Computer Science and Application 计算机科学与应用, 2022, 12(5), 1458-1473
Published Online May 2022 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2022.125145

Hans汉斯

应用于服务组合的改良粒子群算法

丁　洋[1,2]，王红斌[1,2*]

[1]昆明理工大学信息工程与自动化学院，云南　昆明
[2]昆明理工大学云南省人工智能重点实验室，云南　昆明

收稿日期：2022年4月27日；录用日期：2022年5月24日；发布日期：2022年5月31日

摘　要

当今社会，互联网、物联网、云计算以及大数据的快速发展和普及使得网络应用越来越广，越来越多的行业和相应的网络进行越来越深地融合。许多原本属于线下的服务经过封装后被搬上了网络上，同类型的服务也越来越多。如何在互联网大数据环境下快速找到满足用户个性化需求的服务组合已经成为亟需解决的问题。为了解决这个问题，本论文提出了一种改良粒子群服务组合方法，此改良粒子群算法根据服务组合问题的特点分别从四个方面加入了逃出局部最优的机制，根据用户的服务组合请求去快速组合出更优的服务组合方案。本文通过实验与其他相关的服务组合优化算法在最优性、时间复杂度以及收敛性三个指标进行了对比。根据实验结果分析，本文所提出的方法在最优性、时间复杂度以及收敛性三个方面整体上表现出来良好的性能。

关键词

服务组合，服务质量，服务计算，优化算法，粒子群算法

Improved Particle Swarm Optimization for Service Composition

Yang Ding[1,2], Hongbin Wang[1,2*]

[1]Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming Yunnan
[2]Yunnan Key Laboratory of Artificial Intelligence, Kunming University of Science and Technology, Kunming Yunnan

Received: Apr. 27th, 2022; accepted: May 24th, 2022; published: May 31st, 2022

Abstract

In today's society, the rapid development and popularization of the Internet, Internet of Things,

*通讯作者。

文章引用: 丁洋, 王红斌. 应用于服务组合的改良粒子群算法[J]. 计算机科学与应用, 2022, 12(5): 1458-1473.
DOI: 10.12677/csa.2022.125145

cloud computing, and big data make network applications increasingly widely used. And increasing industries and corresponding networks are integrated increasing deeply. Many services that originally belonged to the line have been packaged and put on the Internet, and there are increasing services of the same type. How to quickly find a service combination that meets the personalized needs of users in the Internet big data environment has become an urgent problem to be solved. In order to solve this problem, this paper proposes an improved particle swarm service composition method. According to the characteristics of the service composition problem, the improved particle swarm algorithm adds a mechanism of escaping from the local optimum from four aspects. According to a user's service composition request, quickly compose a better service composition solution. This paper compares the optimality, time complexity and convergence with other related service composition optimization algorithms through experiments. According to the analysis of the experimental results, the method proposed in this paper shows good performance on the whole in terms of optimality, time complexity and convergence.

Keywords

Service Composition, Quality of Service, Service Computing, Optimal Algorithm, Particle Swarm Algorithm

1. 引言

随着云计算[1]、IoT [2]以及 IoD [3]等技术的发展，越来越多的线下服务通过虚拟化技术加入到线上，同线上服务一起协同为用户提供服务。这些线上服务越来越多，需求越来越复杂，逐渐构成了 IoS [4]或 Big service [5]以及跨界服务[6]；为了满足用户的复杂需求，需要进行服务组合。其中面临的挑战就是面对复杂的需求和海量的可用服务，如何通过服务组合技术快速高效地构建服务组合方案。单一的 Web 服务往往不能很好地满足复杂的客户需求，服务组合变得相当重要[7]。在进行服务组合的过程中，每一种类型的服务存在大量功能相同而 QoS (Quality of Service)不同的候选服务，这些候选服务就构成了对应服务类型的候选服务集。QoS 感知的服务组合已经成为研究的热点。另外，互联网上的服务对应存在着大量的服务历史记录，若能利用这些服务历史记录作为服务组合的先验知识去指导服务组合，这将大大提高服务组合的效率。目前的 Web 组合方法一般都是基于 QoS 感知的 Web 服务组合方法，针对基于 QoS 感知的 Web 服务组合优化问题，目前的解决方法很多，大致可以分为三大类[8]：

1) 局部搜索方法[9] [10]，不同的 QoS 属性值通过一个聚合函数被映射为单一指标，参照多指标从每个候选服务集中选择一个具体的服务来构成一个服务组合方案。这种方法对于有少量 QoS 属性限制条件的服务组合有效果，但对于大量的有 QoS 属性限制的服务组合效率较低。

2) 全局优化方法[11] [12] [13]，这种方法将整个服务组合问题转化为混合线性规划问题，通过一个线性函数寻找解决方案。这种方法对选择最优组合服务的算法有一定的限制，所求的目标函数必须是线性的，不能完全用于解决 Web 服务组合问题。

3) 智能优化算法[14] [15] [16] [17]，该方法解决了全局优化算法的缺点，这也是目前用来解决 Web 服务组合问题的主流方法。

以上服务组合方法在面对海量需求和可用服务场景下，效率和性能不能满足用户的需求；本文的解决思路是根据服务组合问题和服务数据集的特点设计出具有逃出局部最优即预防早熟机制的改良离散粒子群服务组合算法 IDPSO (Improved Discrete Particle Swarm Optimization)，该算法分别从对候选服务集进行排序、一定迭代次数后重置粒子位置、合理设置算法控制参数、设置参照最优性四个方面来预防粒子群算法早熟。改进后的粒子群算法有效克服了容易陷入局部最优的窘境，其组合效率得到了很大的提高。

2. 相关工作

基于 QoS 感知的服务组合问题有两个重要的衡量指标，时间复杂度 Time Complexity 和最优性 Optimality。时间复杂度反映的是算法的时间性能，简单的来说是算法所执行的时间，最优性反映的是算法的寻优能力。一般情况下，时间复杂度和最优性是相互矛盾的，寻求高的时间效率会牺牲算法的最优性，反之亦然。所以，如何在时间复杂度和最优性之间找到一个平衡点去满足客户的需求是服务组合的一个关键问题。服务组合就是从候选集合中选择相应的具体服务去组合成满足客户所提的 QoS 限制的方案，这是一个 NP-hard 问题[18]。目前的群智能优化算法对于连续型的问题寻优效果相当好，但是对于 Web 服务组合这一类离散型问题就显得捉襟见肘了。所有的群智能优化算法都存在陷入局部最优的问题，因为群智能优化算法在寻找最优解的过程中遵循一定的方向和顺序性，所以群智能优化算法对于连续型的问题表现出优良的效果。文献[19]给出有关联关系的服务模型，使用遗传算法进行服务组合方案的查找，提出了一种云制造中有关联意识的服务组合方案选择方法。文献[20]使用粒子群优化算法来解决 Web 服务组合优化问题。作为解决 Web 服务组合优化问题的典型算法，与遗传算法相比，粒子群优化算法具有参数少、收敛快的特点，在很多寻优问题上表现出良好的性能，但粒子群算法也会出现早熟现象，容易陷入局部最优问题[21]。文献[22]提出了一种将候选服务分类的方法来解决大规模 Web 服务组合问题，此方法提出了一个缩减模型来减少服务候选集的规模，以企图降低问题的搜索空间来提高寻优效率，此方法需要每次根据新的服务组合问题对新的候选服务集进行缩减，这样就降低了算法的整体效率。文献[16]提出了利用人工蜂群算法来解决 Web 服务组合问题，人工蜂群算法相对于粒子群算法其参数更少，但是其寻优性能略低一些。文献[23]提出了任务粒化的质量约束感知的服务组合方法，该方法把服务组合模型分为 4 个层，最下层为基本层，对应的是服务组合的候选服务集；最上层为任务粒层对应的是服务组合中分割出的任务；任务粒化的分层方法一定程度上提高了组合效率，但是割裂了服务组合方案的整体约束，因此寻找出的服务组合方案未必是最优的。

本文中提出的改良粒子群优化算法引入了逃出局部最优的相关机制，可以很好的避免陷入局部最优以实现寻找到全局最优解。理论分析和实验表明，在 Web 服务组合优化问题上，本文所提出的方法有更好的综合性能。

