UML 时间顺序图的可达性分析*)

龚嘉宇　李宣东　郑国梁

(南京大学计算机软件新技术国家重点实验室　南京210093)

摘　要　对于实时系统来说,UML 顺序图描述了对象之间的交互。对象之间的交互展现了系统行为的场景。本文中,我们针对描述多场景的 UML 顺序图组合中的可达性问题进行研究。尽管这个问题可以转换为相应的时间自动机,然后进行处理,但其转化为之后,状态空间巨大,解决的开销比较大,效率不高。针对部分可达性问题,本文采用更为高效的基于线性规划的解决方案,其思想如下:首先遍历所有到达给定节点的简单路径片断来验证可达性,随后遍历到达给定节点的并且包含所有循环至多一次的路径片断来验证可达性。由于我们并没有遍历所有路径片断,因此用本文的方法判定给定节点的可达性的时候,结果会有三种:可达,不可达和不确定。由于有些循环与可达性是无关的,我们进一步通过识别哪些循环与可达性无关,对算法进行改进。

关键词　实时系统,UML,顺序图,时间约束

Analysis of the Reachability of Timed UML Sequence Diagrams

GONG Jia-Yu　LI Xuan-Dong　ZHENG Guo-Liang

(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093)

Abstract　For real-time systems, UML sequence diagrams describe interaction among objects, which shows the scenarios of system behaviour. In this paper, we discuss about the reachability of the composition of UML sequence diagrams, which can describe multi-scenarios. We can transform this problem to the corresponding timed automata, but the corresponding time automata has a large state space and the cost is large and not efficient. For these reasons, to solve part of the problem of reachability, we adapt a solution based on linear programming for its higher efficiency. The method of the solution is shown as follows: firstly, we traverse all simple path segments that can reach the given node to check the reachability; secondly, we traverse the path segments reaching the given node to check the reachability, such path segments include all the loops at most once. In this paper, when we try to check the reachability of the given node, there may be three results: true, false and uncertain, for we haven't traversed all the path segments. Since some loops has nothing to do with the reachability, we try to distinguish those loops that have no relationship with the reachability in order to improve the algorithm.

Keywords　Real-time system, UML, Sequence diagram, Timing constraints

1　引言

随着软件系统的规模和复杂性的日趋增大,各部件之间的交互活动已成为系统开发过程中规约和建模的重要方面。基于场景的规约(Scenario-Based Specification)用直观、可视的形式描述系统各部件之间的交互活动,在软件开发过程中扮演了越来越重要的角色。

自 ITU-Z120[10]推出后,消息顺序图(Message Sequence Charts, MSCs)逐渐被工业界接受为基于场景的规约语言。MSC 是一种通用的图形化的描述通信协议的符号,可以描述系统的通信框架并能用于寻找设计的错误。

随着 OMG 推出的统一建模语言(Unified Modeling Language, UML)将类似于 MSC 的顺序图(Sequence Diagram)集成其中,类似于 MSC 的基于场景的规约语言被越来越多的软件开发人员认识并使用。顺序图用于描述系统的行为,描述了交互的对象之间的合作关系,而这些交互是对象间的消息交换,并且提供了针对实时系统和分布式系统中通信实体间的消息交换的规约。正如其它的设计和规约的过程一样,UML 顺序图的规约也可能存在错误,于是相应的分析显得尤为重要。

实时系统是一类应用广泛的复杂系统,特别是具有高安全性需求的系统大都是实时系统。时间约束条件是实时系统所要满足的关键需求,在对实时系统各个部件之间的交互活动进行建模的过程中必须要考虑时间因素,从而就形成了基于场景的时间规约(Scenario-Based Specification)。为了对基于场景的时间约束进行描述,目前都是在现有主流基于场景规约描述语言(如 MSC, UML 顺序图)中引入时间约束描述机制。

在本文中,我们描述了一个针对部分情况验证 UML 顺序图的可达性的算法。在接下来的部分,我们简单介绍一下 UML 顺序图和 UML 顺序图的组合(CUD)。一个针对部分情况检查 UML 顺序图的组合的可达性的算法将在第3部分提出。第4部分将给出一个实例分析。

2　带有时间约束的 UML 顺序图和 UML 顺序图的组合

在这个部分中,我们简单介绍带有时间约束的 UML 顺序图和 UML 顺序图的组合。

2.1　UML 顺序图

UML 顺序图描述了交互,交互是对象间的一个影响指定操作或者结果的合作的消息交换的集合,着重于消息流的时间顺序。顺序图存在两个轴:水平轴表示不同的对象;垂直

*)项目基金:国家自然科学基金(6027036,602339291),863计划(2002AA116090),省自然科学基金(BK2002079)。龚嘉宇　硕士,研究方向主要为软件工程,模型检验,形式化方法。李宣东　教授,博士生导师,主要研究方向包括面向对象技术和形式化方法,特别是实时和混成系统的模型验证算法和工具。郑国梁　教授,博士生导师,研究方向主要是软件工程、软件开发环境。

轴表示时间。顺序图中的对象用带有垂直线的矩形框表示，在矩形框内标有类名或对象名。垂直线称为对象的生命线，代表在对象之间的交互作用中该对象的生命周期。时间通常是自上而下增加的，一般只关注时间的前后关系，而不关注具体时间的长短。对象之间的通信消息通过对象生命线之间的箭头表示。当收到消息时，接收对象立即开始执行活动，即对象被激活了。每个消息旁标注消息名和一些控制信息。这里我们仅仅考虑简单的 UML 顺序图，只描述一个场景，不涉及到选择和循环。为了能够描述多场景，我们将考虑可以包括选择和循环的 UML 顺序图的组合。

2.2 带有时间约束的 UML 顺序图

事件通常指的是消息发送、消息接收、对象创建和对象消亡。每个事件都被赋予一个名称代表它发生的时刻。因此时间约束能够被描述成为事件名称的布尔表达式。我们定义时间约束为如下的形式：

$$a\leqslant c_0(e_0-e'_0)+c_1(e_1-e'_1)+\cdots+c_n(e_n-e'_n)\leqslant b$$

其中 $e_0,e'_0,e_1,e'_1,\cdots,e_n,e'_n$ 是事件的名称，a,b 和 $c_0,c_1,\cdots,c_n$ 是实数（b 可能是∞）。

