文章编号:1000-5862(2016)06-0635-05

基于构件的若干图算法开发和生成

鄢梦恬,石海鹤*
(江西师范大学计算机信息工程学院,江西 南昌 330022)

摘要:软件构件技术可显著提高程序的可靠性和开发效率,极大减少开发成本.泛型程序设计有助于降低编程的复杂度,为重用构件开发提供有效支持.介绍了生成式程序设计思想及泛型程序设计技术,分析了图算法领域的关键特征及领域共性问题,并对广度优先搜索、单源最短路径、所有顶点对最短路径等一类问题进行抽象,设计出相应的泛型图算法构件,进一步借助 PAR 方法中的泛型机制进行描述,并在 PAR 平台程序生成系统上进行构件组装生成具体的算法程序.

关键词:图算法生成;构件;生成式程序设计;PAR 方法
中图分类号:TP 301 文献标志码:A DOI:10.16357/j.cnki.issn1000-5862.2016.06.18

0 引言

20 世纪 60 年代中期开始爆发的“软件危机”对软件的可靠性和开发效率提出了更高的要求,软件产业亟需开发效率更高、正确性更好的软件产品.长期以来,计算机工作者研究如何提高软件尤其是算法程序的开发效率,同时保证程序的正确性.当前使用的众多程序设计环境都有规模不等的可复用程序构件库,由于这些构件的存在,使得软件的开发效率得到较大的提升.大量的实际工作已经表明,可以利用软件复用解决“软件危机”这一难题.可复用程序构件库中的构件之间可以相互组装成各种不同的软件,这可以大大减少软件开发中的重复性工作.

在 1968 年的 NATO 软件工程会议上,M. D. McIlroy[1] 提出可以借鉴硬件部件组装成电子计算机的思想,将不同的软件部件进行组装以生成软件产品,这对提高软件的开发效率有很大的帮助.除了保证开发效率,尚需保证可重用软件部件的正确性.人们发现使软件部件具有通用的特性可以较好解决上述问题,Ada 语言最早体现了这一思想.用户在使用 Ada 语言时,可以根据需要定义类型或操作来作为参数,然后在具体的程序执行过程中,对所定义的参数赋值.这一方法有助于设计开发出更具通用性的软件构件,大大提升了部件的可复用性.为提高软件的生产效率和质量,有效途径之一是以一种自动化构件组装的方式来制作软件产品.图灵奖获得者 J. Gray[2] 将软件自动化列为 21 世纪信息技术领域的 12 个重要研究目标之一.D. Batory[3] 在 ACM FSE 会议上指出,软件自动化在未来软件工程中占据着重要的位置.软件自动化的核心是算法的自动生成问题.

图算法广泛应用于生活、生产和科学研究中,其算法设计策略的灵活性使图算法更具复杂性和多样性,在算法研究中难度最大.因此,本文基于形式化方法 PAR[4-7],采用生成式程序设计思想[8] 和泛型程序设计技术,深入剖析广度优先搜索、最短路径等一类图问题及其算法求解的特征,揭示不同算法隐藏的共性结构,设计图算法泛型构件,并借助 PAR 方法泛型机制进行描述与刻划,进一步通过 PAR 平台支撑的构件自动组装生成若干具体图算法,进而提高了一类图算法的可靠性和开发效率,降低其开发成本.所开发的构件有望组装生成其他类图算法程序.

1 相关技术

1.1 生成式程序设计技术

G. T. Leavens 等[9] 指出生成式程序设计(Generative Programming,GP)是实现自动程序设计的有效途径.GP 在对整个软件系统族进行建模的基础上,从领域特定语言描述的软件需求规约和配置知识出发,自动化地组装基层的可复用软件部件,使生产出的软件产品满足客户需求,从而实现软件开发

收稿日期:2015-12-17
基金项目:国家自然科学基金(61363013,61662035)和江西省自然科学基金(20142BAB217026,20151BAB207015)资助项目.
通信作者:石海鹤(1979-),女,江西乐平人,教授,博士,主要从事可信软件和基于组件的软件工程研究.

