文章编号：0253-374X(2017)06-0922-08 DOI:10.11908/j.issn.0253-374x.2017.06.021

可缩放矢量格式地图图层定义及模糊查询模型

杜庆峰，钱浩然
(同济大学 软件学院，上海 201804)

摘要：根据图层定义，提出了针对可缩放矢量图形(SVG)格式地图的模糊查询模型. 通过构建模糊查询词库来储存模糊查询相关词和模糊查询匹配模式. 在此基础上，提出了分词子算法，将查询语句分解为词串. 据此给出了语义分析子算法，将分词子算法产生的词串转化成模糊查询匹配模式，并与SVG格式地图相关层(一层或多层)中的元素关联以实现模糊查询. 最后实验验证证明了该模型的有效性.

关键词：可缩放矢量图形(SVG)格式地图；图层；词库；分词；语义分析
中图分类号：TP399 **文献标志码**：A

Fuzzy Query for Scalable Vector Graphics Format Map Based on Layer Definition

DU Qingfeng, *QIAN Haoran*
(School of Software Engineering, Tongji University, Shanghai 201804, China)

Abstract: A fuzzy query model for scalable vector graphics (SVG) format maps with layer definitions is proposed in this paper. In the model, a word repository is constructed to store the related words and fuzzy query patterns. When users execute queries, a word segmentation sub-algorithm decomposes the query statement into words according to the word repository. Then a semantic analysis sub-algorithm converts the words to the fuzzy query patterns. The fuzzy query patterns are connected with the elements on a related layer or layers in an SVG format map. These elements will be highlighted as query results. Experiments are conducted to validate the effectiveness of the proposed model.

Key words: scalable vector graphics (SVG) format map; layer; word repository; word segmentation; semantic analysis

可缩放矢量图形(SVG)[1]格式是W3C制定的一种新的用于描述二维矢量图形的格式，它规范了网络矢量图形的标准，并严格遵从可扩展标记语言(XML)语法，是一种和图像分辨率无关的矢量图形格式.

SVG格式地图目前已经支持对图形元素的查询功能[2]，但是对用户要求较高，易用性较低，而工程应用中对基于语义的模糊查询的需求越来越大. 国内外在多个领域都有对模糊查询的相关研究，如：针对传统关系型数据库管理系统[3-5]的模糊查询，空间数据[6]的模糊查询和针对XML格式文件的模糊查询[7-8]，但是这些模糊查询方法都不适用于SVG格式地图. 若能研究出一种基于SVG地图的模糊查询模型，将会大大提升工程应用中查询功能的易用性. 基于SVG格式地图的查询需要根据地图的分层数据与属性数据进行查询，而标准的SVG格式仅提供用于显示基础图形的标签，并没有储存分层数据与属性数据. 因此，为了实现模糊查询，必须将分层数据与属性数据储存到SVG格式地图的对应标签中.

1 SVG格式地图图层

基础的SVG格式地图只包含绘制元素必要的信息，不能支持基于语义的模糊查询，而改进后的SVG格式地图定义了用于储存地图分层数据与属性数据的标签.

SVG结构体[9]表示地理元素中的一个对象，使用组标签<g>元素作为最顶层元素. <g>元素的第一个子元素为扩展定义的XML标签<geo-attribute-data>，<g>元素的第二个之后(包含第二个)的所有子元素为该对象的图形数据.

SVG图层[9]使用组标签<g>元素进行表示，通过包含图层标识符(LAYER_)和图层编号(LayerNumber)的id属性进行标识. <g>元素下包

收稿日期：2016-03-09
基金项目：国家自然科学基金(41171303)
第一作者：杜庆峰(1968—)，男，教授，工学博士，主要研究方向为地理信息系统、软件工程、软件质量管理与软件测试.
E-mail:Du_cloud@tongji.edu.cn

含该图层所有的地理元素.

定义 1　假设有 SVG 地图 G，在地图 G 中存在用于储存图形绘制信息和数据的标签 l，则称 l 是 SVG 地图 G 的一个基础图形标签. 基础图形标签包括<line>、<rect>、<circle>、<ellipse>等.

定义 2　假设有 SVG 地图 G，G 中有一个图层 L，在 L 中有一个<geo-attribute-data>标签 l，l 容纳图层 L 对应的 SVG 结构体所包含的属性数据，则称标签 l 是图层 L 的分词属性标签. 其中属性数据采用标签-值对的方式表示，即标签名为属性名称，标签中所包含的文本信息即为属性数据.

