文章编号：1674－9057(2012)02－0271－05 doi:10.3969/j.issn.1674－9057.2012.02.022

基于贪婪策略的局部优化服务组合方法

黎辛晓，叶恒舟

（桂林理工大学 信息科学与工程学院，广西 桂林 541004）

摘 要：提出了一种不依赖于服务关系图、可支持实体之间继承与组合关系、高效的自动服务组合方法。该方法首先标记出必要的可满足服务，然后从中挑选出部分服务直接生成组合路径，从而减少时空开销，避免循环搜索与重复搜索。为达到局部优化的目标，标记可满足服务时运用了贪婪策略。仿真实验表明，该方法具有良好的时间复杂度，能够适用于大规模的语义服务组合。

关键词：Web服务；服务组合；局部优化；贪婪算法

中图分类号：TP311 **文献标志码**：A

Web服务是基于现有互联网协议与公开标准的自包含、自描述、模块化的应用，可以提供一个灵活解决应用集成问题的方案[1]。作为面向服务体系结构（service-oriented architecture，SOA）的实现技术，Web服务已经成为一种为模块化组件定义接口和通信协议的主流标准，逐步应用于各类水平和垂直产业[2]。服务组合用于在现有服务的基础上构造新服务或应用。服务组合方法可以是静态的、自动的或半自动的。静态组合费时、灵活性不足，因此，越来越多的研究人员关注自动或半自动组合方法，尤其是建立在服务依赖关系基础上的图搜索方法[3－5]。然而，当需要考虑语义信息时，构建并维护服务及服务实体之间的依赖关系图或矩阵比较复杂，需要较多的时间与空间，相应的搜索算法往往存在时间复杂度高、循环搜索、重复搜索、丢失解等问题，如搜索空间较小，避免循环搜索，但不支持语义的自动服务组合方法[6－7]；文献［8］在文献［6］的基础上，增加了语义支持，保留了时间复杂度较好、可避免循环搜索等特点；文献[9]能够支持语义继承关系但可能存在循环搜索;文献[10]支持语义组合关系,但可能丢失解;文献[11]采用了关系矩阵，也只能解决直接依赖服务之间的循环搜索问题;文献[12]解决了循环搜索问题，但存在重复搜索，最坏时间复杂度高。因此，本文提出了一种高效的Web服务自动组合方法，可以支持服务实体之间的继承与组合语义关系，避免重复搜索与循环搜索。

1 Web服务组合模型

对于给定的Web服务库与服务请求，Web服务组合（WSC）问题是EXP-hard[13]。因而，探讨简化的WSC模型是需要且值得的。本文所讨论的WSC问题仅仅关注在WSDL中描述的服务的输入与输出实体。

定义1 Web服务。一个Web服务可以描述为 $s=(EI,EO)$，其中 EI 表示 s 的输入实体集，EO 表示 s 的输出实体集。为方便讨论，假定所有实体都是经过本体标注过的，即所有的实体属于同一个集成本体，每个实体代表唯一的一个概念。

定义2 服务请求。一个Web服务请求可以描述为 $r=(RI,RO)$，其中 RI 表示用户给定的实体信息集合，RO 表示用户期望获取的实体信息集合。

通过对服务或请求的简单处理，可以保证：对

收稿日期：2011－08－28

基金项目：广西科学研究与技术开发计划项目（桂科攻10100002－2）；广西教育厅科研项目（201010LX171；200911LX131）

作者简介：黎辛晓（1980—），女，硕士，讲师，研究方向：计算机网络应用、制造业信息化，renpet@glite.edu.cn。

引文格式：黎辛晓，叶恒舟. 基于贪婪策略的局部优化服务组合方法［J］. 桂林理工大学学报，2012，32（2）：271－275.

于任一服务 s,$s.EI \neq \varnothing$,$s.EO \neq \varnothing$ 且 $s.EI \cap s.EO = \varnothing$;对于任一请求 r,$r.RI \neq \varnothing$,$r.RO \neq \varnothing$ 且 $r.RI \cap r.RO = \varnothing$。

定义 3 服务组合问题。一个 Web 服务组合问题可描述为 WSC $= (S,r)$,其中 $S = \{s_1,s_2,\cdots,s_n\}$ 表示所有候选服务的集合,$r = (RI,RO)$ 表示一个特定的用户请求集合。记 $ES = S \cup \{s_0,s_d\}$,其中 $s_0 = (\varnothing,r.RI)$,$s_d = (r.RO,\varnothing)$。