的自动化.

GP 包含 2 个开发周期: 领域工程和应用工程. 前者包括领域分析、领域设计和领域实现 3 个不同的阶段,后者是利用重用进行开发的过程[10]. 同一个领域中的新系统开发是以原有的领域模型为基础,重新组装配置大量的可复用构件,而不用一切从头开始.

就此主题,国际上每年召开一次学术会议(International Conference on Generative Programming and Component Engineering,GPCE). D. Batory 等[11] 提出可将算法程序的功能性定义为特征,将不同的特征进行组合来自动化地开发程序. Li Yulin 等[12] 主要是基于通用软件组件的重用实现程序设计的自动化,他们通过创建一个通用的几何应用程序组件库,利用几何计算自动生成应用程序. J. C. Fu 等[13] 提出一种迭代智能规划方法来生成类参数化的可重用过程,并结合组件技术来生成复杂的代码.

1.2 PAR 方法及其泛型机制

PAR(Partition-and-recur) 是一种行之有效的软件形式化方法,主要运用了分划和递推 2 种思想,并设计了一种抽象程序设计语言 Apla(Abstract Programming Language),它兼具数学语言简洁、严谨的优点,充分体现了数据及功能抽象的算法程序设计思想. Apla 泛型机制的实现主要分为类型参数化和子程序参数化.

1) 类型参数化. 在 Apla 语言中,引入关键字 sometype 来定义程序名类型参数、类型变量、过程函数的参数返回值类型,也可以直接在类型声明中以参数的形式类说明组合数据类型的基本类型,使得程序实现泛型化. 定义类型参数的语法结构为: "sometype 〈类型参数〉".

2) 子程序参数化. Apla 中提供了 func 和 proc 关键字来声明过程参数和函数参数. 泛型子程序中的子程序可以带有普通值参、类型参数或者子程序参数,使得子程序被泛型化. 在 Apla 中定义子程序参数化的语法结构为 "procedure 〈过程名〉|function 〈函数名〉(形参表);".

泛型子程序使用之前需进行实例化,即通过实例化得到类型参数和子程序参数的具体实现细节,才能使用定义的子程序. 实例化时,必须要有具体的类型传递给泛型子程序中的形参表,也必须有非泛型的实参传递给子程序中相应的形参,同时,子程序传递的实参必须和子程序的形参在参数个数、参数类型上完全匹配. 在 Apla 中实例化泛型子程序的语法结构为 "procedure [function] 〈新过程名或新函数名〉: new 〈泛型过程名〉(〈类型实参表〉|〈子程序实参表〉);".

可以看出,Apla 语言可用以方便描述和设计复杂算法程序,所得算法结构具有较高的可重用性.

PAR 较好地支持了生成式程序设计中的 2 个开发过程. 在生产可重用软件构件的过程中,PAR 方法基于软件开发构件化的思想,对数据库操作和算法程序的开发提供了良好的支持,将其作为可以重用的部件独立出来,而将一些 PAR 方法目前还没有涉及的方面如用户界面等分离出去. 此外,PAR 方法已经开发出一些基本的可重用构件,利用构件库中这些已有的基本构件也能够开发出新的构件.

使用 PAR 方法,文献 [14] 推导了单源最短路径、拓扑排序算法,文献 [15] 将 Floyd 最短路径算法、Warshall 图传递闭包算法、最大容量路算法抽象于基于闭半环(Closed semiring) 结构的 Kleene 抽象算法框架.

2 若干图算法开发

2.1 图算法分析

图论问题渗透整个计算机科学,图算法对于计算机学科至关重要,成千上万的计算问题最后都可以归约为图论问题. 在解决各种图论问题时,应用不同的问题划分和规则变换就会得到形式、性能各异的求解算法. 因此,可以设计非常灵活的算法策略. 虽然算法策略各异,但仍然可以对其进行数据、功能的抽象,分析各算法的通用性及可变性,构造出构件并进行组装来生成算法.