为了分析 UML 顺序图，给出 UML 顺序图的形式化定义如下。

定义1（UML 顺序图，UML sequence diagram） UML 顺序图是一个四元组，可表示为 $D=(O,E,V,C)$。

其中 O 是一个对象的有限集合；E 是一个对应于发送消息、接收消息、创建对象和对象消亡的事件的有限集合；V 是这样的一个有限集合，该集合中的元素都形如(e,e')，这里 e，e' 都在 E 中并且 $e\neq e'$。这代表了 D 中的可视顺序。C 是一个布尔表达式的集合，代表了 D 上的时间约束。

每个 UML 顺序图对应于一个可视顺序，该顺序体现了一对事件 e_1 和 e_2 的顺序并且在如下情况下 e_1 先于 e_2：

— 因果性(Causality)：发送事件 e_1 和其相应的接收事件 e_2。

— 可控性(Controlability)：在同一对象生命线上，e_1 在 e_2 上方出现，并且 e_2 是发送事件。这种关系反映了一个发送事件必须等待其它事件的发生。对于接收事件，接收事件的顺序不可控。

— 先进先出(Fifo order)：接收事件 e_1 和接收事件 e_2 发生在同一条生命线上，并且相应的发送事件 e'_1 和 e'_2 发生在同一个对象的生命线上并且 e'_1 先于 e'_2。

我们用事件序列来表示 UML 顺序图的与时间无关的行为。每个事件序列的形式为 $e_0\hat{}\ e_1\hat{}\cdots\hat{}\ e_m$，表示对于任何 i $(0\leqslant i\leqslant m-1)$，$e_{i+1}$ 发生在 e_i 之后。

定义2（事件序列的运行轨迹，Trace of UML Sequence Diagram） 对于一个顺序图 $D=(O,E,V,C)$，事件序列 $e_1\hat{}\ e_2\hat{}\cdots\hat{}\ e_m$ 是一个有效的运行轨迹当且仅当以下两个条件得到满足：

— 所有 E 中的事件在序列中出现且仅出现一次，也就是对于任何 $i,j(i\neq j,0\leqslant i,j\leqslant m),\{e_1,e_2,\cdots,e_m\}=E$ 且 $e_i\neq e_j$；

— $e_1,e_2,\cdots,e_m$ 满足 V 定义的可视顺序，也就是对于任何 e_i 和 e_j，如果 $e_i<e_j$，那么$\leqslant i<j\leqslant t_j m$。

我们用时间事件序列来表示 UML 顺序图的行为。一个事件时间序列的形式为$(e_0,t_0)\hat{}\ (e_1,t_1)\hat{}\cdots\hat{}\ (e_m,t_m)$，对于任何 $i(0\leqslant i\leqslant m)$，$e_i$ 是事件，t_i 是一个非负实数。它表示 e_0 在系统开始后 t_0 个时间单位发生，e_m 在 e_{m-1} 发生后 t_m 个事件单位后发生。对于任何 $i(0\leqslant i\leqslant m)$，$e_i$ 的发生事件是 $\sum_{j=0}^{i}t_j$。用 $\tau(\sigma)=\sum_{i=0}^{m}t_i$ 来记任何时间事件序列 $\sigma=(e_0,t_0)\hat{}\ (e_1,t_1)\hat{}\cdots\hat{}\ (e_m,t_m)$。

定义3 （时间事件序列，Timed Event Sequence） 一个时间事件序列 $\sigma=(e_0,t_0)\hat{}\ (e_1,t_1)\hat{}\cdots\hat{}\ (e_m,t_m)$ 是一个 UML 顺序图 $D=(O,E,V,C)$ 的行为当且仅当以下条件满足：

— $e_0\hat{}\ e_1\hat{}\cdots\hat{}\ e_m$ 表示 D 的与时间无关的行为，并且

— $t_0,t_1,\cdots,t_m$ 满足 C 所描述的时间约束，也就是对于 C 中任何布尔表达式 $a\leqslant\sum_{i=0}^{n}c_i(f_i-f'_i)\leqslant b$，$a\leqslant c_0\delta_0+c_1\delta_1+\cdots+c_n\delta_n\leqslant b$ 这里对于每个 $i(0\leqslant i\leqslant n)$，如果 $f_i=e_j$ 并且 $f'_i=e_k$，那么

$$\delta_i=\begin{cases}t_k+1+t_k+2+\cdots+t_j & \text{if } j>k\\ -(t_j+1+t_j+2+\cdots+t_k) & \text{if } j<k\end{cases}$$

用 $L(D)$ 表示代表 D 的时间事件序列的集合。

为了描述对于事件发生时刻上的时间约束，我们引入一个特殊的事件 ε，表示系统的开始。对于任何 UML 顺序图 $D=(O,E,V,C)$，并且 $\varepsilon\in E$，$L(D)$ 中任何时间事件序列形式如下：

$$(\varepsilon,0)\hat{}\ (e_1,t_1)\hat{}\ (e_2,t_2)\hat{}\cdots\hat{}\ (e_m,t_m)$$

2.3 UML 顺序图的组合

一个简单的 UML 顺序图只描述了一个场景。为了能够描述多个场景并且对实时系统建模，我们需要考虑 UML 顺序图的组合。UML 标准目前还没有提供 UML 顺序图的组合的标准。我们建议引入高层图(high-level graphs)，它类似于高层 MSCs (High-Level MSCs)，来描述 UML 顺序图的组合。

UML 顺序图的组合能通过一个层次图来描述。图1展现了一个 UML 顺序图的组合，描述了在文[2]中的一个电信系统示例的简单连接建立协议。图1中，有三个简单的 UML 顺序图(D_1,D_2,D_3)和一个高层图(D_4)，高层图描述了三个简单的 UML 顺序图的组合。D_1 描述了连接请求，D_2 描述了成功的连接建立，D_3 描述了不成功的连接建立。obj_1 是服务提供方，obj_2 是本地协议机器，obj_3 是远程协议机器。D_4 描述了 D_1,D_2,D_3 的组合：循环分支表示重复的建立连接的请求，非循环分支表示成功的建立连接。由于定义1中的 C 仅仅描述了一个 UML 顺序图中的事件约束，为了描述在不同 UML 顺序图中的事件之间的时间约束，我们引入了形同 $a\leqslant e-e'\leqslant b$ 的布尔表达式的集合作为补充。