2　SVG 格式地图模糊查询模型的相关定义

SVG 格式地图的分层是 SVG 格式地图模糊查询的基础，包含 SVG 图层的 SVG 格式地图将分层数据与属性数据储存到 SVG 地图对应的 SVG 图层中，为 SVG 格式地图的模糊查询提供了必要的数据. SVG 格式地图模糊查询模型由三个部分组成：模糊查询词库的构建，对查询语句的分词处理和对分词处理后查询语句的语义分析. 本章主要从上述三个部分讲述模糊查询模型的相关定义.

2.1　词库构建定义

词库的构建是模糊查询模型首先要完成的部分，因此整个模糊查询模型的定义也是从词库的构建开始展开. 词库构建相关定义包括地理信息元素编码定义，词库的定义和模糊查询匹配模式的相关定义.

2.1.1　地理信息元素编码定义

为了标识不同的图层，图层中还要标记出元素的编号和分类，而图层的编号与分类由图层编码所确定. 图层的编码由三个部分组成：大类编码、小类编码和扩展编码.

(1) 大类编码

依据《基础地理信息要素分类与代码》[10] 中的定义，地理要素分为定位基础、水系、居民地及设施、交通、管线、境界与政区、地貌和植被等八大类. 编码则由四位二进制数构成，从 0000 到 0111 对应八个大类.

(2) 小类编码

依据《基础地理信息要素分类与代码》[10] 中定义的八大类，每个大类都包含若干小类，所有大类下对应的小类数量不超过 16 个，因此使用四位二进制数标识小类.

(3) 扩展编码

为了使模糊查询词库支持扩展，模糊查询词库除了包含大类编码与小类编码以外，本文还定义了对应扩展类别的扩展编码，扩展编码的位数表示为

$$\left|\sqrt[16]{1+\max(c(i))_{i=00000000}^{01110010}}\right|\times 4 \qquad (1)$$

式中：i 为当前类别(由大类编码和小类编码共同确定)；$c(i)$ 为类别 i 的扩展类别数量. 式(1)表示，取 $c(i)$ 的最大值并对其 16 开方后向上取整再乘以 4，即是扩展类别编码位数. 对于所有八个大类中的小类，默认的扩展编码为 0000(由于默认编码已经被占用，因此要在原编码数量上加一，即扩展编码从 0001 开始).

2.1.2　模糊查询词库与匹配模式定义

本节将构建相关查询词库. 相关词词库添加了相关词的近义词，同时针对每个类别，添加了属性、操作符以及它们的近义词. 列出大类编码为 0000 的相关词，如表 1 所示.

表 1　词库编码快照

Tab.1　Snapshot of word coding repository

大类编码	大类相关词	小类编码	小类相关词	扩展编码	属性	操作符
0000	定位	0000 0001	观测、测量、测、控制点 数学	0000	坐标	大于、小于 等于

针对词库中每一个类别建立对应的匹配模式，用于模糊查询中的语义分析. 相关定义如下：

定义 3　假设在一次模糊查询的过程中，包含由大类编码、小类编码和扩展编码共同唯一确定的基础类别 e，则称 e 为基本匹配元素. 例如，0111 0000 0000 对应农林用地的默认扩展类型(默认扩展类型为 0000)，则称该扩展类型为一个基本匹配元素.

定义 4　假设在一次模糊查询的过程中包含基本匹配元素 e，e 包含属性信息 a，则称 a 为基本匹配元素 e 所包含的匹配元素属性. 例如：基本匹配元素河流包含匹配元素属性长度和宽度.

定义 5　假设在一次模糊查询的过程中，包含基本匹配元素 e，e 包含匹配元素属性 a，查询过程中包含匹配元素属性 a 的操作类型 p，则称 p 为匹配元素属性 a 的匹配属性操作符. 例如：匹配元素属性长度有匹配属性操作符等于.

定义 6　假设在一次模糊查询的过程中，包含基本匹配元素 e，e 包含匹配元素属性 a，a 包含匹配

属性操作符 p，查询过程中包含匹配属性操作符 p 的参数 p_d，则称 p_d 是匹配属性操作符 p 的匹配属性操作数. 例如：针对基本匹配元素河流的匹配元素属性长度有匹配属性操作符大于，匹配属性操作数就是数字.

定义 7 假设在一次模糊查询的过程中，包含基本匹配元素 e，e 包含匹配元素属性 a，a 包含匹配属性操作符 p，p 包含匹配属性操作数 p_d，查询过程中规定匹配属性操作数 p_d 的标准量 u，则称 u 是匹配属性操作数 p_d 的匹配属性单位. 例如基本匹配元素河流的匹配元素属性长度有匹配属性操作符大于，其匹配属性操作数是数字，匹配属性单位就是 m.