定义 4 实体关系。在本文的服务组合过程中,考虑两实体 EA、EB 之间的 3 种关系:

Equal(EA,EB) = true 当且仅当 EA 与 EB 表示同一概念;

Subtype(EA,EB) = true 当且仅当 EA 是 EB 的派生概念;

Component(EA,EB) = true 当且仅当 EA 是一个组合概念,且 EB 是 EA 的一个组成部件。

定义 5 实体匹配。对于 WSC(S,r) 中涉及到的任意两个实体 EA 和 EB,基于定义 4 描述的实体关系,EA 与 EB 的匹配关系定义为 Match(EA,EB) = true iff Equal(EA,EB) = true 或者 Subtype(EA,EB) = true 或者 Component(EA,EB) = true。为描述方便,设 ESA、ESB 为两个实体的集合,记

Match(ESA,EB) = true iff$\exists E \in ESA$ 且 Match(E,EB) = true;

Match(ESA,ESB) = true iff$\forall E \in ESB$ 都有 Match(ESA,E) = true。

定义 6 服务满足。对于 WSC (S,r) 中涉及到的任意一个服务 s,s 是否可以满足定义为 fs(s) = true iff Match$(r.RI,s.EI)$ = true。显然,fs(s_0) = true。

定义 7 服务条件满足。给定 WSC (S,r),设 SS 为 ES 的任一子集,s 为 ES 中的任一服务,定义 cs(SS,s) = true iff Match$(\bigcup_{w \in SS} w.EO, s.EI)$ = true。

定义 8 有效条件满足服务集。对于给定 WSC (S,r) 及 ES 的任一子集 SS,SS 的有效条件满足服务集定义为 ECSS$(SS) = \{s \mid s \in ES \;\&\&\; \text{cs}(SS,s) = \text{true} \;\&\&\; s.EO - \bigcup_{w \in SS} w.EO \neq \varnothing\}$。

定义 9 服务组合序列。WSC (S,r) 的一个解可以表示为一个服务序列$(s_0,s_1,s_2,\cdots,s_k,s_d)$,满足:

(1) $\forall i(i = 1,2,\cdots,k)$,cs$(\{s_0,s_1,s_2,\cdots,s_{i-1}\},s_i)$ = true;

(2) cs$(\{s_0,s_1,s_2,\cdots,s_k\},s_d)$ = true。

2 基于贪婪策略的服务组合方法

对于给定的 WSC(S,r),基于贪婪策略的服务组合方法主要包含 2 个步骤:1)针对用户请求,运行贪婪策略生成一个服务组合序列;2)优化生成的服务组合序列,同时生成服务流程。

2.1 生成服务组合序列

根据服务组合序列的定义,对于给定的 WSC (S,r),算法 GSCS 可判断问题是否有解,并在有解时生成一个服务组合序列,存放于队列 usedSq 中。

算法 1 GSCS(Generate Service Composition Sequence)

```
输入:服务组合问题 WSC(S,r)
输出:存放服务组合的队列 usedSq
Set KnownEs = null as a set of entities;
Set usedSq = null as a queue of services
Add s0. EO to KnownEs;
Add s0 to usedSq;
While (true){
If (CS(usedSq,sd) = = true)
    Add sd to usedSq and return;
计算当前有效条件满足服务集 es = ECSS(usedSq);
If (es = = null)
    Set usedSq = null and return;
在 es 中采用贪婪策略选择一个服务 s;
Add s. EO to KnownEs;
Add s to usedSq;
}
```

算法 1 循环搜索可满足的服务,直到目标服务变为可满足,或者再也找不到有效的可满足服务。若目标服务变为可满足,则队列 usedSq 为 WSC(S,r)的一个服务序列;否则,给定的 WSC(S,r)无解,这可参照文献[14]加以证明。

若不考虑服务选择时的贪婪策略,算法 1 的主要时间开销为实体匹配操作。在最坏情况下,每次循环需要检查所有尚未加入到 usedSq 中的服务,整个循环需要进行 n 次(n 为 S 中服务个数),因而时间复杂度为 $O(n^2)$。

2.2 贪婪策略

寻找涉及最小服务数量的服务组合序列是 Web 服务组合的目标之一。为此,在生成服务组合序列过程中,运用贪婪策略,优先选择那些最能满足期望或能够得到最多未知输出实体的服务。下述的式(1)用来度量服务 s 的效用,其中 SS