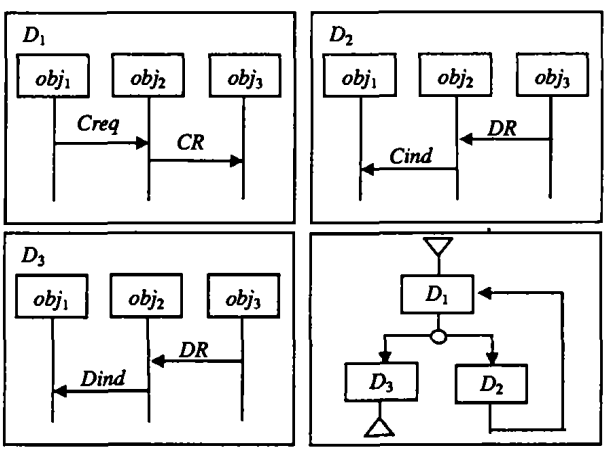


图1 一个 UML 顺序图的组合

定义4（UML 顺序图的组合，Compositions of UML Diagrams，CUD） 一个 UML 顺序图的组合(CUD)是一个五元组 $S=(U,N,succ,ref,M)$。

其中：U 是一个简单 UML 顺序图的集合，且满足对于任

何 $D=(O,E,V,C)\in U$ 和 $D'=(O',E',V',C')\in U$，如果 $D\neq D'$，那么 $E\cap E'=\phi$；$N=\{\top\}\cup I\cup\{\perp\}$是一个有穷的结点集合，能被分成三个部分：包含起始节点的单一集合，中间节点集合，包含终止节点的单一集合；$succ\subset N\times N$ 是反映 N 节点的连接性的关系，N 中的任何一个节点能从起始节点出发到达，N 中的任何一个节点能到达终止节点；$ref:I\vdash B$ 是一个函数关系映射每个中间节点到 U 中的 UML 顺序图；M 是一个有穷的形如 $a\leqslant e-e'\leqslant b$ 的时间约束的集合，这里 e 和 e' 发生在不同的 UML 顺序图中并且$0\leqslant a\leqslant b$(b 可能是∞)。

对于一个 CUD $S=(U,N,succ,ref,M)$来说，一个路径片断(path segment)是一个中间节点的序列，$v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_n$，其中对于任何 $i(2\leqslant i\leqslant n)$满足$(v_{i-1},v_i)\in succ$。一个路径片断是简单的当且仅当所有的节点是唯一的。一条路径是这样的一个路径片断，$v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_n$满足$(\top,v_1)\in succ$ 并且$(v_n,\perp)\in succ$。一个是简单路径片断的路径称为简单路径。对于一个满足$(\top,v_1)\in succ$ 的简单路径片断 $v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_n$ 来说，如果有 $v_i(1\leqslant i\leqslant n)$使得$(v_n,v_i)\in succ$，那么序列 $v_i\hat{}\ v_{i+1}\hat{}\ \cdots\hat{}\ v_n$ 就是一个循环(loop)，而 v_i 就是循环的起始节点(loop-start node)。

3 分析 UML 顺序图可达性的算法

3.1 UML 顺序图的可达性问题

UML 顺序图的可达性问题就是针对其中的某一节点，是否可以找到一条从起始节点出发的路径片断能够到达该节点，并且要满足这条路径片断上的所有时间约束，这里的时间约束不仅是路径片断所经过的每个节点自身的时间约束(定义1中的 C)，而且还包括路径片断中各节点之间的时间约束，即定义6中的 M。

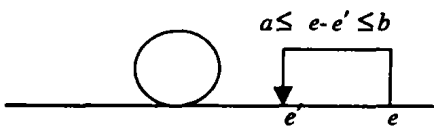
在分析可达性的时候，为了避免在解释与循环相关的时间约束的时候出现不确定性，时间约束不允许发生在循环内外的事件之间，也就是说，对于一个 CUD $S=(U,N,succ,ref,M)$来说，所有在 M 中形如 $a\leqslant e-e'\leqslant b$ 的时间约束必须满足下列条件：

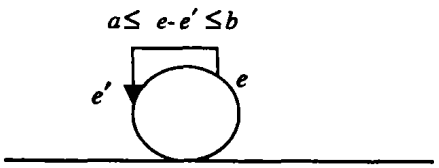
－对于任何循环 $v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$，如果 e 发生在 $ref(v_i)(1\leqslant i\leqslant m)$中并且 e'不发生在任何 $ref(v_j)(1\leqslant j\leqslant i)$中，那么就不存在简单路径片断 $v'_1\hat{}\ v'_2\hat{}\ \cdots v'_n\hat{}$ 满足 e'发生在 $ref(v'_k)(1\leqslant k\leqslant n)$中，并且 $v'_n=v_1$；

－ 对于任何循环 $v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$，如果 e'发生在 $ref(v_i)(1\leqslant i\leqslant m)$中并且 e 不发生在任何 $ref(v_j)(i\leqslant j\leqslant m)$中，那么就不存在简单路径片断 $v'_1\hat{}\ v'_2\hat{}\ \cdots\hat{}\ v'_n$ 满足 $v_1=v'_1$，e 发生在 $ref(v'_n)$中，并且 e'不发生在任何 $ref(v'_k)(1\leqslant k\leqslant n)$中；

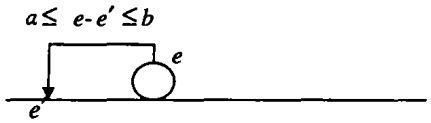
－对于任何循环 $v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$，如果 e 发生在 $ref(v_i)$中并且 e'发生在任何 $ref(v_j)$中，那么$1\leqslant j\leqslant i\leqslant m$。

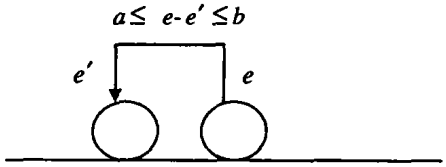
也就是允许：





不允许：





我们通过局部语义(local semantics)来解释 CUD 中的时间约束：在某一时刻选择一条路径，并独立于所选择的路径的其它分支路径来分析它的时间需求。定义一个 CUD S 行为为代表组成 S 的 UML 顺序图的行为的时间事件序列。用 $\hat{}$ 来表示序列的连接。

定义5(CUD 的时间事件序列，timed event sequence of CUD) 对于一个 CUD $S=(U,N,succ,ref,M)$来说，一个时间事件序列 $\sigma=(e_0,t_0)\hat{}\ (e_1,t_1)\hat{}\ \cdots\hat{}\ (e_n,t_n)$表示了 S 的一个行为当且仅当以下条件满足：