图论问题的核心问题之一是最短路径问题,许多更深层次的算法问题都是以最短路径问题为基础的. 同时,最短路径问题在实际中也有很多的应用,有些问题虽然从表面上看与最短路径问题没有什么联系,但是深入分析就会发现它们也可以用最短路径的思想解决. 下面先给出该问题的定义.

给定有向加权图 $G=(V,E)$,在其上定义的加权函数 $W: E \to R$ 为从边到实型权值的映射. 路径 $P=(v_0,v_1,\cdots,v_k)$ 的权是指其组成边的所有权值之和 $w(p)=\sum_{i=1}^{k} w(v_{i-1},v_i)$.

若 u 到 v 之间存在通路,则 u 到 v 的最短路径的权为 $\delta(u,v)=\min\{w(p): u \to v\}$,从结点 u 到结点 v 的最短路径为权 $w(p)=\delta(u,v)$ 的任何路径.

最短路径问题可以进一步分为单源最短路径和全源最短路径. 单源最短路径定义为: 给定起始顶点 s,找出从 s 到图中其它各顶点的最短路径. 全源最短路径问题可以简单地认为是单源最短路径问题的

推广,即分别以每个顶点作为起始顶点,求其余顶点到起始顶点的最短距离.

Dijkstra 算法和 Floyd 算法分别是解决单源最短路径和全源最短路径的经典算法.

Dijkstra 算法主要解决所有边的权为非负的单源最短路径问题,Dijkstra 算法中首先定义一个顶点集合 S,从源点到集合中的顶点的最终最短路径的权值均已经确定. Dijkstra 算法过程反复从 $V-S$(V 即图的顶点集) 中选择一个顶点 v,使得 $d(s,v)$ 为最短,并将 v 加入到集合 S 中,接着对 v 的所有边进行松弛(relaxation) . 其算法过程描述如下: (i) 初始将顶点 s 加入到 S 中,并更新 s 到其他顶点的路径权值; (ii) 选择最短路径 $dist(s,t)$,并将 t 加入到 S 中; (iii) 在(ii) 得到 t,对于 $u \in V-S$,松弛边(t,v) ; 重复上述过程,直到所有的顶点都加入到集合 S 中为止.

Floyd 算法可以检测图中的负环并可以解决不包括负环的图中的全源最短路径问题. 从一个图中的任意节点 i 到另一个任意节点 j 的最短路径只有 2 种情况: (i) 从节点 i 到节点 j 不存在中间结点, (ii) 经过若干个中间节点 k,从结点 i 到达结点 j. 因此,首先可以定义节点 u 到节点 v 的最短路径的距离为 $dis(i,j)$,对于每一个中间节点 k,验证 $dis(i,k)+dis(k,j)<dis(i,j)$ 是否成立,若成立,则说明经过中间结点 k,i 到 j 的距离变短,就可以设置 $dis(i,j)=dis(i,k)+dis(k,j)$,通过这种运算,当所有节点 k 遍历完,$dis(i,j)$ 表示的就是结点 i 到结点 j 的最短路径的距离.

Dijkstra 算法和 Floyd 算法都使用了松弛技术. 开始进行一个最短路径算法时,只知道图中边和权值. 随着处理逐渐得到各对顶点的最短路径的信息. 算法会逐渐更新这些信息,每步都会检查是否可以找到一条路径比当前给定路径更短. 这一过程通常称为“松弛”. 用 $d[]$ 数组表示顶点 s 到顶点 i 的最短路径的长度,用 $p[]$ 表示顶点 i 在最短路径中的父顶点. 可以将边松弛过程用如下代码来描述:

$Relax(v,w,\omega(v,w))$

if $d[w]>d[v]+\omega(v,w)$

$\{d[w]=d[v]+\omega(v,w);p[w]=v;\}$.