3. Web 服务组合问题描述

本部分主要阐述 Web 服务组合中 QoS 的定义以及 Web 服务组合问题的定义。

3.1. QoS 的定义

Web 服务质量 QoS 是一个 6 元组 $QoS=(p,r,a,rep,t,rel)$，其中：p 是服务价格 price，指调用一次服务所需要的费用，r 是响应时间 response time，a 是可用性 availability，指服务执行成功的次数与总执行次数的比率，rep 是信誉度 reputation，$rep=\left(\sum_{i=1}^{n} sc_i\right)/n$ 是用户对当前服务 n 次评价得分 $(sc_1,sc_2,\cdots,sc_i,\cdots,sc_n)$ 的平均值[20]，t 是吞吐量 throughput，rel 是可靠性 reliability，指服务能运行的时间与总时间的比率。

3.2. 服务组合问题的定义

服务组合的目的是在满足用户功能需求和 QoS 总体约束下，针对特定的优化目标，得到一组最优的服务集合，构成组合服务。组合服务是一个 6 元组 $cs=(T,R,QoS,QC,W,EF)$，其中：$T=\{t_1,t_2,\cdots,t_j,\cdots,t_n\}$ 表示构成组合服务的任务集合，关系 $R\subset T*T$ 表示任务之间时序关系集合，QoS 是组合服务的总体质量，$QC=\{C(p),C(r),C(a),C(re),C(t),C(rel)\}$ 表示组合服务必须满足的质量约束，$w=\{w_1,w_2,\cdots,w_6\}$ 是集合 QoS 中各子属性对应的权重并且满足 $\sum_{i=1}^{6}w_i=1$，EF 是组合服务质量评价函数即最优性函数。任务集 T 中的任务 t_j 是抽象服务，在组合服务 CS 执行前，从 t_j 的候选服务集 CSS_j 中选择一个具体的 Web 服务 as_k^j 代替抽象服务 CSS_j，as_k^j 表示 CSS_j 中第 k 个候选服务，$k\in\{1,2,\cdots,|CSS_j|\}$。组合服务的生成过程是根据客户提交的任务信息来寻找满足客户需求的组合服务过程。组合服务生成的过程主要分为两个步骤进行，即服务选择与服务组合。在服务寻找过程中，将客户提交的任务 T 抽象为多个子任务 $\{t_1,t_2,\cdots,t_j,\cdots,t_n\}$，各个子任务间有不同的执行关系，工作流管理联盟(WFMC)定义了 4 种基本执行关系模型，即顺序(Sequence)、并行(Parallel)、选择(Selective)、循环(Circular)等。在定义的基本关系模型中，并行、选择与循环模型均可以转换为顺序模型，本文选取顺序模型作为研究对象。

由于客户对于 QoS 属性有个性化的要求，在对组合服务进行评估时，根据客户的偏好以及这个组合服务的整体 QoS 值，使用组合评价函数 EF 去评估这个组合服务的解决方案，选出最佳的组合服务。由于 Web 服务的 QoS 每个属性取值范围有较大的差别，不在同一计算等级上，在进行 Web 服务评估之前必须先将 QoS 属性值进行预处理。本文使用被广泛采纳的文献[20]中的方法对 Web 服务的属性值进行预处理，将属性值区间限定为[0, 1]，经过预处理后的 QoS 各子属性值越小代表服务越优。

4. 离散粒子群改良算法 IDPSO

当客户提交服务组合请求时，需要采用合适的算法进行服务组合，现有的粒子群算法普遍具有陷入局部最优的问题，其寻优性能不太理想，为此，本文根据 Web 服务问题的特点进行了深入的分析和研究，设计出了针对 Web 服务问题具有很好性能的离散粒子群改良算法 IDPSO，下面给出了粒子的相关概念以及 IDPSO 算法的具体分析和设计过程。

4.1. 标准粒子群算法

粒子是存在于 n 维欧氏空间中无体积、无质量的对象，粒子群是是空间中 m 个粒子的集合[20]。粒子群算法是一种启发式进化计算技术，来源于对简化社会群体智能行为模型的模拟，由 Eberhart 和 Kennedy 于 1995 年提出[24]在标准粒子群算法 SPSO(Standard Particle Swarm Optimization)下，在第 $t+1$ 代时，第 i 个粒子的第 j 维速度和位置迭代更新公式如公式(1)、(2)表示：

$$V_{i,j}^{t+1}=w*V_{i,j}^{t}+c_1r_1\left(p_{i,j}-X_{i,j}^{t}\right)+c_2r_2\left(p_{g,j}-X_{i,j}^{t}\right) \tag{1}$$

$$X_{i,j}^{t+1}=X_{i,j}^{t}+V_{i,j}^{t+1} \tag{2}$$

其中非负常数 C_1，C_2 是学习因子，决定 $p_{i,j}$ 和 $p_{g,j}$ 对新速度的影响程度，r_1 和 r_2 是区间[0, 1]上均匀分布的随机变量。w 值越大，全局寻优能力越强。目前采用较多的是 Shi 建议的线性递减惯性权重值[24]，如公式(3)所示：

$$w=w_{\min}+\left(w_{\max}-w_{\min}\right)/\left(T_{\max}-t\right)/T_{\max} \tag{3}$$

其中 $w_{\max}$ 为初始惯性权重值，$w_{\min}$ 为最大迭代次数对应的惯性权重值，一般取 $w_{\max}=0.9$，$w_{\min}=0.4$。

在进行服务组合的选择时本文将一个粒子看作一个服务组合解决方案，然后去寻找最优组合服务方

案。本文分解的抽象子任务 $T=\{t_1,t_2,\cdots,t_i,\cdots,t_n\}$ 代表粒子的各维搜索空间，n 代表分解的抽象子任务的个数，同时 n 也表示粒子的搜索空间有 n 维。每个抽象子任务 t_i 对应的候选服务集 CSS_i 中 Web 服务个数设为 m_i 个，即表示粒子在这一维搜索空间中的最大向量长度，$i\in[1,n]\wedge i\in N^+$，CSS_i 中的每一个元素为单元服务 as_j^i，$j\in[1,m_i]\wedge j\in N^+$，具体编码模式如图 1 所示：

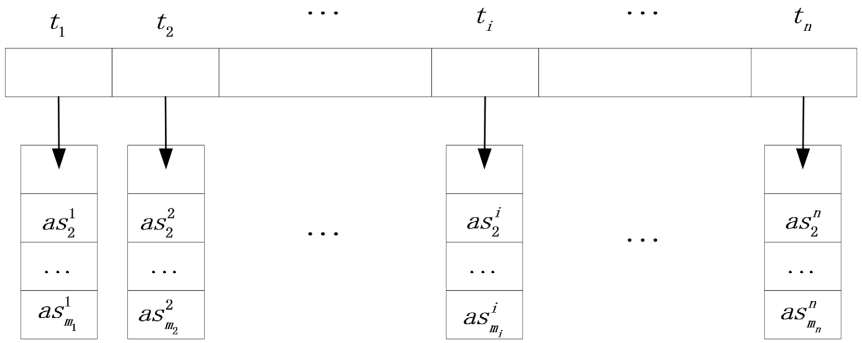


Figure 1. Particle coding in service composition
图 1. 服务组合中粒子编码

粒子选出对应的各个维的向量值后就完成了 Web 服务组合解决方案的选择。通过适应度值评估函数对每一个粒子所对应的 Web 服务组合解决方案进行评估，来寻找最优解的服务组合解决方案。使用这种编码方式，Web 服务组合选择问题被转化为 n 元向量的求解问题，即寻找最优向量 $(X_1,X_2,\cdots,X_i,\cdots,X_n)$ 过程。任务 $T=\{t_1,t_2,\cdots,t_i,\cdots,t_n\}$ 为组合服务 CS 的任务集合，T 中各抽象服务对应的候选数量分别为 $m_1,m_2,\cdots,m_i,\cdots,m_n$ 且 $m_i\in N^+\wedge i\in[1,n]$，则组合服务解空间大小为 $\prod_{i=1}^{n}m_i$，这也是是采用穷举法求最优解的时间复杂度，当任务 T 中子任务数量以及每一个子任务 t_i 对应的候选服务 m_i 的值非常大时，其时间复杂度是非常庞大的，这也是服务组合不能采用穷举法的直接原因。IDPSO 算法将粒子在其所有维度上的运动区间设置为连续的圆周轨道，当采用适应度函数评价粒子位置的优劣时，用约数函数将粒子的具体位置映射到离散区间上。