定义 8 假设在一次模糊查询的过程中，包含基本匹配元素 e，e 包含匹配元素属性 a，a 包含匹配属性操作符 p，p 包含匹配属性操作数 p_d，p_d 包含匹配属性单位 u，则将 e、a、p、p_d、u 的组合称为模糊查询匹配模式.

模糊查询匹配模式的建立规则如图 1 所示. 图中 1，n 表示连线左边元素可以连接多个连线右边元素，如一个属性可以有多个操作符.

图 1 模糊查询模式建立规则

Fig.1 Rules of fuzzy query matching mode

每个基本匹配元素至少包含一个匹配元素属性，每个匹配元素属性至少包含一个匹配属性操作符，每个匹配属性操作符至少包含一个匹配属性操作数，每个匹配属性操作数对应一个匹配属性单位，则{基本匹配元素，匹配元素属性，匹配属性操作符，匹配属性操作数，匹配属性单位}的结构就是一个模糊查询匹配模式. 具体示例如下：

针对表 1 中大类编码 0000，小类编码 0000，扩展编码 0000(默认扩展编码)的元素，基于该元素的长度属性，可以构建出以下三种匹配模式：

河流—长度—等于—数字(操作数)—m(单位)

河流—长度—大于—数字(操作数)—m(单位)

河流—长度—小于—数字(操作数)—m(单位)

2.2 模糊查询通配符分词定义

在进行模糊查询的过程中，必须将用户输入的语句进行分词操作，分解成模糊查询系统可以解析的一组词串，因此分词过程也是模糊查询不可缺少的过程. 分词操作相关定义如下：

使用 Σ 表示汉字表，用“*”和“?”表示通配符，并有 $*\in\Sigma^*$，$?\in\Sigma^?$. C 表示一个字符串，且有 $C\in\Sigma^+$，它由 Σ 中的字符组成，其长度为 $|C|$.

定义 9 包含通配符的字符串表示为 $\Pi\in(\Sigma\cup\{*,?\})^*$. 假设有 $l\in\Pi$，存在映射 $F(l)$，$F(l)$ 指用 Σ 中的字符替换 l 中的通配符“*”和“?”后构成的字符串集合，则称 F 为通配符的映射.

定义 10 假设有查询字符串 $Q\in(\Sigma\cup\{*,?\})^*$，$Q$ 可以表示成 $q_1t_1q_2t_2q_3t_3q_4t_4\cdots q_nt_n$，且有 $q_i\in\Sigma^*$，$t_i\in\{*,?\}^*$ $(i\in\mathbf{N}^+, i\leqslant n)$. 如果有匹配字符串 $R\in\Sigma^*$，其中 $Y_{qi}=|q_i|$，$Y_{?i}$ 表示 t_i 中“?”的个数，Y_{*i} 表示 t_i 中“*”的个数，$\Pi(q_i,R)$ 表示 R 中包含子串 q_i，且返回子串的位置. 当满足：①R 中包含所有子串 q_i $(1\leqslant i\leqslant n)$，且 $\Pi(q_i,R)+Y_{qi}\leqslant\Pi(q_{i+1},R)$；② $\Pi(q_i,R)+Y_{qi}+Y_{?i}\leqslant\Pi(q_{i+1},R)$，则称 Q 与 R 是通配符的匹配.

2.3 SVG 格式地图模糊查询语义分析定义

在完成分词操作后，要将整个词串进行语义分析，与词库中定义的模糊查询匹配模式进行匹配，最后返回匹配结果，从而取得 SVG 格式地图中对应的元素. 语义分析相关定义如下：

在模糊查询语义分析算法中，用 S 表示查询语句，使用 w 表示查询语句分解后的词串，P 表示根据模糊查询词库定义的语义分析模式.

定义 11 假设存在查询语句 S，如果查询语句 S 可以表示为多个子句 S_i $(i\in\mathbf{N}, i\leqslant n)$，且可以将查询语句分解表示成如下格式：$S=\{S_1,S_2,S_3,S_4,\cdots,S_n\}$ $(n\in\mathbf{N})$，则将上述分解过程称为模糊查询语句的分解.

定义 12 假设存在查询语句子句 S_i $(i\in\mathbf{N})$，如果查询子句 S_i 可以表示为 m 个词 W_{ij} $(j\in\mathbf{N}, j\leqslant m)$，且可以将子句分解表示成如下格式 $S_i=\{W_{i1},W_{i2},W_{i3},W_{i4},\cdots,W_{im}\}$，则将上述分解过程称为模糊查询子句的分解.

