.1　总体结构

本文探讨设计的系统 TRSE 主要分为建模模块（Model）、引擎模块（Engine）和视图模块（View）。运行者（Runner）通过调用这三个模块来执行时态推理求解。系统结构见图 4-1，其中的脚本（Script）用于生成随机 DTP，详见 5.1 节。系统最外层包图和类图见图 4-2。

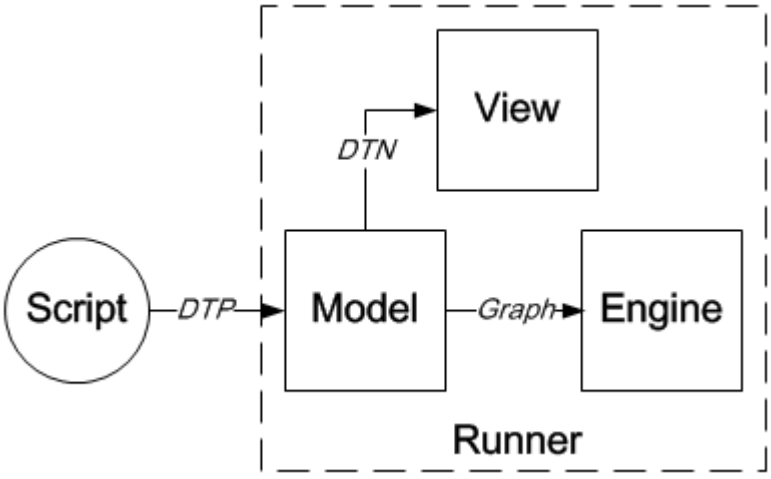


图 4-1 TRSE 逻辑框图

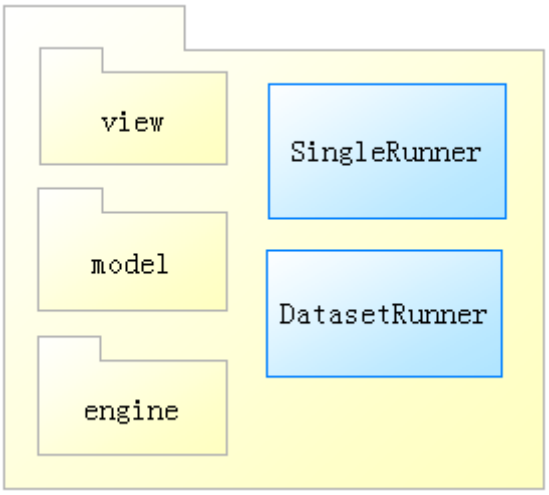


图 4-2 TRSE 最外层包图和类图

4.1.2 建模模块

建模模块主要负责接收脚本文件随机生成的 DTP 文件，并转换成对应的 DTN 图形结构，再将其提供给引擎模块。内部结构见图 4-3，类图见图 4-4。其中，DTNGraph 是根据 DTN 构图需要定制的图形类，使用了工具 JGraphT 进行 DTN 的结构建模；Translator 是一个翻译器，它将 DTP 翻译为对应的 DTN 并将其信息用于结构建模。