4.2. IDPSO 算法

针对服务组合问题，使用改良的粒子群算法 IDPSO 来寻找最优的组合服务方案。本文首先采用如图 1 所示的编码方式对种群粒子进行编码处理。本算法设置相应的预防早熟机制，在必要的时机进行操作以便使粒子逃出局部最优位置，根据适应度函数 $f(X)$ 对种群粒子选出的组合服务方案进行评估，选出种群中最优的服务组合方案。

4.2.1. 适应度函数设计

在 Web 服务组合方案的选择中，服务组合方案的整体 QoS 对服务评估有较大影响，根据服务组合方案中单个节点服务的 QoS 属性值进行聚合计算，得出服务组合方案的整体 QoS 指标。适应度函数作为 Web 服务组合方案的评估函数，计算方法有多种，本文中聚合函数和适应度值计算函数为：

$$Q_{cs}^r=Aggregation(cs_r) \tag{4}$$

$$fitness=\sum w_r*Q_{cs}^r \tag{5}$$

其中，cs_r 是当前服务组合方案的第 r 个属性，$Aggregation(cs_r)$ 是对服务组合方案 cs 中每一个原子服务的第 r 个属性进行聚合，w_r 表示客户对于 Web 服务组合方案的第 r 个 QoS 属性的偏好即权重值，n 为服

务组合方案的 QoS 属性总个数。本文使用文献[20]提出的组合服务属性值的计算方法，在对服务组合方案 QoS 属性值进行预处理，得出顺序模型下的组合服务方案整体 Q_{cs}^{r} 属性值，本文中的适应度值 *fitness* 越小代表最优性越优。

4.2.2. 预防早熟机制

1) 对候选服务集进行排序

通过本文对粒子群算法进行分析后发现，粒子群算法运行时粒子群中的每一个粒子首先会随机初始化一个位置，然后每一个粒子会在各种的运行轨道上沿着特定的方向上进行运动，因此粒子群算法对于最优解有一定方向秩序的问题表现出很好的性能，如图 2 所示的利用粒子群算法求抛物线函数 $y=f(x)$ 最小函数值的位置，粒子群中某一个粒子的位置当前在 x_0 的位置，此时这个粒子只有继续不断的向右移动就可以逼近最低点 x_1 的位置，进而就可以寻找到最小函数值的位置。对于这种有秩序的问题，粒子群中至少有一个粒子的方向恰好与解的方向一致，此时寻优问题就迎刃而解。

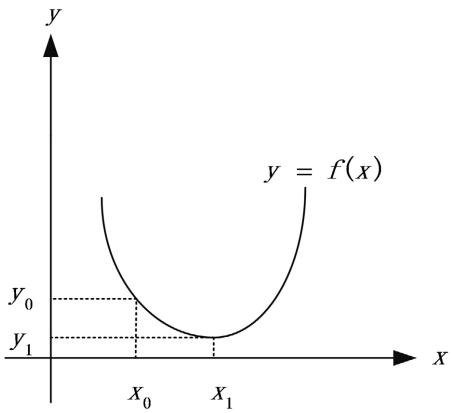


Figure 2. An example of finding the optimal solution
图 2. 求最优解示例

基于上述发现的粒子群算法的一个特点，本文试图对使用 IDPSO 算法的服务组合问题中的某个对象或者某些对象进行排序，IDPSO 算法操纵的基本对象为候选服务集，因此本文需要对候选服务集 CSS_i 进行排序，对服务候选集 CSS_i 怎么排序是一个问题，经过本文深入分析发现，IDPSO 算法对候选服务集的操纵最终体现在 CSS_i 中的每一个原子服务 as_j^i 的每一个 QoS 属性，因此本文进一步想到根据 as_j^i 中的某一个或某几个 QoS 属性对 CSS_i 进行排序，究竟根据 as_j^i 中哪一个或哪几个 QoS 属性对 CSS_i 进行排序呢，as_j^i 中的每一个 QoS 属性对应着权重 w，哪一个 QoS 对应的权重越大，表明这个 QoS 对服务组合方案的影响即最优性的影响越大，因此本文根据具有最大权重的 QoS 属性对 CSS_i 中的 as_j^i 进行排序。根据前面的定义可知，$QoS=(p,r,a,re,t,rel)$，对应的权重为 $w=(w_1,w_2,w_3,w_4,w_5,w_6)$，即属性 p 的权重为 w_1，属性 r 的权重为 w_2，属性 a 的权重为 w_3，属性 re 的权重为 w_4，属性 t 的权重为 w_5，属性 rel 的权重为 w_6，假如 $w=(0.4,0.1,0.1,0.2,0.1,0.1)$，那么本文就根据第 1 个属性 p 对候选服务集 CSS_i 进行升序排序，假如 $w=(0.1,0.2,0.3,0.1,0.2,0.1)$，那么本文根据第 3 个属性 a 对候选服务集 CSS_i 进行降序排序。QoS 的前 2 个属性为反向属性即属性值越大越不优，后 4 个属性为正向属性即属性值越大越优，所以若根据前 2 个属性中的任一个属性对 CSS_i 进行排序，那么就需要进行升序排序，若根据后 4 个属性的任意属性对 CSS_i 进行排序，那么就需要进行降序排序。最后，本文通过相关试验表明，本文采用以上的策略对候选服务集 CSS_i 的排序方法对预防 IDPSO 早熟的问题起到了一定的作用。

2) 一定迭代次数后重置粒子位置

粒子早熟意味着粒子陷入局部最优长时间无法逃离，粒子群算法开始运行时，首先会给每一个粒子随机生成一个位置，然后每一个粒子会在各自的运行轨道上扫描查找最优位置，假如当前所有粒子各自的运行轨道恰好绕过了全局最优或更优位置，那么整个粒子群算法就陷入了局部最优，即粒子算群法出现了早熟。如图 3 所示，此粒子群有三个粒子组成，分别为 p_1， p_2， p_3， X_{op} 为整个空间中最优位置或者更优位置， X_{op} 在粒子 p_1 的运行轨道和粒子 p_2 的运行轨道之间的某一位置，三个粒子 p_1， p_2， p_3 各自的运行轨道上运行，三个粒子的运行轨道恰恰绕过了 X_{op} 的位置，没有一个粒子的运行轨道扫过 X_{op} 的位置。此时，这种情况意味着粒子陷入了局部最优，无论此后多久粒子都不会找到全局最优或者更优位置的 X_{op}。因此当粒子群陷入局部最优时，本文需要检测出或预测出粒子陷入局部最优，然后使其逃出局部最优的情况。为此，本文需要分两步走来促使粒子逃出局部最优位置，第一步，检测出或者预测出粒子陷入了局部最优的时刻，直接检测出粒子陷入了局部最优位置是非常困难的，因此本文采用预测粒子陷入局部最优的策略，当然直接预测粒子陷入局部情况也是有困难的，本文采用间接的预测，本文知道当粒子圈算法运行足够长的时间其最优性值仍没有发生变化，此时意味着大概率陷入了局部最优，因此当粒子群算法经过足够长的迭代次数后其最优性值没有改变，本文就判定为陷入局部最优，其这个经过的足够长的迭代次数设为 t_{pre}， t_{pre} 满足 $t_{pre} \in \left[(1/5)T_{\max},(1/3)T_{\max}\right] \wedge t_{pre} \in N^+$，第二步，当预测出粒子群算法陷入局部最优时，立刻进行干预使其逃出局部最优位置，显而易见，事先本文并不知道全局最优位置到底在什么地方，对于本文来说最优位置或者更优位置具有一定的盲目性，从概率上来讲就是随机性，所以当粒子群算法陷入局部最优时，本文采用随机重置的方法来重置每一个粒子的位置，尽可能的使某一个或某些粒子的运动轨道恰好扫过最优位置或更优位置。