同时,Dijkstra 算法和 Floyd 算法都使用了类似广度优先搜索算法(Breadth First Search,BFS 算法) 的思想,在寻找最短路径的过程中,它们是以起始点为中心一级一级向外层层扩展,直到扩展到终点为止. 广度优先搜索能够在非加权图 $G=(V,E)$ 中快速地寻找从一个固定顶点 s 到其余顶点之间的最短距离,所以可以将广度优先搜索看作一个特殊的求取最短路径的算法. 广度优先搜索是基于如下的简单想法: 从 s 点开始,访问每一个被 s 支配的顶点 x. 设 $dis(s,x)=1$ 和 $s:=pred(x)$ (顶点 s 是 x 的前驱) . 访问那些与 s 距离不为 1 且受 x 所支配的顶点 y,则有 $dis(s,y):=2$ 和 $x:=pred(y)$. 一直执行这个过程,直到所有能到达的顶点都被找到,这些顶点都是从 s 点可达的. 因为搜索过程是逐级进行的,所以这个搜索过程能够快速地计算出 s 点到其可达顶点的最短路径. 而那些从顶点 s 不可达的顶点 z,则令 $dis(s,z):=\infty$.

2.2　算法构件设计

虽然 BFS 算法、Dijkstra 算法、Floyd 算法 3 个算法思想相似,但它们各自输出的结果是不一样的,因此可以构造出一个抽象的泛型子程序构件和 3 个具体的输出构件,这些构件都可以基于 PAR 可重用构件库中的类型构件来定义和实现. 为了表示一个构件对另一个构件的依赖关系,可以用一条带箭头的连线来体现,则上述构件间的依赖关系可描述成图 1,下一层的构件为上一层的构件提供某些服务,如 relax 需要使用到 $p1$ 提供的服务.

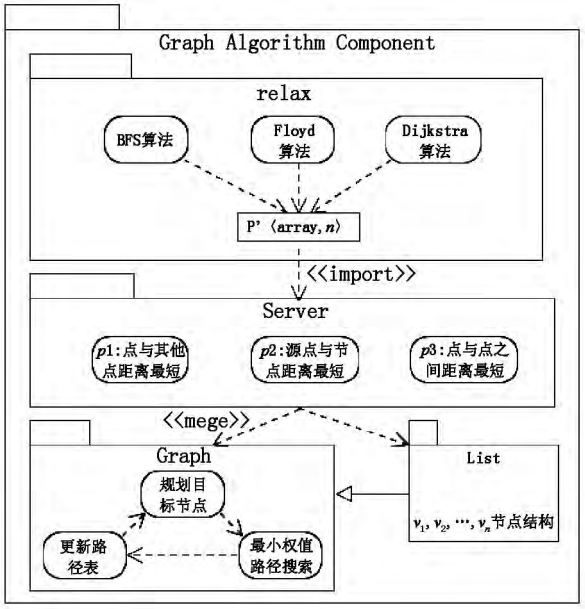

图 1　构件依赖关系

其中图类型构件 graph 和序列构件 list 已在 PAR 平台实现. 泛型子程序构件 relax 所带形参表为一个普通整型参数,一个 2 维类型数组以及一个函数参数名为⊙,这些未知类型将会在程序实例化时获得具体的类型实参.

2.3　构件的形式化开发

将上述设计出的算法构件用 Apla 语言描述如下,其中,①、②、③、④分别为构件 $relax$、$p1$、$p2$ 和 $p3$:

①//泛型子程序构件//

Procedure relax(n: integer; c: array [0.. maxnumv,a];

```
func⊙(d: array [0..maxnumv,a]);
var  i,j,k: integer;
begin
i:=0; k:=0;
do k<numv→i:=0; do i<numv→j:=0; do j<numv→
d[i,j]:=min(d[i,j],d[i,k]+d[k,j]); j:=j+1;
od; i:=i+1; od;
     k:=k+1;
od;
⊙(d);
end;
```