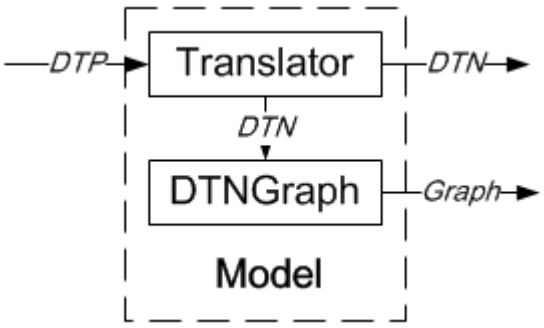


图 4-3 建模模块内部结构

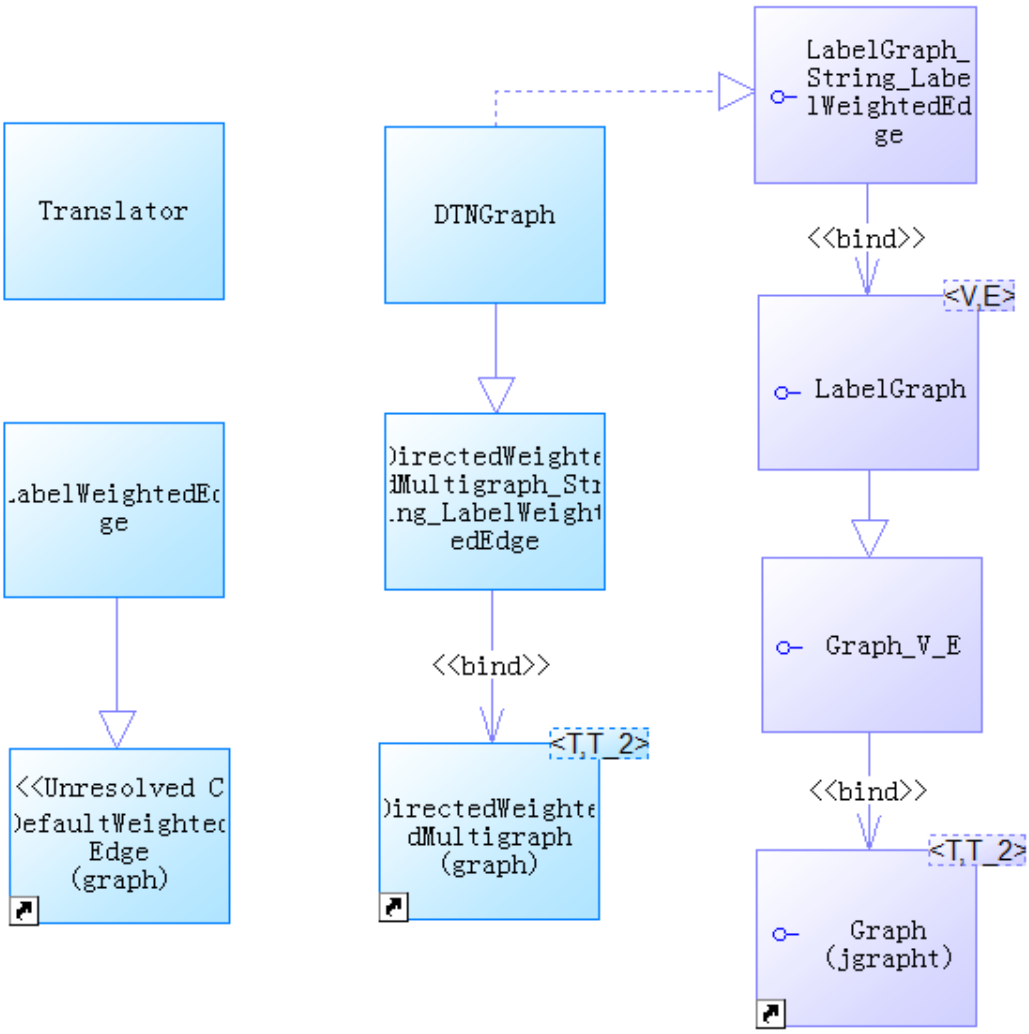


图 4-4 建模模块类图

4.1.3 视图模块

视图模块主要负责实现 DTN 的可视化，使用了可视化框架 JUNG，主要包括 DTNView 类，主要用于分析展示。类图见图 4-5。

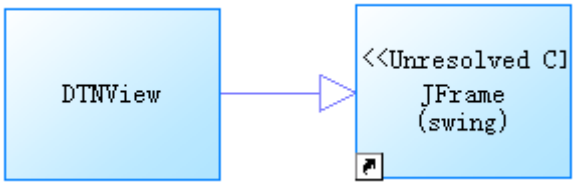


图 4-5 视图模块类图

4.1.4 引擎模块

引擎模块主要负责定义分支和传播（约束），通过使用建模模块递交的图形可以获得问题的求解结果。该模块使用了求解工具 Gecode，内部结构见图 4-6，类图见图 4-7。其中，BaseBranching（无实现启发式）和 MRVBranching（实现 MRV 启发式）同为分支（Branching）策略，两者在系统使用中必须是二选一，FWAPropagator（Floyd-Warshall 算法）和 PrePropagator（实现 FC 和 RSV 启发式）同为传播（Propagator）策略，前者为必选，后者为可选。分支和传播是引擎的运行核心，两者在实际运行中需要轮流执行并迭代，即分支→传播→分支→…（见图 4-8），在每次迭代中分支会提交一个选择（Choice）给传播，直到完成搜索（或到达死点）并得出求解结果，必要时将部分解传递给建模模块的 DTNView 予以回显。