－存在一条路径 $v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$满足 $\sigma=\sigma_1\hat{}\ \sigma_2\hat{}\ \cdots\hat{}\ \sigma_m$，这里对于任何 $i(1\leqslant i\leqslant m)$，$\sigma_i$ 是 $ref(v_i)$的一个行为；

－σ 满足 M 中的任何布尔表达式，也就是说对于任何 $a\leqslant f-f'\leqslant b\in M$，对于任何 $i,j(0\leqslant i\leqslant j\leqslant n)$满足如果 $f'=e_i$，$f=e_j$，那么就不存在任何 $k(i<k<j)$满足 $f=e_k\vee f'=e_k$，$a\leqslant t_{i+1}+t_{i+2}+\cdots t_j\leqslant b$

定义6(CUD 的时间事件序列的集合，the set of timed event sequence of CUD) 对于任何 CUD $S=(U,N,succ,ref,M)$来说，对于任何路径片断 $\rho=v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$，用 $L(\rho)$表示所有形如 $\sigma=(e_0,t_0)\hat{}\ (e_1,t_1)\hat{}\ \cdots\hat{}\ (e_n,t_n)$的时间事件序列的集合并且满足：

－$\sigma=\sigma_1\hat{}\ \sigma_2\hat{}\ \cdots\hat{}\ \sigma_m$，这里对于任何 $i(1\leqslant i\leqslant m)$，$\sigma_i$ 是 $ref(v_i)$的一个行为；

－σ 满足 M 中的任何布尔表达式，也就是说对于任何 $a\leqslant f-f'\leqslant b\in M$，对于任何 $i,j(0\leqslant i\leqslant j\leqslant n)$满足如果 $f'=e_i$，$f=e_j$ 那么就不存在任何 $k(i<k<j)$满足 $f=e_k\vee f'=e_k$，$a\leqslant t_{i+1}+t_{i+2}+\cdots t_j\leqslant b$

对于一个 CUD $S=(U,N,succ,ref,M)$，一条路径 ρ 是时间一致的当且仅当 $L(\rho)\neq\phi$，假设有一个节点 v_m，要验证该节点是否可达，就是验证是否存在一条从$\top$出发能够到达 v_m 的路径片断$\top\hat{}\ v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$ 并且该路径片断是时间一致的，也就是 $L(\rho)\neq\phi$。假定对于任何 $i(1\leqslant i\leqslant m)$，有 $ref(v_j)=(O_i,E_i,V_i,C_i)$。从定义3和定义6可知，所有的时间事件序列 $\sigma\in L(\rho)$形式为 $\sigma_1\hat{}\ \sigma_2\hat{}\ \cdots\hat{}\ \sigma_m$，这里对于任何 $i(1\leqslant i\leqslant m)$，$i=(e_{i1},t_{i1})\hat{}\ (e_{i2},t_{i2})\hat{}\ \cdots\hat{}\ (e_{imi},t_{imi})$。而且，所有 $t_{ij}(1\leqslant i\leqslant m,1\leqslant j\leqslant n_i)$必须满足 M 和 $C_i(1\leqslant i\leqslant m)$中的时间约束。

3.2 基于线性规划的解决方案

$S=(U,N,succ,ref,M)$是一个 CUD，$\rho=v_1\hat{}\ v_2\hat{}\ \cdots\hat{}\ v_m$ 是从起始节点出发到达节点 v_m 一条有穷路径片断。设对于任何 i $(1\leqslant i\leqslant m)$都有 $ref(v_i)=(O_i,E_i,V_i,C_i)$。从定义3和定义6可知，所有的时间事件序列 $\sigma\in L(\rho)$形式为 $\sigma_1\hat{}\ \sigma_2\hat{}\ \cdots\hat{}\ \sigma_m$，这里对于任何 $i(1\leqslant i\leqslant m)$，$\sigma_i=(e_{i1},t_{i1})\hat{}\ (e_{i2},t_{i2})\hat{}\ \cdots\hat{}\ (e_{imi},t_{imi})$。而且，所有 $t_{ij}(1\leqslant i\leqslant m,1\leqslant j\leqslant n_i)$必须满足 M 和 C_i $(1\leqslant i\leqslant m)$中的时间约束。这个时间约束的集合构成了一个线性不等式的集合。因此我们能通过这个线性不等式组是否有解来判定是否 $L(\rho)\neq\phi$，这个可以通过线性规划解决。我们

也可以用自动机来分析这样的问题。UML 顺序图可以转换为全局的时间自动机(Timed Automata),因此相应的问题也能转化为时间自动机进行处理。但是转化为时间自动机的时候,状态空间巨大,解决的开销比较大,效率不高,因此我们采用更为高效的基于线性规划的解决方案。

3.3 分析 UML 顺序图可达性的算法

3.3.1 *算法描述* 这部分解决可达性问题的算法的思想如下:首先遍历所有到达给定节点的简单路径片断来验证可达性,随后遍历到达给定节点的并且包含所有循环至多一次的路径片断来验证可达性。由于我们并没有遍历所有路径片断,因此用本文的方法判定给定节点的可达性的时候,结果会有三种:可达、不可达和不确定。但是有些循环与可达性是无关的,我们进一步通过识别哪些循环与可达性无关,对算法进行改进。

对于从起点到给定节点的简单路径片断,也就是有穷路径片断,可以通过深度优先搜索找出,并进行检查。算法如图2所示。

```
currentpath:=⟨⊤⟩;isReachable := uncertain;
repeat
  node:= the last node of currentpath;
  if node has no new successive node
  then delete the last node of currentpath
  else begin
         node:=a new successive node of node;
         if node=v then
          begin
            if the path P corresponding to currentpath is such that L
            (ρ)≠ϕ
            then begin
                   isReachable:= true;return true;
                 end;
          end;
         else if node ≠⊥ and node is not in currentpath
              then append node to currentpath;
       end
until currentpath=⟨⟩;
```

图2 检查简单路径片断

对于带有循环的路径片断,简单起见,仅仅针对路径片断中每个循环至多出现一次进行分析。用深度优先搜索找出所有的每个循环至多出现一次的路径片断,然后进行检查。

对于一个给定的 CUD $S=(U,N,succ,ref,M)$和一个给定的节点 v,我们给出一个该节点是否可达的算法。这个算法中的主要数据结构包括一个节点的表 *currentpath* 用于记录当前路径,和一个集合 *loopset* 用于记录循环。