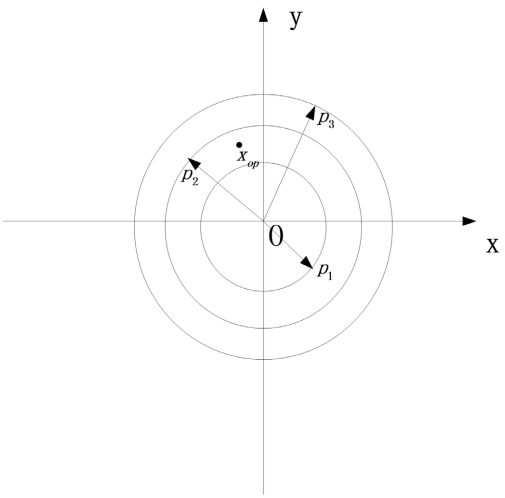


Figure 3. Particle trap into local optimum example
图 3. 粒子陷入局部最优示例

3) 合理设置 IDPSO 的控制参数值

IDPSO 算法的控制参数控制着 IDPSO 算法的执行，因此 IDPSO 算法的某个参数或者某些参数可能影响着算法的收敛情况，即影响着时间复杂度 Time Complexity 和最优性 Optimality。IDPSO 的五个控制参数分别为 PS、$T_{\max}$、c_1、c_2、w、v_{ini}，PS 代表粒子群的规模即粒子个数，$T_{\max}$ 为 IDPSO 算法的最大迭代次数即算法的最大迭代次数，c_1、c_2 两个参数为非负常数是学习因子，决定全局粒子最优位置和当前粒子历史最优位置对粒子速度的影响，w 是惯性权重，v_{ini} 为粒子的初始化速度。显而易见，参数 PS

和 $T_{\max}$ 直接影响着时间复杂度和算法的最优性，即 $PS*T_{\max}$ 的值越大更有利于最优性，但同时会增加算法的时间复杂度，因此需要对 $PS*T_{\max}$ 值进行一个取舍，使其在客户可以容忍的时间内尽量保证算法的最优性即在客户可以容忍的时间内使 $PS*T_{\max}$ 足够大，本文知道 IDPSO 算法的所有组合服务方案解的空间大小为 $\prod_{i=1}^{n} m_i$ ，当 $PS\times T_{\max}=\prod_{i=1}^{n} m_i$ 时，IDPSO 算法穷尽了整个解的空间，此时是可以找到全局最优解，当 $\prod_{i=1}^{n} m_i$ 很大时， $PS*T_{\max}$ 是不可以等于 $\prod_{i=1}^{n} m_i$ 的，经过本文相关试验表明，当 $PS*T_{\max}$ 至少取值为 $(1/1000)\prod_{i=1}^{n} m_i$ 时，IDPSO 算法可以取得很好的最优性，因此本文设置 $PS*T_{\max}\approx(1/1000)\prod_{i=1}^{n} m_i$ ，此时 IDPSOS 算法只要穷尽整个解空间的大约千分之一就可以取得很好的最优性，当客户对 $PS*T_{\max}\approx(1/1000)\prod_{i=1}^{n} m_i$ 的值仍然不可容忍时可以根据客户的需求适当下调。另外，当 $PS*T_{\max}$ 设定后，PS 与 $T_{\max}$ 值如何分配也是一个问题，若 PS 值分配比值过多，意味着迭代次数 $T_{\max}$ 过少，此时不利于 IDPSO 算法随机重置粒子位置，可能会影响算法的最优性，若 $T_{\max}$ 分配比值过大，意味着 IDPSO 算法随机重置粒子位置的次数会多，如果先前某个粒子或某些粒子的轨道会覆盖全局最优解，但还没来得及扫描到这个最优解就被重置了，这样也不利于 IDPSO 算法的寻优能力，因此本文按照 1:1 的方式去分配 PS 和 $T_{\max}$ 的值。关于 c_1、c_2 值的设置，c_1、c_2 是学习因子，决定历史全局最优位置和当前粒子历史最优位置对新速度的影响，若 c_1、c_2 的值较大，则意味着粒子群历史全局最优位置和当前粒子历史最优位置对新速度的影响就较大，反之亦然，c_1、c_2 的值目前没有很好的设定标准，本文的建议是首先为 c_1、c_2 设置很大的值，如果此时 IDPSO 算法的最优性值频繁的改变，意味着历史全局最优位置和当前粒子历史最优位置实际上对算法搜寻新的最优性值的帮助很大，因此，此时本算法需要保持或增加历史全局最优位置和当前粒子历史最优位置对算法寻优的影响力，所以本文可以选择不改变 c_1、c_2 的值或适当增加 c_1、c_2 的值，如果 IDPSO 算法的最优性值很多迭代次数后仍未改变，意味着历史全局最优置和当前粒子历史最优位置实际上对算法搜寻新的最优性值的帮助不大或无帮助，因此，此时本文需要减小历史全局最优位置和当前粒子历史最优位置对算法寻优的影响力，所以本文可以适当减小 c_1、c_2 的值。关于参数 w 值的设定，参数 w 值可以根据公式(3)进行设定。参数 V_{ini} 为粒子初始化速度，由于全局最优位置对于本文来说具有一定的盲目性，因此本文采用随机策略为参数 V_{ini} 设置一个值，随机策略就是随机生成一组整数来作为 V_{ini} 的值。

4) 设置参照最优性

在整体与组成部分的关系中，本文知道组成部分最优并不意味着整体的最优，但是最优的组成部分组合而成的整体是比较优的，所以最优的组成部分对于整体最优具有一定的参考价值，本文把这种理念应用到 IDPSO 算法中，前面本文提到任务 $T=\{t_1,t_2,\cdots,t_i,\cdots,t_n\}$ ，其对应的候选服务集为 $CSS=\{CSS_1,CSS_2,\cdots,CSS_i,\cdots,CSS_n\}$ ，即子任务 t_i 对应的候选服务集为 CSS_i ，首先本文对每一个子任务 CSS_i 的候选服务集 t_i 中的单元服务 as_j^i 计算单元适应度值，其计算公式如下：

$$fitness_{as}=\sum_{r=1}^{n} w_r*Q_{as}^r \tag{6}$$

其中 w_r 为单元服务 as 的第 r 个属性的权重，Q_{as}^r 为单元服务 as 第 r 个属性，然后选出每一个候选服务集 CSS_i 中单元适应度值最小的单元服务 as_{MF} ，组成参照最优服务组合方案 CS_{ref} ，把方案 CS_{ref} 用公式(4)、(5)计算出适应度值，本文称为参照适应度值 $fitness_{ref}$ 。当 IDPSO 算法执行最后一次迭代后，用得到的全局最优粒子的适应度值 $fitness$ 与参照适应度值 $fitness_{ref}$ 作比较，若 $fitness<fitness_{ref}$ ，则表明粒子群算法选出了全局最优或更优的解决方案，此时使用全局最优粒子对应的解决方案，若 $fitness>fitness_{ref}$ ，则表明粒子群算法实际上还是陷入了局部最优，且粒子群算法选出的最优方案并不是很好，如果此时客户不能接受粒子群算法继续寻优所导致的长时间的等待，则此时应选择参照最优服务组合方案 CS_{ref} 作为服务组合的最优方案，若 $fitness=fitness_{ref}$ ，则表明粒子群算法选出的解决方案的最优性与参照最优服务组合方案 CS_{ref} 的最优性相当，粒子群算法可能仍陷入局部最优，如果此时客户不能接受粒子群算法继