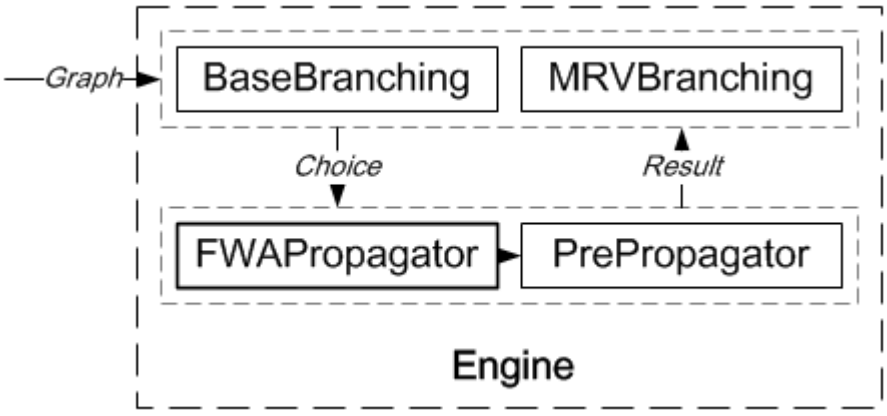


图 4-6 引擎模块内部结构

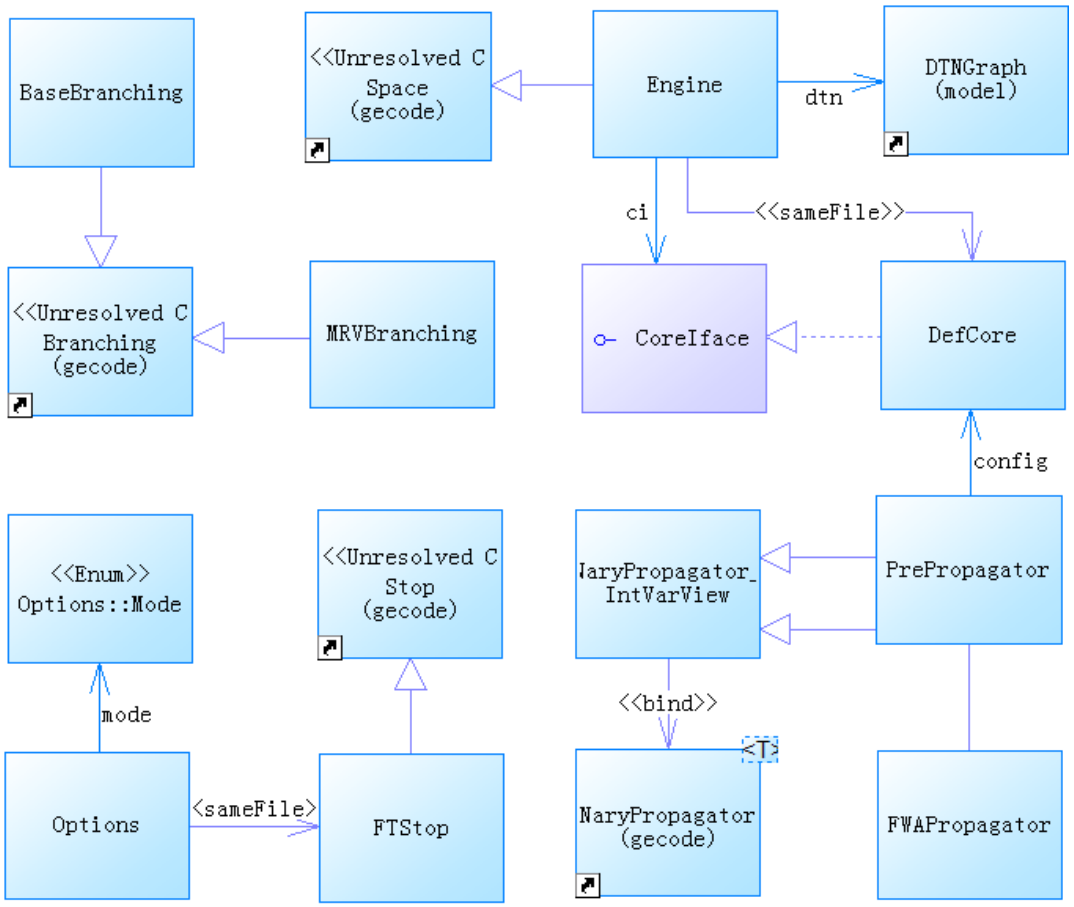


图 4-7 引擎模块类图

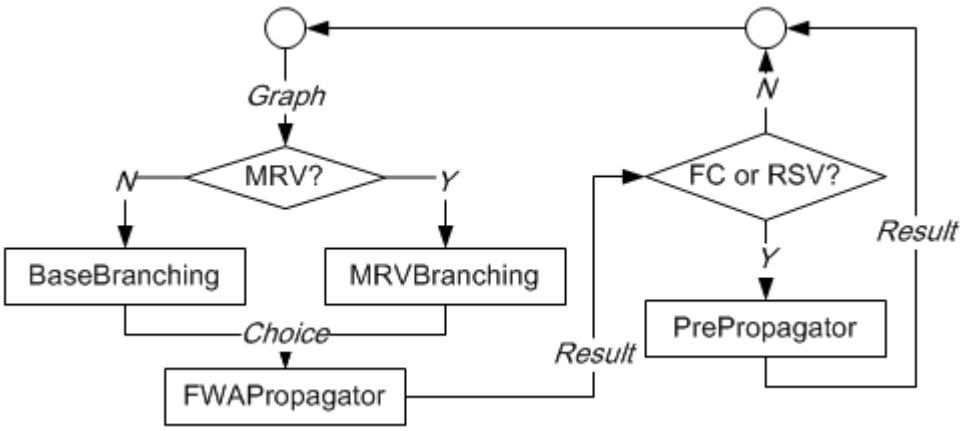


图 4-8 分支和传播轮流执行迭代过程

4.2 系统运行

图 4-9 是 TRSE 的时序图，其过程如下：研究人员把对计算 DTP 一致性的请求交给运行者 Runner，然后 Runner 会将 DTP 信息交给翻译器 Translator 以得到对应的 DTN，相应的 DTN 信息分别传递给 DTNView 和 DTNGraph，前者是 DTN 的可视化，后者是 DTN 的内部存储，DTNGraph 会将形成的图形结构传递给引擎 Engine，引擎首先会初始化分支 Branching 和传播 Propagator，然后发出搜索消息，让分支和传播迭代运行，在每次运行中，分支会将选项发送给传播，传播收到选项执行约束判断后，返回结果给分支。