由定义 8 约定，是所有当前可满足并且当前执行后，可得到新的输出实体的服务的集合；$|s.EO - \bigcup_{w \in SS} w.EO|$ 表示服务 s 执行后，可得到新的输出实体的个数；$|(s.EO - \bigcup_{w \in SS} w.EO) \cap r.RO|$ 表示服务 s 执行后，可得到新的用户期望实体的个数；α 为 0 ~ 1 的一个常量，可根据用户偏好设定。在算法中选择服务的贪婪策略可表述为：分别计算 SS 中的任一服务 s 的 EONum(s,SS) 的值，优先选择值最大的那个服务。

$$\text{EONum}(s,SS) = \alpha |s.EO - \bigcup_{w \in SS} w.EO| + (1-\alpha)|(s.EO - \bigcup_{w \in SS} w.EO) \cap r.RO| \quad (1)$$

2.3　生成组合路径

由算法 1 生成的队列 usedSq 已经是给定的 WSC (S,r) 的一个服务组合序列，但该序列可能包含无用的服务，也没有给出服务的有效生成路径。

容易证明：对于 usedSq 中的任一元素 s（设其对应的索引值为 index），可以在 usedSq[0,…, index - 1] 中挑选出一些元素构成集合 SS，使 cs(SS,s) = true，$\forall e \in s.EI$，$\exists ps \in$ usedSq [0,…, index - 1]，且 Match($ps.EO$,e) = true。usedSq 的这个性质保证了算法 GSCP 的可行性与完备性。

算法 2　GSCP(Generate Service Composition Path)

输入：队列 usedSq

输出：表示组合路径的有向图 DG

```
Set EES = null 记录尚需扩展的服务;
Set EDS = null 记录已经在图 DG 中的服务
Add s_d to ESS and create node s_d;
While (! ESS.Empty()){
    从 ESS 中取出一个服务 s;
    计算 s 在 usedSq 中的索引 index;
  For each e ∈ s.EI{
    在 usedSq [0,…,index - 1] 搜索服务 ps 使 Match(ps.EO,e) = true;
    Create node ps and edge <ps,s>;
    If (ps not in EDS && ps ! = s_0){
      Add ps to ESS;
      Add ps to EDS;
    }
  }
}
```

设 usedSq 有 n 个服务，最多有 $n-1$ 个服务需要扩展（s_0 不需要扩展），每次扩展需要的时间为 $O(mn)$，其中 m 为服务中输出实体的最大值。由于 m 可以认为是常量，算法 2 的时间复杂度可表示为 $O(n^2)$。

3　实验分析

通过仿真实验验证该方法在不同服务总数与 IO 总数的服务规则库中进行服务组合时的可满足率、usedSq 中元素的个数、解树中节点个数和服务组合时间开销。由于当前缺少公共的 Web 服务标准库，通过程序模拟生成 Web 服务库及用户请求，并在相同实验环境下计算文献［7］的算法的解树中节点个数和时间开销，进行对比分析。

3.1　仿真实验准备

仿真所需要的 Web 服务库及用户请求采用随机生成办法。对于一个 WSC (S,r)，S 中的服务个数设定为 n，从 500 开始取值，每次增加 50；k 为 S 中服务总数与涉及到的不同实体总数的比值，k 的取值集为{1,1.5,2,2.5,3}。在实体中 5% 的实体与另外一个实体具有继承关系；3% 的实体是组合类型，并与另外的 2 ~ 4 个实体组合。每个服务随机选取 1 ~ 5 个实体作为输入集，再从剩下的实体中随机选取 1 ~ 5 个作为输出集；用户请求的输入与输出实体集采用与服务生成一致的方法。为减少特例影响，每种规模的 WSC (S,r) 都随机生成 100 次，取其平均值。实验环境为 2.70 GHz CPU，2 GB 内存，Windows XP，JDK1.6。

3.2　实验结果

图 1 ~ 4 显示了当 k 分别取值为 1、1.5、2、2.5、3 时，本文算法与文献［7］算法的服务组合有解时解树中包含的服务个数（没有统计 s_0 及 s_d）及时间开销。