②//输出构件//

```
function p1(d1: array [0..maxnumv,a]);
var  i,j,k: integer;
begin
writeln("请输入源节点:");
read(i);
writeln(i,"节点与其他节点之间的最短路长度为:");
k:=0; do k<numv→j:=0; do j<numv→if(d[i,j]=k),writeln(i,"→",j,"最短路度:",d[i,j]); j:=j+1;
od;
end;
```

③

```
function p2(d1: array [0..maxnumv,a]);
var  i,j: integer;
begin
writeln("请输入源节点:");
read(i);
writeln(i,"源点与其他节点之间的最短路长度为:");
j:=0; do j<numv→writeln(i,"→",j,"最短路长度:",d[i,j]); j:=j+1;
od;
end;
```

④

```
function p3(d1: array [0..maxnumv,a]);
var  i,j: integer;
begin
writeln("各个节点之间的最短路长度为:");
i:=0; do i<numv→j:=0; do j<numv→writeln(i,"→",j,"最短路长度:",d[i,j]); j:=j+1; od; i:=i+1; od;
end;
```

构件 *relax* 刻画了 3 个算法的共性部分,即松弛操作,每找到一个新节点 k,就要以 k 为新考虑的中间点,用 $d[i,j]:=\min(d[i,j],d[i,k]+d[k,j])$ 修改源节点到剩余各顶点的距离. BFS 算法输出的是一棵广度优先生成树,Dijkstra 算法输出的是从源顶点到其它顶点间的最短路径长度,Floyd 算法输出的是每两点之间的最短路径. 由于 3 个算法的输出结果不一样,所以 $p1$、$p2$、$p3$ 依次为 BFS 算法、Dijkstra 算法、Floyd 算法 3 种算法的输出构件.

3 特定算法的组装生成

在运用 Apla 语言描述构件后,可使用 PAR 平台中的自动转换系统将 Apla 程序转换为 C++或者 Java 等可执行语言的构件. 组装这些构件时,可以运用 Apla 语言提供的泛型实例化机制,用参数替换的方式组装成具体的算法程序. 组装构件生成相应图算法的语句如下:

```
procedure BFS: new relax(p1);
procedure Dijkstra: new relax(p2);
procedure Floyed: new relax(p3);
```

将泛型构件 relax 分别与输出构件 $p1$、$p2$、$p3$ 装配可得到 BFS 算法、Dijkstra 算法和 Floyed 算法.

Apla->Java 自动转换系统可以将抽象的 Apla 程序转换为 Java 程序,是支持 PAR 方法开发可靠的 Java 程序的必要支撑工具. 其功能是基于自定义 Java 可重用构件库的支持,遵循预定的转换规则,把抽象的 Apla 程序,自动地转换为可执行的 Java 程序. 利用 PAR 平台,将上述设计构件组装自动生成了具体图算法的 Java 程序. 选取了 10 组数据进行测试实验,图 2 展示了其中一组数据,运行结果见图 3. 这表明利用构件及自动转换系统可生成具体的算法程序,从而达到提高算法程序的可靠性及设计效率的目的.

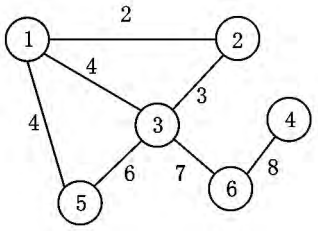

图 2 测试图例

4 总结

本文将生成式程序设计思想、泛型程序设计技术和形式化方法综合应用于图算法生成领域,基于泛型构件开发和生成了 BFS、Dijkstra 和 Floyed 3 种图算法. 通过分析、提取这 3 种算法的通用性和可变性,设计并形式化开发了抽象的 Apla 泛型构件,并在 Apla->Java 程序生成系统的支持下,以构件组装的方式自动生成各种图算法程序,有效保证了生成

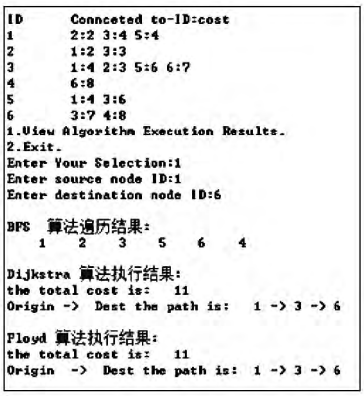

图 3　BFS、Dijkstra、Floyd 算法运行结果

算法程序的正确性,提高了算法程序的设计效率.