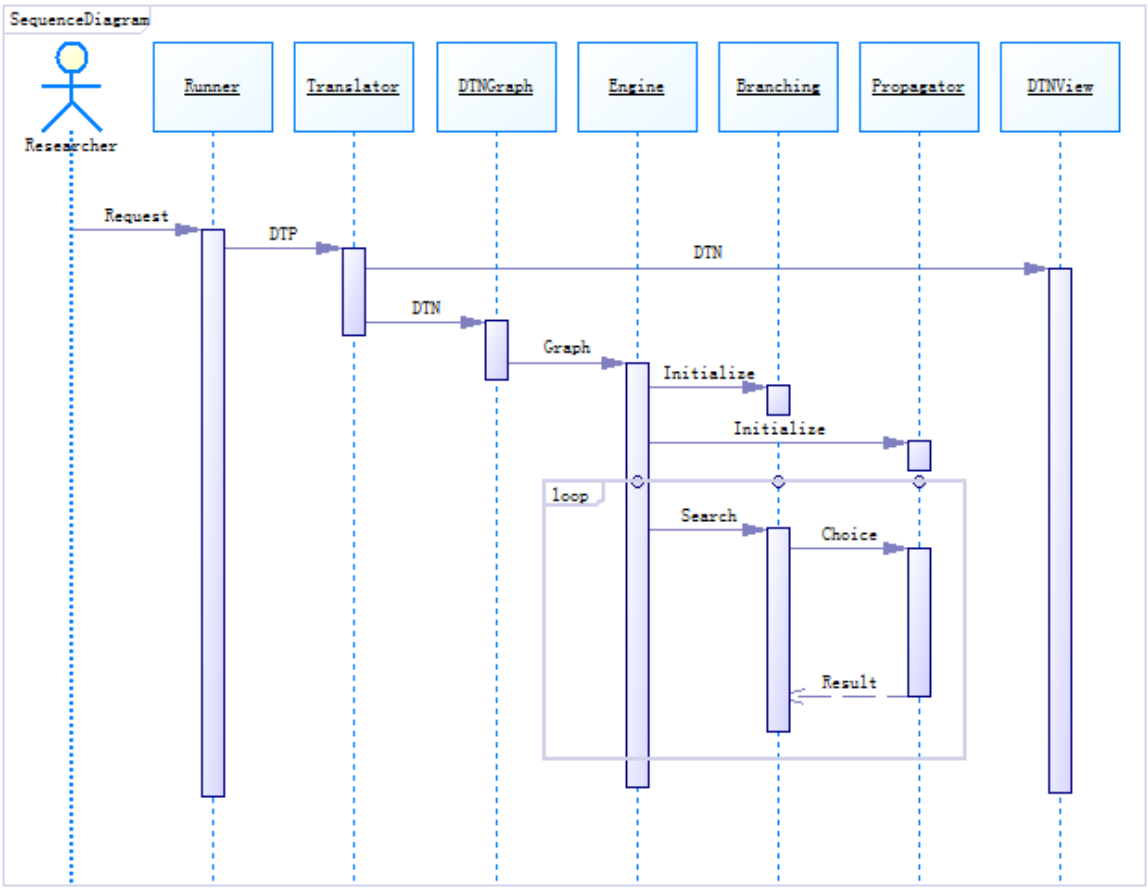


图 4-9 TRSE 时序图

4.3 工具适配

4.3.1 JGraphT 的适配

在对特定 DTP 建立 DTN 时，需要有具备以下图形结构的功能支持：有向图、带权图、多重图和标签图。由于 JGraphT 默认只支持前三者，并且功能较为有限，因此需要对其进行适配处理，即添加对标签图的支持及相关实用数据成员。

4.3.2 Gecode/J 的适配

在 2.3.2 节讲到，本文探讨的 TRSE 系统在求解 DTP 时将基于元约束可满足模型（meta-CSP），然而作者发现，由于 Gecode/J 及其相关文档未指出其对泛型变量的支持，不能直接对元变量进行定义。很明显，Gecode/J 对整型变量的定义是最为直观的，因此作者选用映射表（Map）集合类型来实现索引与元变量的元值之间的映射关系。

第5章　系统测试

5.1　测试用例

本文使用了由 Massimo Idini 和 Claudio Castellini 编写的 DTP 文件生成脚本 DTPgen.pl 来产生随机数据集，以衡量 TRSE 中各种启发式组合的实际运行效率。脚本代码参见附录。