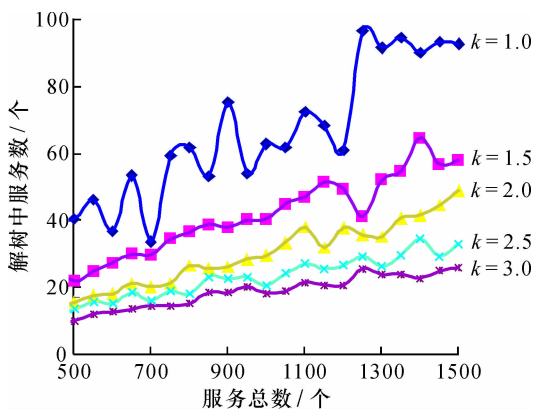

图 1　本文算法的解树中服务个数

Fig. 1　Service numbers in solution tree for the algorithm

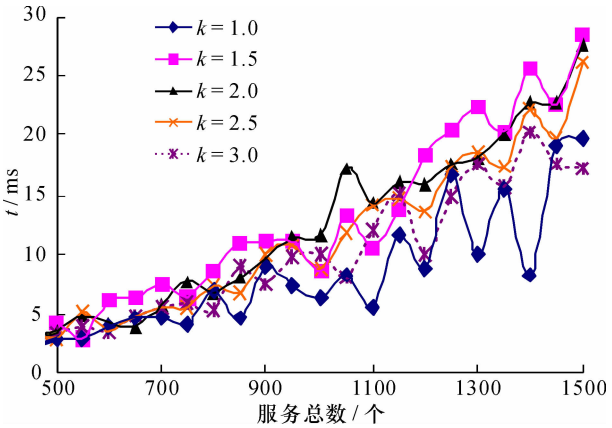

图 2 本文算法的服务组合时间开销

Fig. 2 Service composition time cost for the algorithm

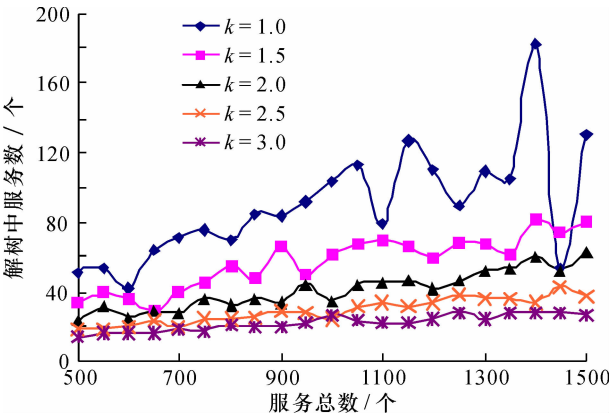

图 3 文献［7］算法的解树中服务个数

Fig. 3 Service numbers in solution tree for the algorithm［7］

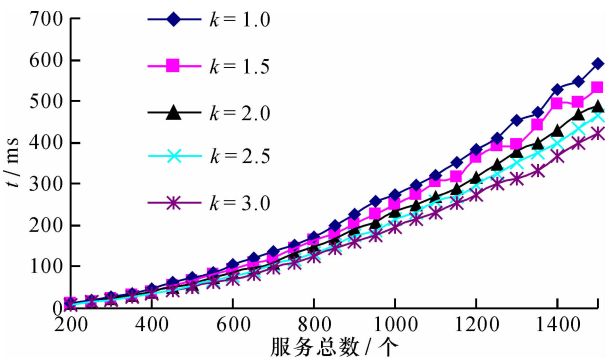

图 4 文献［7］算法的服务组合时间开销

Fig. 4 Service composition time cost for the algorithm［7］

3.3 数据分析与比较

图 1 表明服务组合解树的服务个数与 k 值相关，服务总数相同的情况下，k 值越大，服务个数越少。k 值反映服务库中服务之间的关联程度：k 值越大，服务之间耦合性越强，因而可满足率越大，可组合性也越强。图 1 还表明该方法可以得到比较复杂的组合解（最多时包含了 100 多个服务）。

图 2 反映了时间开销的变化规律，服务组合耗时随服务总数的增加而增大。服务总数越大，搜索空间越大，服务组合时间开销越大。由于通常并不需要搜索整个空间，实际时间开销低于 $O(n^2)$。

对比图 1 和图 3，本文算法与文献［7］算法的解树中服务个数随 k 值的变化规律类似，都是 k 值越大，服务个数越少。两种算法需要用到的服务数量相差不大。对比图 2 和图 4，本文算法与文献［7］算法的服务组合耗时都随服务总数的增加而增大，但前者的时间开销远小于后者。因为本文算法先标记出必要的可满足服务加入 usedSq 列表，再从中挑选部分服务直接生成服务组合路径。实验中，usedSq 中服务的个数大约是服务总数的 1/10，表明该算法可以在很大程度上缩小搜索空间，从而减少了时空开销。