定义 13 假设存在查询语句子句 S_i $(i\in\mathbf{N})$，根据定义 12 将子句 S_i 分解成一组词串，如果该组词串可以根据模糊查询词库分为基本匹配元素、匹配元素属性、匹配属性操作符和匹配属性操作数(根据定义 3～6)，即 W_e、W_p、W_o、W_u、W_n，且可以将查询子句表示成如下格式：$S_i=\left\{\sum W_e,\ \sum W_p,\ \sum W_o,\ \sum W_u,\ \sum W_n\right\}$，则称将子句被分解为模糊查询匹配词串.

定义 14 假设语义分析模式库 P 由多个模糊查询匹配模式构成，分解过程可以表示成如下格式：

$P=\{P_1,P_2,P_3,\cdots,P_n\}(n\in\mathbf{N})$，则称是对语义分析模式库的分解.

3　模糊查询模型

依据定义 2～14，首先完成对模糊查询词库的构建，将用户的输入分词操作后得到模糊查询匹配词串，然后对模糊查询匹配词串进行语义分析后得到模糊查询匹配模式，即是模糊查询模型的完整流程. 因此，模糊查询模型包括模糊查询词库子模型、支持通配符的分词子模型和语义分析子模型.

3.1　模糊查询词库子模型思想

模糊查询模型首先需要完成对查询词库的建立，查询词库由两个部分组成：模糊查询相关词和模糊查询匹配模式.

(1)模糊查询相关词

对于每个类别，除了将类别名称加入词库，还将类别名称的近义词加入词库. 同时，针对每个类别添加了属性、操作符、操作数、单位以及它们的同义词. 根据定义 3～7，将这些词分别标识为基本匹配元素、匹配元素属性、匹配属性操作符、匹配属性操作数和匹配属性单位，用于后续的模糊查询匹配模式的匹配. 在分词操作过程中会使用到分词，分词过程结束后，得到系统可以解析的词串.

(2) 模糊查询匹配模式

除了储存相关词，词库需要储存模糊查询匹配模式，匹配模式由基本匹配元素、匹配元素属性、匹配属性操作符、匹配属性操作数和匹配属性单位共同确定. 匹配模式用于查询词串的语义分析，将查询词串匹配到 SVG 格式地图中对应的元素.

3.2　支持通配符的分词子模型思想

在完成建立查询词库的基础上，实现分词子模型是不可或缺的部分. 支持通配符的分词子模型由两个部分组成：分词算法[11]和通配符的处理.

(1)分词

分词系统建立在词库的基础上，需要使用到词库中的相关词. 首先根据定义 11 将查询语句分解成多个子句，针对每个子句根据定义 12 和 13 分解成词串. 分词过程用到的模型算法是正向最大匹配法[12]，即整个语句从左至右将所有可能的词块与查询词库中的相关词进行匹配，返回匹配结果，直至整个语句完成匹配.

(2)通配符的处理[13]

由于查询词中会存在通配符“*”和“?”，而分词结果的词串中不能包含“*”和“?”，因此需要将词块中存在的通配符处理后映射成词库中的相关词，即将包含“*”和“?”的词块根据定义 9 进行映射，取得不包含“*”和“?”的正常词. 具体做法是将包含“*”和“?”的词块与词库中的相关词根据定义 10 中的匹配方法进行匹配. 如果匹配成功，则将包含“*”和“?”的查询词映射为匹配后的相关词.

3.3　语义分析子模型思想

模糊查询模型最后一个部分是语义分析子模型，语义分析的过程会使用到分词子模型返回的词串和查询词库中的匹配模式.

语义分析系统首先将分词系统返回的词串根据定义 3～6 分别标记为基本匹配元素、匹配元素属性、匹配属性操作符和匹配属性操作数. 然后，根据定义 13 将基本匹配元素、匹配元素属性、匹配属性操作符和匹配属性操作数与词库中的模糊查询匹配模式(根据定义 14 分解匹配模式)匹配，返回匹配成功的模糊查询匹配模式.

3.4　模糊查询模型构建

(1)模糊查询流程

依据第 3.1～3.3 节中各子模型思想，本节构建完整的模糊查询模型. 模糊查询模型由上述三个部分组成. 首先，要使用模糊查询词库子模型来建立词库，用于后续查询语句的分词处理和模糊查询语义分析；然后，将用户的查询输入至模糊查询分词子模型，模糊查询分词子模型将原始查询词串与模糊查询词库中的相关词进行匹配后输出匹配后的查询词串；最后，调用语义分析子模型，语义分析子模型将处理后的查询词串组合与查询词库中的模糊查询匹配模式匹配，返回对应的模糊查询匹配模式，用于 SVG 格式地图的元素查询. 下文将介绍模糊查询词库的具体构建方法、带通配符的分词算法与 SVG 格式地图模糊查询语义分析算法的详细实现.