首先,查找所有的循环(如图3所示)并把循环放入 *loopset* 中。如果遍历简单路径片断无法验证给定节点的可达性,并且 *loopset* 为空,那么就可以判定该节点不可达,因为除了简单路径片断外不存在其余的路径片断到达该节点,而遍历简单路径片断无法验证该节点可达。

如果 *loopset* 不为空,采用深度优先搜索的方法遍历所有到达 v 的路径片断(如图4所示),这样的路径片断必须是所有的循环至多出现一次的。将 *currentpath* 初始化为仅包含起始节点,并将 *loopset* 中的所有循环标记为 *unused*。在深度优先搜索的时候,当前节点为 *currentpath* 中的最后一个节点,如果当前节点不存在新的后继节点,则删除当前节点,并且如果删除的当前节点破坏了 *currentpath* 中存在的循环,将相应的循环在 *loopset* 中标记为 *unused*。如果遇到的新的后继节点是指定结点 v,那么根据 *currentpath* 中保存的路径片断来检查 v 是否可达。如果遇到的新的后继节点已经在 *currentpath* 中,那么检查这样的节点是否仅仅在 *currentpath* 中出现一次。如果是的话,把这个节点加入 *currentpath* 中,并且将这两个相同结点之间的循环在 *loopset* 中标志为 *used*。如果这样的节点在 *currentpath* 中出现不仅一次,则查找可能加入的新循环是否在 *loopset* 中标记为 *unused*,若是则把这个节点加入 *currentpath* 中,并且将这最后两个相同结点之间的循环在 *loopset* 中标志为 *used*。这样直至 *currentpath* 为空。如果此时依然无法验证节点 v 可达,那么我们就认为节点 v 的可达性不确定。

```
currentpath:=⟨T⟩;loopset:=ϕ;
repeat
  node:=the last node of currentpath;
  if node has no new successive node
  then delete the last node of currentpath
  else begin
         node:= a new successive node of node;
         if node is in currentpath then put the corresponding loop into
         loopset;
         if node≠⊥ and node is not in currentpath
         then append node to currentpath;
       end
until currentpath =⟨⟩ ;
if loopset =ϕ and isReachable = uncertain
then begin
       isReachable:= false; return false;
       end;
```

图3 查找所有的循环

```
currentpath:=⟨T⟩;set all loops in loopset unused;
repeat
  node:=the last node of currentpath;
  if node has no new successive node
  then begin
         if the last node of currentpath occurs in currentpath more
         than once
         then begin
                lnode:= the nearest node same as the last node of
                currentpath except itself;
                set the loop between lnode and node unused in
                loopset;
              end
              delete the last node of currentpath;
       end
  else begin
         node:=a new successive node of node;
         if node=v
         then if the path P corresponding to currentpath is such that L
              (P)≠ϕ
              then begin
                     isReachable:= true; return true;
                   end
              if node ≠⊥ and node is not in currentpath
              then append node to currentpath;
              if node has appeared in currentpath only once
              then beg (in
                        append node to currentpath;
                        find the loop occurs in currentpath that starts
                        from node;
                        mark that loop used in loopset;
                   end
              else begin
                     lnode := the nearest node occurs in currentpath
                     same as node
                     if the loop starting from lnode to node in current-
                     path is unused in loopset
                     then begin
                            append node to currentpath;
                            find the loop occurs in currentpath that
                            starts from node;
                            mark that loop used in loopset;
                          end
                   end
       end
until currentpath =⟨⟩;
if isReachable =uncertain
then return uncertain;
```

图4 遍历所有到达 v 的路径片断,且其所有的循环至多出现一次并检查之

我们可以对上述算法进行改进,因为有些循环是与可达性无关的,因此在遍历到达给定节点的路径片断的时候,这些循环不需要出现在路径片断中。

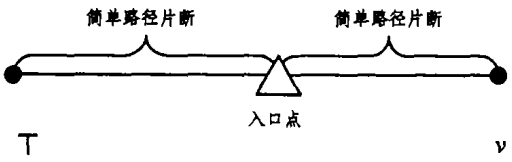
定理1 在检查 CUD 中某个节点可达性的时候,如果存

在这样的循环，那么这样的循环是与可达性无关的，在遍历的过程中是不需要出现在路径片断中的。这类循环满足这样的条件：在所有能到达给定节点的路径片断上，不存在 $a \leqslant e-e' \leqslant b \in M$，并且包含 e 和 e' 的节点都出现在路径片断中，并且循环的入口点在包含 e 和 e' 的节点之间。

证明：对于一个给定的 CUD $S=(U,N,succ,ref,M)$ 和一个给定的节点 v，简单起见，不妨假设 $\forall D \in U$，D 总满足其内部的时间约束 C，否则的话，任何经过不满足 C 的 D 的路径或路径片断必定不是时间一致的。考虑如下情况，对于一条到达给定节点 v 的路径片断 ρ，如果 $L(\rho)=\phi$，那么必定是该路径片断上的时间约束条件 M 没有满足。于是考虑添加循环，目的便在于使路径片断中的时间约束条件 M 被满足。如果在某一路径片断上添加循环，而不存在 $a \leqslant e-e' \leqslant b \in M$，并且包含 e 和 e' 的节点都出现在该路径片断中，并且添加的循环的入口点在包含 e 和 e' 的节点之间，那么所加入的循环将不会对我们试图满足 M 的工作产生任何影响，因此这样的循环是没有必要加入的。 □

我们采用如下方法查找这样的循环。

在查找完所有循环的基础上，用深度优先搜索寻找从起点到某一入口点的简单路径片断，接着再从这个入口点出发，用深度优先搜索查找到给定节点的简单路径片断，如果在所有到达给定节点的路径片断中不存在任何 $a \leqslant e-e' \leqslant b \in M$，并且包含 e 和 e' 的节点都出现在路径片断中，使得这个入口点在包含 e 和 e' 的节点之间，那么这样的循环在寻找每个循环至多一次的路径片断的过程中不必加入。两次深度优先搜索如下所示。



在改进算法的时候，我们对遍历时可能经过的循环进行筛选。判定哪些循环是与可达性无关的，其算法的主要数据结构包括两个节点的表 *currentpath1*，*currentpath2* 用于记录当前路径，和两个集合 *loopset*，*loopsetA* 用于记录循环。其中 *loopset* 已经记录了查找到的所有循环。