DTPgen 的运行需要定义 5 个参数<k, n, m, L, seed>，其中 k 表示每个析取约束中析取肢的数目，n 表示变量数目，m 表示析取式的数目，L 表示用来限定每个不等式约束中常数项的取值范围的正整数（即限定范围为[-L,L]），seed 表示随机数生成种子。本文主要考察的参数取值为 k=2、n∈{5,10,15,20}、m=r×n(r=1,2,…,14)、L=100、seed=1,101,201,…,4901 的数据集，每个取值组合都生成 50 个 DTP 样本，分别进行运算得到耗费的 CPU 时间，取运行结果的中位数作为该取值组合的代表结果。

本文针对 TRSE 所实现的 3 种启发式技术进行组合测试。所实验测试的组合搭配如下：

- NONE: Base-Branching
- MRV: MRV-Branching
- PRE: Base-Branching + FC-Propagator + RSV-Propagator
- ALL: MRV-Branching + FC-Propagator + RSV-Propagator

需要说明一点的是，TRSE 系统中虽然已经使用了 Spring IoC 对 RSV 和 FC 进行模块分离，但是在实际测试时，为了方便将两者同时启用当作一个测试模块 PRE。

5.2　实验结果与讨论

根据 TRSE 的运行结果，作者绘制多个图表进行比较（以变量数目 n 的不同

赋值为条件），并得到一些相应的结论。

以下数据运行的主要软硬件环境为：Windows 7 32 位操作系统、i3-2350M CPU、4GB DDR3 内存。实验数据的 CPU 时间以 100 秒为上限。