续寻优所导致的长时间的等待，那么使用当前粒子群选出的最优方案或者使用参照最优服务组合方案 CS_{ref} 都可以。

基于以上分析和设计，下面给出用于服务组合的 IDPSO 算法描述。

算法 1. IDPOS 算法

输入： PS, $T_{\max}$，c_1，c_2, w, V

输出： P_{gbest} /*全局最优粒子的位置*/

1. /*初始化 PS，c_1，c_2 */
2. /*为粒子群设置维度*/
3. /*根据最大权重的 QoS 属性对服务候选服务集 *CSS* 进行降序排序*/
4. :Randomly initialize V,X /*随机初始化 V, X, X 为粒子的位置矩阵*/
5. Compute *fitness*/*计算适应度值 fitness*/
6. Set P_{ibest}，P_{gbest} according to *fitness*/*根据 *fitness* 填充 P_{ibest} 和 P_{gbest} 的值，P_{ibest}，P_{gbest} 分别为粒子个体历史最优位置和粒子全局历史最优位置*/
7. Set $t_{pre}=1$
8. While (iterations <= $T_{\max}$)
   8.1. If (t_{pre} = set count)
      8.1.1. Randomly initialize V, X
   8.2. Else
      8.2.1. Update c_1，c_2, w/*更新 c_1，c_2, w*/
      8.2.3. Update V
   8.2.4. Update X
   8.3. Compute *fitness*
   8.4. Update P_{ibest}，P_{gbest} according to fitness
   8.5. $t_{pre}++$
9. Compute $fitness_{ref}$ /*为参照最优方案的适应度*/
10. If ($fitness_{ref} < fitness_g$)/*为循环语句给出的全局最优方案的适应度值*/
    10.1. return P_{ref} /*参照最优位置*/
11. Else
      11.1. return P_{gbest}

5. 实验设计与结果分析

5.1. 实验数据与环境

本文实验中单元服务部分采用 Zeng 等的 QWS 公共数据集[25] [26]。在本文实验中，本文分别测试不同的子任务数，并将 QWS 数据集上的服务分别分配到这些子任务上，用来作为子任务的候选服务集，本文设定每个子任务的候选集包含 100 条单元服务。本文选择数据集中的 4 个属性以及用仿真生成工具插入数据集中作为补充的 2 个属性总共 6 个属性作为本文 QoS 评价指标属性，即响应时间(r)，可用性(a)，吞吐量(t)，可靠性(rel)，服务价格(p)，信誉度(rep)，其中前面 4 个为 QWS 数据中原有的属性，最后 2 个属性是仿真生成插入 QWS 数据中的属性，数据集的特征如表 1 所示。设置 $weight(r,a,t,rel,p,rep)=(0.2,0.3,0.1,0.1,0.2,0.1)$ 为客户偏好。本文的实验环境为 windows 7 OS, Intel(R)

Core(TM) i5 3230 CPU @ 3.60 GHZ, 12 GB RAM, 64 位操作系统，采用的实验工具为 eclipse + pyDev2.7 + python 3.0。

Table 1. Characteristics of atomic services in the service
表 1. 服务数据集中单元服务的特征

Total	QoS attributes & Value range
2000	*r* (1 − 5000), *a* (1 − 100), *t* (1 − 10), *rel* (1 − 100), *p* (1 − 100), *rep* (1 − 10)

5.2. 实验参数设置

本文根据不同子任务数设置一些组实验，第一组实验的子任务数为 2，其他组实验的子任务数逐次加 2，最后一组实验的子任务数为 20，共 10 组实验，每组实验均进行 50 次，实验最终结果为 50 次实验的平均值，10 组实验的具体参数设定如表 2 所示。其中，学习因子的 c_1， c_2 值根据 4.2.2 的相关概念由算法动态设定，此处给出了手动设置的初值，惯性权重 *w* 根据公式(3)由算法动态改变，此处根据公式(3)给出了初值，粒子速度 *V*、粒子位置 *X* 根据公式(1)、(2)由算法动态改变，此处由随机生成器给出初值，符号 *ram* 代表随机值。另外，本实验的每一组实验中其他对照算法的粒子数或个体数以及迭代次数或循环次数都与本文提出的 IDPSO 算法完全一样。

Table 2. Parameter value setting under different number of subtasks
表 2. 不同子任务数下参数值的设置

TasksNum	Parameters							
	PST	*T*	t_{pre}	c_1	c_2	*w*	*V*	*X*
2	4	4	2	100	100	0.7750	*ram*	*ram*
4	50	50	10	100	100	0.8900	*ram*	*ram*
6	100	100	20	100	100	0.8950	*ram*	*ram*
8	250	250	50	100	100	0.8980	*ram*	*ram*
10	500	500	100	100	100	0.8990	*ram*	*ram*
12	700	700	140	100	100	0.8993	*ram*	*ram*
14	900	900	180	100	100	0.8994	*ram*	*ram*
16	1000	1000	200	100	100	0.8995	*ram*	*ram*
18	1500	1500	300	100	100	0.8997	*ram*	*ram*
20	2000	2000	400	100	100	0.8998	*ram*	*ram*

5.3. 实验结果分析

在本文中，本文将改良的离散型粒子群算法 IDPSO 与 OPSO [20]、FWPSO [27]、ZP-PSO [28]、S-ABC [16]等算法进行对比，来验证本文提出算法的性能，本文将从算法的最优性(Optimality)、收敛性(Convergence)以及时间复杂度(Time Complexity)去衡量各个算法的性能，其中 FWPSO、ZP-PSO 是目前性能比较优良的粒子群算法，S-ABC 算法是性能优良的人工蜂群算法，本实验中相同子任务数下的各个算法的迭代次数和个体数相等，具体分析情况如下。

5.3.1. 最优性

最优性 *Optimality* 反映算法的寻优能力的高低，即算法给出的服务组合方案在满足用户 QoS 限制的前提下其满足服务组合的质量有多高，本文中算法选择出的服务方案的适应度值 *fitness* 越小代表其最优性越优，最优性公式如公式(7)所示。其中同一子任务数下，最优算法适应度值保留最大有效位后的数，显然 *Optimality* 的值越小，其最优性越高。为验证本文提出的算法在 Web 服务组合选择领域的最优性，每个子任务可选择的候选单元服务数为 100 个，考察不同的子任务下，五种算法寻找最优解决方案的能力。具体实验结果如表 3 以及图 4 所示。

$$Optimality = fitness \tag{7}$$

Table 3. Comparison of optimality values of different algorithms under different number of tasks
表 3. 不同任务数下不同算法的最优性值对比

TaskNum	IDPSO	FWPSO	OPSO	ZP-PSO	S-ABC
2	**0.018623**	0.028340	0.048990	0.045560	0.033410
4	**0.023804**	0.035786	0.051323	0.048970	0.038906
6	**0.036513**	0.052156	0.072334	0.071823	0.056318
8	**0.044916**	0.053987	0.072674	0.072481	0.057911
10	**0.048867**	0.055823	0.072563	0.073652	0.059221
12	**0.051596**	0.057389	0.079865	0.076673	0.060945
14	**0.051529**	0.059784	0.083707	0.078875	0.063467
16	**0.058463**	0.061465	0.086237	0.081768	0.065797
18	**0.065115**	0.067316	0.089654	0.086387	0.077837
20	**0.069979**	0.073837	0.092345	0.089124	0.080346