分词子模型的输入为用户输入的查询语句，输出为模糊查询词库中对应的相关词集合.

分词子模型依据最大匹配法的思想进行分词，T 是整个分词结果的集合，T 可以表示为

$$T=\bigcup_{i=0}^{|Q|}\bigcup_{j=i}^{\min(H_Q,|Q|-1)}\mathrm{MATCH}(Q_{ij})$$

式中：Q 是查询字符串；H_Q 是词典中字符串的最大长度；$\min(H_Q,|Q|-1)$ 是 H_Q 与 $|Q|-1$ 的最小值；Q_{ij} 是 Q 的子串，从 Q 的第 i 位到第 j 位；$\mathrm{MATCH}(Q_{ij})$ 的思想是使用定义 9 中的映射获得 $F(Q_{ij})$，根据定义 10 将获得的集合中的词与词库匹配，返回匹配成功的结果. 根据公式可以看出算法有

两层循环,子模型的执行步骤如下:

步骤1 外层循环设置 Q 的子串 Q_{ij} 的起始位置,即 i 的值,从第一个字符的位置0到最后一个字符的位置 $|Q|$,每次外层循环更改 i 的值.

步骤2 内层循环设置 Q 的子串 Q_{ij} 的结束位置,即 j 的值,从子串的起始位置 i 到 $\min(H_Q, |Q|-1)$. 因为子串的长度不会大于词典中最长词的长度 H_Q,也不会大于查询词的末尾到子串起始位置的长度,所以结束位置最大为 $\min(H_Q, |Q|-1)$. 每次内层循环更改 j 的值.

步骤3 针对每个子串 Q_{ij} 调用函数 MATCH. MATCH 函数将找出词库中与 Q_{ij} 相匹配的词并返回.

(2)语义分析子模型构建

语义分析子模型的输入为分词子模型的输出,即相关词的集合,输出为模型查询匹配模式.

对于任意查询语句 S,使用模糊查询词库中定义的查询模式库 P,定义函数 $F(S,P)$ 表示语义分析模型的分析过程. 子模型分析过程为

$$F(S,P)=\bigcup_{k=i}^{n}\{F(S_k,P)\}=$$
$$\bigcup_{k=i}^{n}\{\bigcap_{j=1}^{m}\{F(W_{kj},P)\}\}=$$
$$\bigcup_{k=i}^{n}\left\{\prod\{F(W_e,P)F(W_p,P)F(W_o,P)\cdot F(W_u,P)F(W_n,P)\}\right\}=\bigcup_{k=i}^{n}\left\{\prod\{F(W_e,P_l)\cdot F(W_p,P_l)F(W_o,P_l)F(W_u,P_l)F(W_n,P_l)\}\right\}$$

式中:函数 F 是语义分析模型的分析过程;S 是查询语句;P 是模糊查询定义的查询词词库中的匹配模型库;S_k 是 S 分解后的子句;W_{kj} 是 S_k 分解后的词串;P_l 是由 $F(W_e,P)$ 确定的一个特定匹配模型. 子模型的执行步骤如下:

步骤1 对于查询语句 S,将 S 与查询模式 P 作为参数进行语义分析.

步骤2 根据定义11,将 S 分解成多个子句,并多次进行语义分析,并将结果求并集;再根据定义12将每个子句 S_k 分解成多个词串,多次进行语义分析.

步骤3 根据定义13,将每个词串定义成特定类别词型 W_e、W_p、W_o、W_u、W_n.

步骤4 根据定义14,通过语义分析一组词串 $\{W_e,W_p,W_o,W_u,W_n\}$,确定特定的语义分析模型 P_l. 将 P_l 与 $\{W_e,W_p,W_o,W_u,W_n\}$ 进行语义分析,返回分析结果.