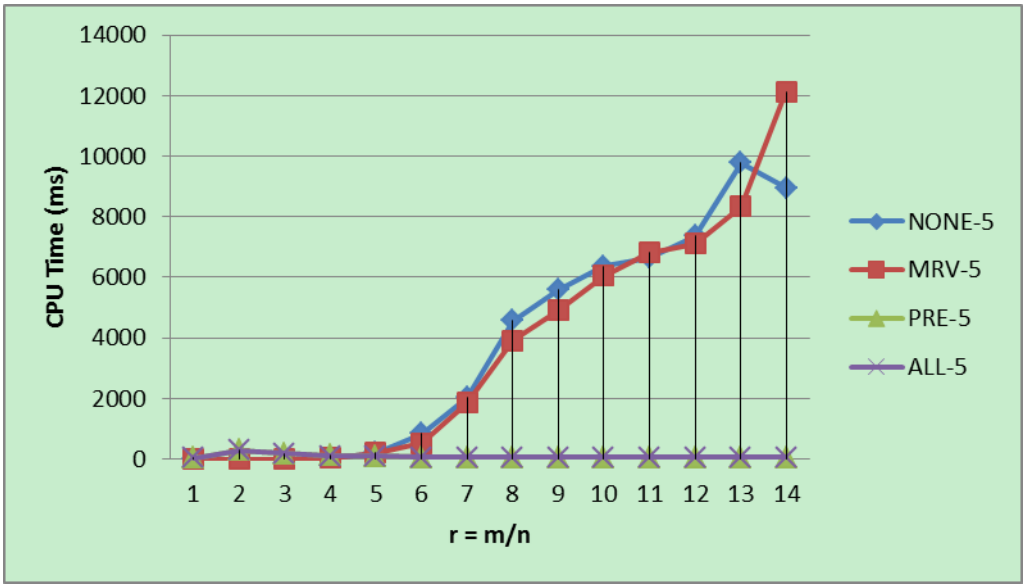


图 5-1 各种启发式组合在 n=5 时的 CPU 时间比较

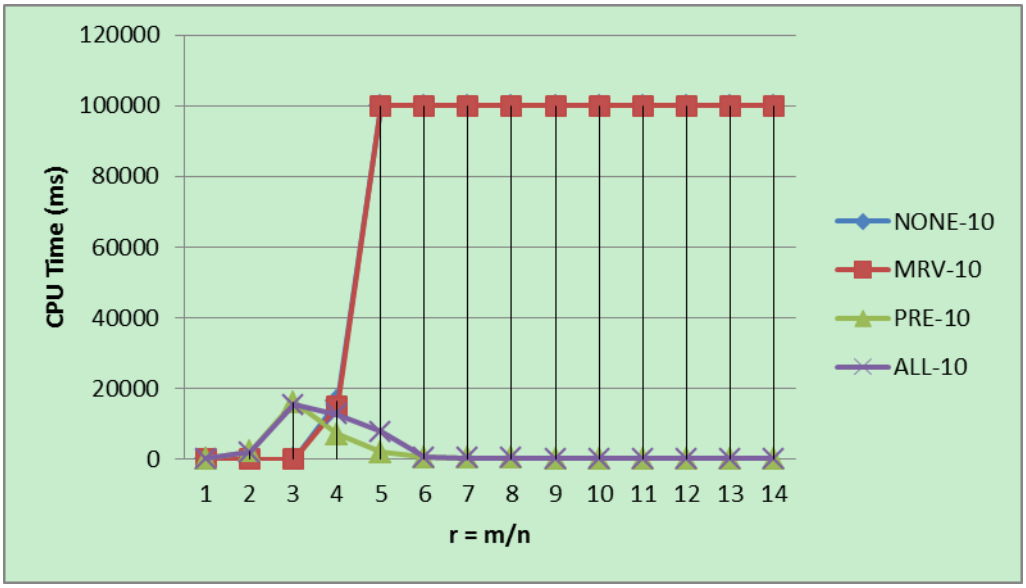


图 5-2 各种启发式组合在 n=10 时的 CPU 时间比较

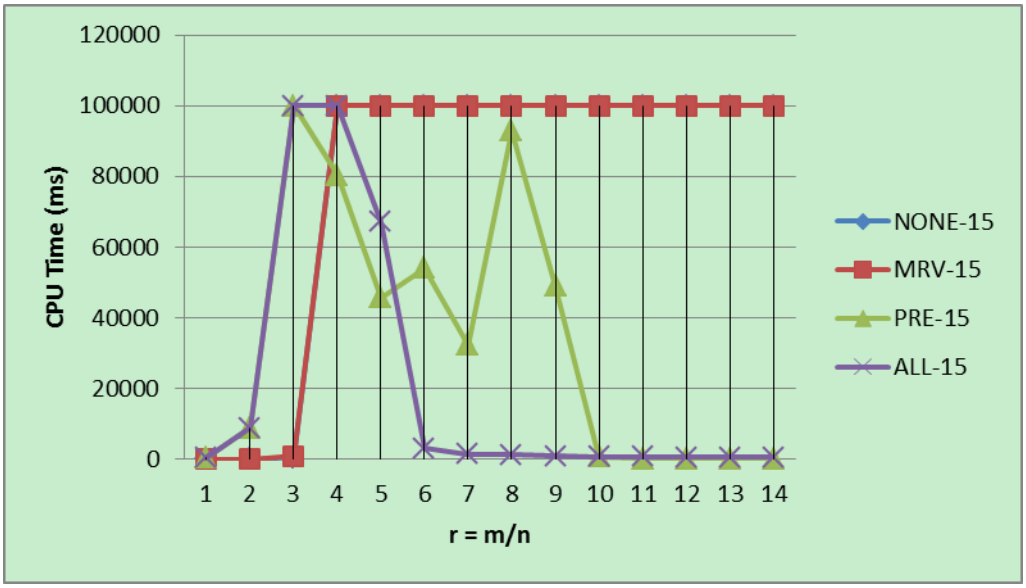


图 5-3 各种启发式组合在 n=15 时的 CPU 时间比较

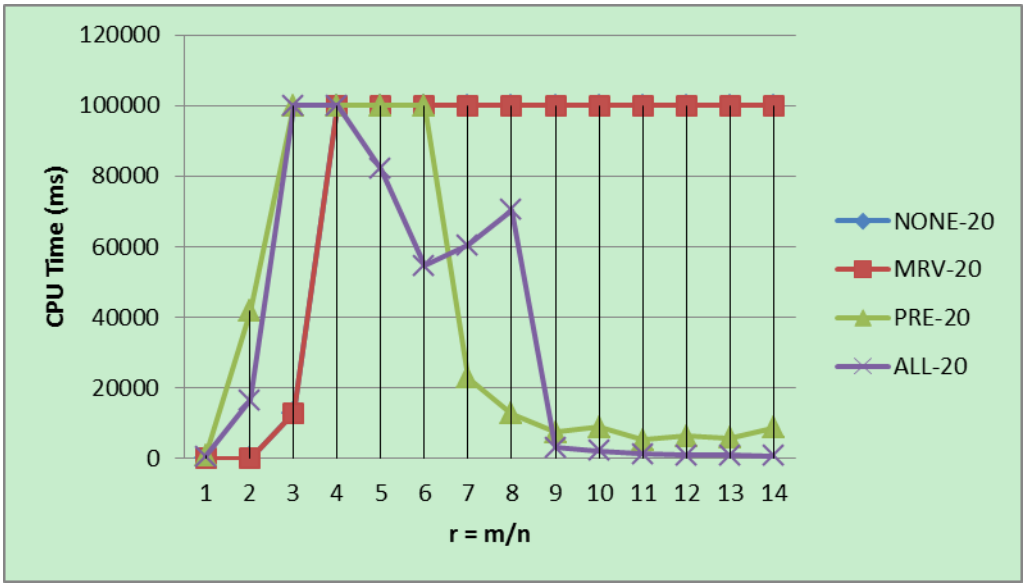


图 5-4 各种启发式组合在 n=20 时的 CPU 时间比较

根据图 5-1~4 的数据走势，作者总结出以下结论：

- 由于设定了参数 k=2，因此单纯使用 MRV 启发式并不能发挥其作用，其效果与 NONE 是一致的，需要与其他启发式进行组合才能发挥其剪枝能

力。

- 在变量数目较少的情况下（n=5 或 10），使用 PRE 组合和 ALL 组合的剪枝效果几乎是一样显著的（即 MRV 的作用不明显）。
- 虽然使用了 FC 启发式和 RSV 启发式的组合在 r=4 之前相比 NONE 组合和 MRV 组合需要耗费更多的 CPU 时间，但是随着 r 的增大，其剪枝效果将变得十分明显。特别是 ALL 组合，其 CPU 时间在 r=6 后几乎就趋于平缓，效率获得了很大的提升。
- 当变量数目较多（n=20）且 r 值较大时，ALL 组合会得到最大的效率提升，充分体现了启发式技术的剪枝魅力。