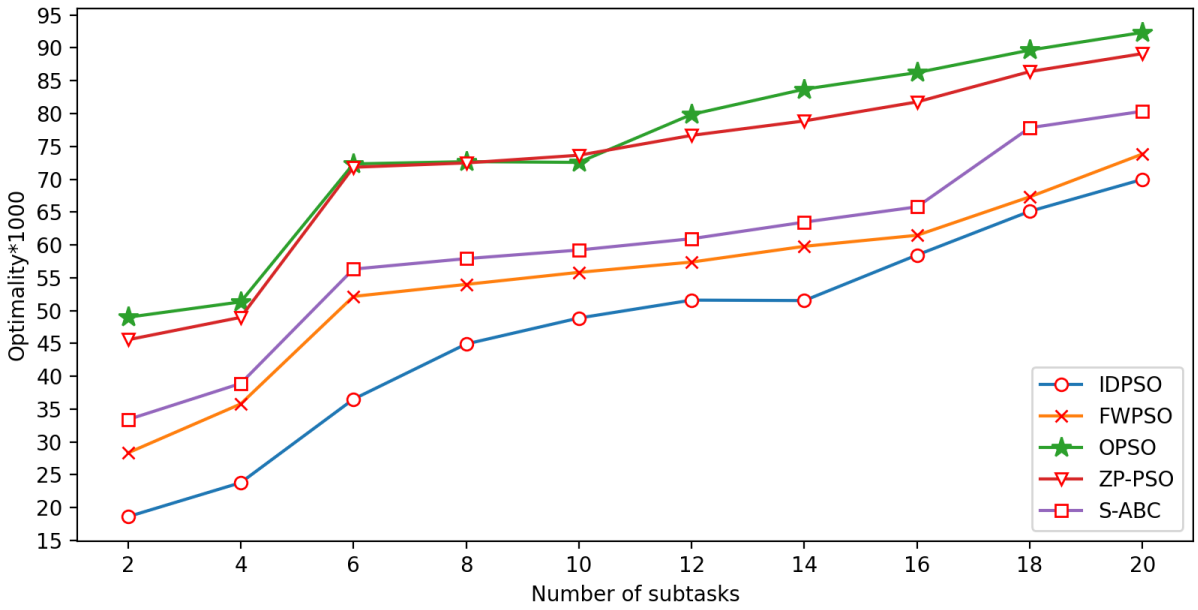
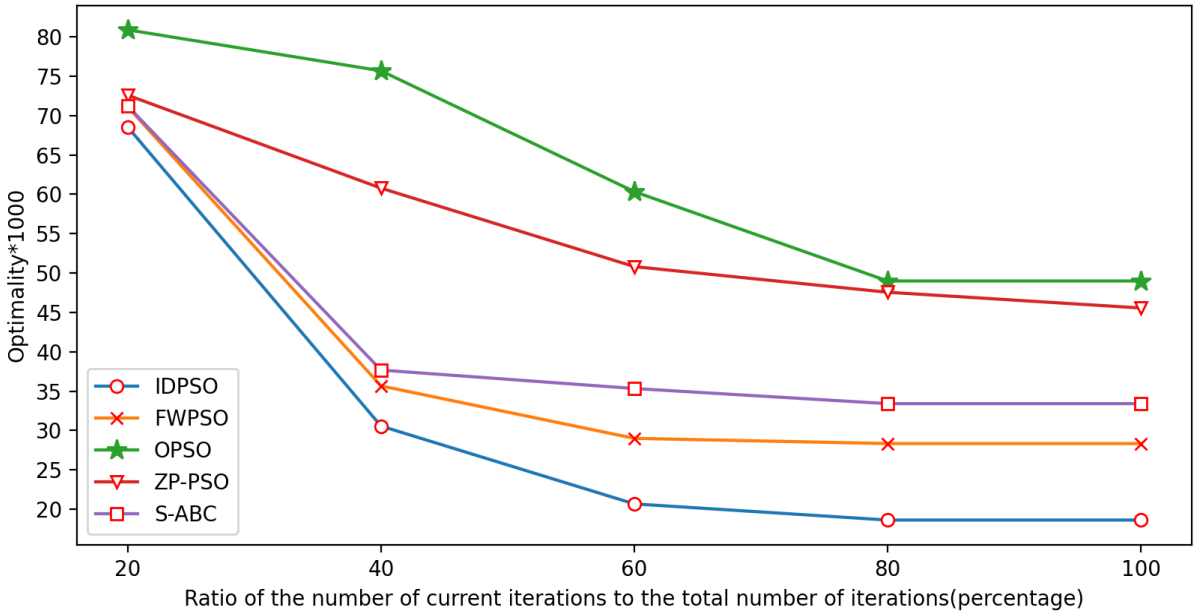


Figure 4. Comparison of optimality values of different algorithms under different number of tasks
图 4. 不同任务数下不同算法的最优性值对比

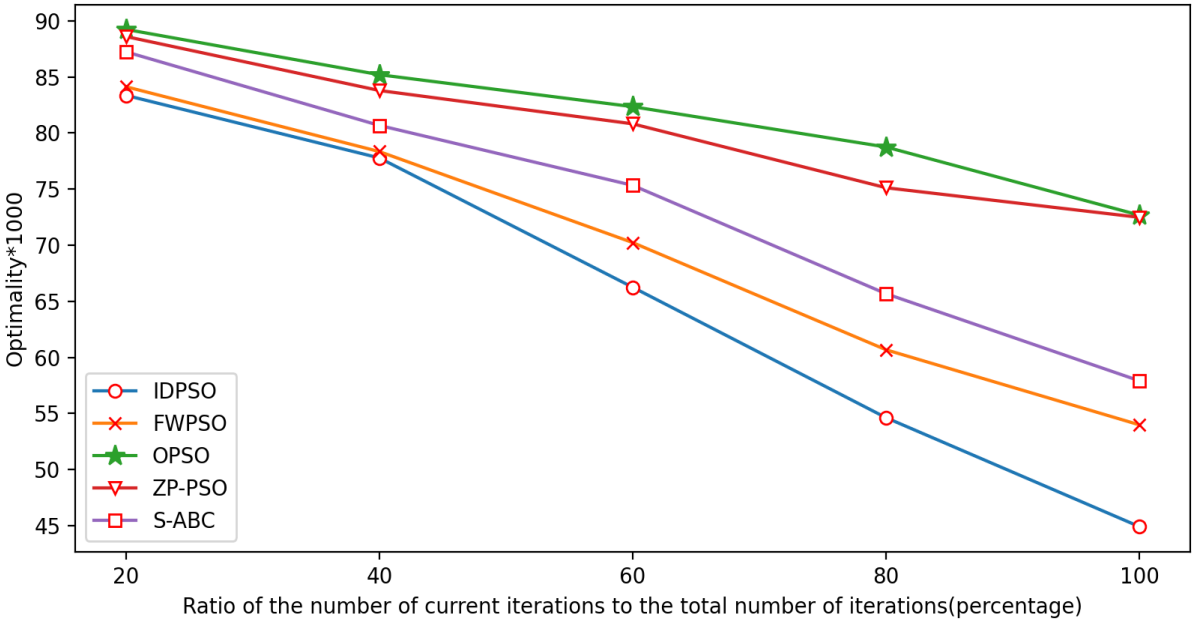
从表 3 和图 4 中可以看出，在测试的不同的子任务实验中，IDPSO 算法所找到的服务组合方案的适应度值最小，即适应度值最优，这是因为本文设置的预防早熟机制对最优性起到了很大的作用，是本文实验中所有算法寻优性能中最高的。

5.3.2. 收敛性

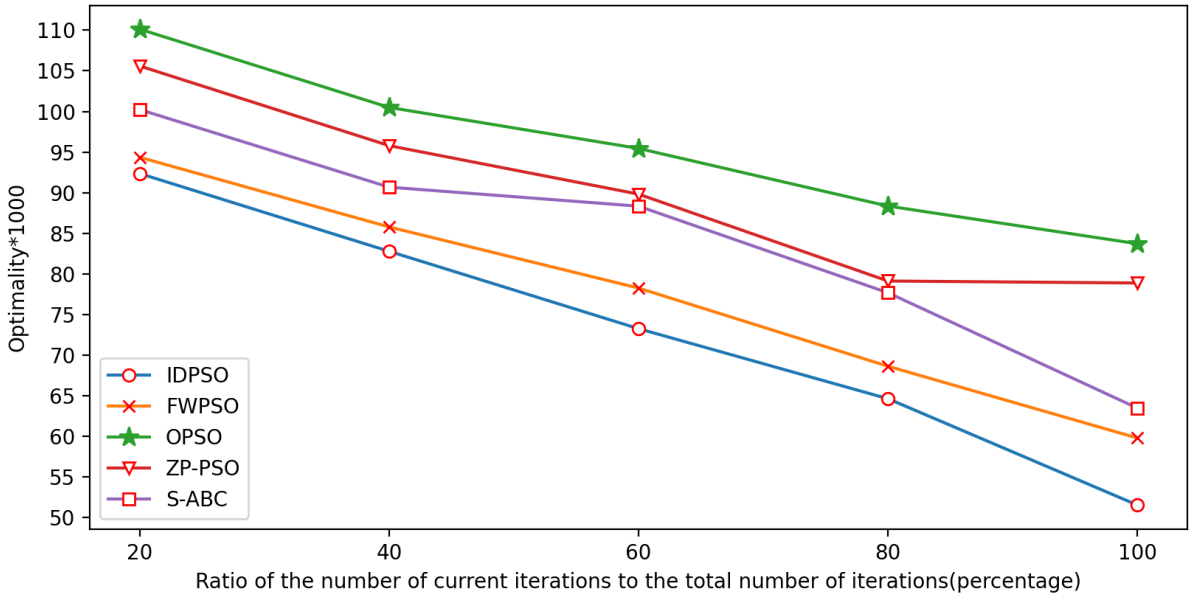
收敛性 Convergence 反映算法寻优的速度快慢，本实验主要分析对比各个算法在不同子任务数下，迭代次数对服务组合的影响。相同子任务数下的各个算法的迭代次数和个体数相等，实验结果如图 5 所示。