下一步的工作将深入探索本文设计的算法构件的理论拓展和潜在应用价值,以基于构件开发的新思路,进一步挖掘更加完善、抽象、可重用性高的算法构件,提高图算法的可靠性和开发效率.

5　参考文献

[1] McIlroy. M D Mass Produced Software Components [EB/OL]. [2015-11-17]. https://www.researchgate.net/publication/200827763_Mass_Produced_Software_Components.

[2] Gray J. What next? A dozen information-technology research goals [J]. Journal of the ACM, 2003, 50(1): 41-57.

[3] Batory D. Thoughts on automated software design and synthesis [C]. New York: ACM Press, 2010: 29-32.

[4] Xue Jinyun. A unified approach for developing efficient algorithm of programs [J]. Journal of Computer Science and Technology, 1997, 12(4): 314-329.

[5] Xue Jinyun. PAR method and its supporting platform [EB/OL]. [2015-11-17]. https://www.researchgate.net/publication/285841454_PAR_method_and_its_supporting_platform.

[6] 谢武平,薛锦云. Radl 算法到 Apla 程序的生成系统 [J]. 计算机研究与发展, 2014, 51(4): 856-864.

[7] 左正康,薛锦云. Apla 中泛型约束机制研究 [J]. 软件学报, 2015, 26(6): 1340-1355.

[8] Czarnecki K, Eisenecker U. Generative programming: methods, tools, and applications [M]. New Jersey: Addison-Wesley, 2000

[9] Leavens G T, Abrial J R, Batory D, et al. Roadmap for enhanced languages and methods to aid verification [C]. New York: ACM Press, 2006: 221-236.

[10] 范少锋,张乃孝. 生成式程序设计研究概述 [J]. 计算机科学, 2005, 32(3): 12-16.

[11] Batory D, Hofner P, Kim J. Feature interactions, products, and composition [C]. New York: ACM Press, 2011.

[12] Li Yulin, Novak G. Generation of geometric programs specified by diagrams [C]. New York: ACM Press, 2011.

[13] Fu J C, Bastani F B, Yen I. Iterative planning in the context of automated code synthesis [C]. California: IEEE-Computer Society Press, 2007.

[14] Xue Jinyun. Formal derivation of graph algorithmic programs using partition-and-recur [J]. Journal of Computer Science and Technology, 1998, 13(6): 553-561

[15] Xue Jinyun. Developing the generic path algorithmic program and its instantiations using PAR method [EB/OL]. [2015-11-17]. https://www.researchgate.net/publication/221323186_Developing_the_Generic_Path_Algorithmic_Program_and_its_Instantiations_Using_PAR_Method.

Components-Based Graph Algorithms Development and Generation

YAN Mengtian, SHI Haihe*

(College of Computer Information and Engineering, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: Software component technology not only significantly improves the reliability and development efficiency of programs, but also reduces the development cost. Generic programming contributes to reducing the complexity of programming, and supports the development of reusable components. Generative programming and generic programming are introduced here, and some characteristics and common features of graph algorithms are analyzed, based on which generic graph algorithm components are abstracted from a class of problems, such as the breadth-first search problem, single source shortest path problem and all vertex-pairs shortest path problem. Further, it is described by means of generic mechanisms of PAR, and several graph algorithm programs are generated via component assembly supported by PAR platform.

Key words: graph algorithms generation; component; generative programming; PAR method

(责任编辑: 冉小晓)