我们采用两次深度优先搜索来查找这样的循环。将 *currentpath1* 初始化为仅包含起始节点，然后当前节点 *node1* 为 *currentpath1* 中的最后一个节点，如果当前节点 *node1* 不存在新的后继节点，则删除当前节点。否则的话如果当前节点的新的后继节点为循环的入口点，那么将 *currentpath2* 初始化为仅包含此后继节点。然后当前节点 *node2* 为 *currentpath2* 中的最后一个节点，如果当前节点 *node2* 不存在新的后继节点，则删除 *node2*。否则当前节点 *node2* 的后继节点为给定节点 v，那么对于每一个 $a \leqslant e-e' \leqslant b \in M$，如果包含 e 的节点在 *currentpath2* 中且包含 e' 的节点在 *currentpath1* 中，那么把该入口点对应的循环放入 *loopsetA* 中。如果当前节点 *node2* 的后继节点不为终点并且不存在于 *currentpath2* 中，那么把该后继节点加入 *currentpath2* 中，直至 *currentpath* 为空。如果 *node1* 的新的后继节点不为终点，不为 v，并且不在 *currentpath1* 中，则把该后继节点加入 *currentpath1* 中。这样直至 *currentpath1* 为空。这样所得到的 *loopsetA* 中的循环就是在检查可达性时候可以加入的循环，而此外的循环则与可达性无关，因此可以不予考虑。如果遍历简单路径片断无法验证给定节点的可达性，并且 *loopsetA* 为空，那么我们就可以判定该节点不可达，因为除了简单路径片断外不存在其余的与可达性有关的路径片断到达该节点，而遍历简单路径片断无法验证该节点可达。算法如图5所示。

```
currentpath1:=⟨T⟩;loopsetA :=ϕ;
repeat
  node1:= the last node of currentpath1;
  if node1 has no new successive node
  then delete the last node of currentpath1
  else begin
       if node1 is the entry point of a loop
       then begin
         currentpath2:=⟨node1⟩;
         repeat
           node2 :=the last node of currentpath2 ;
           if node2 has no new successive node
           then delete the last node of currentpath2
           else begin
             node2:=a new successive node of node2;
             if node2 = v
  then for every a≤e−e′≤b of M do
       begin
         nodea := the node including e;
         nodeb := the node including e′;
         if nodea in currentpath2 and nodeb in currentpath1
         then add the loop with entry point node2 into loopsetA;
       end
             else if node2≠⊥and node2 is not in currentpath2
                  then append node2 to currentpath2;
           end
         until currentpath2=⟨⟩;
         end
       if node1≠⊥ and node1≠v and node1 is not in currentpath1
       then append node1 to currentpath1;
     end
until currentpath1=⟨⟩;
if loopsetA=ϕ and isReachable = uncertain
then begin
       isReachable:= false;return false;
       end;
```

图5 查找可以被加到路径片断中的所有循环

3.3.2 *算法分析* 下面，我们分析这个算法的复杂度和开销。这个算法基于深度优先搜索。对于查找到达给定节点的每个循环至多出现一次的路径片断，复杂度与到达给定节点的每个循环至多出现一次的最长路径片断成正比，空间复杂度是 O(*longest-path*)，也就是 $space=C1*longest\_path+C2$。在这个算法中，每当检查对应于 *currentpath* 的路径片断是否是时间一致的，就要求解一个线性规划。因此，最坏的情况下，需要求解的线性规划的数目等于所有到达给定节点的每个循环至多出现一次的路径片断的数目。如果检查所有的这样的路径片断而不加以选择将使得整个开销过于庞大，因此在改进算法中，为了减少开销，我们要判断哪些循环是与可达性无关的。这样，总的需要使用线性规划进行检查的路径片断的数目会有所降低。尽管如此，对于复杂的情况，效率还是不够高，因此需要进一步改进算法，降低复杂度。

4 实例分析

我们给出一个 ATM 的实例。高层的 UML 顺序图和该 CUD 中涉及到的部分简单 UML 顺序图如图6所示。不同 UML 顺序图中的事件间的时间约束如下：

$\{0 \leqslant cancel'\text{-}req\_pin \leqslant 4\}$，$\{5 \leqslant enter\_pin'\text{-}req\_pin \leqslant 60\}$，

$\{0 \leqslant valid'\text{-}verify \leqslant T_1\}$，$\{return\text{-}card\text{-}verify \geqslant T_1\}$，$\{0 \leqslant invalid'\text{-}verify \leqslant T_1\}$，

$\{q_1 \leqslant amt\_approved'\text{-}approve\_amt \leqslant q_2\}$，$\{q_1 \leqslant not\_approved'\text{-}approve\_amt \leqslant q_2\}$

$\{w_1 \leqslant give\_money'\text{-}ent\_amount \leqslant w_2\}$，$\{w_1 \leqslant not\_Possi-$

$ble' - ent\_amount \leqslant w_2\}$

现在我们仅仅考虑图中虚线框内的部分。假设我们已经能到达 ProcessPin，现在考虑能否到达 EndTrans，作为示例，简单起见，仅仅考虑能否从 ProcessPin 到达 EndTrans。