4 模糊查询模型算法及验证

4.1 模糊查询算法

4.1.1 模糊查询词库建立

针对每一元素(由大类编码、小类编码和扩展编码共同确定),定义出了元素属性、操作符、单位和操作数. 此处不一一列出,仅列出词库中大类编码和小类编码为0000 0000的相关词结构,如图2所示.

```
<element>
  <name>河流</name>
  <synonym>河,流域</synonym>
  <attribute>
    <name>长</name>
    <synonym>长度</synonym>
  </attribute>
  <operators>
    <operator>
      <name>大于</name>
      <synonym>大,&gt;</synonym>
      <unit>m</unit>
      <operand>number</operand>
    </operator>
    <operator>
      <name>小于</name>
      <synonym>小,&lt;</synonym>
      <unit>m</unit>
      <operand>number</operand>
    </operator>
    <default_operator>
    <! ——当没有指定操作符时,默认使用,例如河流长度100 m,这时使用等于作为默认操作符——>
      <name>等于</name>
      <synonym>近似,=</synonym>
      <unit>m</unit>
      <operand>number</operand>
    </default_operator>
  </operators>
</element>
<element>
</element>
```

图2 词库快照

Fig.2 Snapshot of word repository

元素为河流,包含元素属性长度和宽度,基于长度属性有操作符大于、小于和等于,每个操作符又包括单位和第二操作数. SVG格式地图模糊查询词库以XML格式储存,其中<element>标签对应基本匹配元素,<attribute>标签对应匹配元素属性,<operator>标签对应匹配属性操作符,<operand>

标签对应匹配属性操作数，<unit>对应匹配属性单位. 其中，<name>标签用于储存名称，<synonym>标签用于储存同义近义词. 用于语义分析的模糊查询匹配模式也隐式地包含在<element>标签中.

4.1.2　分词子模型算法

分词子模型中使用的分词词库即是第 4.1.1 节中模糊查询词库，分词结果即是模糊查询词库中相关词的集合，具体算法如图 3 所示.

```
0   Function Participle(input,output)
1       Begin
2           Integer startPos=0;
3           Integer endPos=input.length;
4           Integer currentPos=startPos+1;
5           While(startPos<endPos) //对应步骤 1 的外层循环
6               then call coreFunc(input,startPos,currentPos,out)→result;
7                   If(result==true)
8                       then startPos=currentPos+1;
9                   Else
10                      then currentPos=currentPos+1;
11      End
12
13  Function coreFunc(input,startPos,currentPos,out)
14      Begin
15          String[] strStore;
16          String str
17          For i=startPos to currentPos
18              if(input[i]=="*" OR input[i]=="?")
19                  then strStore.put(str)
20                      strStore.put(input[i])
21              Else
22                  Then str.put(input[i])
23          Bool flag=true;
24          For i=0 to strStore.length//对应步骤 2 的内层循环
25              If str[i]=="*"//25~38 对应步骤 3 的 Match 函数
26                  Then if LibraryContain(str[i+1]) And LibraryContain(str[i-1]) And
27                          LibraryContain(str[i+1])>= LibraryContain(str[i-1])
28                          Then flag=flag
29                      else
30                          Then flag=false
31                              Retrun flag
32              If str[i]=="?"
33                  Then if LibraryContain(str[i+1]) And LibraryContain(str[i-1]) And
34                          LibraryContain(str[i+1])== LibraryContain(str[i-1])+1
35                          Then flag=flag
36                      else
37                          Then flag=false
38                              Retrun flag
39          Output.put(theLastStrUsedInLib)
40          Return true
41      END
```

图 3　分词伪代码

Fig.3　Pseudocode of word segmentation

主函数为 Participle，Participle 根据输入的原始查询词串循环调用核心函数 coreFunc，直至整个输入都被处理完. coreFunc 将原词分解成常用字符部分和{*，?}部分，并与词库中所有词进行比较，直至匹配结束.

假设该子算法词库含量为 M，输入为 N，词库中最长词长度为 d. coreFunc 的第一个循环时间复杂度为$(N-d)$，第二个循环最坏情况下循环$(N-d)$次，每次都将与词库匹配，每次匹配最坏情况为 M 次，则实际复杂度为$(N-d+1)M$. Participle 循环调用 coreFunc，最坏情况下调用$(N-d)$次，则整个函数的时间复杂度为$(N-d)(N-d+1)M$，即$O(N^2M)$.

4.1.3　语义分析子模型算法

语义分析子模型将第 4.1.2 节中的输出作为输入，使用第 4.1.1 节中的模糊查询词库，与词库中的模糊查询匹配模式匹配，输出匹配的模糊查询匹配模式的集合，具体算法如图 4 所示.

主函数为 fuzzyQuery，首先对处理后的查询词数组进行排序（基本匹配元素>匹配元素属性>匹配属性操作符>匹配属性操作数），将排序后的相关词数组与词库中所有的匹配模型进行比较，即调用 matchModel. matchModel 根据数组找到对应的元素词，继续将元素词和对应的多个模型依次与输入数组进行比较，直至匹配.