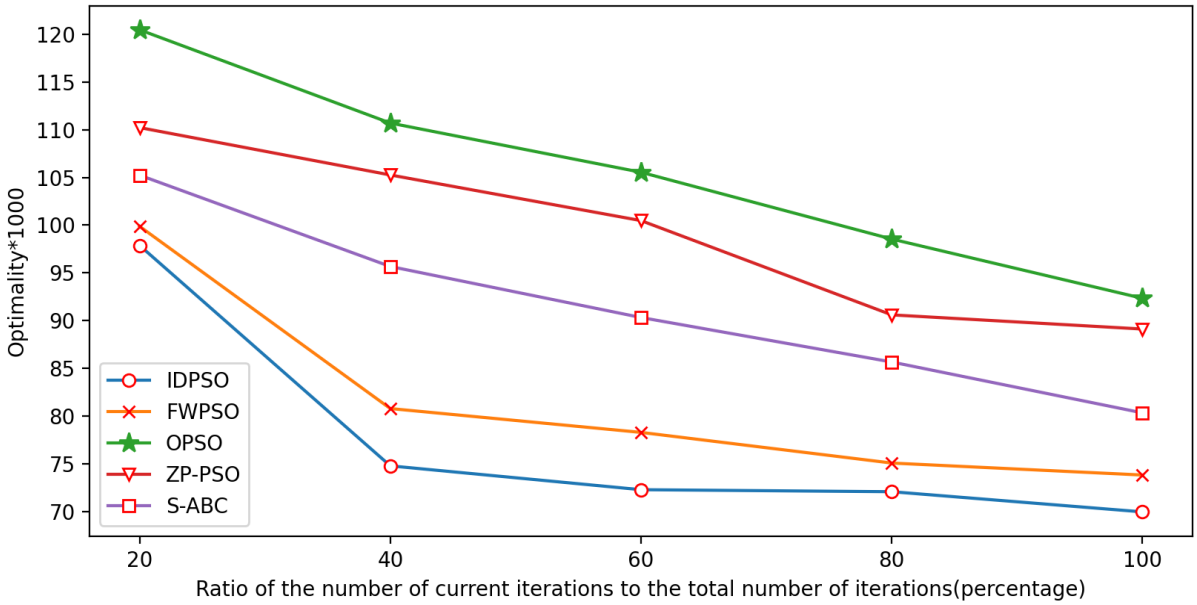
(a) Influence of the number of iterations on optimality under 2 subtasks



(b) Influence of the number of iterations on optimality under 8 subtasks



(c) Influence of the number of iterations on optimality under 14 subtasks



(d) Influence of the number of iterations on optimality under 20 subtasks

Figure 5. Influence of the number of iterations on optimality under different number of subtasks
图 5. 不同子任务数下，迭代次数对最优性的影响

从图 5 中可以看出，在不同的子任务数下，各种算法所找到的平均最优适应度值随着迭代次数的增加呈现出不同的变化趋势。无论子任务数为多少，IDPSO 算法收敛速度最快，这是因为本文设置的预防早熟机制对收敛性起到了很大的作用，因此 IDPSO 算法是本文实验中所有算法收敛速度中最高的。

5.3.3. 时间复杂度

时间复杂度 Time Complexity 为优化算法在相关参数设置一定的情况下，其算法的执行时间，时间复杂度反映算法在时间上的性能，本文将每个子任务对应的候选集设置为 100 个单元服务，在不同的子任

务数下分别进行实验，在每组实验中，考察不同算法的时间复杂度即执行时间的多少。实验结果如表 4 和图 6 所示，其中执行时间单位为秒(s)。

Table 4. Comparison of time complexity of different algorithms under different number of subtasks
表 4. 不同子任务数下不同算法的时间复杂度对比

TaskNum	IDPSO	FWPSO	OPSO	ZP-PSO	S-ABC
2	0.0301	0.0311	**0.0289**	0.0356	0.0303
4	0.1794	0.1935	**0.1651**	0.1865	0.1806
6	0.9987	1.1265	**0.8976**	1.0031	1.0024
8	8.2725	9.2365	**7.9348**	8.6554	8.4336
10	30.4751	40.3354	**28.6791**	33.5643	32.6574
12	79.2681	99.32137	**5.238**	780.2145	81.3367
14	149.3201	190.8931	**140.8879**	160.3576	157.9017
16	220.0475	265.3461	**201.9341**	233.9001	231.9077
18	493.1813	601.3267	**460.5609**	540.2231	518.7039
20	1198.967	1438.896	**1003.89**	71289.790	1220.908

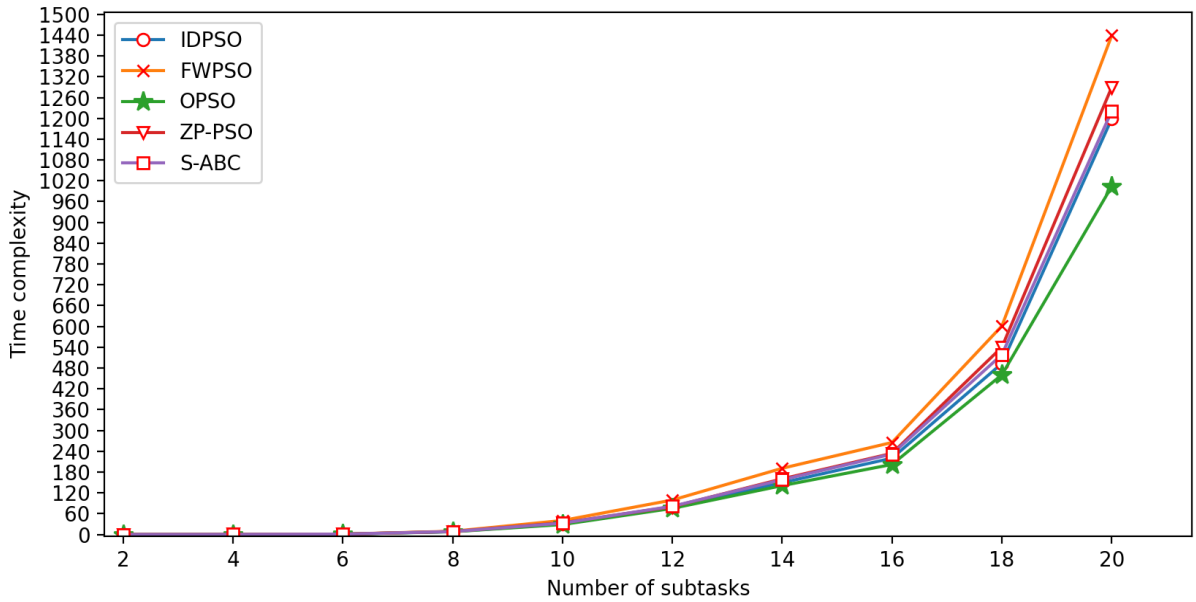


Figure 6. Comparison of time complexity of different algorithms under different number of subtasks
图 6. 不同子任务数下不同算法的时间复杂度对比

从表 4 和图 6 可以看出 OPSO 算法的时间性能是最好的，这是因为 OPSO 算法是最普通的粒子群算法，没有加入防止陷入局部最优机制，所以 OPSO 算法时间性能最高。但是在其他的几个算法中 IDPSO 算法的时间性能是最高的，这表明 IDPSO 算法的时间性能具有良好的表现。

6. 结论

本文针对服务组合问题提出了改良粒子群服务组合算法，该算法根据服务组合问题的特点从对候选服务集进行排序、合理设定控制参数、一定迭代次数后重置粒子位置、设置参照最优值四个方面来预防粒子早熟。实验表明本文所提出的算法在最优性、收敛性、时间复杂度三个方面都表现出良好的性能。但是，本文所提出算法的控制参数的设定并没有实现自动化的设定，此外，本文所提出的算法时间性能并没有明显的优势。未来，本研究打算使用机器学习的相关技术去实现算法控制参数的自动化设定，然后使用大数据的相关技术从历史服务方案中发掘先验知识去指导服务组合，企图降低算法的时间复杂度。

基金项目

国家重点研发计划资助“科技服务协同技术及平台研发”(编号 2018YFB1402900)。

参考文献

[1] Vouk, M.A. (2008) Cloud Computing—Issues, Research and Implementations. *Journal of Computing and Information Technology*, **16**, 235-246.