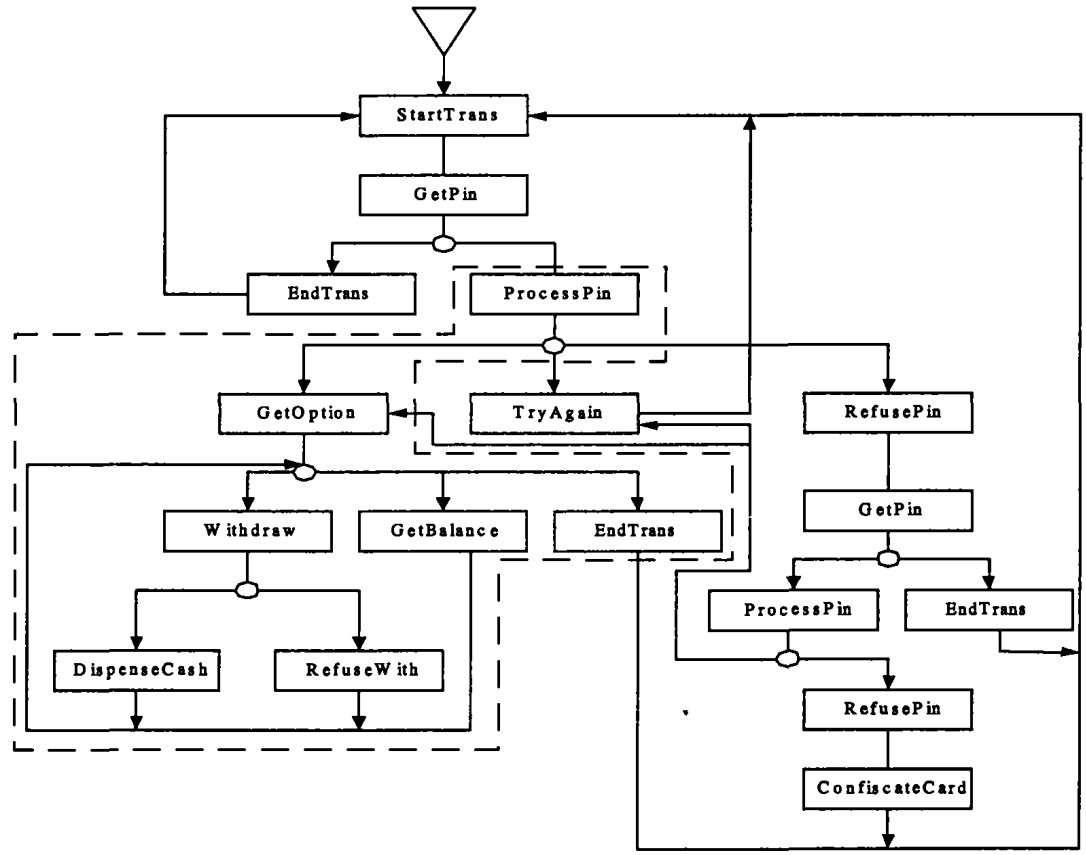
除了上述的这些时间约束外，我们假设每个 ATM 用户的通信时间延迟为[1,2]秒，反之亦然，每条垂直轴的默认延迟为[0,1]秒。为了简单起见，所有的延迟没有在图上标出。

取 $T_1=20, q_1=0, q_2=\infty, w_1=0$ 和 $w_2=\infty$。通过路径片断 ProcessPin ^ GetOption ^ EndTrans 进行验证，节点 EndTrans 不可达因为时间约束没有满足。而通过遍历包含循环的路径片断 ProcessPin ^ GetOption ^ WithDarw ^ DispenseCash ^ GetOption ^ EndTrans，节点 EndTrans 可达。因此该节点可达。

结束语 在本文中，我们研究了 UML 顺序图的可达性。首先我们考虑了更一般的时间约束，然后我们提出了扩充的 UML 顺序图来描述多个场景，并且给出一个针对部分情况验证可达性的算法。

在文[1～3,6,7]中，给出了一些针对带时间约束的 MSC 及其组合和 UML 顺序图进行一致性分析的算法和工具。在文[6]中，UML 顺序图被添加上循环和分支来用于实时系统的建模，并考虑了时间一致性的分析问题：时间约束仅以 $a \leqslant e-e' \leqslant b$ 的简单形式出现，并且从不跨越循环。本文的工作着重于 UML 顺序图的可达性分析，但仅仅针对通过遍历包含所有循环至多一次循环的路径片断来判定给定节点是否可达，而针对更为普遍的情况解决方案还在探索中。

以后的工作将着重于针对更为普遍的问题进一步解决 UML 顺序图可达性问题，利用开发的工具做些实例研究，并扩充工具以分析更多 UML 图。



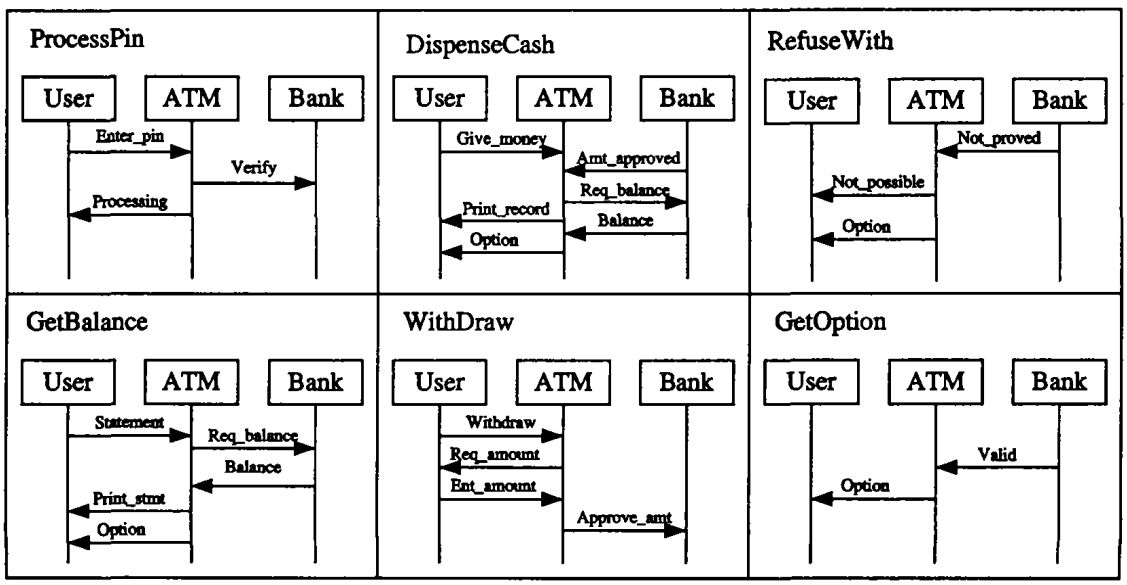


图6 一个 ATM 实例及其相关 UML 顺序图

参 考 文 献

1 Alur R, Holzmann G J, Peled D. An Analyzer for Message Sequence Charts. Software-Concepts and Tools, 1996, 17: 70～77
2 Ben-Abdallah H, Leue S. Expressing and analyzing timing constraints in message sequence chart specifications: [Technical Report 97-04]. Department of Electrical & Computer Engineering, University of Waterloo
3 Ben-Abdallah H, Leue S. Timing Constraints in Message Sequence Chart Specifications. In: Formal Description Techniques X, Proc. of the Tenth Intl. Conf. on Formal Description Techniques FORTE/PSTV'97, Osaka, Japan, Chapman & Hall, 1997
4 Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Addison-Wesley, 1998
5 France R, Evans A, Lano K, et al. The UML as a formal modeling notation. Computer Standards & Interfaces, 1998, 19: 325～334
6 Seemann J, WvG J. Extension of UML Sequence Diagrams for Real-Time Systems. In: Proc. Intl. UML Workshop, Lecture Notes in Computer Science, Springer, 1998
7 Li Xuandong, Lilius J. Timing Aanlysis of Message Sequence Charts: [TUCS Technical Report 255]. Turku Centre for Computer Science, Finland, March 1999
8 Li Xuandong, Lilius J. Timing Analysis of UML Sequence Diagrams. In: UML 99 - The Unified Modeling Language. Lecture Notes in Computer Science 1723, Springer, 1999. 661～674
9 Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Guide. Addison-Wesley, 1998
10 ITU-T Recommendation Z. 120. Message Sequence Chart (MSC). March 1993
11 Muscholl A, Peled D, Su Zhendong. Deciding Properties for Message Sequence Charts
12 Levin V, Peled D. Verification of Message Sequence Charts via Template Matching
13 A Toolset of Requirement Engineering using Message Sequence Charts. Jan. 1998