假设元素对应的模糊查询匹配模式的数量最多为 m，输入中元素词的个数为 A，输入大小为 N. matchModel 的时间复杂度即是 $m\times A$（除去元素，模型包含的关键词的个数、属性、操作符、操作数和单位），则主函数的时间复杂度为$NA(m\times 4)$，由于

```
1   Function fuzzyQuery(input)                    //output from function Participle
2   Begin
3       Sqort the input                    // by the value of category
4       For I=0 to input.length            //对应步骤 3
5           Foreach model in library
6               matchModel(input[I],input[i+1...input.length])
7               If(matched)
8                   Then search the element in map
9                       Return the results
10
11  Function    matchModel(element,restElements)    //对应步骤 4
12  Begin
13      Foreach element in the model
14          Element.match(restElements)
15      IF all elements matched
16          Then return model;
17      else
18          Then return null model
```

图 4 查询伪代码

Fig.4 Pseudocode of semantic analysis

$mA<M$,则时间复杂度为 $O(NM)$.

综上分析,完整的模型查询算法包含分词子模型算法和语义子模型分析算法,总的时间复杂度为 $O(N^2M)$.

4.2 模糊查询验证

为验证模糊查询模型的原型实现,采用 JAVA 语言,在 Eclipse(Version:Kepler Service Release 2)环境下进行开发,实验环境为 Corei5 处理器,2.60 GHz 双处理器,8.00 GB 内存,Win8 系统.

4.2.1 模型算法功能性验证

针对上述模糊查询模型,本节将根据三个不同的模型类型对模糊查询模型进行验证,所有的验证都是基于图 5,其中包含酒店和河流图层(酒店使用圆圈表示,河流使用折线表示).

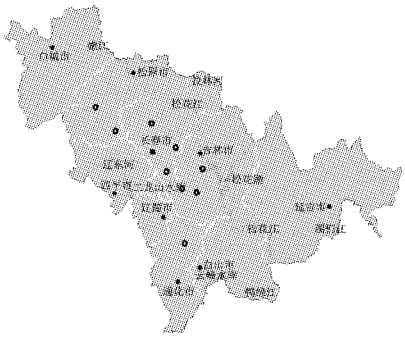

图 5 查询原图

Fig.5 Original query map

多元素多模式是指查询结果包含多个基本匹配元素对应的多个模糊查询匹配模式.

输入的搜索词:长度大于 5 000 m 的河流,价格低于 150 元的酒店.

语义分析过程:针对基本匹配元素河流和酒店,查找河流和酒店的所有模糊查询匹配模式,找到模糊查询匹配模式{基本匹配元素,匹配元素属性,匹配属性操作符,匹配属性操作数,匹配属性单位},对分词结果模糊查询匹配模式进行匹配.

匹配结果为{河流,长度,大于,5 000,m}和{酒店,价格,低于,150 元}.匹配结果满足河流模型和酒店模型,返回找到的所有模型,然后在地图中针对河流图层和酒店图层元素根据模糊查询匹配模式进行查询.

图 6 为结果展示,地图中河流元素中长度属性大于 5 000 m 的河流被加粗,酒店元素中价格属性低于 150 元的酒店都被放大(使用圆圈表示).

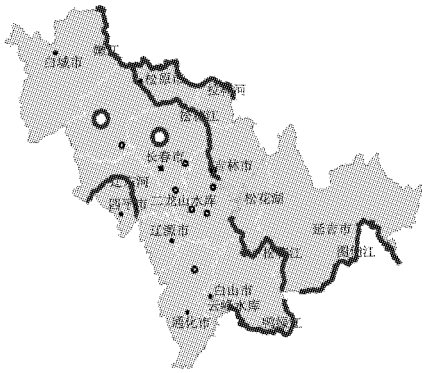

图 6 多元素多模式查询结果

Fig.6 Query results of multiple elements and multiple modes

4.2.2 模型算法效率验证

(1)基于输入长度 N 的验证

针对同一匹配模型(即最终匹配结果固定),在词库固定(词库的含量 M 固定)的情况下,在输入中加入无关字符,改变输入查询字符串的长度,记录程序运行时间.

在输入字符串长度分别为 112、212、312、412 和 512 的情况下各运行 10 次,记录运行时间,结果如表 2 所示.

观测同一N下所有试验的平均,判断是否符合

表 2　基于输入的算法效率验证

Tab.2　Validation of algorithm efficiency based on input　　ms

试验次数	不同输入长度下运行时间				
	112	212	312	412	512
1	849	2 425	5 469	9 305	13 887
2	661	2 269	5 335	9 721	14 005
3	605	2 197	5 031	10 438	13 947
4	659	2 415	4 884	10 717	13 484
5	680	2 146	5 151	9 718	14 389
6	641	2 384	5 294	9 851	13 611
7	835	2 016	5 132	9 463	14 019
8	547	2 237	5 115	9 392	13 777
9	699	2 858	4 900	9 243	14 049
10	578	2 405	4 783	9 345	14 414
平均值	675	2 335	5 109	9 719	13 958

时间复杂度的预测，可以得出结论程序执行时间与 N 的平方线性相关，验证正确.