第6章　总结与展望

智能规划作为人工智能的一个研究方向，一直受到许多研究人员的关注。时态规划作为对经典智能规划的补充，近年来已经取得了很多的研究成果。为了不断适应十分复杂的现实环境，时态规划系统希望取得较高的求解效率。因而，研究人员希望通过优化算法来提高效率，以兴一家之长。

工欲善其事，必先利其器。算法效率的提高只有通过实践才能得以验证。本文探讨实现了基于约束满足模型的时态推理求解环境 TRSE，为研究如何设计和组合启发式以提高析取时态问题求解效率提供了有力的测试环境。

本文是建立在导师的研究工作（独立开发完整推理系统 DTN-DTP）之上，并做了如下创新：

1）使用了国际上最新的通用约束推理工具（瑞典皇家理工学院 Christian Schulte 教授团队开发的最新的约束求解框架 Gecode）进行了重新实现，这样可以充分利用约束推理领域的最新研究成果用于时态推理，实现方法较新颖。

2）本文还使用开源图形建模工具 JgraphT，充分利用其数据结构定义，在数据结构的设计上更有助于系统的扩展和完善。

由于时态规划和推理的核心动力是算法，设计推理系统的目的也仅是为了运行和测试启发式算法，因此未来还有更多更为重要的任务等着我们去实现。

参考文献

[1] 刘越畅. 基于约束的时态推理和时态规划[D]. 广州:中山大学, 2008.

[2] 宋泾舸, 查建中, 陆一平. 智能规划研究综述——一个面向应用的视角[N]. 北京:北京交通大学, 2007.

[3] 丁德路,姜云飞. 智能规划及其应用的研究[J].广州:中山大学, 2002.

[4] 严蔚敏, 吴伟民. 数据结构(C 语言)[M].北京:清华大学出版社, 2009.

[5] Haralick Robert M, Elliott Gordon L. Increasing Tree Search Efficiency for Constraint Satisfaction Problems [J]. Artificial Intelligence, 1980, 14: 263-313.

[6] Dechter Rina, Meiri Itay, Pearl Judea. Temporal Constraint Networks [J]. Artificial Intelligence, 1991, 49: 61-95.

[7] M R Garey and D S Johnson. Computers and intractability: A guide to the theory of NP-completeness [J]. W. H. Freeman, New York, 1979.

[8] Kostas Stergiou, Manolis Koubarakis. Backtracking Algorithms for Disjunctions of Temporal Constraints [J]. In proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), Madison, Wisconsin, July 26-30, 1998: 248-253.

[9] Ioannis Tsamardinos and Martha E Pollack. Efficient solution techniques for disjunctive temporal reasoning problems [J]. Artificial Intelligence, 2003, 151(1-2): 43-89.

[10] O'Madadhain Joshua, Fisher Danyel, Nelson Tom. JUNG [CP/OL]. (2010-1-24). http://jung.sourceforge.net/

[11] Naveh Barak, Sichi John, et al. JGraphT [CP/OL]. (2012-1-20). http://jgrapht.org/.

[12] Schulte Christian. gecode-2.2.0.tar.gz Source Package and

gecode-2.2.0 Reference Documentation [CP/OL]. (2008-8-25). http://www.gecode.org/.

[13] Lagerkvist Mikael, Tack Guido. gecodej-2.2.0.tar.gz Source Package and gecodej-doc-2.2.0.tar.gz, gecodej-javadoc-2.2.0.tar.gz Reference Documentation [CP/OL]. (2008-8-25). http://www.findthatzipfile.com/.

[14] Schulte Christian, Tack Guido, Lagerkvist Mikael Z. Modeling and Programming with Gecode [CP/OL]. (2013-3-13). http://www.gecode.org/doc/4.0.0/MPG.pdf.

[15] Johnson Rod, Hoeller Juergen. Spring Framework [CP/OL]. http://www.springsource.org/

[16] Horstmann Cay S, Cornell Gary. Core Java, Volume I: Fundamentals [M]. Eighth Edition. Pearson Education, 2008.