[2] Sailer, J. (2014) M2M-Internet of Things-Web of Things-Industry 4.0. *e & i Elektrotechnik und Informationstechnik*, **131**, 3-4. https://doi.org/10.1007/s00502-013-0191-8

[3] AbualIgah, L., Dlabat, A., Sumari, P., *et al*. (2021) Applications, Deployments, and Integration of Internet of Drones (IoD): A Review. *IEEE Sensors Journal*, **21**, 25532-25546. https://doi.org/10.1109/JSEN.2021.3114266

[4] Momeni, K. (2021) Service Integration: Supply Chain Integration in Servitization. In: Marko, K., Tim, B., Rodrigo, R., *et al*., Eds., *The Palgrave Handbook of Servitization*, Palgrave Macmillan, Cham, 471-485. https://doi.org/10.1007/978-3-030-75771-7_30

[5] Zhang, Z.T. (2020) WITHDRAWN: Big Data Service in Distributed Network Environment Based on FPGA. *Microprocessors and Microsystems*, **79**, Article ID: 103586. https://doi.org/10.1016/j.micpro.2020.103586

[6] Falch, M., Williams, I. and Tadayoni, R. (2020) Cross-Border Provision of E-government Business Registration Service. *ITS Online Event*, Online, 14-17 June 2020, 1-27. http://hdl.handle.net/10419/224852

[7] Klai, K. and Ochi, H. (2016) Model Checking of Composite Cloud Services. 2016 *IEEE International Conference on Web Services* (*ICWS*), San Francisco, 27 June-2 July 2016, 356-363. https://doi.org/10.1109/ICWS.2016.53

[8] Huo, Y., Zhuang, Y., Gu, J., *et al*. (2015) Discrete Gbest-Guided Artificial Bee Colony Algorithm for Cloud Service Composition. *Applied Intelligence*, **42**, 661-678. https://doi.org/10.1007/s10489-014-0617-y

[9] Maâtouk, O., Ayadi, W., Bouziri, H., *et al*. (2021) Evolutionary Local Search Algorithm for the Biclustering of Gene Expression Data Based on Biological Knowledge. *Applied Soft Computing*, **104**, Article ID: 107177. https://doi.org/10.1016/j.asoc.2021.107177

[10] Cook, W., Held, S. and Helsgaun, K. (2021) Constrained Local Search for Last-Mile Routing. arXiv.2112.15192.

[11] Kurokawa, S. and Matsui, T. (2021) Dynamic Programming and Linear Programming for Odds Problem. arXiv:2107.13146.

[12] Liu, C., Wan, Z., Liu, Y., *et al*. (2021) Trust-Region Based Adaptive Radial Basis Function Algorithm for Global Optimization of Expensive Constrained Black-Box Problems. *Applied Soft Computing*, **105**, Article ID: 107233. https://doi.org/10.1016/j.asoc.2021.107233

[13] Gao, Z.M., Zhao, J., Hu, Y.R, *et al*. (2021) The Challenge for the Nature-Inspired Global Optimization Algorithms: Non-Symmetric Benchmark Functions. *IEEE Access*, **9**, 106317-106339. https://doi.org/10.1109/ACCESS.2021.3100365

[14] Abualigah, L., Yousri, D., Elaziz, M.A, *et al*. (2021) Aquila Optimizer: A Novel Meta-Heuristic Optimization Algorithm. *Computers & Industrial Engineering*, **157**, Article ID: 107250. https://doi.org/10.1016/j.cie.2021.107250

[15] Abualigah, L., Elaziz, M.A., Sumari, P., *et al*. (2021) Reptile Search Algorithm (RSA): A Nature-Inspired Meta-Heuristic Optimizer. *Expert Systems with Applications*, **191**, Article ID: 116158. https://doi.org/10.1016/j.eswa.2021.116158

[16] Liu, R., Wang, Z. and Xu, X. (2019) Parameter Tuning for S-ABCPK: An Improved Service Composition Algorithm Considering Priori Knowledge. *International Journal of Web Services Research*, **16**, 88-109. https://doi.org/10.4018/IJWSR.2019040105

[17] Kashyap, N., Kumari A.C. and Chhikara R. (2021) Service Composition in IoT Using Genetic Algorithm and Particle

Swarm Optimization. *Open Computer Science*, **10**, 56-64. https://doi.org/10.1515/comp-2020-0011

[18] Mabrouk, N.B., Beauche, S., Kuznetsova, E., *et al*. (2009) QoS-Aware Service Composition in Dynamic Service Oriented Environments. *Middleware* 2009, Urbana, 30 November-4 December 2009, 123-142. https://doi.org/10.1007/978-3-642-10445-9_7

[19] Jin, H., Yao, X. and Chen, Y. (2017) Correlation-Aware QoS Modeling and Manufacturing Cloud Service Composition. *Journal of Intelligent Manufacturing*, **28**, 1947-1960. https://doi.org/10.1007/s10845-015-1080-2

[20] Wen, T., Sheng, G.J., Quan, G., *et al*. (2017) Web Service Composition Based on Modified Particle Swarm Optimization. *Chinese Journal of Computers*, **36**, 1031-1046. https://doi.org/10.3724/SP.J.1016.2013.01031

[21] Chen, Y., Huang, J. and Lin, C. (2014) Partial Selection: An Efficient Approach for QoS-Aware Web Service Composition. 2014 *IEEE International Conference on Web Services*, Anchorage, 27 June-2 July 2014, 1-8. https://doi.org/10.1109/ICWS.2014.14

[22] Zhan, Y., Jing, Z. and Zhang, Y. (2015) MR-IDPSO: A Novel Algorithm for Large-Scale Dynamic Service Composition. *Tsinghua Science and Technology*, **20**, 602-612. https://doi.org/10.1109/TST.2015.7349932

[23] Zhang, Y., Gui, G., Yan, Y., *et al*. (2019) Quality Constraints-Aware Service Composition Based on Task Granulating. *Journal of Computer Research and Development*, **55**, 1345-1355. https://doi.org/10.7544/issn1000-1239.2018.20170234

[24] Eberhart, R. and Kennedy, J. (1995) A New Optimizer Using Particle Swarm Theory. *Mhs*'95 *Sixth International Symposium on Micro Machine & Human Science*, Nagoya, 4-6 October 1995, 39-43. https://doi.org/10.1109/MHS.1995.494215

[25] Zeng, L., Benatallah, B., Dumas, M., *et al*. (2003) Quality Driven Web Services Composition. *Proceedings of the* 12*th International Conference on World Wide Web*, Budapest, 20-24 May 2003, 411-421.

[26] Zeng, L., Benatallah, B., Ngu, A., *et al*. (2004) QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, **30**, 311-327. https://doi.org/10.1109/TSE.2004.11

[27] Guo, X., Chen S.S., Zhang, Y.W., *et al*. (2019) Application of Fireworks Particle Swarm Optimization Algorithm in Web Service Composition. *Journal of Chinese Computer Systems*, **39**, 1312-1316. https://doi.org/10.3969/j.issn.1000-1220.2018.06.035

[28] Kashyap, N., Kumari, A.C. and Chhikara, R. (2021) Service Composition in IoT Using Genetic Algorithm and Particle Swarm Optimization. *Open Computer Science*, **10**, 56-64. https://doi.org/10.1515/comp-2020-0011