(2)基于词库含量 M 的验证

针对同一查询词(N 固定)，改变词库的含量，记录程序运行时间.

在词库含量分别为 10、100、200、600 和 1 000 的情况下各运行 10 次，记录运行时间，结果如表 4 所示.

观测同一 M 下所有试验的平均值，判断是否符合时间复杂度的预测，可以得出结论程序执行时间与 M 线性相关，验证正确.

表 3　基于词库的算法效率验证

Tab.3　Validation of algorithm efficiency based on word repository　　ms

试验次数	不同词库含量下运行时间				
	10	100	200	600	1 000
1	849	6 431	13 401	39 191	58 673
2	661	6 624	14 327	34 758	62 471
3	605	7 013	15 021	36 462	70 358
4	659	6 982	11 430	37 243	69 762
5	680	6 614	12 485	36 393	62 748
6	641	6 823	13 771	32 787	65 423
7	835	6 647	15 012	39 812	66 237
8	547	5 930	13 447	36 946	80 435
9	699	6 997	14 304	35 763	62 179
10	578	8 320	15 279	34 439	65 314
平均值	675	6 216	12 589	33 072	60 327

5　结语

本文基于 SVG 格式地图的图层数据建立了模糊查询模型，可以通过分析查询语句的语义后与预定义的模糊查询模式进行匹配，最后从 SVG 文件中找到匹配的节点. 本文工作主要由以下四个部分组成：

(1)在 SVG 格式地图图层中储存分层数据，使 SVG 格式地图可以支持模糊查询.

(2)建立模糊查询词库，储存模糊查询相关词和匹配模式.

(3)对查询语句进行分词，分解成模糊查询模型可以解析的词串，将分词结果返回给语义分析算法.

(4)使用模糊查询词库对分词后的词串进行模糊查询模式匹配，返回匹配结果，在 SVG 格式地图中查找对应的 SVG 元素.

SVG 格式模糊查询模型已经实现了基于 SVG 格式地图的模糊查询功能，可以根据查询词的语义进行分析后查询，返回查询结果. SVG 格式地图模糊查询功能的实现大大提升了工程应用中 SVG 格式地图查询功能的易用性，为 SVG 格式地图模糊查询提供了较好的方法和理论基础.

参考文献：

[1] W3C. Scalable vector graphics (SVG) 1.2 specification [EB/OL]. [2016-02-15]. http://www.w3.org/TR/2004/WD-SVG12-2004-10-27/2004-10-27.

[2] 杜庆峰，卢冬蕊. 改进规则的可放缩矢量图形地图的查询模型[J]. 同济大学学报(自然科学版)，2014，42(5)：790.
DU Qingfeng，LU Dongrui. Query model of improved rules scalable vector graphics map[J]. Journal of Tongji University (Natural Science)，2014，42(5)：790.

[3] LAI L F，WU C C，HSIEH Y T，*et al*. A fuzzy query approach to human resource web services[C]//Proceedings of the e-Business Engineering (ICEBE)，2013 IEEE 10th International Conference on. Washington DC：IEEE，2013：461-466.

[4] MISHRA J. Fuzzy query processing[C]//BUCUR R，BREAZ D，eds. Proceedings of the International Journal of Research and Reviews in Next Generation Networks. Naiguatá：IEEE，2011，1(1)：35-38.

[5] CADENAS J T，TINEO L，PEREIRA R T. About fuzzy query processing [C]//Proceeding of the Computing Conference (CLEI)，2013 XXXIX Latin American. Washington DC：IEEE，2013：1-11.

[6] 胡骏，范举，李国良，等. 空间数据上 Top-k 关键词模糊查询算法[J]. 计算机学报，2012，35(11)：2237.
HU Jun，FAN Ju，LI Guoliang，*et al*. Top-k fuzzy spatial keyword search [J]. Journal of Computer，2012，35(11)：2237.

[7] 孟祥福，张霄雁，马宗民，等. 一种基于领域知识的 XML 数据模糊查询[J]. 智能系统学报，2013，7(6)：525.
MENG Xiangfu，ZHANG Xiaoyan，MA Zongmin，*et al*. An XML fuzzy query answering approach based on domain knowledge[J]. CAAI Transactions on Intelligent Systems，2013，7(6)：525.

(下转第 940 页)