本文算法需要用到的辅助存储空间为生成服务组合序列时用到的 usedSq 队列及当前有效条件满足服务集 es，空间复杂度为 $O(n)$。

实验表明，本文算法搜索空间小，避免环搜索和重复搜索，算法实现简单，具有较好的时间复杂度，达到了局部优化的目标，能够适用于大规模的语义服务组合。

4 结 论

本文通过简化 Web 服务组合模型，提出了一种高效的 Web 服务自动组合方法。与传统的依赖于服务关系图的图搜索方法不同，该方法直接有序标记服务库中的可满足服务，然后通过计算相关服务的前趋集直接构造组合解树。与现有图搜索方法相比，该方法具有如下特点：能够支持实体概念之间的继承与组合关系；无需构造并维护服务关系图；无需进行复杂的图搜索；可以快速判断服务组合是否有解；在服务组合有解时，可确保找到一个解，可以直接生成解树。

参考文献：

［1］ Wang H, Huang J Z, Qu Y, et al. Web services: problems and future directions［J］. Journal of Web Semantics, 2004, 1 (3): 309 – 320.

[2] Zhang L J. EIC editorial: Introduction to the body of knowledge areas of services computing [J]. IEEE Transactions on Services Computing, 2008, 1 (2): 62 – 74.

[3] Aydogan R, Zirtiloglu H. A graph-based web service composition technique using ontological information [C] //Proc. IEEE International Conference on Web Services (ICWS 07). Salt Lake City, UT, 2007: 1154 – 1155.

[4] Ramasamy V. Syntactical & semantical web services discovery and composition [C] //Proc. the 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE 06). San Francisco, 2006: 68 – 68.

[5] Calado I, Barros H, Bittencourt I I. An approach for semantic web services automatic discovery and composition with similarity metrics [C] //Proc. the 2009 ACM Symposium on Applied Computing (SAC 09). New York, 2009: 694 – 695.

[6] 叶恒舟, 罗晓娟, 牛秦洲. 基于归约图的 Web 服务自动组合 [J]. 桂林工学院学报, 2009, 29 (3): 395 – 401.

[7] 叶恒舟, 罗晓娟, 牛秦洲. 基于与或图的 Web 服务自动组合 [J]. 计算机工程与设计, 2010, 31 (11): 2645 – 2647

[8] 叶恒舟, 罗晓娟, 牛秦洲. 基于归约图的语义 Web 服务自动组合 [J]. 桂林理工大学学报, 2010, 30 (3): 441 – 444.

[9] Yan Y X, Xu B, Gu Z F. Automatic Service Composition Using AND/OR Graph [C] //Proc. 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services (CEC/EEE 08). Washington DC, 2008: 335 – 338.

[10] Liang Q A, Stanley Y W Su. And/or graph and search algorithm for discovering composite web services [J]. International Journal of Web Services Research, 2005, 2(4): 48 – 67.

[11] Omer A M, Schill A. Web services composition using input/output dependency matrix [C] //Proc. the 3rd Workshop on Agent-oriented Software Engineering Challenges for Ubiquitous and Pervasive Computing (AUPC 09). New York, 2009: 21 – 26.

[12] 邓水光, 吴健, 李莹, 等. 基于回溯树的 Web 服务自动组合 [J]. 软件学报, 2007, 18 (8): 1896 – 1910.

[13] Kil H, Nam W, Lee D. Computational complexity of web service composition based on behavioral descriptions [C] //Proc. the 2008 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 08). Washington DC, 2008: 359 – 363.

[14] 叶恒舟, 罗晓娟, 牛秦洲. Web 服务请求输入闭包与可满足性分析 [J]. 微电子学与计算机, 2011, 28 (7): 168 – 170.

Local Optimal Service Composition Approach Based on Greedy Algorithm

LI Xin-xiao, YE Heng-zhou

(*College of Information Science and Engineering*, *Guilin University of Technology*, *Guilin* 541004, *China*)

Abstract: Web service composition is the key technology to achieve the service-oriented computing. However, the research on composition approach with high efficiency is still a critical challenge for semantic, large scale web services. Unlike traditional graph based methods, an effective approach is proposed for automatic service composition which does not depend on service dependency graph and can support for inheritance and composition relationships between entities. Firstly, it marks out necessary and satisfied services, and then selects parts of them to generate a composition path directly. It can reduce the time and space overhead, avoid circulation and repetition search. To achieve the goal of local optimization, greedy strategy is used to mark satisfied services. Experimental results show that this approach has a good level of time complexity and can be applied to large-scale semantic service composition.

Key words: web service; service composition; local optimization; greedy algorithm