致 谢

首先，感谢我的导师刘越畅博士，是他将我带入热门的人工智能领域，让我在本科阶段就能接触到如此精彩的计算机科学。刘老师的悉心指导不仅让我学习到更多的专业知识，而且使我的自主学习能力得到了充分的锻炼。在本科阶段，刘老师给予我许多帮助，让我获得了很多珍贵的学习实践机会，令我受益匪浅，相信四年的所学所思在未来的学习工作中必将发挥其正能量。

然后，要感谢在本科阶段教授我知识、给予我帮助的其他老师，没有他们的艰辛付出，我也不能取得今天的成绩。此外，还要感谢我的舍友和同学，在他们的鼓励和支持下，我才能坚持不懈，乐观地去克服各种困难。最后，还要特别感谢我的父母和兄弟姐妹，在我迷茫无助的时候，给予我精神上的莫大支持。

2013 年 5 月于嘉大南 5

附 录

```
#!/bin/sh
k=2
l=100
n=10
while [ $n -le 30 ]
do
    r=1
    while [ $r -le 14 ]
    do
        m=$(expr $r \* $n)
        mkdir "dataset/"$n"_"$r"_"$m
        echo "mkdir dataset/"$n"_"$r"_"$m
        i=1
        j=1
        while [ $i -le 50 ]
        do
            perl DTPgen.pl $k $n $m $l $j > \
                "dataset/"$n"_"$r"_"$m"/"$n"_"$r"_"$m"_"$i".tsat"
            echo "perl DTPgen.pl "$k" "$n" "$m" "$l" "$j" > "\
                "dataset/"$n"_"$r"_"$m"/"$n"_"$r"_"$m"_"$i".tsat"
            i=$(expr $i + 1)
            j=$(expr $j + 100)
        done
        r=$(expr $r + 1)
    done
    n=$(expr $n + 5)
done
```

代码-1 批量 DTP 文件生成脚本 datasetGen.sh

```
#!/usr/bin/perl

# ------------------------------------------------------------------
# DTPgen.pl - generates a random Disjunctive Temporal Problem,
#             given a set of parameters k,n,m,L
# Author     : Massimo Idini
# Updated by : Claudio Castellini
# Date       : Summer 2003
#
# Use DTPgen <k> <n> <m> <L> <seed>, where
#   k     is the number of disjuncts per disjunction (forced to be 2)
#   n     is the number of variables
#   m     is the number of disjunctions
#   L     is the bound on the constant term (usually 100)
#   seed  is the random seed.
# ------------------------------------------------------------------

# parse command line switches

$k =$ARGV[0];
$n =$ARGV[1];
$m =$ARGV[2];
$L =$ARGV[3];
$seed =$ARGV[4];

# check their consistency

($k>=2)      || die "ERROR: only k >= 2 is admitted.\n" ;
($n>=3)      || die "ERROR: n must be greater than 2.\n";
($m>=3)      || die "ERROR: m must be greater than 2.\n";
($L>=0)      || die "ERROR: L cannot be negative.\n";
($#ARGV==4) || die "ERROR: use DTPgen <k> <n> <m> <L> <seed>\n";

# initialise random seed

srand($seed);

# stamp the problem with the problem's parameters

print STDOUT "#\n";
print STDOUT "# A random Disjunctive Temporal Problem\n";
print STDOUT "#\n";
print STDOUT "# k = $k literals per clause,\n";
print STDOUT "# n = $n arithmetic vars,\n";
print STDOUT "# m = $m clauses,\n";
print STDOUT "# L = 100 is the coefficients' range,\n";
print STDOUT "# random seed is $seed\n";
print STDOUT "#\n";
```

```
49
50  # generate m constraints:
51
52  for ( $i=0; $i<$m; $i++ ) {
53      # place an AND after every clause but the last one
54      if ( $i!=0 ) {
55          print STDOUT " ^\n";
56      }
57      # now generate six random numbers a,b,c,d,e,f such that
58      do {
59          $repeat=0;
60          for ( $j=0; $j<$k; $j++ ) {
61          # the variable indexes (a,b) differ,
62          while ( ($a=int(rand()*$n)) == ($b=int(rand()*$n)) ) { }
63          # the constant term c is chosen randomly in [-L,L]
64          $c = int(rand()*$L);
65          $neg = rand();
66          if ($neg>= 0.5) {
67              $c=$c*-1;
68          }
69          $param[$j*3]= $a;
70          $param[$j*3+1]= $b;
71          $param[$j*3+2]= $c;
72        }
73        for ( $h=0; $h<=$k*3-6; $h=$h+3 ) {
74          for ( $g= $h+3; $g<=$k*3-3; $g=$g+3 ){
75            if( ($param[$h]==$param[$g]) && ($param[$h+1]==$param[$g+1])
76                  && ($param[$h+2]==$param[$g+2])) {
77              $repeat=1;
78              last if ($repeat==1);
79            }
80          }
81        }
82      }
83      # finally check that we haven't generated the same relation twice!
84      while ( $repeat==1 );
85
86      # write otu the relations as a binary clause
87      for($p=0; $p<=$k*3-3; $p=$p+3) {
88       if($p>0) { print STDOUT " v "; }
89       print STDOUT "x$param[$p]-x$param[$p+1]<=$param[$p+2]";
90    }
91  }
92
93  print STDOUT "\n";
```

代码-2 单个 DTP 文件生成脚本 DTPgen.pl