(上接第146页)

车辆自动导航系统中的一个重要部分就是车辆的行驶路线的设计。图2和图3两幅图片是在两维平面上 GIS 中车辆行驶路线设计的结果。其中图2是采用复合聚类分析方法在 MATLAB 中实现的结果；图3是采用传统的基于密度的方法所得的聚类结果。由这两幅图，我们可以得出结论：复合聚类分析方法比传统的基于密度的方法设计的行车路线要清晰明朗得多。

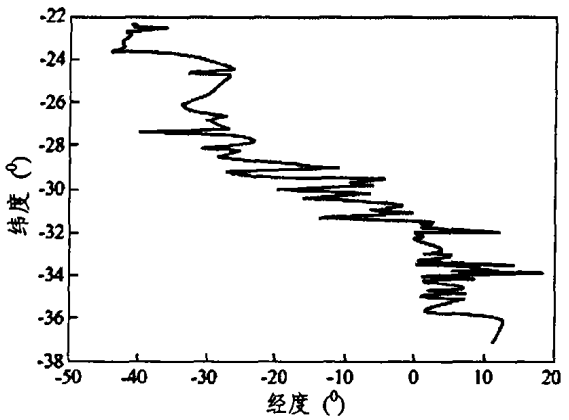


图3 密度方法的聚类结果

3 DM 和 GIS 在车辆自动导航系统中的应用

本文的实例数据来自于 GPS，通过 SQL SERVER 建立专门的数据库，见图4。对数据进行存储、组织和管理。利用 SQL SERVER 建立数据库时，设定相应的规则，保证数据的有效性、安全性和完整性。

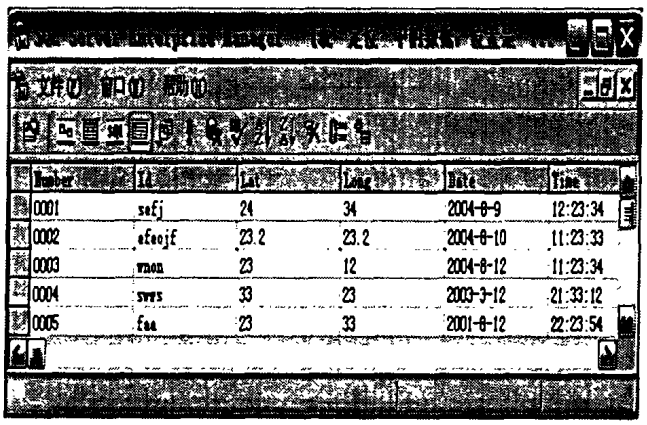


图4 SQL SERVER 建立专门的数据库

这个数据库就包含1张表，表中的每条记录包括5个字段，分别为 Number(编号)、Id(车辆名称)、Lat(纬度坐标)、Long(经度坐标)、Date(日期)、Time(时间)，这样每条记录都完整地记录着某辆车在某一个时刻的确切的位置。所用的地理信息系统的软件为 MapInfo，在 MapInfo 中调用 SQL SERVER 中的数据，并且利用 MapInfo 中提供的工具去除不必要的数据，对空间数据进行处理后，可以将数据库中包含的所有的车辆的位置，描到电子地图的相应地点。由于数据量非常的庞大，因此需要采用 DM 的方法从数据库中提取有价值的信息。我们就以自动导航系统中车辆行驶路线的设计为例来说明一下 DM 是如何应用在 GIS 中的。主要的步骤如下：

1) 从卫星上得到所有加入到自动导航系统中的车辆的分布数据。

2) 选取样本数据，例如，选择任意的两个城市，并导出该区域所包含的数据。

3) 用我们前面所介绍的复合聚类算法进行聚类分析，提取能确定最优路线的数据，最终将所得数据所确定的路线，在地图中以描点的方式显示出来。

因为从 GPS 得来的自动导航系统的车辆分布数据是杂乱无章的、随机的，所以通过测试数据量、样本排序将数据进行预处理，再进行区域分割，经多次试验证明 k 的选取与 n 有关，至少保证每个 μ_j 内样本点数大于10，因为，如果样本点太少，聚类效果不明显；反之，如果样本点数太多的话，设计的路线将会失真。我们取的两个城市之间的样本点为1908，取 $k=120$，试验证明，此时的聚类效果最好。

结论 我们在本文中提出了复合聚类的方法，这种方法在初始的时候设定多个聚类中心，这样的初始中心在数据的空间分布上是很广泛的，具有多样性的特点。这种特点使得最初的聚类基本上能保证每个小区域 μ_i 有一个密度中心，然后再根据适当的准则在小区域 μ_i 找出几个子中心，删除冗余数据，再计算这个区域 μ_i 的平均密度中心，来修正原来的密度中心。事实证明，这种方法在基于 GIS 的自动导航系统中非常有效。

参 考 文 献

1 朱明. 数据挖掘[M]. 合肥：中国科学技术大学出版社，2002
2 Wang X Z. Data Mining and Knowledge Discovery for Process Monitoring and Control[M]. Spring-Verlag London limited, 1999
3 Liu Bing. Knowledge Discovery and Data Mining. 21世纪青年科学论坛，2001, 19(6): 70～74
4 Han Jiawei. Micheline Kamber. Data Mining: Concepts and Techniques. Morgan Kufnamm Publishers, Aug. 2000
5 Business Objects Ltd. Growth in Decision Supports System. Database&Network Jounal, 1998, 28(1): 3～4
6 朱勇华，郜淑彩，孙福玉. 应用数理统计[M]. 武汉：武汉水利电力大学出版社，1999. 362～376
7 龚健雅. 当代 GIS 的若干理论与技术. 武汉测绘科技大学出版社，1999